Nets of Polyhedra

Netze von Polyedern

Diplomarbeit

Wolfram Schlickenrieder

Technische Universität Berlin

e-mail: `schlicke@math.tu-berlin.de`

16. Juni 1997

Abstract

In 1525, the painter *Albrecht Dürer* introduced the notion of a *net* of a polytope, and published nets for some of the Platonian and Archimedian polyhedra, along with directions about how to construct them.

An *unfolding* of a 3-dimensional polytope P is obtained by cutting the boundary of P along a collection of edges that spans the vertex set of P and then flattening the remaining set to a polygon in the plane. An unfolding is a *net* if it does not overlap itself. Conversely, a simple connected plane polygon with specific *folding lines* is a net, if it is possible to fold it into (the boundary of) a polytope.

We consider the question whether every 3-dimensional polytope has a net. Although the problem is intuitive and easy to state, and there are nets known for all regular and uniform polytopes, in general it is still unsolved.

After giving an overview of related questions and conjectures about the nature or existence of nets for 3-polytopes, we present an account of our experiments with a number of different "unfolding rules" and their results. Some of the experiments produced counterexamples to (up to this point) open conjectures about how nets can be constructed. Although all except one of the unfolding rules we tested produce overlapping unfoldings for *some* given polytopes, we still present most of them: First, in order to give an idea of how different kinds of nets look like; second, to show that despite being easy to state, the problem seems to be of the "old, unsolved and wicked" type; and finally, to provide a compilation of unfolding methods that do *not* work, so that others are exempted from inventing them again.

The best unfolding rule we found (that is, the one that is empirically least likely to produce self-overlapping unfoldings) essentially cuts the boundary of P along paths which are in a sense "as straight as possible": At each vertex, cut along the steepest ascending edge w.r.t. a certain objective function c.

This approach does not always produce a net. However, experimenting with different (deterministic as well as randomized) choices of c, we found strong empirical evidence that for every polyhedron P there exists an objective function c_P such that this rule *does* produce a net.

Contents

Acknowledgements

I would like to express my appreciation to those who (intentionally or unintentionally) contributed to this thesis.

First of all, thanks to Günter M. Ziegler for bringing up the topic of nets, for many useful comments and suggestions and for his patience. Komei Fukuda posed some challenging conjectures that found their way into this document. Technical thanks also go to the *polymake* surgeon team — Ewgenij Gawrilow & Michael Joswig; special credits go to Ewgenij Gawrilow who accidentally provided me with a name for the "turtle polytopes". Makoto Namiki & Günter Rote kindly allowed me to steal pictures from their publications and reproduce them here.

I also want to thank my parents for their support.

Last, but certainly not least: Thank you, Gabriele, for your encouragement and support, for debugging my mind like nobody else could, and for making the SHUTTLE succeed.

Wolfram Schlickenrieder
Berlin, June 16th, 1997

"Experiment has always been, and increasingly is,
an important method of mathematical discovery. [...]
Yet this tends to be concealed by the tradition of
presenting only elegant, well-rounded and rigorous results."[1]

1 Introduction

If you have ever tried to make a model of a cube from paper or similar material, you already have an idea of what a net of a polyhedron is: Informally, take the surface of a polyhedron and cut it open along edges in such a way that you can open it out in one piece and flatten it out in the plane. If this *unfolding* doesn't overlap, it is a *net*. Figure 1(a) shows the well-known "cross" net of a cube, Figure 1(b) shows a less often published net.

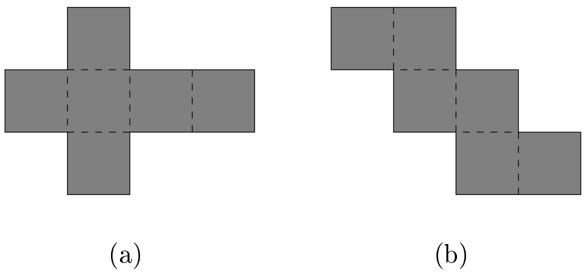


Figure 1

The idea of a net is a fairly old one; in 1525 Albrecht Dürer introduced the concept of a net of a polyhedron in his Book "*Unterweysung der Messung mit dem Zyrkel und Rychtscheydt*"[2]) [11]. Nonetheless Shephard [38] seems to be the first to have explicitly posed the following question:

[1]D. Epstein, S. Levy, R. de la Llave: *Experimental Mathematics; Statement of philosophy and Publishability Criteria* [15].

[2]The title could be translated to contemporary English as "Instructions for Measuring with Compass and Ruler"; there is an English translation — "The Painter's Manual" — with a commentary by W.L. Strauss.

(A) Does *every* polytope have a net?

The particularly interesting aspect of the problem is that which mathematicians and non-mathematicians alike have always been fascinated by: it is an "intuitive" geometrical problem in the sense that it is very easy to state and the answer to Shephard's question is "obviously" yes. Apart from this, there is considerable empirical evidence supporting a positive answer to (A): Up to now, no one has been able to construct a convex polytope which does *not* have a net.

On the other hand, there have been similar questions with equally "obvious" answers, the most closely related of which probably is Cauchy's *angle opening lemma*:

> If one transforms a 2-dimensional convex polygon Q into another convex polygon Q', such that all but one side length is fixed and one or more of the angles not incident to this side increase, then the last side length must increase (see Figure 2).

However, as "obviously" correct as this may seem, it wasn't easy to prove; Cauchy's original proof was incorrect and was later corrected by Steinitz [28].

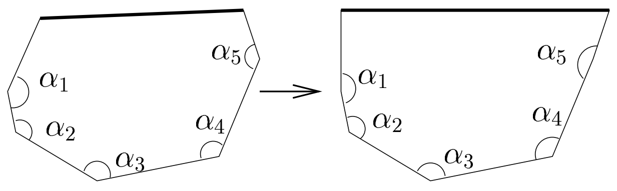


Figure 2: Cauchy's *angle opening lemma*: If the lengths of the thin edges are fixed, at least one of the angles $\alpha_1, \ldots, \alpha_5$ increases and P as well as P' are convex, then the thick line must increase in length.

Finding a (1-dimensional) net of a 2-polytope (i.e. unfold the boundary of a polygon, such that the unfolding is 1-dimensional) is a simple task: Cut the boundary of the polygon at a vertex and unfold the edges to a single straight line segment.

In three dimensions this task turns out to be considerably harder; the difficulties start with trying to imagine what a net of a specific polytope looks

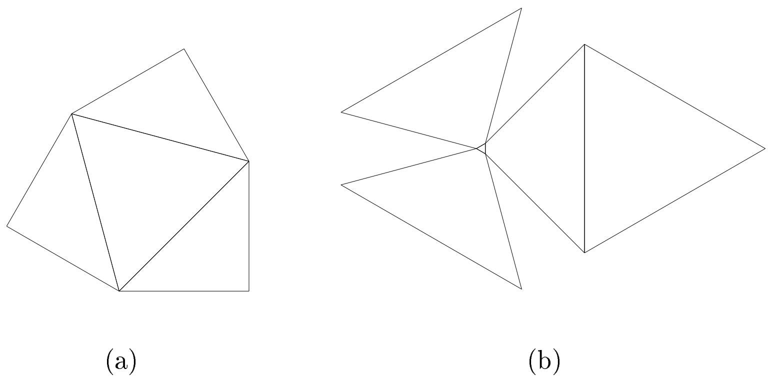
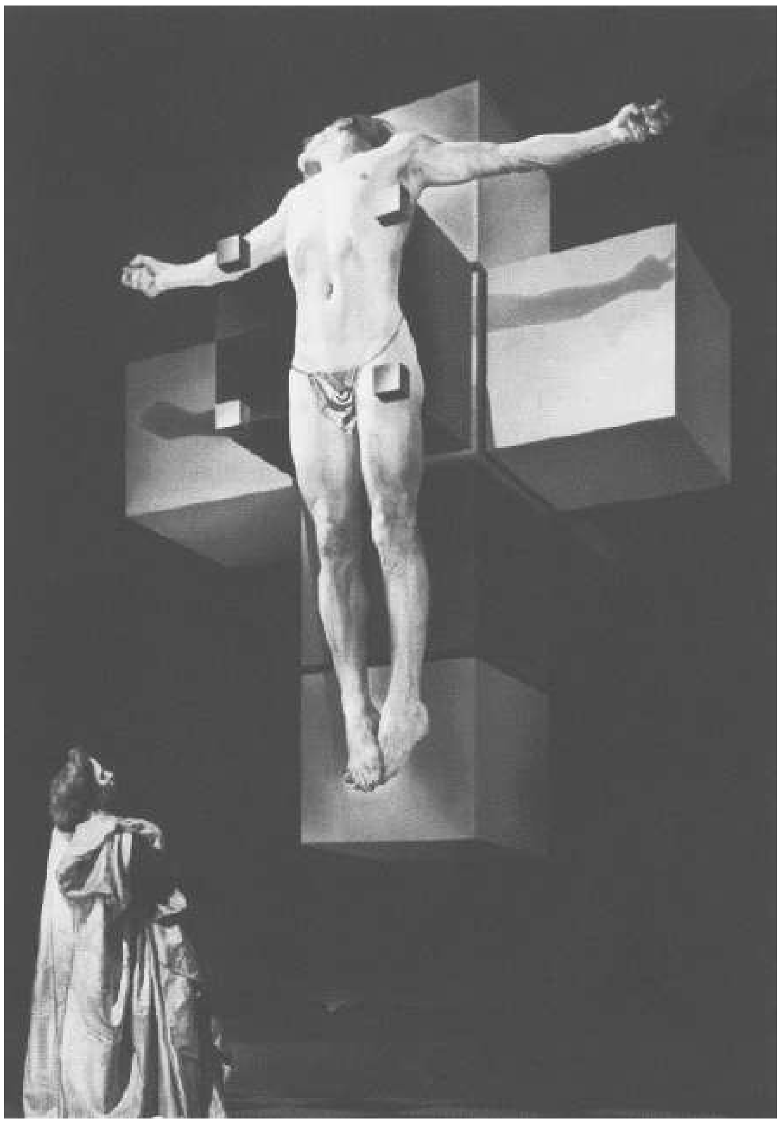


Figure 3: The net in (a) is a net of a tetrahedron. After cutting off a vertex, the net changes dramatically (b). Both nets were obtained by FLAT-SPANNING-TREE; see section 4.7.3.

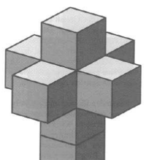
like — that is, unless the polytope is as familiar as a tetrahedron or a cube. Also, unlike in the 2-dimensional case, small local changes in the polytope (such as cutting off a vertex, or small deformations) may result in remarkable changes in the resulting net; the nature of the change of course depends on the unfolding method (see Figure 3 for an example).

The generalization of the concept of unfoldings and nets to dimensions higher than 3 is straightforward. Several (2-dimensional projections of) 3-dimensional nets for special 4-polytopes have been published (see Figure 4 or Banchoff's "Beyond the Third Dimension" [3] for example); apart from that, to our knowledge the problem of finding nets for higher-dimensional polytopes has not been systematically investigated as yet.

Several other questions and conjectures about the existence or special properties of nets for a given polytope — and vice versa — have been posed. We will present them in Section 3, along with answers, as far as they are known. Section 4 shows several ways to obtain an unfolding, many of which we conjectured to produce a net for any given polytope. Their implementa-



(a)



(b)

Figure 4: *Crucifixion — Corpus Hypercubicus* (Dali, 1954). The underlying structure of the "cross" is a (3-dimensional) net of a 4-dimensional hypercube (b).

tion disproved all of them — except for one, which we conjecture to produce a net for any given polytope, supported by considerable empirical evidence.

2 Preliminaries

Before we start, let's define the objects being addressed in this thesis:

Definition 2.1 (Hulls)
A set $M \subseteq I\!R^d$ is convex, if for any two points $x, y \in M$ the connecting straight line segment

$$[x, y] = \{\lambda x + (1 - \lambda)y \mid \lambda \in I\!R, 0 \leq \lambda \leq 1\}$$

is entirely contained in M. (See Figures 5 & 6(a).)

For n points $x_1, \ldots, x_n \in I\!R^d$, the convex hull is

$$\mathrm{conv}(x_1, \ldots, x_n) = \Big\{\lambda_1 x_1 + \ldots + \lambda_n x_n \,\Big|\, \lambda_i \geq 0, \sum_{i=1}^{n} \lambda_i = 1\Big\} \text{ (see Figure 6(b)).}$$

Or, alternatively: The convex hull of a set of points is the smallest convex set containing these points [32].

The affine hull of n points $x_1, \ldots, x_n \in I\!R^d$ is

$$\mathrm{aff}(x_1, \ldots, x_n) = \Big\{\lambda_1 x_1 + \ldots + \lambda_n x_n \,\Big|\, \sum_{i=1}^{n} \lambda_i = 1\Big\}.$$

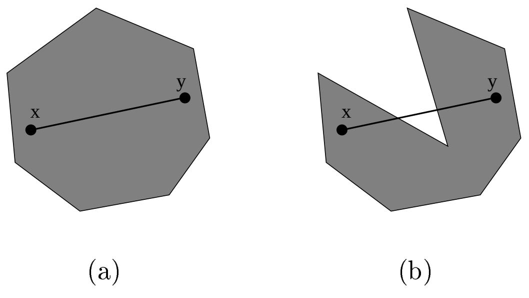


Figure 5: A convex (a) and a non-convex set (b).

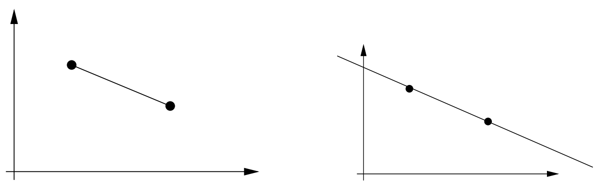

(a) (b)

Figure 6: Convex (a) and affine (b) hull of two points in $\mathbb{R}^2$

Definition 2.2 (Polyhedron, Polytope)
A halfspace in $\mathbb{R}^d$ is a set of the form

$$\{x \in \mathbb{R}^d \mid ax \leq b; a \in \mathbb{R}^d, b \in \mathbb{R}\}.$$

A d-polyhedron in $\mathbb{R}^d$ is the intersection of a finite number of closed halfspaces in $\mathbb{R}^d$. If the resulting polyhedron is bounded, we call it a polytope. The dimension of a polyhedron is the dimension of its affine hull.

In this thesis we will almost exclusively deal with 3-polytopes in $\mathbb{R}^3$.

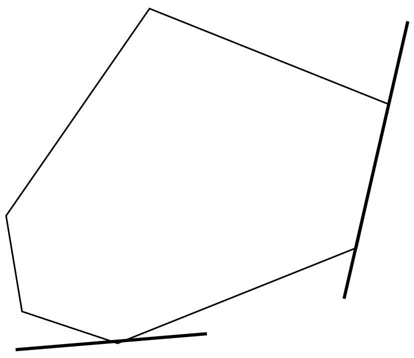
Definition 2.3 (Faces, Facets)
Let P be a polyhedron, $a \in \mathbb{R}^d$ $(a \neq 0)$, and $b \in \mathbb{R}$. $ax \leq b$ is a valid inequality for P, if $ax \leq b$ for all $x \in P$.

A subset $F \subseteq P$ is a face of P, if

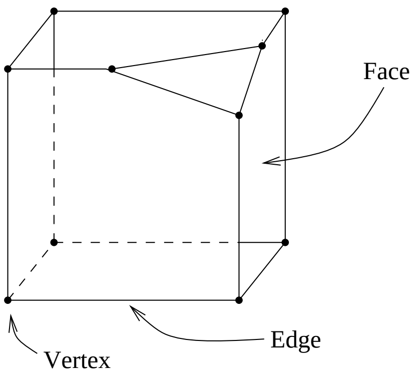
$$F = P \cap \{x \in \mathbb{R}^d | ax = b\},$$

where $ax \leq b$ is a valid inequality for P. The dimension of a face F is the dimension of the affine hull $\mathrm{aff}(F)$. The faces of dimension 0, 1, and $\dim(P) - 1$ are called vertices, edges and facets; $V(P)$, $E(P)$ and $F(P)$ denote the set of vertices, edges and facets of P, respectively.

Two vertices v and w of P are adjacent, if there is an edge $e = \mathrm{conv}\{v, w\} \in E(P)$. Two facets f and g of P are adjacent, if their intersection is an edge of both of them.



(a) Two valid inequalities for the shown polygon, which define a vertex and an edge.



(b)

Figure 7

The normal vector n_f of a facet $f \in F(P)$ is a vector with $||n_f|| = 1$, such that

$$\langle n_f, p - q \rangle = 0 \quad \textit{for any two points } p, q \in f,$$

and the closed line segment $[p, p + n_f]$ intersects P only in p.

Definition 2.4 (Equivalence)
Two polytopes P and P' are combinatorially equivalent, if there is a bijection φ between their faces that preserves the inclusion relation, i.e. for any face f of P, $\varphi(f)$ is a face of P', and for any two faces f_1 and f_2 of P with $f_1 \subset f_2$

$$\varphi(f_1) \subset \varphi(f_2) \textit{ holds.}$$

One of the first theorems being proved in many books on polyhedra is that every polyhedron as defined in Definition 2.2 can be obtained by taking the convex hull of a finite set of points:

Theorem 2.1 (see **McMullen, Shephard [29]**)
The convex hull of a finite set of points in $I\!R^d$ is a convex polytope; conversely, a convex polytope is the convex hull of a finite set of points.

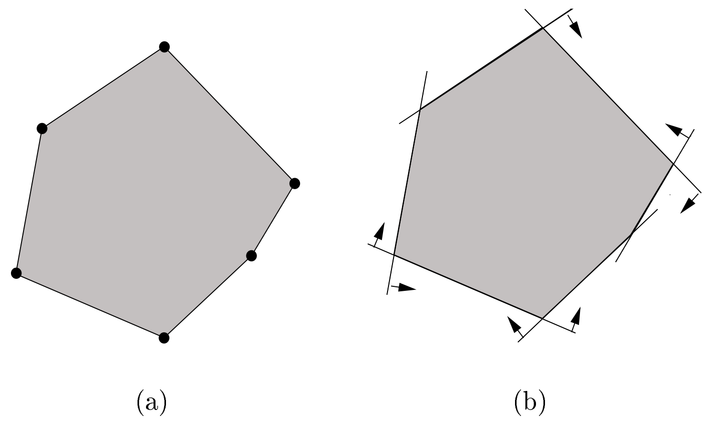


Figure 8: A 2-polytope — depicted as the convex hull of points (a), and as intersection of halfspaces (b). Arrows indicate on which side of the line the halfspace lies.

Two further structures that we will make frequent use of are the *graph* and *dual graph* of a polytope.

Definition 2.5 (Graph & dual graph of a polytope)
Let P be a 3-polytope. The graph

$$\begin{array}{rcl} G(P) & = & (V_G, E_G), \quad \textit{where} \\ V_G & = & \{v \mid v \in V(P)\}, \quad \textit{and} \\ E_G & = & \{(v,w) \mid v \;\&\; w \textit{ are endpoints of a common edge } e \in E(P)\} \end{array}$$

is the graph (or 1-*skeleton*) *of P. The graph*

$$\begin{array}{rcl} D(P) & = & (V_D, E_D), \quad \textit{where} \\ V_D & = & \{f \mid f \in F(P)\}, \quad \textit{and} \\ E_D & = & \{(f,g) \mid f \cap g \in E(P)\} \end{array}$$

is the dual graph of P.

The number of edges incident to a node v is called the degree $\gamma(v)$.

Obviously, there is a 1-to-1-correspondence between the edges of $G(P)$ and of $D(P)$: For each edge of P there is an edge e in $G(P)$ and a corresponding

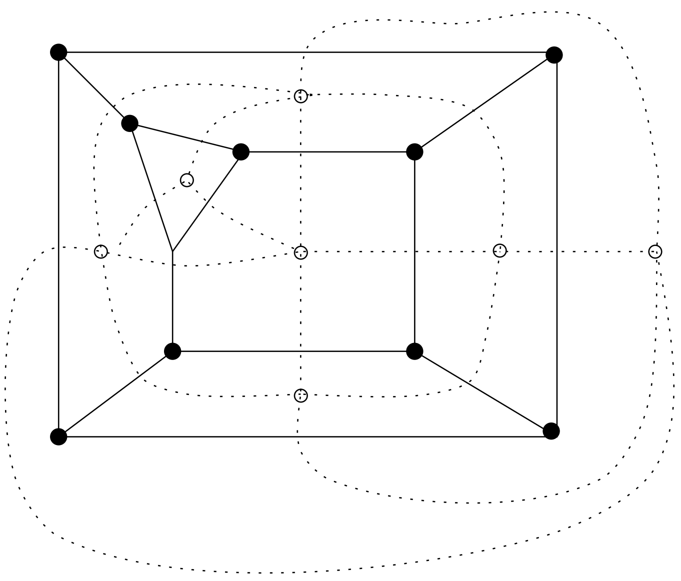


Figure 9: The graph (straight lines) and dual graph (dotted lines) of the polytope in Figure 7(b).

dual edge e^* in $D(P)$; thus, $(e^*)^* = e$. In general, for an edge e of $G(P)$ or $D(P)$, e^* denotes the dual edge of e.

Definition 2.6 (Planar embedding)
Let $G = (V, E)$ be a graph. A planar embedding *of G is a mapping of each node in V to a point in the plane and of each edge to a simple curve between the images of its vertices, such that no two images of the edges intersect except at their endpoints. A graph G is* planar, *if it has a planar embedding.*

If the images of the edges are straight line segments, the mapping is a straight line embedding.

Definition 2.7 (Connectedness)
A graph $G = (V, E)$ is connected, *if for every two non-adjacent nodes $v, w \in V$ there is a* path *$(v, x_1), (x_1, x_2), \dots, (x_s, w)$ in G. It is k-connected, if there are k such paths whose* inner *nodes are pairwise disjoint (that is, the only*

common nodes of the k paths are their endpoints). G is simple, if there is at most one edge between any pair of nodes and there are no loops (edges of the form (v, v)).

Steinitz provided a characterization of polytopal graphs:

Theorem 2.2 (Steinitz [39])
G is the graph of a 3-polytope if and only if it is simple, planar, and 3-connected.

Definition 2.8 (Tree, spanning tree)
A graph G contains a cycle, if there exists a sequence of pairwise disjoint edges $e_0, \ldots, e_k, e_{k+1} = e_0 \in E$ such that e_i is incident to $e_{i+1 \bmod n}$.

A spanning tree of a graph $G = (V, E)$ is a connected subgraph of G with vertex set V, which doesn't contain a cycle.

Now we will give a definition of the *net* of a 3-polytope: Informally, an *unfolding* of a 3-polytope is a mapping of its facets to $I\!R^2$, obtained by cutting along certain edges of P such that the remaining connected set can be flattened out in the plane along the remaining edges; we then call the unfolding a *net*, if it doesn't overlap itself.

As you may already have suspected, we will define it more formally:

Definition 2.9 (Polyhedral complex)
A polyhedral complex $\mathcal{C}$ is a finite collection of polyhedra in $I\!R^d$ such that

- *(i)* $\emptyset \in \mathcal{C}$,
- *(ii) if $P \in \mathcal{C}$, then all the faces of P are also in $\mathcal{C}$, and*
- *(iii) for all $P, Q \in \mathcal{C}$, the intersection $P \cap Q$ is a face of both P and Q (note that $\emptyset$ is a face of every polytope).*

The dimension $\dim(\mathcal{C})$ *is the largest dimension of a polyhedron in $\mathcal{C}$. If all polyhedra in $\mathcal{C}$ are bounded, $\mathcal{C}$ is a polytopal complex. The d-dimensional elements of $\mathcal{C}$ are called d-cells.*

Definition 2.10 (Boundary complex)
Let P be a polytope. The boundary complex $\mathcal{B}(P)$ is the collection of all faces of P of dimension at most $\dim(P) - 1$.

Definition 2.11 (Net)
Let N be a 2-dimensional polytopal complex in $I\!R^2$, together with a prescribed identification of some of its edges. We call N a net of a 3-polytope P if

(i) the union of the cells of N is a connected polygon,

(ii) no vertex (0-cell) of N is a cut-set of N, (i.e. for any two 2-cells $f, g \in N$, $f \cap g$ is either empty, or a 1-cell),

(iii) each edge (1-cell) which lies in only one 2-cell of N is identified with exactly one other such edge,

(iv) N is isomorphic to, and isometric with the boundary complex $\mathcal{B}(P)$ of P.

When unfolding a polytope, one has to cut at least one edge at each vertex in order to allow the incident faces to spread apart. The boundary of the unfolding consists of $2m$ edges, where m is the number of *cut edges*; each cut edge appears exactly twice on the boundary. Moreover, the cut edges form a spanning tree, the *cut tree*, of the polytope's graph.

Lemma 2.1
The set of cut edges for a net is a spanning tree of $G(P)$.

Proof: Since $G(P)$ is connected, it has a spanning tree. The set of cut edges cannot contain a cycle, since otherwise the resulting unfolding would not be connected. Since the boundary of the unfolding is a connected cycle of edges, the cut edges must form a connected graph, which is necessarily a tree, because it does not contain a cycle. Furthermore, since there has to be at least one cut edge at every vertex, the tree spans the vertex set of $G(P)$. $\square$

Lemma 2.2
On the other hand, given a net of a polytope, the set of join edges

$$T = \{e^* \in D(P) | e \textit{ is not a cut edge}\}$$

forms a spanning tree of $D(P)$.

Proof: Since we required the unfolding to be an edge-connected polygon, T has to be connected. Since all facets are unfolded, T has to span the vertex set of $D(P)$. Moreover, T cannot contain a cycle, since otherwise the set of *cut edges* would not be connected. Thus, T is a spanning tree of $D(P)$. □

Therefore, we can construct an unfolding from any given spanning tree of $G(P)$ or $D(P)$. For the experiments in Section 4 we also need a formal definition of an *unfolding*: What we will actually do is

(1) construct an unfolding and

(2) check for overlaps.

Definition 2.12 (Unfolding)
Let P be a polytope, T a spanning tree of $D(P)$. An unfolding is an isometric mapping $\phi : F(P) \to I\!R^2$ of the facets of P to the Euclidean plane, such that for all $(f_1, f_2) \in D(P)$, $\varphi(f_1) \cap \varphi(f_2)$ is an edge of both $\varphi(f_1)$ and $\varphi(f_2)$.
$N(P,T) = \varphi(F(P))$ is the unfolding of P induced by T.

If an unfolding of a polytope P does not contain any overlaps, we can apply the map φ to its boundary complex $\mathcal{B}(P)$, and the resulting polytopal complex $\mathcal{N} = \varphi(\mathcal{B}(P))$ is a net in the sense of Definition 2.11. Otherwise, if there *are* overlaps, the resulting complex is not polytopal, since some of its edges intersect in their relative interior, and thus, their intersection is not a facet of either of them.

Finally we introduce some notation:

Definition 2.13 (Neighbours)
Let f and g be two facets that are adjacent on P, and let e be the common edge of f and g. Then define the neighbour of f or g (w.r.t. e):

$$f^+(e) := g \quad \textit{and} \quad g^+(e) := f.$$

3 Overview

In this section we will present known answers to questions regarding nets of polyhedra, and related work being done.

3.1 Polytope unfolding

Countless books which show how to construct nets of commonly encountered polyhedra, like the Platonian or Archimedian solids have been published [37], [41]. However, Shephard seems to be the first who explicitly stated the following simple question [38]:

(A) Does every convex 3-polytope have a net?

To our knowledge, up to now nobody has constructed a convex polytope that does not have a net. On the other hand, in many cases there seems to be a way to cut "perversely" and obtain an overlapping unfolding; we will later present a few unfolding algorithms ("rules") which are likely to generate would-be nets. Even "nice" polytopes like the regular 12-sided prism allow an overlapping unfolding — although it is of course possible to construct a net (see Figure 10).

Actually, it seems to be even worse: Schevon found empirical evidence supporting the conjecture that "almost all unfoldings of a polytope overlap":

Conjecture (Schevon [35])
The probability that a random unfolding of a random polytope overlaps approaches 1 *as the number of vertices of the polytope approaches infinity.*

This conjecture seems to be true even if only *local* overlaps are considered. Of course, the notion of randomness has to be stated more precisely: What is a random polytope? What is a random unfolding?. Two kinds of *random polytopes* were considered in [35]: *random spherical polytopes* that are the convex hull of uniformly distributed points on the unit sphere, and *random cubical polytopes* which were generated by selecting points uniformly distributed in a unit cube and then taking their convex hull. The definition of a *random unfolding* is simple: Among all spanning trees of $D(P)$ (or $G(P)$), choose

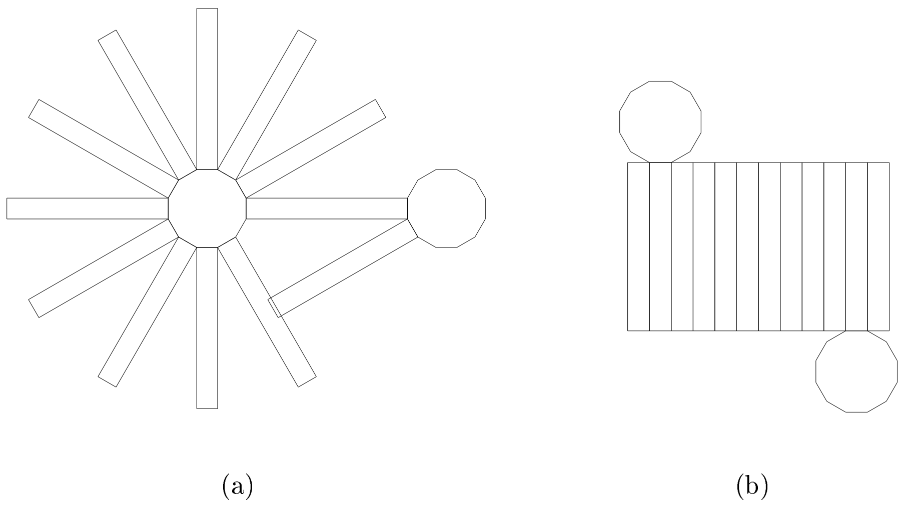


Figure 10: A net and a would-be net of a regular 12-sided prism.

any tree with equal probability. Experiments showed that the percentage of (even locally) overlapping unfoldings approaches 100% quickly for both kinds of random polytopes, as the number of vertices increases.

If the above conjecture is true, this leads to a curious situation: On the one hand, a polytope hasn't been found which doesn't have a net but, on the other hand, almost all unfoldings overlap.

Apart from our work, there is much more evidence to be found supporting the conjecture that every 3-polytope has a net, among it the *Hypergami* software package developed by Eisenberg and Nishioka, a *System for Creating Polyhedral Models*, which — up to now — was able to find a net for every given convex polytope, and even for many non-convex polytopes [13], [14].

For *non-convex* polyhedra, the question has been answered: There are non-convex polyhedra for which every unfolding has overlaps [9].

Shephard also investigated the problem of when a polytope has a (strictly) *convex* net (that is, when it can be unfolded to a non-overlapping convex polygon) [38], which is very rare. Considering *combinatorially equivalent* polytopes, Shephard found several classes of polytopes with combinatorially equivalent representatives which have (strictly) convex *Hamiltonian* nets (that is, nets which are obtained by cutting edges of the polytope along a sequence of edges that visits all its vertices exactly once), or, equivalently, nets which consist of a "strip" of facets, where each facet is adjacent to only one or two others). He also provided a set of transformation rules for polytopes with a (strictly) convex Hamiltonian net, which produce new polytopes with the same property.

He conjectured, *inter alia*, that any polytope with a Hamiltonian path is combinatorially equivalent to one that has a convex Hamiltonian net. This would imply that every polytope with up to 13 vertices would have this property, since such polyhedra all have a Hamiltonian path [24], [38].

A number of related questions have been asked; for example:

(B) Does every spanning tree of a polyhedron lead to a net?

We have already seen that this conjecture is generally wrong (recall Figure 10). On the other hand there *are* polytopes for which it *is* true, e.g., the regular tetrahedron. It is not true for all tetrahedra, however; as a minimal example of a polytope with an overlapping unfolding, Namiki constructed the skinny tetrahedron in Figure 11.

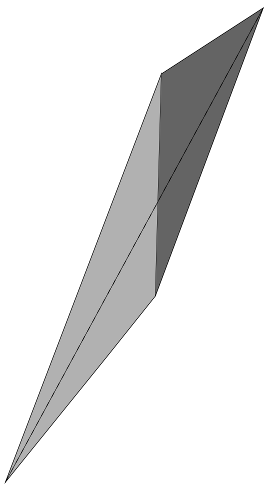
Since the answer to (A) is not known, it is sensible to ask the following question:

(C) Given a polytope P, is there a polytope Q combinatorially equivalent to P, for which every spanning tree leads to a net?

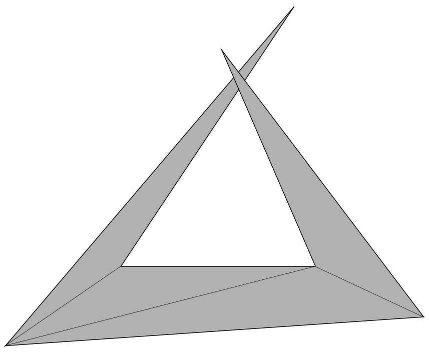
Grünbaum conjectured that the answer is negative in general and that every polytope that is combinatorially equivalent to the n-sided prism has some would-be nets, if n is large enough [24]. Note that apart from the kind of overlap shown in Figure 10, there are other overlaps possible, for example when the prism is "flat"; see Figure 12.

Another question closely related to (A) is the following:

(D) Given a polytope P, is there a polytope Q combinatorially equivalent to P, which has a net?



(a) A tetrahedron. . .



(b) . . . and a “bad” unfolding.

Figure 11

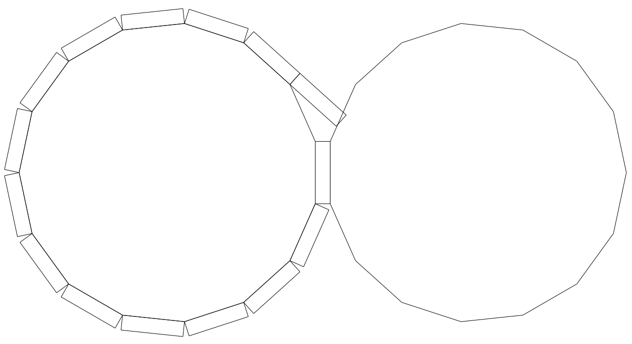


Figure 12: A bad unfolding of a flat 15-sided prism.

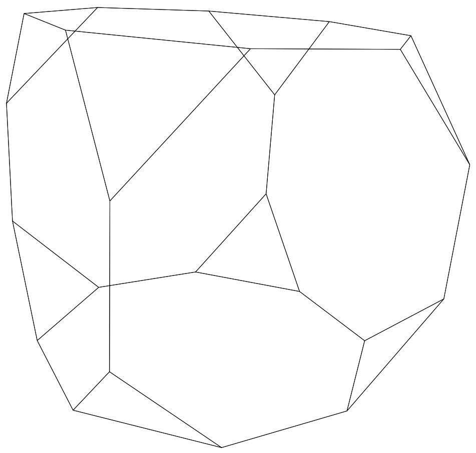


Figure 13: Truncated cube

As in (A), the answer is still unknown, but — since (A) implies (D) — it is widely believed to be positive.

(E) Does every polyhedron have a single-chain net? (A net is a "single-chain" net, if every facet of the unfolding shares an edge with only one or two other facets.)

The answer is negative [24]. The reason lies in combinatorics: For example, consider the truncated cube in Figure 13. No two triangles have a common edge. Therefore in an arrangement of the facets in a single chain between any two triangles there has to be a non-triangular facet. Since there are eight triangles, but only six other facets, this is impossible. On the other hand, the five Platonian solids do admit single-chain nets; in fact, they even have the property that every chain consisting of all facets yields a net. This is a result of Reggini's enumeration of all such chains for the five Platonian polytopes [34]; he showed that there are the following numbers of distinct chains of this type (not counting mirror images and rotations as distinct):

1	for the tetrahedron,	4	for the cube,
3	for the octahedron,	340	for the dodecahedron and
18	for the icosahedron.		

This leads to a more general question:

(F) How many different (up to isomorphism) nets does a polytope have?

A general answer isn't known (which is not surprising, since it isn't even known whether every polytope has *one* net). Jeger showed that the answer is 11 if the polytope is a cube or an octahedron [25].

Barnette [4] proved that dual graph of every polytope has a spanning tree where each node has degree at most 3. This leads to a question analogous to (D):

(G) Does every polytope have a net in which each facet has at most 3 neighbours?

A different unfolding approach in [35] results in a theorem on the development of convex polygonal curves: A directed *polygonal curve* is a connected sequence of line segments on the surface of a polygon. At a point p where two adjacent segments s_1 and s_2 meet, one can measure the *surface angle*, which is

$$\sphericalangle_P(s_1, s_2) = \begin{cases} \sphericalangle(s_1, s_2) & \text{if } p \text{ is in the interior of a facet,} \\ \sphericalangle(s_1, s_2) - \phi(s_1, s_2) & \text{otherwise,} \end{cases}$$

where $\phi(s_1, s_2)$ is the angle between the normal vectors of the two facets on which s_1 and s_2 lie.

Intuitively, then, a *development* of a polygonal curve is the trace it leaves on the plane when the polytope is rolled along the curve without slippage (see the sketch in Figure 14). More precisely, the development is obtained by laying out the curve's line segments on the plane so that the angle on one side of the developed curve between each adjacent pair of segments equals the surface angle between them. One of the main results here is the following:

Theorem 3.1 (Schevon, [35])
A closed convex polygonal curve on the surface of a 3*-polytope develops in the plane without self-intersection.*

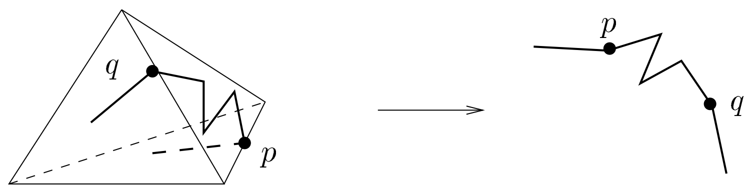


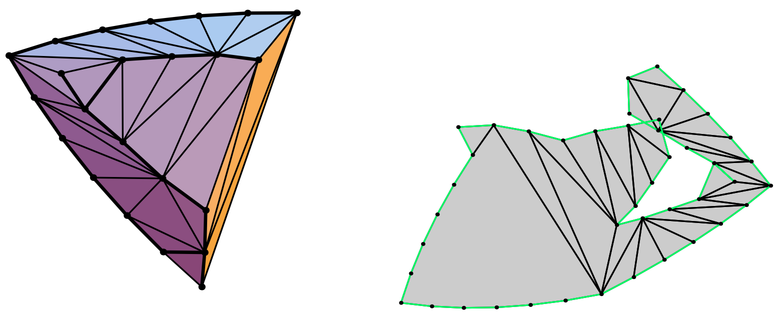
Figure 14: Development of a polygonal curve: "Roll" the polytope along the curve.

This may be used to construct a net for a polytope P by developing convex polygonal curves on P. In particular, if the convex polygonal curve is a sequence of *edges* of P, this leads to a net:

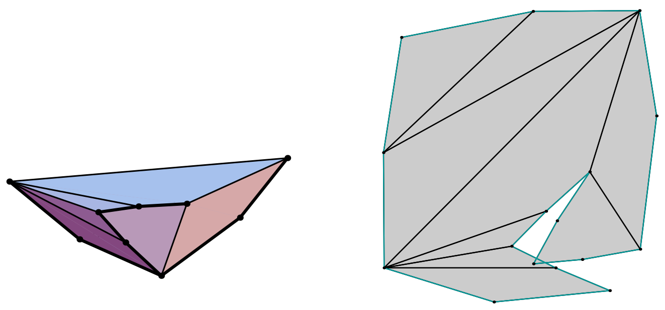
Corollary 3.1
If a polytope is cut open along a Hamiltonian path, such that the surface angle is strictly less that π at all inner vertices of the path, the resulting unfolding does not overlap.

There were a few conjectures posed by Fukuda [16]: The first is that a minimum-perimeter unfolding of a polytope is always a net. (A minimum perimeter unfolding is an unfolding obtained from a minimum spanning tree of $G(P)$, where the length of an edge taken to be the length of the corresponding edge on P.) Just recently, Rote constructed counterexamples to this conjecture (see Figure 15). We will also give counterexamples to another open conjecture of Fukuda that cutting along a shortest path tree of $G(P)$ leads to a net (see Section 4.6).

Yet another approach to polytope unfolding is not to require that the cuts are a spanning tree of the 1-skeleton of P, but rather allow arbitrary (straight line) cuts through the interior of facets. A special kind of unfolding is the *star unfolding.* It is constructed as follows: Let the *source* x be a point on the surface of a polytope P, such that there is a unique shortest path to every vertex of P (distance measured on the surface of P). The star unfolding is then obtained by cutting the boundary of P along these paths and flattening the resulting surface S_x in the plane; see the sketch in Figure 16. For the star unfolding the following theorem holds:



(a)



(b)

Figure 15: Counterexamples for the “minimum perimeter” conjecture of Fukuda; reproduced with kind permission of Günter Rote.

Theorem 3.2 (Aronov & O'Rourke [2])
When viewed as a metric space with the natural definition of interior metric, S_x is isometric to a simple polygon in the plane (with the internal geodesic metric).

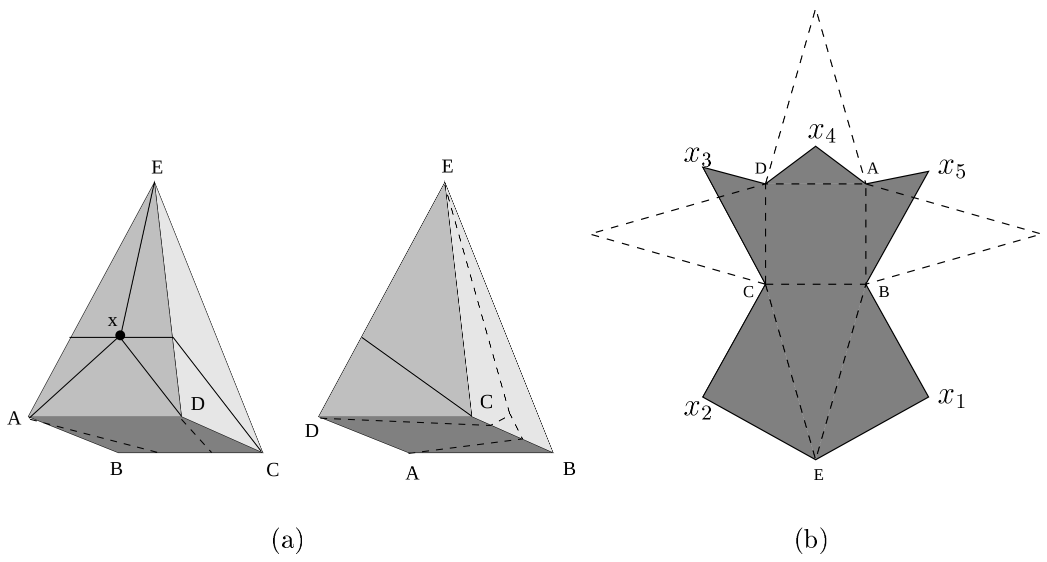


Figure 16: A pyramid with a shortest path tree emanating from x (front view (a) and side view (b)), and the corresponding star unfolding. Dotted lines in (c) indicate the "natural" unfolding obtained by cutting along edges.

Theorem 3.2 can be used to prove that certain polytopes have nets:

Corollary 3.2
Let P be a polytope. If P has a vertex which is adjacent to all other vertices of P, then P has a net.

Proof: The shortest paths from x to all other vertices p_i of P are the edges $[x, p_i]$. The star unfolding S_x is a net in the sense of Definition 2.11, since the shortest paths introduce cuts along the edges emanating from x. According to Theorem 3.2, S_x is isometric to a simple polygon, i.e. it doesn't overlap.
$\square$

Viewing polytope unfolding as an optimization problem, many more issues can be considered, for example, finding a net whose convex hull has minimal diameter, area, or other properties.

3.2 Polytope folding

The primary reference in this area is probably Aleksandrov's *Konvexe Polyeder* [1]. He proved that a given polygon, together with edge matching information and folding lines can be folded into at most one polytope, and gave a scheme for determining if a given polygon is a net [1].

Surprisingly and, as Fukuda put it, "against human intuition", if one starts with a simple connected polygon (that is, *without* an *a priori* identification of some of its edges), it is sometimes possible to fold it into polyhedra that are geometrically or even combinatorially different.

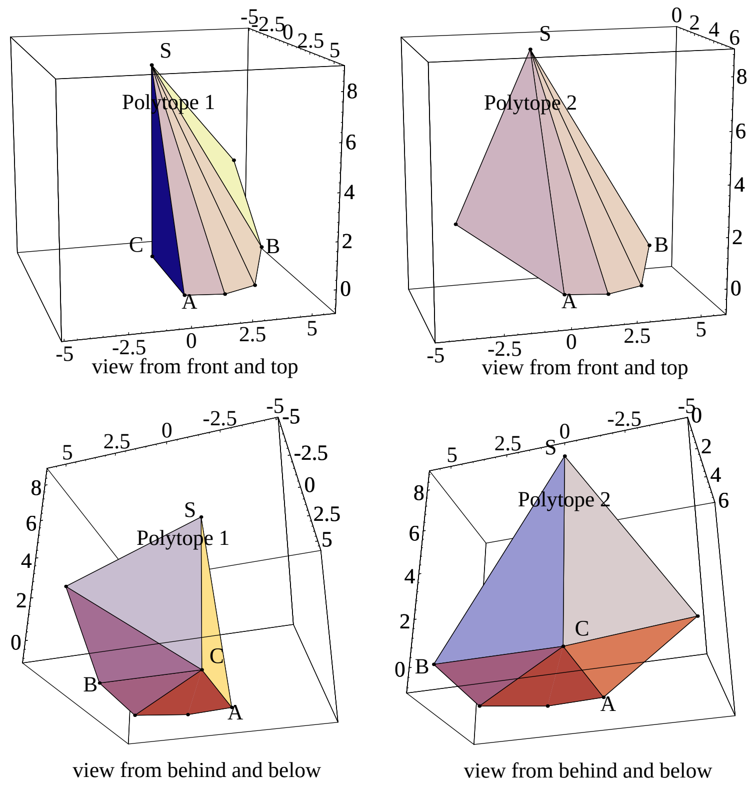
The question whether a given polygon can be folded to a polytope has been answered theoretically by Aleksandrov [1], who proved the following theorem:

Theorem 3.3 (Aleksandrov, [1], pp. 88 & 91)
Every net which is homoeomorphic to a sphere, and for which the angle sums at each vertex are $\leq 2\pi$ is the net of a unique closed convex polyhedron (up to motions and flips).

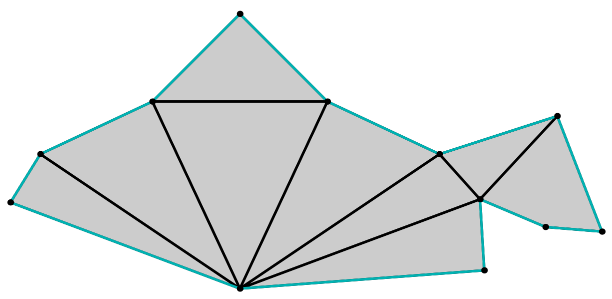
He also mentioned that it is of course possible to *construct* this polytope, but he wouldn't find that interesting because one could just go ahead and *fold* the polygon, and the convex polyhedron would come out "by itself"; the only thing one has to maintain is that the resulting polyhedron is *convex*.

Thus, in principle it is possible to distinguish between the two polygons in Figure 18: One is a net of a triangular prism (a), the other one is not a net of any convex polytope (b).

An algorithm which solves this folding problem was published by Lubiw and O'Rourke [27]. They developed an $O(n^2)$-algorithm, which answers the question if a given polygon (without any edge matching information) can be folded to a polytope, by testing the angle condition mentioned above for (a sufficient subset of) all possible edge matchings. Their algorithm also solves the problem if vertices of the net are allowed fold "flat", or conversely,

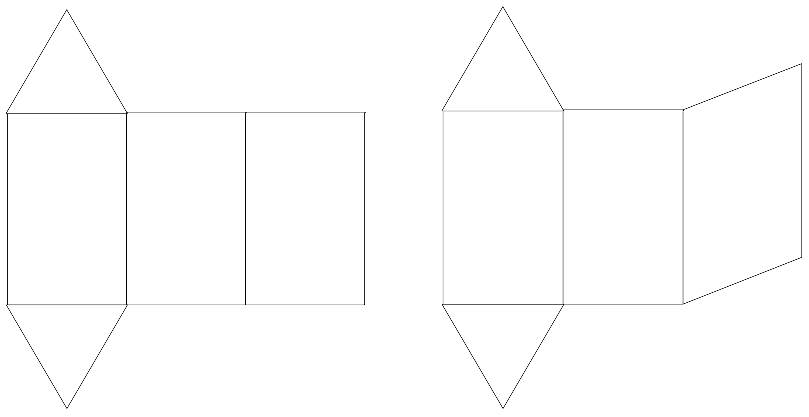


(a) Two combinatorially different polytopes...



(b) . . . with a common unfolding.

Figure 17: The pictures were reproduced with kind permission by Günter Rote.



(a) Net of a triangular prism.

(b) Not a polytopal net.

Figure 18

nets which are obtained from a polytope by cutting through the interior of one or more facets, a case which is also covered by Aleksandrov's theorem. In this case the resulting polytope may not be unique. For example, the "cross" net of the cube in Figure 1(a) may be folded into as many as five distinct polytopes: A cube, a tetrahedron, an octahedron, a pentahedron, and a doubly covered quadrangle; here we show two of these nets in detail: Identifying the edges

$$e_0 \equiv e_3, e_1 \equiv e_2, e_4 \equiv e_{13}, e_5 \equiv e_6, e_7 \equiv e_{12}, e_8 \equiv e_{11} \quad \text{and} \quad e_9 \equiv e_{10}$$

leads to a polytope which is (combinatorially equivalent to an) octahedron (see Figure 19). Likewise, identifying the edges

$$e_0 \equiv e_1, e_2 \equiv e_3, e_4 \equiv e_{13}, e_5 \equiv e_6, e_7 \equiv e_{12}, e_8 \equiv e_{11}, \quad \text{and} \quad e_9 \equiv e_{10},$$

leads to a tetrahedron (see Figure 20).

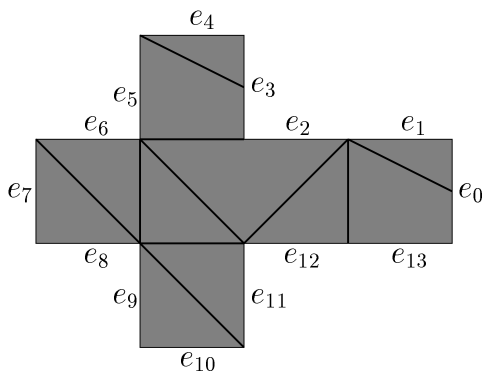


Figure 19: Folding net for an octahedron.

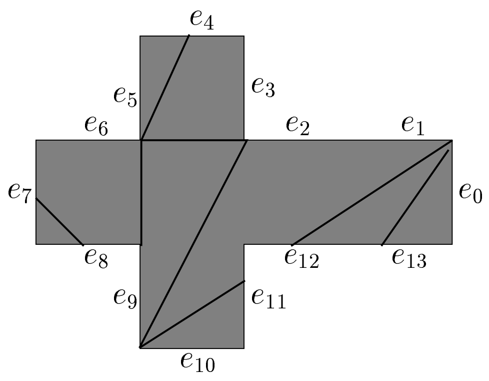


Figure 20: Folding net for a tetrahedron.

4 Experiments

For the rest of the thesis, we will focus on the initial question:

(A') Does every 3-polytope possess a net (and, if so, how does the net look like)?

The following section describes the implementation tools we used. Section 4.2 presents the polytope *test set* all unfolding rules were applied to. Section 4.3 introduces a few primitive unfolding operations and further notation. After that we exhibit the actual unfolding rules along with their results.

4.1 Environment

One of the most useful tools for the manipulation of polytopes that was available was *polymake* [19] — a tool for algorithmic treatment of polytopes and polyhedra. In particular, it allows the user to switch back and forth between different descriptions or representations of a polytope, and obtain combinatorial information about it, like its graph or the corresponding dual graph. In fact, the most often used *polymake* features were, given a set V of points in $\mathbb{R}^3$, compute their convex hull (that is, separate those points which are vertices of $P = \operatorname{conv}(V)$ from the points in the relative interior of P), $\dim(P)$, $G(P)$ and $D(P)$. That is, the input for the unfolding rules is a complete description of the polytope.

Another useful tool is *Geomview* [21] — a software package written at the Geometry Center, University of Minnesota, which we used to visualize the polytopes we were working with; most of the polytope drawings here have been created with *Geomview*.

All algorithms described below have been implemented in C++, and make heavy use of the LEDA library [30], as well as the C++ interface of *polymake*.

4.2 Test Polytopes

We tested each of the unfolding rules described below by applying them to a test set of about 10 000 polytopes of different kinds:

- 55 regular and semi-regular polytopes such as the regular tetrahedron, cube, octahedron, dodecahedron and icosahedron and several other "non-random" polytopes (see Figures 22(a) and 22(b)).

- 4200 random *spherical* polytopes which are the convex hull of 4–500 random points uniformly distributed on a unit sphere (see Figure 22(c)). In order to compute points uniformly distributed on the unit sphere, let x_1, x_2, x_3 be independent and normal distributed. Then the density of their common distribution in the point (x_1^*, x_2^*, x_3^*) is

$$e^{-{x_1^*}^2} \cdot e^{-{x_2^*}^2} \cdot e^{-{x_3^*}^2} = e^{-({x_1^*}^2+{x_2^*}^2+{x_3^*}^2)} = e^{-||(x_1^*,x_2^*,x_3^*)||^2},$$

  since the x_i are *independent.* That is, the distribution of (x_1, x_2, x_3) is independent of longitude and latitude, and thus,

$$x = \frac{(x_1, x_2, x_3)}{||(x_1, x_2, x_3)||}$$

  is uniformly distributed on the unit sphere. (When simulating these random variables, of course one has to omit all zero triples.)

- The *cyclic* polytopes $C_3(4), \ldots, C_3(50)$, where

$$C_d(n) = \operatorname{conv}\{(t, t^2, \ldots, t^d) \mid t \in \{1, \ldots, n\}\} \quad \text{(see Figure 22(d)).}$$

- 672 polytopes which were constructed by repeated random truncation of the standard tetrahedron

$$\Delta = \operatorname{conv}\left\{ \begin{pmatrix}0\\0\\0\end{pmatrix}, \begin{pmatrix}1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\end{pmatrix} \right\}.$$

  Here, "repeated random truncation" means to repeatedly select a random vertex of the current polytope, compute the *truncation* $P_{\measuredangle(v)}$ of P, and make $P_{\measuredangle(v)}$ the current polytope. Intuitively, $P_{\measuredangle(v)}$ is constructed from P by cutting off the vertex v (see the sketch in Figure 21). Formally, let $ax \leq b_0$ be a valid inequality for P, such that

$$\{v\} = P \cap \{x \mid ax = b\},$$

  and choose $b_1 < b_0$ such that $cw < b_1$ for all $w \in V(P) \setminus \{v\}$. Then the *truncation* of P is defined as

$$P_{\measuredangle(v)} = P \cap \{x \mid cx \leq b_1\}.$$

- 350 regular and irregular prisms.
- 2350 random *cubical* polytopes which are the convex hull of 5–100 uniformly distributed points in the unit cube (see Figure 22(h)). (The term *cubical polytope* often refers to polytopes whose facets are cubes — different from the polytopes just described.)
- 1575 "flat" polytopes $P_{i,j,k}$ which are the convex hull of k points uniformly distributed in $[0,i] \times [0,j] \times [0,1/2]$, with $i,j \in \{1,\dots,10\}$ (see Figure 22(g)).
- 630 polytopes that are the convex hull of $10\dots100$ points

  $$(x,y,z) = (\sin\theta\cos\phi, \cos\theta\cos\phi, \sin\phi),$$

  with (ϕ,θ) uniformly distributed in $[0,\pi]\times[0,2\pi]$; these polytopes look like the example in Figure 22(f). Note that the points are not uniformly distributed on the "half sphere", but are more dense at the north pole.
- "Turtle" polytopes $T(i,j)$, with

  $$T(i,j) = \mathrm{conv}\{(x,y,x^2+y^2) \mid x=-i,\dots,i; y=-j,\dots,j\},$$

  for $i,j \in \{1,\dots,7\}$, $i \le j$ (see Figure 22(e)). We expected the *flat* polytopes above to be the "nastiest" possible polytopes for many unfolding rules — to our surprise the *turtle* polytopes turned out to be the ones where most of the unfoldings overlap (see Section 4.11).
- "ε-triangulations" of most of the polytopes above; that is, for each facet f of P, we choose a point x_f just beyond f, and compute

  $$T_\varepsilon(P) = \mathrm{conv}(V(P) \cup \{x_f \mid f \in F(P)\}) \quad \text{(see Figure 22(i))}.$$

4.3 Unfolding Operations

Before we describe the unfolding experiments, we will introduce a few more conventions for describing the process of constructing a net.

Technically, starting from the empty set $N = \emptyset$, we construct a collection of 2-dimensional convex polygons in $I\!R^2$ by adding polygons isometric with

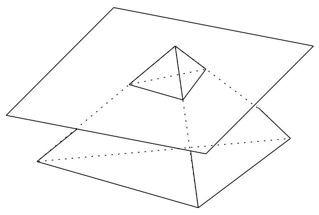


Figure 21: Truncation of a polytope: Cutting off a vertex.

the facets of a given polytope P, such that their union is connected and the intersection of any two polygons in N is either empty or an edge of both polygons. Additionally, each polygon $f_N \in N$ is associated with the facet of $f \in P$ it was constructed from; the same holds for all edges and vertices of the polygons in N.

The first thing that always happens during the construction of a net is that a single facet f is *embedded* or *laid out* somewhere in the Euclidean plane $I\!R^2$. By this we mean that f is mapped to a polygon $f_N \in I\!R^2$ isometric with f. We call this operation $\textsc{Layout}\,(f)$.

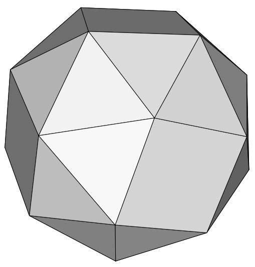
Another primitive operation we need is to extend a *partial net* by *join*ing a facet $g \in P$ to a facet f_N in the partial net N along an edge $e \in f$ by mapping g to an isometric polygon g_N, such that their common edge $e = f \cap g$ is mapped to $f_N \cap g_N$ (f and g are of course required to be adjacent facets of P.) This operation is called $\textsc{Layout}\,(g, f)$ (see Figure 23).

Recall from Section 2 that a net can be constructed from two different trees — a *cut tree* and a *join tree* (see Figure 24):

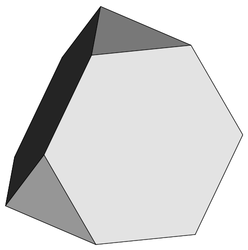
(i) Given a spanning tree T_G of P's graph $G(P)$ (a *cut tree*), the boundary of P is "cut open" along the edges of T, and the facets of P are laid out by joining them along the remaining edges.

(ii) Given a spanning tree T_D in the dual graph $D(P)$ (a *join tree*), two adjacent facets f and g of P are laid out next to each other only if $(f, g) \in T$.

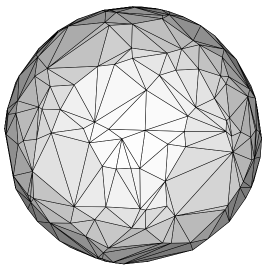
It is actually more straightforward to implement the latter method, because incrementally *joining* facets together is what really happens during the construction. This is what $\textsc{Join}\,(P, T_D)$ does (Procedure 1).



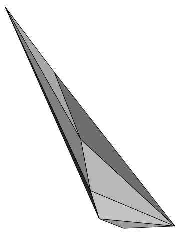
(a) “Snub cube”



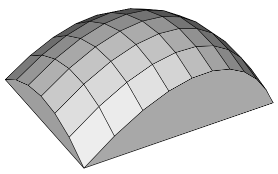
(b) Truncated tetrahedron



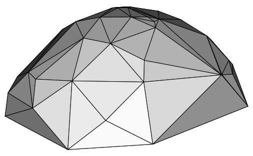
(c) Convex hull of 400 random points on a sphere



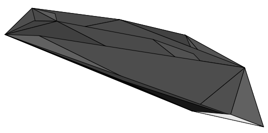
(d) Cyclic polytope $C_3(8)$



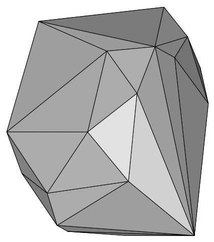
(e) “Turtle” $T(3,5)$



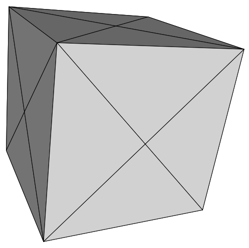
(f) Random points on a half-sphere



(g) Flat polytope $P_{4,3,30}$



(h) Random cubical polytope



(i) ε-triangulation of a cube

Figure 22

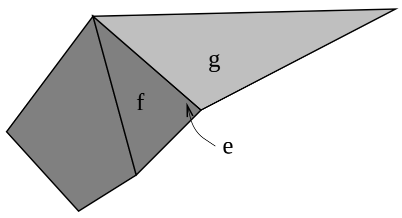


Figure 23: Joining f and g by LAYOUT (g, f)

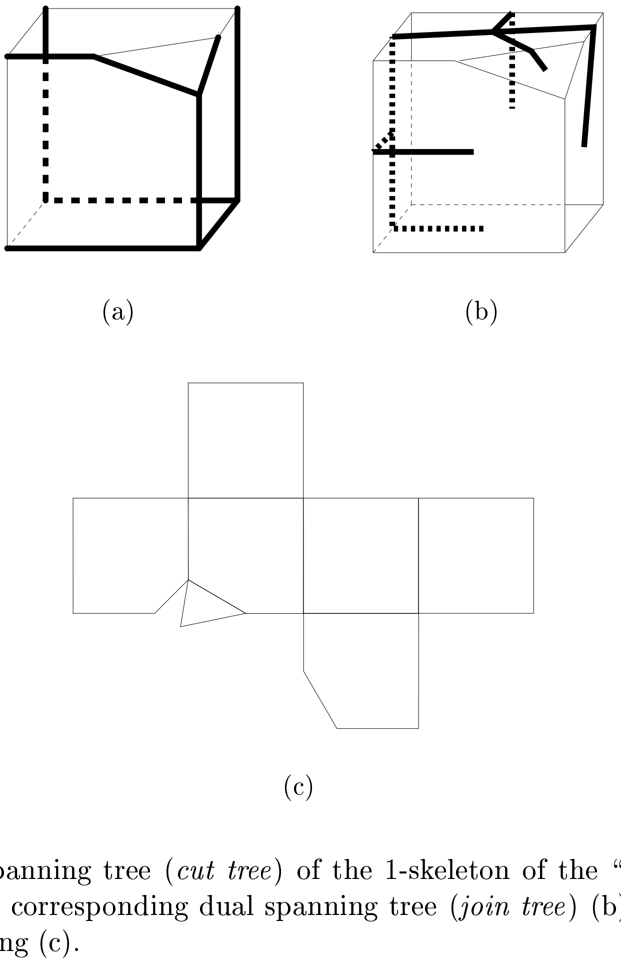


Figure 24: A spanning tree (*cut tree*) of the 1-skeleton of the "cut cube" from Figure 9 (a), the corresponding dual spanning tree (*join tree*) (b) and the corresponding unfolding (c).

Procedure 1 JOIN (P, T_D)

```
$D = D(P)$
$E = E(T)$
$N = \emptyset$
$e = (f, g) =$ *some arbitrary (dual) edge in* $T$
$N =$ LAYOUT $(f)$
$N = N \cup$ LAYOUT $(g, f)$
$E = E \setminus \{e\}$
**while** $E \neq \emptyset$ **do**
  *let* $e = (f, g) \in E$
  $N = N \cup$ LAYOUT $(f, g)$
  $E = E \setminus \{e\}$
**return** $N$
```

In Procedure 1, we assume the edges $E = E(T)$ be represented as a list of edges $e_1, \ldots, e_{|E|}$, such that each edge e_i is adjacent to some edge e_j, $j < i$. Such an ordering can be constructed in $O(|E|)$ time by, for example, appending the edges of T to E in the order they are visited by *breadth-first search*, since breadth-first search only visits nodes which are adjacent to nodes that have been visited before [7]. Hence, if $(e_1, \ldots, e_{|E|})$ are the edges of a breadth-first tree T in the order they are visited by the search, every prefix $(e_1, \ldots, e_i)$, $i < |E|$, forms a connected subtree of T.

Clearly, it is superfluous to provide a different procedure for a *cut tree*: each of the above trees can be computed from the other one because

$$e \in T_G \iff e^* \notin T_D.$$

Running time analysis of the unfolding rules

We will use the usual "$\mathcal{O}$ notation" when giving a brief running time analysis of the unfolding rules.

A remark on the running time of JOIN (P, T) and CUT (P, T) in terms of the number of vertices or facets of the input polytope: In general (i.e. in arbitrary dimension) the number of facets of a polytope can be exponential in the

Procedure 2 $\textsc{Cut}\,(P, T_G)$

```
$T_D = \emptyset$
// *Compute a join tree from $T_G$*
**for all edges** $e \in G(P)$ **do**
  **if** $e \notin T_G$
    $T_D = T_D \cup e^*$
// *Unfold $P$ using $T_D$ as the join tree*
**return** $\textsc{Join}\,(P, T_D)$
```

number of its vertices and vice versa. For 3-polytopes, however, according to Euler's famous formula

$$v - f + e = 2, \text{ where } v = |V(P)|,\ f = |F(P)|,\ e = |E(P)|,$$

the number of vertices, edges and facets are linearly related. Thus, all lines of procedure 1 can be performed in constant time. Therefore, given a join or cut tree of P, $\textsc{Join}\,(P, T_D)$ and $\textsc{Cut}\,(P, T_G)$ will compute an unfolding in time $\mathcal{O}(|F(P)|) = \mathcal{O}(|E(P)|) = \mathcal{O}(|V(P)|)$.

Rounding problems

The final thing to do after the construction of an unfolding is to check whether it is a net. We used the *sweep line* algorithm described in [7], pp. 892–897, to check whether any of the bounding line segments of the unfolding intersect. For n segments this can be done in time $\mathcal{O}(n \log n)$. It turned out, however, that it is numerically difficult to reliably implement this check because of rounding errors that occur during the computation of the vertex coordinates of the net:

- Although we only considered polytopes with rational vertex coordinates, these rounding errors can still occur since we resorted to finite precision arithmetic when computing the length of an edge, which can of course be irrational.
- Intersections at endpoints of line segments had to be dealt with: all line segments meeting at a vertex of N intersect at their endpoints, but we

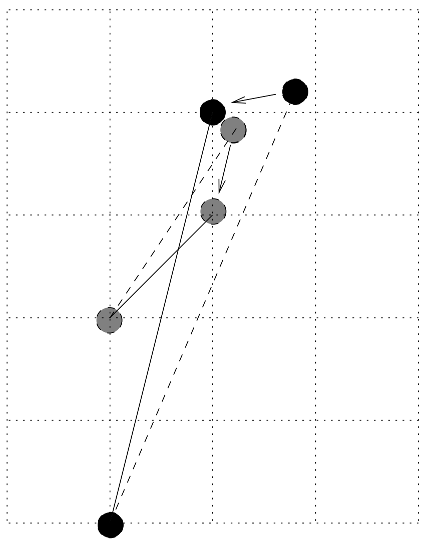


Figure 25: An intersection caused by rounding, when rounding downwards, that is, moving a point to the nearest grid point below and left of it: Before rounding (dotted) the two segments do not intersect, after rounding (straight lines) they do.

don't want to "count" those as intersections, although the *sweep line* algorithm finds them. A straightforward approach would check whether the intersection point is the endpoint of a segment and, if so, do not count this as an overlap, as we defined an overlap as the intersection of two boundary edges in their *interior.* Due to rounding errors as mentioned above, it is possible that, *numerically,* the intersection of two incident line segments does *not* happen at the endpoints, but in their interior.

One way out of this dilemma is to *shorten* all segments prior to the check using a transformation

$$\text{shorten}((v,w),\rho) = (v + \frac{\rho}{2}(w-v), w + \frac{\rho}{2}(v-w)),$$

for some small $\rho > 0$. The choice of ρ is crucial here: If it is too big, very small overlaps may not be detected; if it is to small, we will detect "false" overlaps that are caused only by rounding errors. Furthermore, a sensible choice for ρ is hardware dependent, or even compiler dependent, because it depends on the number of significant digits that are used for storing a floating point number. In the compiler we used for the implementation[3], the `double` data type has 52 significant bits. This means, for example, that

$$1 + 2^{-53} \text{“=”} 1 \text{ (!)} \quad \text{and} \quad 1 + 2^{-52} \neq 1,$$

at least as far as the compiler is concerned. Since

$$2^{-52} \approx 2.22204 \cdot 10^{-16},$$

and the worst case distance between a “real” and a “rounded” point is $\sqrt{2}/2^{53}$ we decided to set

$$\rho := 10^{-10} \cdot n \cdot \ell_{\max},$$

where n is the maximal degree of a node in $D(P)$, and $\ell_{\max}$ is the length of the longest edge of P. The added “safety” is meant to compensate effects such as the following: Suppose an endpoint of a *short* edge e_1 is displaced by, say, $\delta > 0$. This edge will slightly change its direction. Another edge e_2 which is appended to this edge at a certain (fixed) angle will also change its direction by the same amount as e, but in the worst case the other endpoint of e_2 will be displaced by $\delta \cdot ||e_2||$. From a facet f we compute the embedding f_N by successively appending the edges of f; thus, if f has n bounding edges, the rounding error may occur n times; that's where the “n” comes from in the definition of ρ. This choice of ρ worked well, except for the ε-triangulations, where we had to manually adjust the value of ρ.

4.4 Unfolding rules

Here we will present our unfolding experiments. For each unfolding rule we will show “good” and “bad” unfoldings it produces, and briefly note the overall rate of overlapping unfoldings. See Table 1 on page 93 for a more detailed overview over the performance of the unfolding rules w.r.t. different types of input polytopes. All unfolding algorithms were developed under the assumption that the following conjecture is true:

Conjecture
There is a simple rule which produces a net for every 3-polytope. (Here with “simple rule” we mean a procedure that can be explained in less that 10 minutes to anyone who knows what a polyhedron is.)

[3]GNU `g++` Version 2.7.2.

With this in mind, we will present a number of different unfolding rules along with their results. We did not come up with an unfolding rule for which we could *prove* that it always finds a net. Nonetheless, most of the unfolding rules are explained below in order to

- provide images of how the unfoldings produced by different rules *look like*, and an intuitive understanding of why some unfolding rules are more likely to produce nets (or overlaps) than others,
- save others from inventing and trying the same things again, once it is clear that they do not work,
- provide supporting empirical evidence for the above conjecture: There *is* an unfolding rule which found a net for any ever so obstinate polytope and it *is* simple in the above sense.

4.5 Simple trees

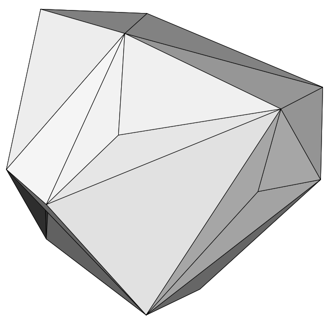
4.5.1 Breadth-first unfolding

The first unfolding rule we tried uses *breadth-first search*. Breadth-first search is one of the simplest algorithms for searching a graph and the archetype for many important graph algorithms. Given a connected graph $G = (V, E)$ and a distinguished *source* node s, breadth-first search produces a "breadth-first tree" with root s that contains all vertices of G. Furthermore, the path from s to any other vertex is a shortest path in terms of the number of edges [7].

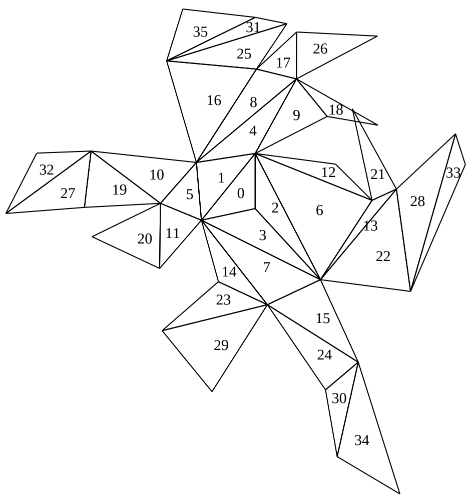
Rule 1 computes a breadth-first tree T of $D(P)$ and then produces an unfolding of P according to T. This rule is used as the default rule in Fukuda's and Namiki's "Mathematica package for unfolding polytopes" [31]. As a breadth-first search ignores the *geometric* structure of the polytope it is not surprising that it eventually fails. Well it *does*, for example, when applied to the polytope in Figure 26(a).

```
Rule 1      BREADTH-FIRST-SEARCH-UNFOLD ($P$)

    $T$ =BREADTH-FIRST-SEARCH ($D(P)$)
    return JOIN ($P, T$)
```



(a) A random cubical polytope,. . .



(b) . . . and a breadth-first unfolding; numbers showing the order in which the facets are laid out.

Overlaps: 16.4% of the unfoldings overlapped.

Running time: A breadth-first tree of a graph with $|E(P)$ edges can be computed in time $\mathcal{O}(|E(P)|)$ [7], so the total running time is $\mathcal{O}(|E(P)|)$.

4.5.2 Depth first unfolding

Speaking of *breadth first search*: Another well-known graph search algorithm is *depth-first search.* As its name implies, depth-first search follows the strategy to search "deeper" in the graph whenever possible; edges are explored starting from the most recently discovered vertex v that still has unexplored edges leaving it. When all of v's edges have been explored, the search "backtracks" to explore edges leaving the vertex from which v was discovered. An unfolding rule using depth-first search is also implemented in [31].

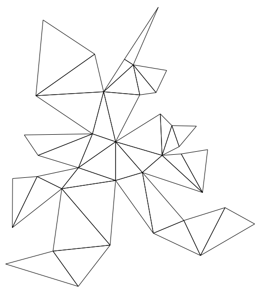
We implemented this rule mainly in order to be sure to have a method that seemed very likely to fail: It not only ignores the polytope's geometric structure, but also tends to generate longer paths than breadth-first search, that is, the resulting unfolding contains longer chains of facets.

Rule 2 DEPTH-FIRST-SEARCH-UNFOLD (P)

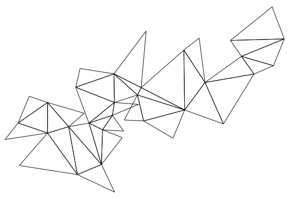
```
    $T$ =DEPTH-FIRST-SEARCH $(D(P))$
    return JOIN $(P, T)$
```

Overlaps: 49.2% of the unfoldings overlapped. Nonetheless, this unfolding rule is not even the worst unfolding rule we found. In the next section we will present two variations of breadth-first and left-first search; both of which lead to more would-be nets than their "normal" counterparts. The latter is one of the two unfolding rules that produce overlaps in more than half of the test polytopes.

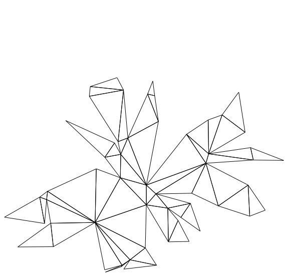
Running time: Since a depth-first tree can be computed in linear time, the total running time is $\mathcal{O}(|E(P)|)$.



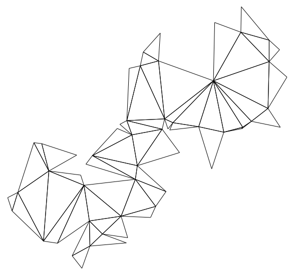
(c)



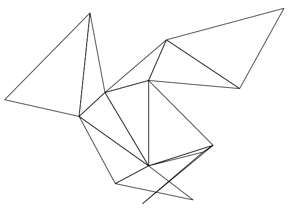
(d)



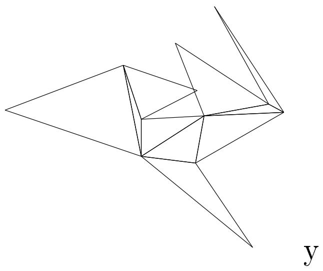
(e)



(f)



(g)



(h)

Figure 26: Typical breadth-first unfoldings ((a), (c) & (e)) and depth-first unfoldings ((b), (d) & (f)) of spherical polytopes; there are overlaps in (e) and (f).

4.5.3 Left-first unfolding

We will now introduce a different approach which uses a graph search technique called "left-first search", and present two unfolding rules based on it — one we expect to produce less overlaps than breadth-first unfolding and one we expect to produce even more overlaps than depth-first unfolding.

Definition 4.1
Let $G = (V, E)$ be a planar graph, with a fixed straight line embedding. For $v \in V$, let $L(v) = ((v, w_1), \dots, (v, w_{\delta(v)}))$ be the edges incident to v, such that $(w_1, \dots, w_{\delta(V)})$ appear in counterclockwise order around v (viewed from the "inside" of P).

For a facet f of G with barycenter b, let $L^*(f) = (e_1, \dots, e_{\delta(f)})$ be the edges of f, such that their midpoints appear in counterclockwise order around b (again, viewed from the "inside" of P; see Figure 28).

$L(v)$ is a left-first ordering of the edges incident to v; $L^*(f)$ is a left-first ordering of the edges that bound the facet f.

From a polytope, one can easily construct a straight line embedding of its graph: Let f be a facet of P, and x_f be a point beyond f, such that for all vertices v except those of f the closed line segment $[x, v]$ intersects f in its interior. (Ziegler [42] or Lóvász [26], for example, describe how to compute such a point explicitly.) Then, the projection of the polytope's vertices and edges onto f along "rays of vision" results in the desired straight line embedding (see Figure 27).

When computing the left-first ordering for the edges of a certain facet, we don't even have to compute the straight line embedding explicitly: It is sufficient to "look at" the facet from a point inside the polytope — its barycenter, for example, in order to compute the ordering. If f is surrounded by edges $(e_0, \dots, e_{\gamma(f)})$, we compute the angles α_i between

$$m_0 - b_f \quad \text{and} \quad m_i - b_f \quad \text{for } i = 1, \dots, \gamma(f),$$

where m_i is the midpoint of e_i.

We will now use this left-first ordering to produce a left-first version of the two graph search techniques *depth-first search* and *breadth-first search*, in order

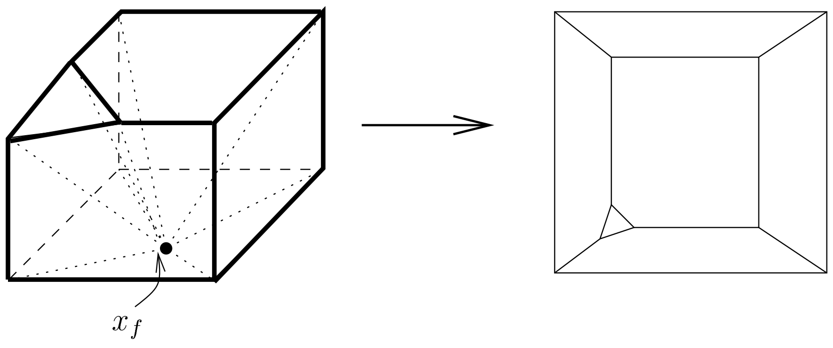


Figure 27: How to compute a straight-line embedding of a polytope's graph

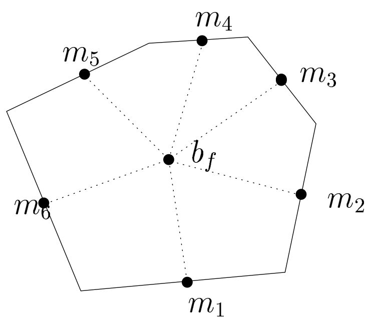


Figure 28: A left-first numbering of the bounding edges of f.

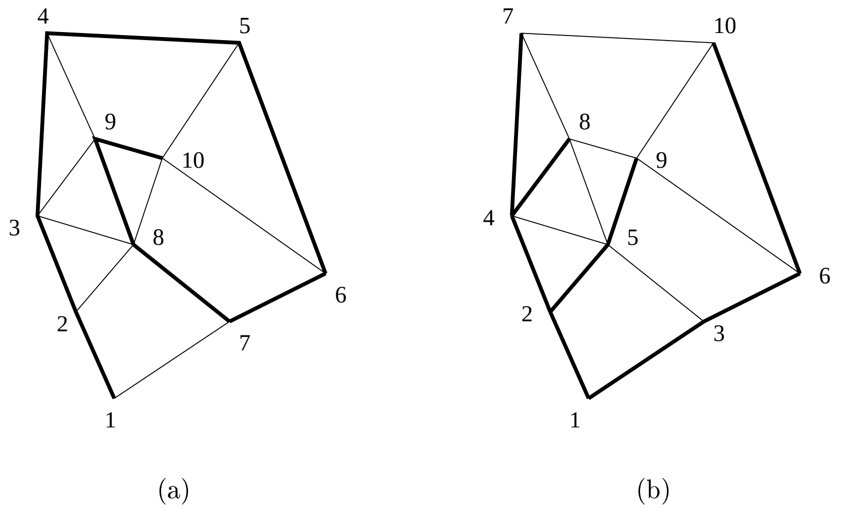


Figure 29: A depth-first (a) and a breadth-first left first tree (b); numbers indicate the order in which nodes are discovered by the search.

to obtain cut or join trees. They are essentially specializations of depth-first search and breadth-first search: in the left-first versions, the adjacent edges of a node are visited in a counterclockwise order, such that the leftmost edge comes first (see Figure 29).

When using a *combinatorial* embedding of $G(P)$, this leftmost neighbour of a node does not necessarily correspond to the *real* leftmost neighbour as defined above, since for each facet f of the graph there exists a planar embedding such that this facet lies on the outside of the embedding. For a polytopal graph, one can obtain a planar embedding such that a certain facet f is the outer facet, if one constructs the embedding like in Figure 27, projecting along rays to a point x_f beyond f. For different embeddings of a planar graph, this may permute the left-first order of the edges at some of the vertices (see the example in Figure 30).

The rationale behind this unfolding rule is (at least for the left-first version of breadth-first search) that a “normal” breadth-first search almost entirely ignores the *geometric* structure encoded in P. Therefore, it isn’t surprising that unfoldings constructed from a breadth-first join tree may overlap. From using a left-first breadth-first join tree we expect less overlaps to occur in comparison with “normal” breadth-first search, because at each facet the

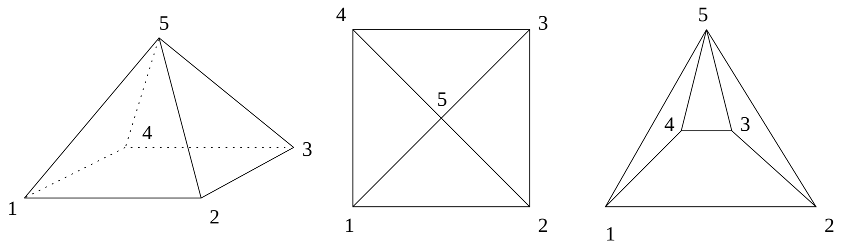


Figure 30: A pyramid and two different combinatorial embeddings of its graph with different left-first orderings of the edges.

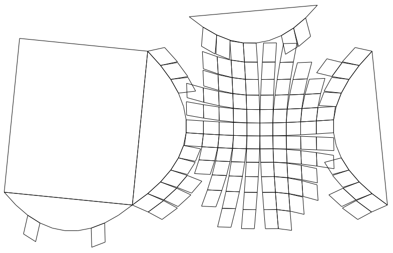
subtrees are attached left-first and thus (in a sense we will not attempt to make precise), at least some of P's geometric structure is reflected by the tree. Conversely, unfoldings obtained from a left-first depth-first tree should look rather like an "apple-peel", consisting of only a few long "strips" starting from the first facet. Indeed, these unfoldings turn out as expected (see Figure 31).

Rule 3 shows the pseudo-code for left-first breadth-first search. We omit the code for the depth-first version, because the only modification is to replace the queue Q by a stack.

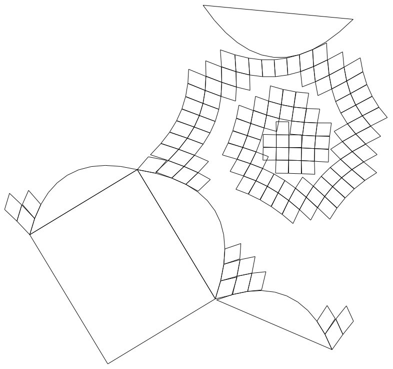
Rule 3 LEFT-FIRST-BREADTH-FIRST-UNFOLD (P, v)

```
INITIALIZE  $T$            $= \emptyset$
            $Q$            $= L^*(v)$
            visited[$v$]   $=$ false   $\forall v \in D(P)$
while $Q \neq \emptyset$ do
  pop $(v, w)$ from the front of $Q$
  $T = T \cup \{v\}$
  visited[$v$]= true
  forall $(w, x) \in L^*(v)$ do
    if visited[$w$] = false then
      append $(w, x)$ to $Q$
return JOIN $(T)$;
```

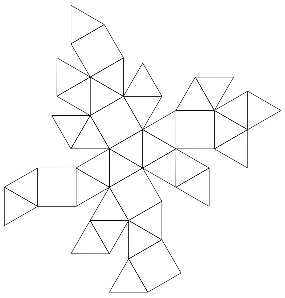
Overlaps: LEFT-FIRST-DEPTH-FIRST-UNFOLD is indeed the worst rule we found — 57.3% of its unfoldings overlap. As we expected, LEFT-FIRST-



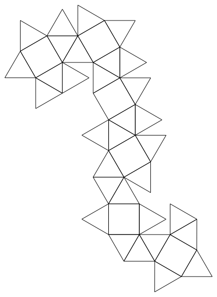
(a)



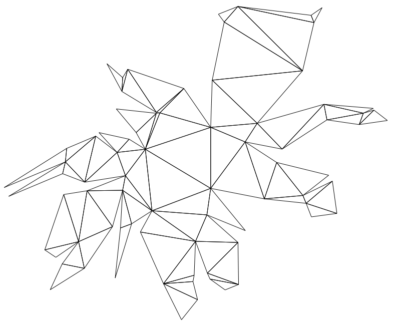
(b)



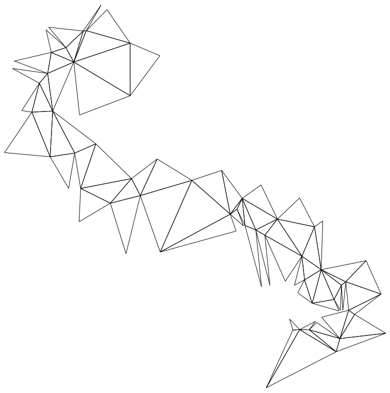
(c)



(d)



(e)



(f)

Figure 31: Typical left-first breadth-first ((a), (c) & (e)) and left-first depth-first unfoldings ((b), (d) & (f)). In particular, (a) and (b) have (local) overlaps.

BREADTH-FIRST-UNFOLD produces less overlaps than BREADTH-FIRST-UNFOLD: Only 16.9% of the unfoldings overlap.

Running time: The worst case time for finding a left-first ordering for all facets is $\mathcal{O}(|E(P)| \log |E(P)|)$: In the case of the pyramid over an n-gon there is a facet that is bounded $|E(P)|/2$ edges; hence we need $\mathcal{O}(|E(P)| \log |E(P)|)$ time to sort them. Thus, the overall running time of LEFT-FIRST-...-UNFOLD is $\mathcal{O}(|E(P)| \log |E(P)|)$.

4.6 Shortest paths tree unfolding

An (up to this point) open conjecture of Fukuda was the following:

> Cutting the boundary of a polytope along the edges of a *shortest path tree* originating from a fixed vertex of P leads to a non-overlapping unfolding.

A *shortest path tree* of a graph $G = (V, E)$ with *root* $v \in V$ is a set of edges $T \subset E$, such that the shortest path from every node $w \in V$ to v uses only edges in T. Such a tree can, for example, be computed using Dijkstra's algorithm [7].

We found a counterexample for this conjecture using the unfolding rule shown below, which implements this conjecture; see Figure 33.

Rule 4 SHORTEST-PATHS-UNFOLD (P, v)

```
for all edges $e \in G(P)$ do
  costs$[v] = ||e||$
// compute a shortest-paths tree in $G$ w.r.t. costs.
$T$ = SHORTEST-PATHS-TREE $(G, v, \text{costs})$
return CUT $(P, T)$
```

4.6.1 Variations

There are many thinkable choices for the root node of the shortest path tree:

(a) Shortest-Path-Min-Unfold
Choose a node with minimal degree as the root node.

(b) Shortest-Path-Max-Unfold
Choose a node with maximal degree.

(c) Short-Shortest-Path-Unfold
Compute the shortest paths trees for *every* node, and use the *shortest* tree as te cut tree. Here, the *length* of a tree is the sum of the lengths of all edges in T.

(d) Long-Shortest-Path-Unfold
Likewise, use the *shortest* shortest paths tree as the cut tree.

Note that not all shortest paths trees of a graph necessarily have the same length: Consider a graph consisting of a cycle $(v_1, v_2), \ldots, (v_n, v_{n+1} = v_1)$, where

$$\text{costs}(v_i, v_{i+1}) = \begin{cases} n-3 & \text{for } i = 1 \\ 1 & \text{otherwise.} \end{cases}$$

Then, a shortest path tree with root v_1 will contain the edge (v_1, v_2) (because it is the shortest path from v_1 to v_2); it will not contain this edge if the root is v_2; hence, the two shortest-paths trees have the lengths $n-1$ and $n-2$, respectively.

A straightforward implementation of this method is shown in Rule 5: For each vertex $v \in V(P)$, it computes the length of the shortest-paths tree with root v and then cuts P along the edges of a shortest shortest-paths tree. We omit the pseudo-code for Long-Shortest-Paths-Unfold, since it is essentially the same as Short-Shortest-Paths-Unfold.

Rule 5 Short-Shortest-Paths-Unfold (P)

```
for all nodes $v \in G(P)$ do
  costs$[e] = ||e||\ \forall e \in G(P)$
  $T_v$ = Shortest-Paths-Tree $(G, v, \text{costs})$
let $T_{\min}$ be the tree of minimal length
return Cut $(P, T_{\min})$
```

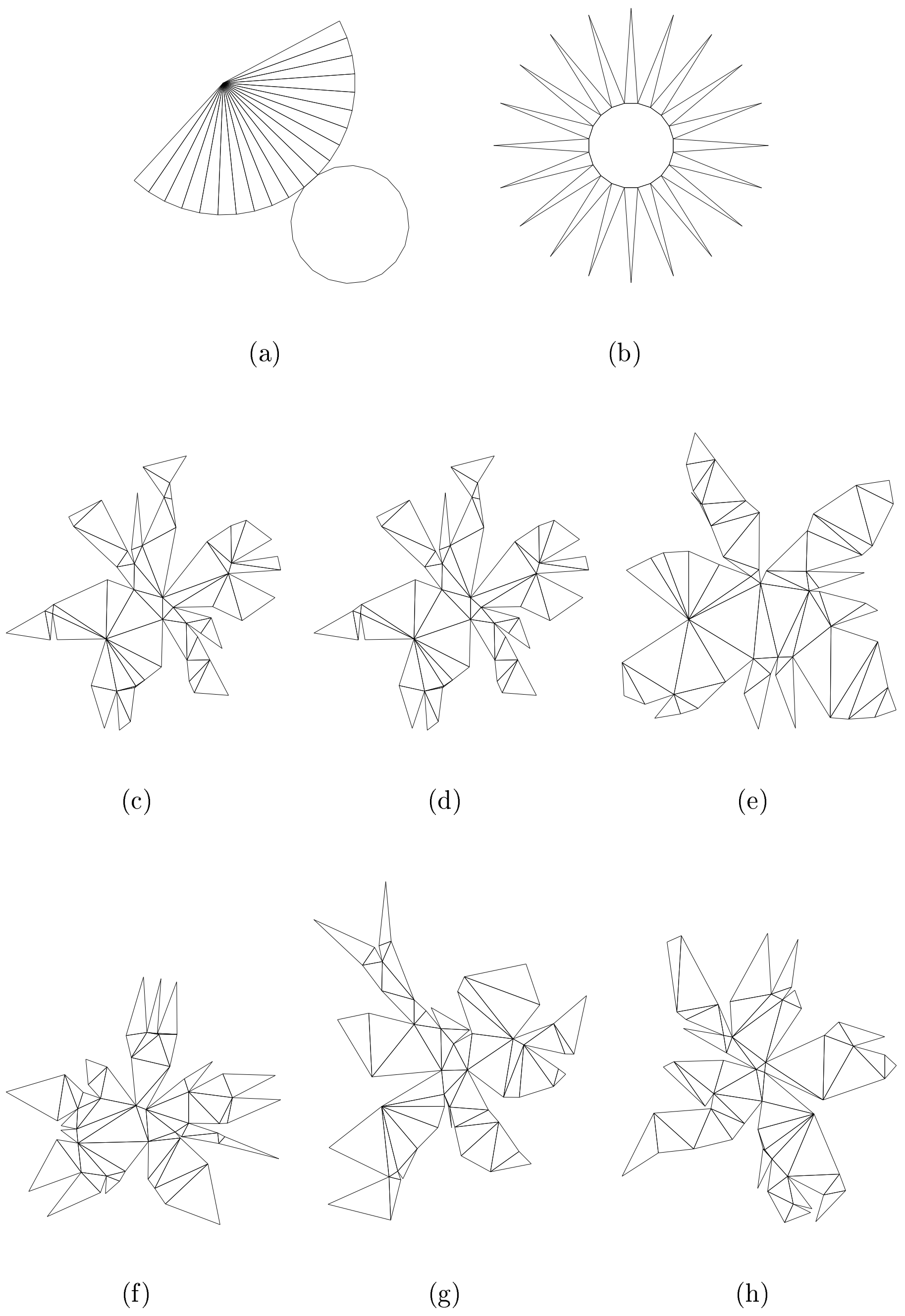


Figure 32: Shortest-paths unfoldings constructed by the unfolding rules introduced on page 48 ((a)–(d)) and 51 ((e)–(h)). Figures (a) & (b) are nets of the same cone, but with different root nodes (minimal & maximal degree). Figures (c)–(h) are different nets of a spherical polytope, obtained by the unfolding rules described on page 48 and 51.

Another idea is to find a shortest path tree in $D(P)$ and use it as a join tree, where the costs of an edge $(f, g) \in D(P)$ is the distance between the barycenter of f and the barycenter of g. This leads to four more shortest-paths unfolding rules.

(a′) SHORTEST-PATH-JOIN-MIN-UNFOLD
Choose a node with minimal degree as the root node.

(b′) SHORTEST-PATH-JOIN-MAX-UNFOLD
Choose a node with maximal degree.

(c′) SHORT-SHORTEST-PATH-JOIN-UNFOLD
Compute the shortest-paths tree with minimal length.

(d′) LONG-SHORTEST-PATH-JOIN-UNFOLD
Compute the shortest-paths tree with maximal length.

Rule 6 SHORTEST-PATHS-JOIN-UNFOLD (P, v)

```
for all edges $(f_1, f_2) \in D(P)$ do
  costs$[e] = ||\text{barycenter}(f_1) - \text{barycenter}(f_2)||$
$T_v$ =SHORTEST-PATHS-TREE $(G, v, \text{costs})$
return CUT $(P, T_v)$
```

Overlaps: After the variations of STEEPEST-EDGE-UNFOLD, the best five variations of SHORTEST-PATHS-UNFOLD immediately follow STEEPEST-EDGE-UNFOLD in the ranking of page 93:

(-)	SHORTEST-PATHS-UNFOLD	3.1 %
(a)	SHORTEST-PATHS-MIN-DEGREE-UNFOLD	14.7 %
(b)	SHORTEST-PATHS-MAX-DEGREE-UNFOLD	15.7 %
(c)	SHORT-SHORTEST-PATHS-UNFOLD	3.6 %
(d)	LONG-SHORTEST-PATHS-UNFOLD	2.3 %
(a′)	SHORTEST-PATHS-JOIN-MIN-UNFOLD	3.4 %
(b′)	SHORTEST-PATHS-JOIN-MAX-UNFOLD	3.1 %
(c′)	SHORT-SHORTEST-PATHS-JOIN-UNFOLD	19.2 %
(d′)	LONG-SHORTEST-PATHS-JOIN-UNFOLD	11.0 %

Figure 32 shows nets produced with these unfolding rules; Figure 33 shows a would-be net for each rule.

Running time: A shortest-paths tree of $G(P)$ can be computed in time $\mathcal{O}(|E(P)| \log |V(P)|)$, which is of the same order as $\mathcal{O}(|F(P) \log |F(P)|)$, since the number of facets, edges and vertices of P are linearly related. This is also the total running time of Shortest-Paths-Unfold (P, v).

To determine the shortest or longest shortest-paths tree, we need $|V(P)|$ calls to Shortest-Paths-Tree; thus the unfolding rules that compute a shortest or longest shortest-paths tree can be implemented to run in time $\mathcal{O}(|V(P)|^2 \log |V(P)|)$.

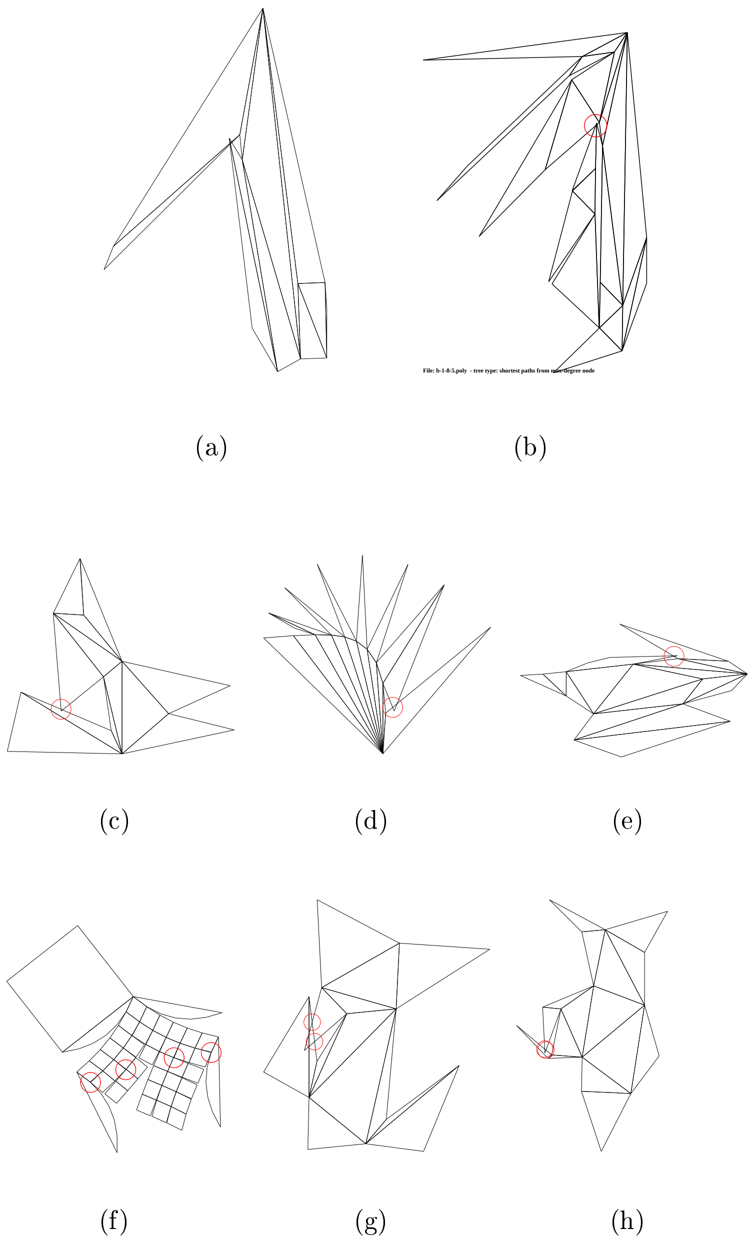

File: b-1-8-5.poly - tree type: shortest paths from max degree node
(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)

Figure 33: A bad unfolding for each of the various ...-Shortest-Paths-...-Unfold() rules.

4.7 Direction unfolding

The general idea behind the next class of unfolding rules is the following: Let $c \in I\!R^3$ be an objective function in general position with respect to P (again, see Ziegler's or Lóvász' Lecture notes for how to compute such a vector explicitly, [42] & [26]), and let v_- and v_+ the lowest and highest vertex of P w.r.t. c. With the exception of the unfolding rule presented in Section 4.7.4 we want to obtain a net by choosing cut edges along (in some sense) "straight" paths from v_- to v_+ (see Figure 34).

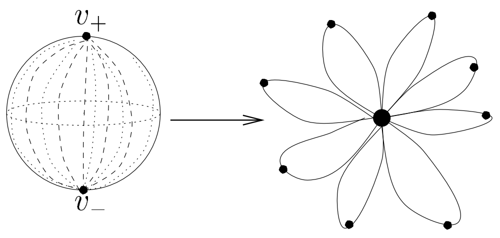


Figure 34

4.7.1 Steepest edge cut tree

A conjecture about the kind of cut paths that lead to a non-overlapping unfolding was to cut along paths that are "as straight as possible". In this case, we will interpret "straight" as "steep": If c is an objective function in general position, here is one possible way to choose cut edges which belong to (in some sense) "straight" paths: Let v_- and v_+ be the bottom and top vertex of P w.r.t. c. For each vertex, we choose the *steepest ascending edge* at that vertex as a cut edge. This should intuitively lead to straight paths from v_- to v_+.

The union of these edges clearly forms a spanning tree of $G(P)$: If P has n vertices, $n - 1$ edges are chosen, and the resulting graph doesn't contain a cycle. (Proof: Suppose there is a cycle. c is in general position w.r.t. P. Hence the cycle has a lowest vertex v_- with *two* ascending edges adjacent to it, which is a contradiction.)

This rule is the most promising rule we found; Figure 35 shows a net.

While producing nets for almost all of our test polytopes, there are a few

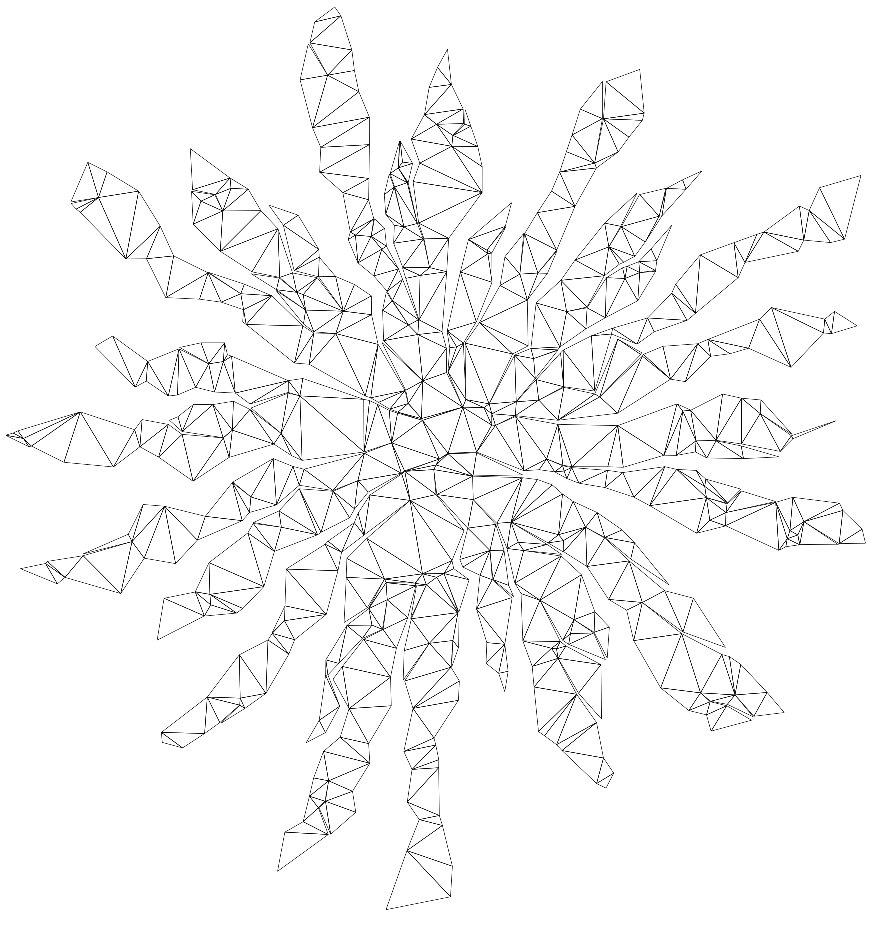


Figure 35: Net of a spherical polytope (400 vertices, 796 facets) constructed with STEEPEST-EDGE-UNFOLD.

Rule 7 STEEPEST-EDGE-UNFOLD (P, c)

```
INITIALIZE   $T = \emptyset$
for all vertices $v \in P, \quad v \neq v_+$ do
  compute the steepest ascending edge incident to $v$ (w.r.t. $c$),
    i.e. the edge $e = (v, w)$, for which

        $\frac{\langle c, v - w\rangle}{|v - w|}$   is maximal.

  $T = T \cup \{e\}$
return CUT $(P, T)$
```

polytopes (118 out of 9703, to be precise) where this rule produces would-be nets; for example, for the polytope in Figure 36.

Overlaps: This rule is the one that produces the fewest would-be nets for the set of test polytopes described in Section 4.2: Only 1.2% of the unfoldings are would-be nets.

We also implemented several variations of STEEPEST-EDGE-UNFOLD which differ in the way they choose the objective function c. When comparing the results of these unfolding rules, we discovered a surprising fact: Whichever “cut direction” c was chosen, for all these rules there were roughly the same number of overlapping unfoldings. Furthermore, for STEEPEST-EDGE-UNFOLD (P, C), there were always $\approx 110(\pm 10)$ polytopes for which it produced a would-be net.

We will reconsider a variation of this rule in Section 4.13.

Running time: The steepest edge incident to a node v can be found in time $\mathcal{O}(\gamma(v))$, where $\gamma(v)$ is the number of edges incident to v. Each edge is considered twice during the procedure, since each edge is incident to exactly two nodes. Thus the running time is $\mathcal{O}(|E(P)|)$.

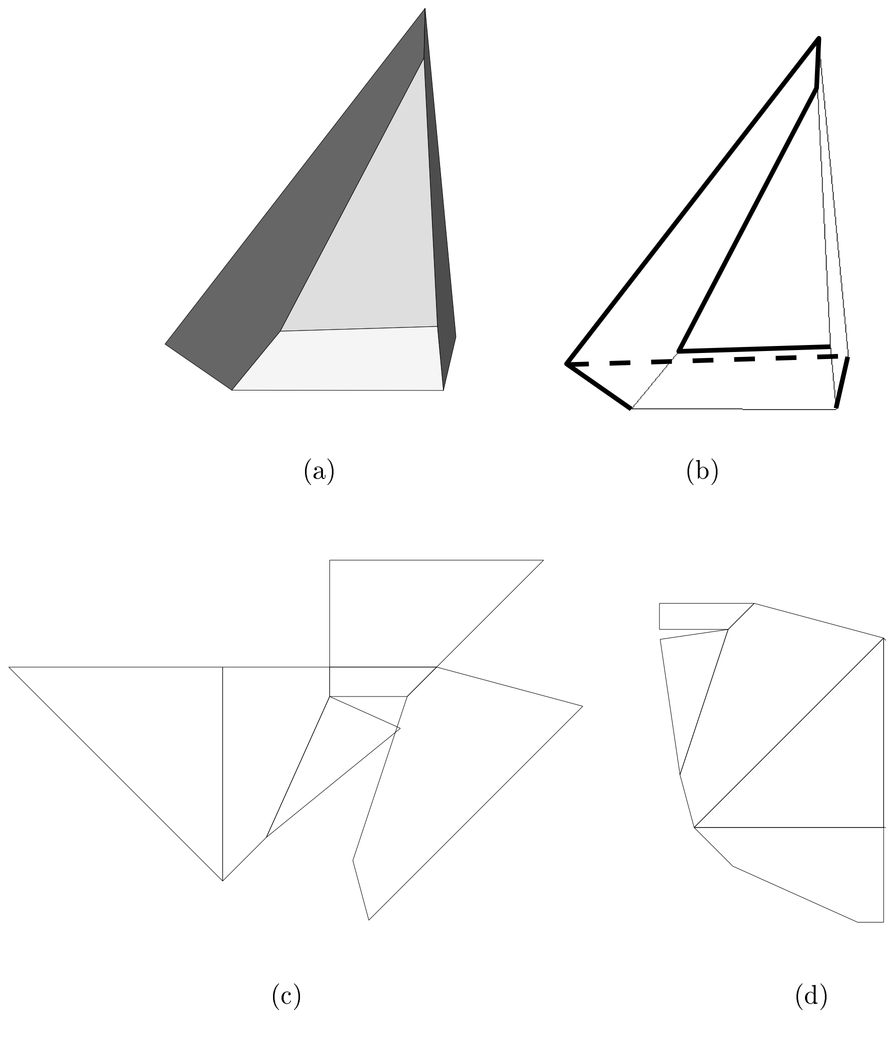


Figure 36: A polytope P (a), its skeleton with steepest edges shown (thick lines) in (b), the steepest-edge unfolding with overlaps (c) and a minimum-perimeter net (d).

4.7.2 Greatest increase cut tree

Another possible method to obtain “straight” paths is used in the following rule. Here, for each vertex v except the highest vertex of P, among all ascending edges incident to v we choose the edge with *greatest increase* w.r.t. a given objective function c in general position, that is, the edge $e = (v, w)$ for which

$$\langle c, w - v \rangle \quad \text{is maximal and positive.}$$

The edges thus obtained form a spanning tree for the same reason as in Section 4.7.1. Figure 37 shows a net and a would-be net produced by GREATEST-INCREASE-UNFOLD (P, c) for two different choices of c.

Rule 8 GREATEST-INCREASE-UNFOLD (P, c)

```
INITIALIZE   $T = 0$
for all vertices $v \in P$ do
   compute the edge with greatest increase (w.r.t. $c$) incident to
     $v$, .i.e. the edge $e = (v, w)$ , for which

                         $\langle c, w - v \rangle$   is maximal.

   $T = T \cup \{e\}$
return CUT $(P, T)$
```

Overlaps: 8.6% of the unfoldings overlap.

Running time: Analogously to STEEPEST-EDGE-UNFOLD the running time is $\mathcal{O}(|E(P)|)$.

Variations

Up to now, we didn’t say anything about the particular choice of c. We investigated a few variations with both of the two unfolding rules above (Sections 4.7.1 & 4.7.2):

(1) Compute a pair of diametral vertices p and q, and use $c = p - q$ as the ”cut direction”.

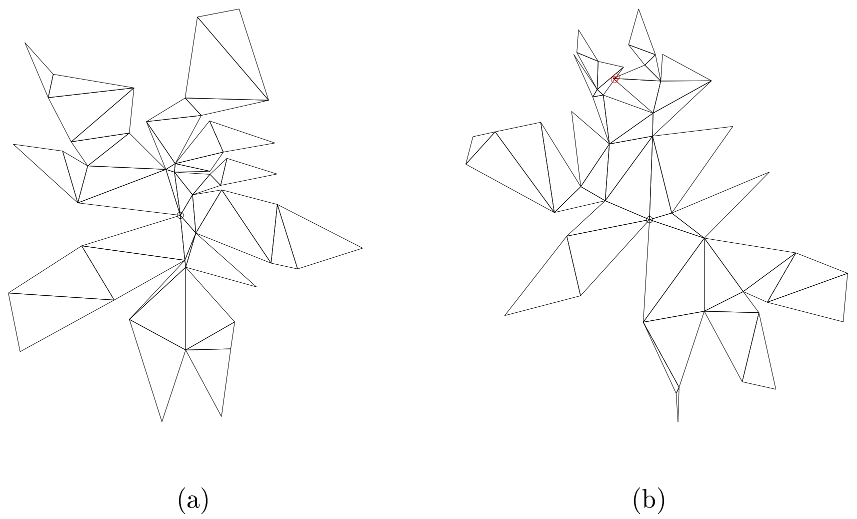


Figure 37: Two greatest-increase unfoldings of a spherical polytope obtained using different objective functions. (b) is a would-be net.

(2) As (1), but use $c = (p - q) + r$, where

$$r = (\lambda, \lambda^2, \lambda^3)$$

for a *small* $\lambda > 0$, for perturbation.

(3) Choose a random vector c with $||c|| = 1$.

It appears that the particular choice of c does not affect the overall performance of Steepest-Edge-Unfold and Greatest-Increase-Unfold.

4.7.3 Flat edge unfolding

A rule which is in a sense "dual" to cutting the steepest ascending edge at each vertex is, informally: Instead of cutting the boundary along "steep" edges, we *join* facets along *flat* edges (that is, along edges which are as perpendicular as possible to a given vector c). One possible way to do this is the following: Assign *costs* to each edge $e^* \in D(P)$:

$$\text{costs}[e^*] = \left| \frac{\langle c, p - q\rangle}{||c||\,||p - q||} \right|,$$

where $e = (p, q)$ is the corresponding primal edge of e^*. Then compute a *minimum spanning tree* T of $D(P)$ w.r.t. these costs, and use this tree as the join tree (see Rule 9).

Rule 9 FLAT-SPANNING-TREE-UNFOLD (P, c)

```
INITIALIZE   $T = 0$
for all edges $e = (p, q) \in P$ do
   let $e^*$ be the corresponding dual edge of $e$
   let $\text{costs}(e^*) = \left| \frac{\langle c, p-q \rangle}{||c|| \, ||p-q||} \right|$
compute a minimal spanning tree $T$ in $D$ w.r.t. costs
return JOIN $(P, T)$
```

Overlaps: At first sight one might assume that the unfoldings should be similar to the ones produced by STEEPEST-EDGE-UNFOLD (P, c). It turned out that this is only half of the truth — Figure 38 shows a typical would-be net of a spherical polytope, and the kind of overlaps this rule leads to. It is almost the worst rule we came up with: 51.3% of its unfoldings overlap.

Running time: There are several algorithms to compute a minimum spanning tree of a graph $G = (V, E)$ efficiently in time $\mathcal{O}(|E| \log |E|)$ [7]. Thus we yield a total running time of $\mathcal{O}(|E| \log |E|)$.

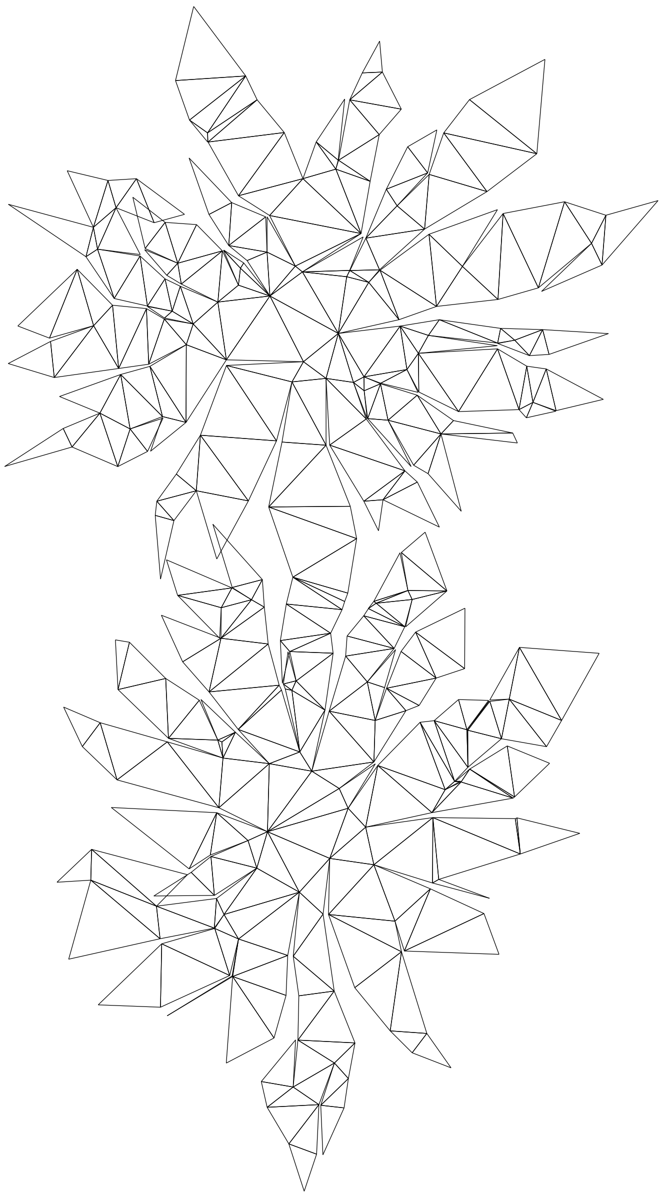


Figure 38: Typical flat spanning tree unfolding of a spherical polytope.

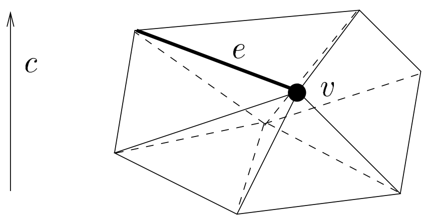


Figure 39: e is the rightmost ascending edge at v (w.r.t. c).

4.7.4 Cutting the rightmost ascending edge

Let c be a vector in general position and v_- and v_+ the lowest and highest vertex of P w.r.t. c. For each vertex $v \in V(P) \setminus \{v_-\}$, we can then compute the *rightmost ascending edge* incident to v (viewed from the inside of P).

The collection of these edges, together with the "flattest ascending edge" incident to v_-, forms a spanning tree of $G(P)$: There are $|V(P) - 1|$ edges. Furthermore, there is at least one edge from a vertex w which is adjacent to v_+, such that (w, v_+) is the rightmost ascending edge emanating from w: Suppose otherwise, then the rightmost ascending edges of all nodes adjacent to v_+ would form a cycle. This is impossible, since they are all strictly ascending.

Rule 10 RIGHTMOST-ASCENDING-EDGE-UNFOLD (P, c)

```
INITIALIZE  $v_-$ = the minimal vertex of $P$ w.r.t. $c$
            $b$  = the barycenter of $P$
            $T$  = $\emptyset$
for all vertices $v \in P$ do
    let $e = (v, w)$ be the rightmost ascending edge incident to $v$,
      i.e. the edge, for which
            $$\frac{\langle w - v, c\rangle}{||w - v||\,||c||} > 0, \quad \text{and}$$
    $\det(c, \frac{v-b}{||v-b||}, \frac{w-v}{||w-v||})$ maximal
    $T = T \cup \{e\}$
return CUT $(P, T)$
```

Overlaps: We expected this rule to produce unfoldings like the ones obtained by Left-First-Depth-First-Unfold. Not behaving as expected, a mere 12.8% of the unfoldings produced by this rule overlap.

Figure 40 shows a few typical unfoldings produced by Rightmost-Ascending-Edge-Unfold. For the objective function c, we tried the same choices as for Steepest-Edge-Unfold; the particular choice of c, however, did not influence the overall performance of this rule.

Running time: The statements in the for-loop can be implemented to run in constant time; each edge is considered exactly twice during the execution of the whole rule; thus the total running time is $\mathcal{O}(|E(P)|)$.

4.7.5 Normal order unfolding

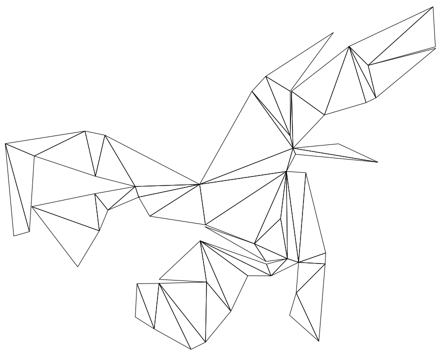
A “natural” order of the facets is as follows: Let c be a vector in general position, and for each facet compute the angle α_f between c and the normal vector on f. Then let $L = (f_1 \ldots, f_m)$ be a list of all facets of P such that $\alpha_i \leq \alpha_j$ if $i < j$. We start with $T = \{f_1\}$ as the initial tree and then incrementally build a join tree as follows: Let f_i be the first facet in L that is adjacent to a facet $f_j \in T$. Add the (dual) edge (f_i, f_j) to T, and remove f_i from L.

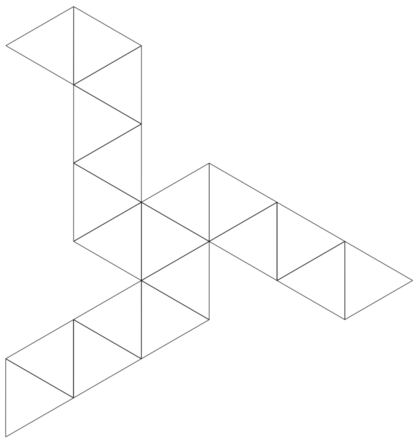
As before, there are many possible ways to choose f_j, if f_i is adjacent to more than one facet in T. We settled for the same rules as in Section 4.8 above:

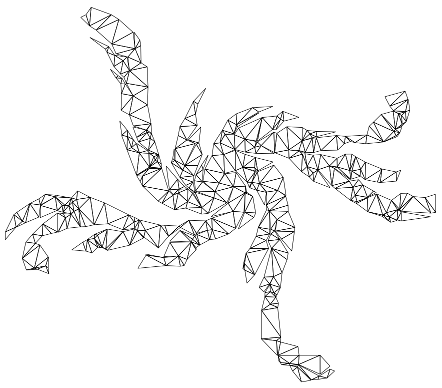
Min
: Choose f_j such that j is minimal, that is, choose f_j to be a facet that has been embedded as early in the process as possible — this should result in breadth-first search like join trees,

Max
: Choose f_j such that j is maximal — this is expected to produce depth-first search like join trees,

Flat
: Choose f_j such that the primal edge e shared by f_i and f_j is as “flat” as possible, that is, the (absolute) angle between e and c is maximal, and


(a) Cubical polytope, 40 vertices


(b) Icosahedron


(c) Spherical polytope, 300 vertices

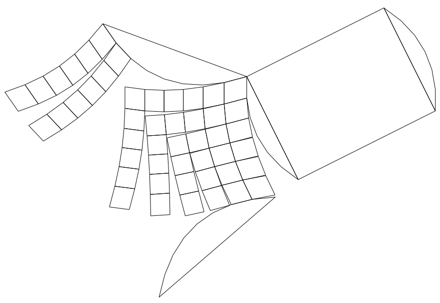

(d) Turtle polytope $T(3,4)$

Figure 40: Unfoldings constructed with RIGHTMOST-ASCENDING-EDGE-UNFOLD.

LONG

Choose f_j such that the primal edge e shared by f_i and f_j is as long as possible.

The tree T constructed as above is a spanning tree of $D(P)$ again: At any time during the construction, T is connected and contains $|T| - 1$ edges; therefore, it must be a tree.

Rule 11 NORMAL-ORDER-UNFOLD (P, c)

```
$m = |F(P)|$ *let* $n_{f_i}$ *be the normal vector on* $f_i$ *for* $i = 1, \ldots, m$
$\alpha_i = \arccos \frac{\langle n_{f_i}, c \rangle}{||c||}$ for $i = 1, \ldots, m$
*Let* $L = (f_1, \ldots, f_m)$ *be the sequence of all facets, such that*
  $\alpha_1 < \alpha_2 < \ldots < \alpha_m$
$t = \{f_1\}$
**while** $L \neq \emptyset$ **do**
  *Let* $f_i$ *be the first facet in* $L$ *that is adjacent to a facet in* $T$
  $f_j$ = CHOOSE-ADJACENT-TREE-FACET $(f_i)$
  *let* $e$ *be the common edge of* $f_i$ *and* $f_j$
  $T = T \cup \{e\}$
  $L = L \setminus \{e\}$
**return** JOIN $(P, T)$
```

Overlaps: 15.5% of the unfoldings overlapped.

Running time: The computation of $\alpha_1, \ldots, \alpha_m$ can be done in $\mathcal{O}(|F(P)|)$, sorting them takes $\mathcal{O}(|F(P)| \log |F(P)|)$. Finding the first facet in L which is incident to a given facet g can be done in $\mathcal{O}(\gamma(g))$; since each facet is a candidate for being the first facet in L at most as often as its number of neighbours, the total time for these lookups during the algorithm is $\mathcal{O}(|E(P)|)$, which leads to a total running time of $\mathcal{O}(|F(P)|^2 \log |F(P)|)$.

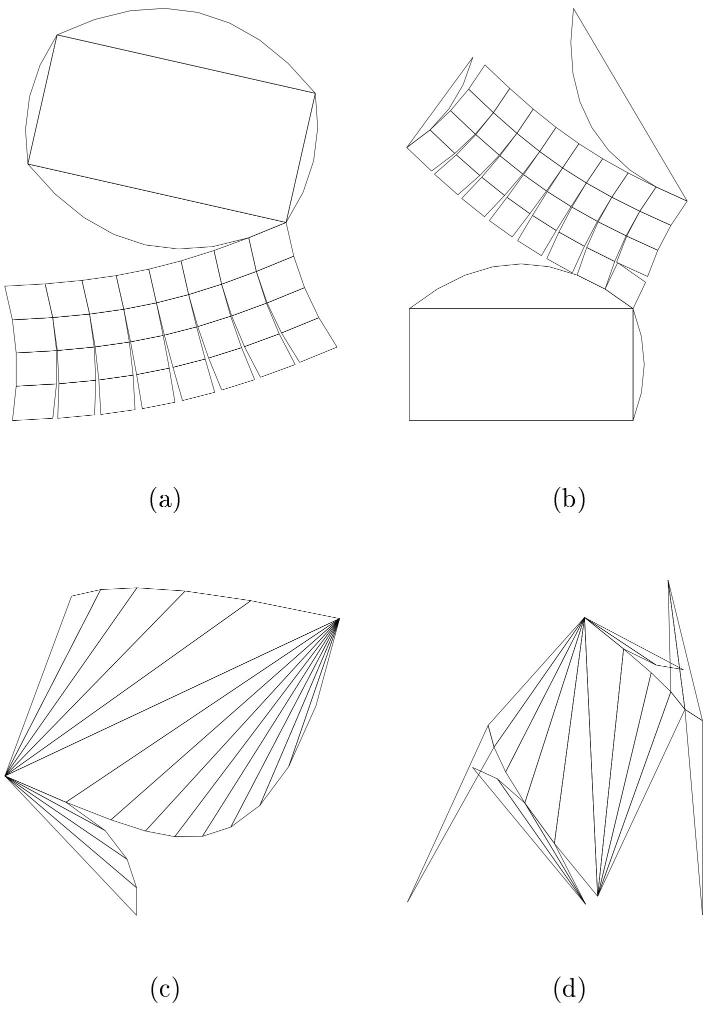
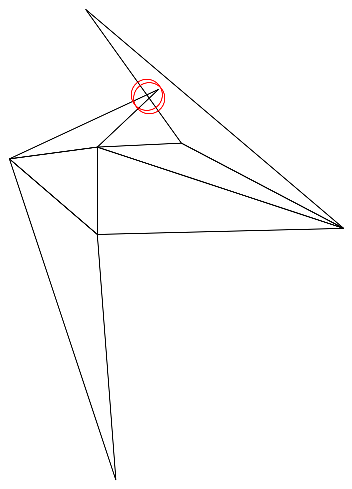
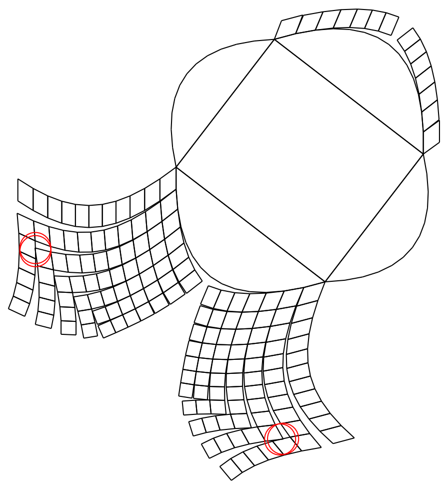


Figure 41: Two pairs of different unfoldings of the turtle polytope $T(2,4)$ ((a) & (b)) and the cyclic polytope $C_3(11)$ ((c) & (d)), using different objective functions for Normal-Order-Min-Unfold.


(a)


(b)

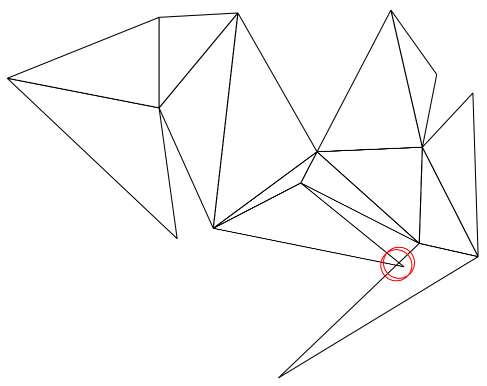

(c)

Figure 42: An overlapping unfolding for each of the remaining variations of Normal-Order-Unfold(): Max (a), Long (b) & Flat (c).

4.8 Shellings

A shelling is a special ordering of the faces of P and does not have much to do with unfolding at first sight. The idea of using them to construct unfoldings comes from the proof of a theorem by Bruggesser & Mani, which guarantees the existence of shellings with a special property [6].

Definition 4.2 (Pure)
A polyhedral complex $\mathcal{C}$ is pure, if each of its facets is contained in a facet of dimension $\dim(\mathcal{C})$ *(i.e. all inclusion-maximal facets of $\mathcal{C}$ have the same dimension.*

This is clearly the case for the boundary complex $\mathcal{B}(P)$ of a polytope.

Definition 4.3 (Shelling)
Let $\mathcal{C}$ be a pure k-dimensional polytopal complex. A linear ordering of its facets $(f_1, f_2, \dots, f_r)$ is a shelling if either $\mathcal{C}$ is 0-dimensional or the following conditions hold:

(i) The boundary complex $\mathcal{C}(\partial f_1)$ has a shelling.

(ii) For $1 < j \leq r$ the intersection of f_i with the previous facets is non-empty and is a beginning segment of a shelling of the $(k-1)$-dimensional boundary complex of f_j, i.e.

$$f_j \cap (\bigcup_{i=1}^{j-1} f_i) = g_1 \cup \ldots \cup g_s$$

for some shelling $(g_1, \dots, g_s, \dots, g_t)$ of $\mathcal{C}(\partial f_j)$, and $1 \leq s \leq t$.

A polytope is shellable, if its boundary complex $\mathcal{C}(\partial P)$ has a shelling.

It is not difficult to prove that every polytope is shellable [6], [42]. For 3-dimensional polytopes, this means in particular: For every 3-polytope P there exists an ordering $(f_1, \dots, f_r)$ of its facets such that for each $i \in \{1, \dots, r\}$ the intersection

$$f_i \cap \bigcup_{j=1}^{i-1} f_j$$

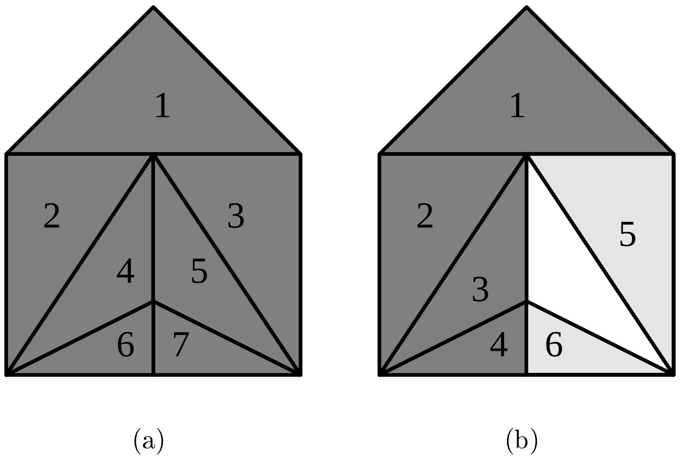


Figure 43: Two 2-dimensional complexes, a shellable (a) and a non-shellable one (b): When adding the last one of the facets incident to the bottom right vertex, condition (ii) of Definition 4.3 will be violated since the intersection with the previous facets is not pure 1-dimensional. If you omit one of the facets 5 or 6, the remaining complex is shellable.

is a non-empty set of *edges* of P.

Here is the way back to unfolding polytopes: If $(f_1, \ldots, f_r)$ is a shelling of a polytope, we can start with an embedding of f_1 and construct an unfolding by successively joining the remaining facets — *in the order they appear in the shelling* — to one of the facets $f_1, \ldots, f_{i-1}$, because the intersection with at least one of the latter is an edge of both facets.

Theorem 4.1 (Bruggesser & Mani [6])
Let $P \in \mathbb{R}^n$ be a polytope. Let $x \in \mathbb{R}^d$ a point in general position w.r.t. P and outside of P. Then the boundary complex $\mathcal{C}(\partial P)$ has a shelling in which the facets of P that are visible from x come first.

A facet $f \subset P$ is *visible from x* if for every $y \in F$ the closed line segment $[x, y]$ intersects P only in y.

(Idea of the) proof [42]: Given P and x, choose a *shelling line* ℓ through x and an interior point p of P. Assume ℓ is not parallel to any of the facet

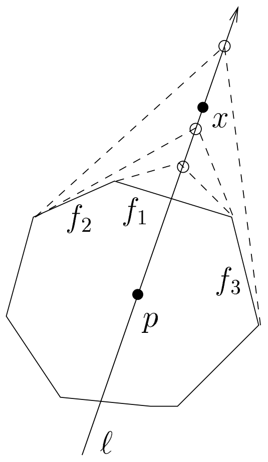


Figure 44: Illustration for the proof of Theorem 4.1.

hyperplanes $\operatorname{aff}(f)$, such that the intersections of $\ell \cap \operatorname{aff}(f)$ are distinct, and orient it from p to x. Let f_1 be the facet where ℓ leaves P. Now imagine a rocket that lifts off of f_1 and flies along ℓ, away from P. Since the intersections $\ell \cap \operatorname{aff}(f)$ are all distinct, as the rocket flies off "to infinity", facets of P will appear at the horizon, one by one (see Figure 44). Then, let the rocket "pass through infinity" and come back to P from the other side. After that, the remaining facets of P will disappear on the horizon, one by one again. The thus obtained ordering of the facets is a shelling and, since the rocket eventually passes through x, in the shelling the facets visible from x come first. ($\square$)

At first sight, this looks like an ordering of the facets similar to the one from the previous section, but it isn't: The normal ordering of the facets is completely determined by the objective function c, whereas for a specific direction of the shelling line there can be different shelling orders, depending on where the shelling line intersects P. In particular, given two distinct facets f_1 and f_2 of P, there is always a shelling order in which f_1 is first and f_2 is last (just choose a shelling line through the interiors of f_1 and f_2).

A remaining point is that each facet f might be adjacent to *more* than one facet appearing prior to f — so, which facet should we join it to? There are many thinkable ways to make a decision here – we decided to try five

Rule 12 SHELLING-UNFOLD (P)

```
INITIALIZE  $T = \emptyset$
            $S = (f_1, \dots, f_m)$ a shelling order of
                  $P$'s facets like in the proof of
                  Theorem 4.1.
for $i = 2, \dots, m$ do
  $f_j$ = CHOOSE-ADJACENT-PRIOR-FACET ($f_i, S$)          (*)
  $T = T \cup \{(f_i, f_j)\}$
return JOIN ($P, T$)
```

different strategies:

CHOOSE-MINIMAL-INDEX (f_i, S) Select the facet $f \in S$ for which

$$j = \min_{k=1,\dots,i-1} \{k \mid f_i \ \&\ f_j \text{ are adjacent}\}$$

CHOOSE-MAXIMAL-INDEX (f_i, S) Select the facet $f \in S$ for which

$$j = \max_{k=1,\dots,i-1} \{k \mid f_i \ \&\ f_j \text{ are adjacent}\}$$

CHOOSE-FLAT-EDGE (f_i, S) Let $\{e_1, \dots, e_k\}$ be those edges of f_i, for which the facet $f_i^+(e) \in S$, and let

$$s_j = \left| \frac{\langle e_j, c \rangle}{||e||\,||c||} \right|, \quad j = 1, \dots, i-1,$$

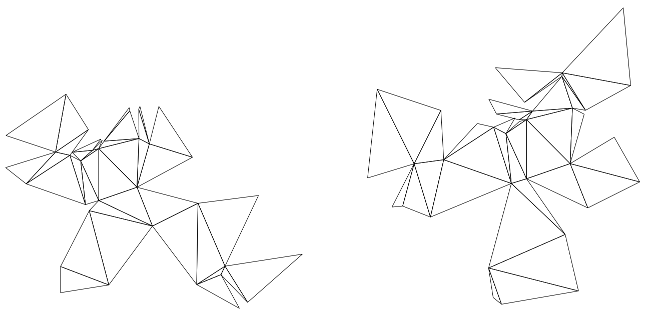
where c is the objective function used to obtain the shelling. Select the facet $f_i^+(e_j)$ such that s_j is as small as possible — that is, such that the (absolute) angle between c and e_j is as big as possible, or, in other words, such that the edge is as "flat" as possible, w.r.t. c.

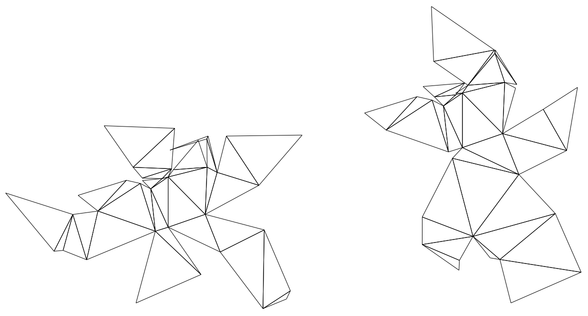
CHOOSE-LONG-EDGE (f_i, S) Let $\{e_1, \dots, e_k\}$ be the edges of f_i, for which the facet $f_i^+(e) \in S$, and let $e_{\max}$ be the longest such edge. Select the facet $f_i^+(e_{\max})$.

CHOOSE-MIN-NORMAL-DIFFERENCE (f_i, S) Among all facets $\{f_j \mid j < i\}$, choose f such that the difference of the normal vectors of the two facets, $|n_{f_i} - n_f|$, is as small as possible.

Overlaps: Here it is of great importance, how the joining facet is chosen in step (*) of Rule 12: Selecting the facet that comes first in the shelling order (Choose-Minimal-Index) leads to breadth-first search like unfoldings and — not surprisingly then — a similar performance: 24.9% of the unfoldings overlap. Likewise, Choose-Maximal-Index leads to depth-first search like unfoldings with almost equally many overlapping unfoldings: 47.1%. The remaining three variations lie in between these two (see the "Statistics" Section 4.11 for more detailed figures).

Running time: For each facet f, the intersection of $\mathrm{aff}(f)$ with the shelling line can be computed in time $\mathcal{O}(1)$, sorting the intersection points on the shelling line takes $\mathcal{O}(|F(P)| \log |F(P)|)$, which also is the total running time.


(a) Min-Index (b) Max-Index


(c) Flat-Edge (d) Long-Edge

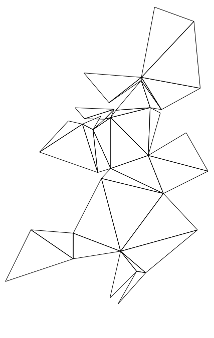

(e) MinDiff

Figure 45: Five different unfoldings of the same polytope, obtained by Shelling-Unfold, using the five different rules to select the adjacent facet.

4.9 Incremental unfolding

Up to now, we use a three-step procedure:

(1) Compute a spanning tree of $G(P)$ or $D(P)$.

(2) Compute the unfolding of P according to this tree.

(3) Check the unfolding for overlaps.

Since this didn't succeed, we will now try a different approach:

(1) Start with embedding a single facet. While there are still unaccounted-for facets left, join them to the net one by one, possibly taking into account how the partial net constructed thus far looks like, and/or the positions at which they can be joined.

(2) Check the resulting net for overlaps.

In order to avoid writing "already accounted-for facet", "facet which is not yet embedded and has an embedded neighbouring facet in N" and the like over and over again during the descriptions of the layout processes, we introduce a little more notation:

Definition 4.4
For a (finished or partial) net, let $\partial(N)$ be the set of edges e where exactly one *of the facets incident to e is in N; that is, $\partial(N)$ is the* boundary *of N.*

Let $\partial^*(N) = \{e^* = (f, g) \mid e \in \partial(N), f \in \partial^+(N), g \in \partial^-(N)\}$.

Let $\partial^+(N)$ be the facets incident to edges in $\partial(N)$ that are not *(yet) in N (the "to do list") and $\partial^-(N)$ be the facets of N which are incident to edges in $\partial(N)$.*

Obviously, if N is finished, then

$$\partial(N) = \partial^*(N) = \partial^+(N) = \partial^-(N) = \emptyset.$$

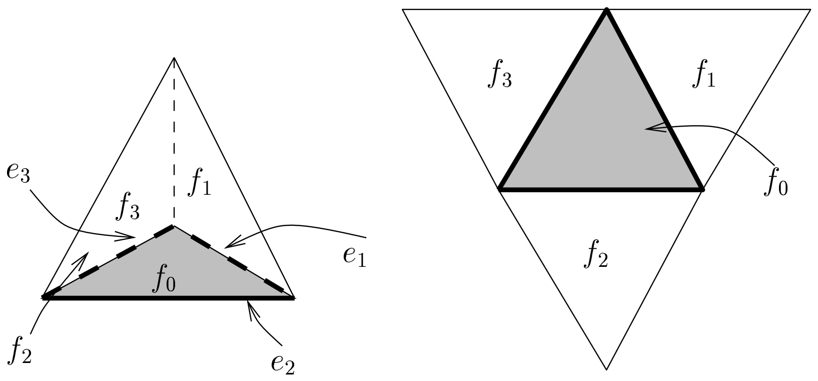


Figure 46: A partial net of a tetrahedron; $\partial(N) = \{e_1, e_2, e_3\}$ (indicated by thick lines), $N = \partial^-(N) = \{f_0\}$, $\partial^+(N) = \{f_1, f_2, f_3\}$, $f_i = f_0^+(e_i)$ for $i = 1, 2, 3$.

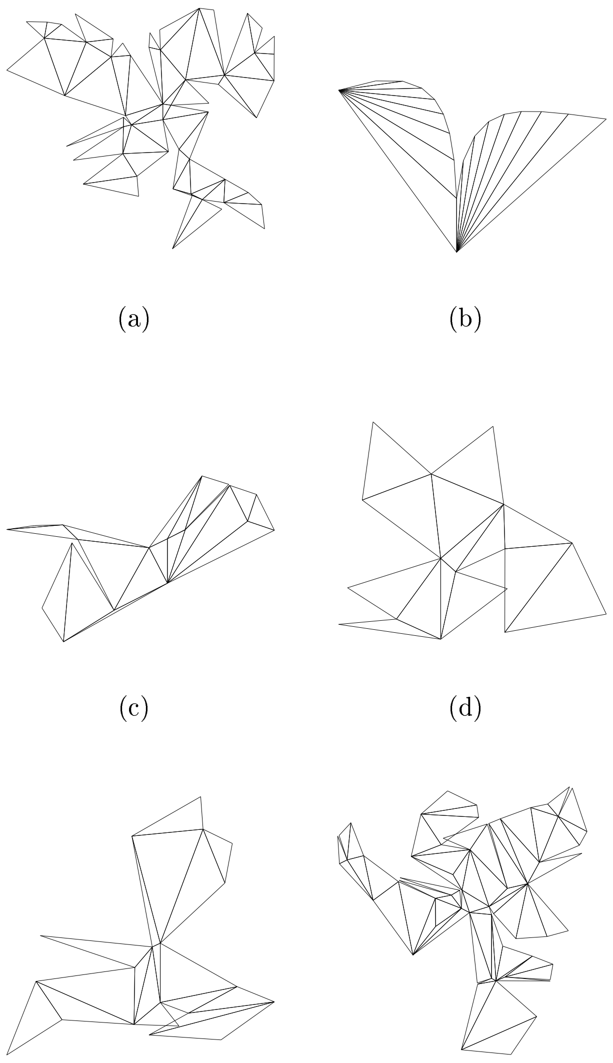
4.9.1 Nearest facet first

The first unfolding rule illustrates what we mean by "taking the current partial net into account". It results in unfoldings similar to those obtained by breadth-first join trees, and works as follows: We start with some facet f of P, and compute the barycenter b (of it's unfolded image f_N). Then we incrementally join the facets of P to the unfolding along an edge whose midpoint is as near to b as possible.

This amounts to the heuristic "if a facet is *near* the first facet f of the unfolding (the center facet) on P, then it should also be *near* f_N in the unfolding of P".

Rule 13 NEAREST-FACET-FIRST (P)

```
choose a facet $f$
$N = \{f_N\} = $ LAYOUT $(f)$
compute the barycenter $b$ of $f_N$
while $\partial^+(N) \neq \emptyset$ do
   let $e \in \partial(N)$ be an edge such that is midpoint has minimal
      distance to $b$ among all edges in $\partial(N)$
   $N = N \cup$ LAYOUT $(e^+, e^-)$.
return $N$
```



(a) (b)

(c) (d)

(e) (f)

Figure 47: Unfoldings obtained by NEAREST-FACET-FIRST.

Overlaps: 20.51% of the unfoldings overlapped.

Running time: The distance of a facet f from b (w.r.t. a bounding edge of f) can be computed once the boundary edge becomes a member of $\partial(N)$. Using a priority queue as a data structure for $\partial(N)$, where insertion and deletion of an item takes $\mathcal{O}(\log |N|)$ time and the extraction of the edge with minimal distance can be done in $\mathcal{O}(1)$ time, all updates of $\partial(N)$ can be done in time $\mathcal{O}(|E(P)| \log |E(P)|)$, since each edge is inserted and removed from $\partial(N)$ exactly once. This yields a total running time of $\mathcal{O}(|E(P)| \log |E(P)|)$.

4.9.2 Place high vertices far away

The next unfolding rule is sort of the "other side of the coin", compared to the previous one: For a polytope P and an height function c in general position, let one of the facets incident to the lowest vertex of P be the start facet, and for each facet $f_i \in F(P)$ let $f_i^{\max}$ be the highest node of f_i. Furthermore, for a facet $f \in \partial^+(N)$ and an edge $e \in f$, let $f^{\max}(e)$ be the image of $f^{\max}$ if f is joined to the unfolding along e.

After embedding a start facet and computing its barycenter b, in each further step we join a facet f along an edge $e \in \partial(N)$ (i.e. $f = e^+$), such that the distance between b and $f^{\max}(e)$ is as large as possible.

The rationale behind this strategy is that "if a vertex is the highest vertex of a facet f, we want it to have the greatest distance from the center of the unfolding".

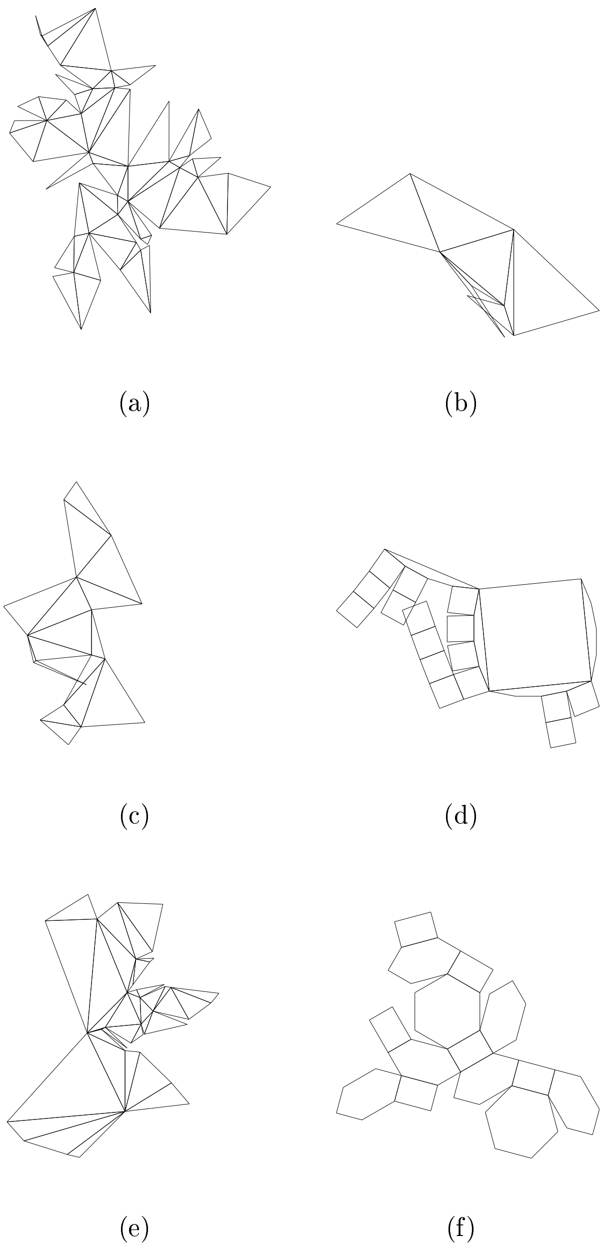


Figure 48: Unfoldings obtained by HIGHEST-VERTEX-FAR-AWAY.

Rule 14 HIGHEST-VERTEX-FAR-AWAY (P)

```
let $c \in I\!R^3 \setminus \{0\}$
let $f_0$ be one of the facets incident to the lowest node of $P$
$N =$ LAYOUT $(f_0)$
and compute the barycenter $b$ of the embedding of this facet.
compute the height $\text{height}(v) = \langle v, c \rangle$ of each vertex of $P$.
for all facets $f \subset P, \quad f \neq f_0$ do
   compute the highest node $f^+$ of $f$ w.r.t. $c$
while $\partial^+(N) \neq \emptyset$ do
   for all edges $e \in \partial(N)$ do
      compute the mid-point of $m_e$ of $e$
      compute $d_f = f_+ - m_e$
   choose the facet for which $\langle m_e - b, d_e \rangle$ is maximal, and
      embed it
return CHECK $(N)$
```

Overlaps: 40.2% of the unfoldings overlapped.

Running time: As before, using a priority queue as the data structure for $\partial(N)$ yields a total running time of $\mathcal{O}(|E(P)| \log |E(P)|)$.

4.9.3 Outward-pointing facets

The dream of finding unfoldings like the one in Figure 35 is still alive. Our next attempt to find such unfoldings in general is to join facets to the current partial net if they "point away" from some center facet of the unfolding. First, we define what we mean by "point away" here.

Definition 4.5 (Opposite)
Let $p = (p_1, \ldots, p_k)$ be a polygon, where $p_1, \ldots, p_k$ are the vertices of p in counterclockwise order. For an edge $e = (p_i, p_{i+1})$ of p, let

$$\text{opposite}(p, e) = \begin{cases} p_{i+k/2 \bmod k} & \text{if } k \text{ is odd,} \\ \frac{1}{2}(p_{i+k/2 \bmod k} + p_{i+k/2+1 \bmod k}) & \text{if } k \text{ is even.} \end{cases}$$

See Figure 49 for a sketch.

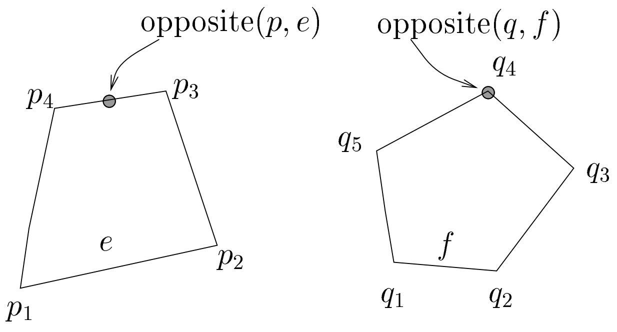


Figure 49

Now let N be a partial net, b a point in the interior of a polygon in N, and $e \in \partial(N)$. We say that the facet $f = e^+$ *points away from* b when joined along e, if b and opposite(f_N, e) lie on different sides of f_N, where $f_N =$Layout(f, e^-).

For an edge $e \in \partial(N)$, let m_e be its midpoint. During the construction of N, in each step we choose a facet $f \in \partial^+(N)$ for which the absolute angle between the vectors

$$m_e - b \quad \text{and} \quad \text{opposite}(f,e) - m_e$$

is minimal; here e denotes the edge along which f is joined to N.

A different notion of "pointing away"

Here is a different plausible definition of "outward-pointing": For an edge $e = (v_1, v_2) \in \partial(N)$, let $f_n =$Layout(e^+, e^-), and let $e_1 = (v_1, w_1)$ and $e_2 = (v_2, w_2)$ be the two edges of the polygon f_N which are incident to e. Furthermore, let b be an interior point of a polygon in N. We say that the facet $f = e^+$ *points away from* b when joined along e, if the points

$$\frac{1}{2}(w_1 + w_2) \quad \text{and} \quad b$$

lie on different sides of aff(e) (see Figure 51).

In the unfolding rule we then choose an edge $e \in \partial(N)$, such that the absolute angle between

$$((w_1 - v_1) + (w_2 - v_2)) \quad \text{and} \quad ((v_1 - v_2) - b)$$

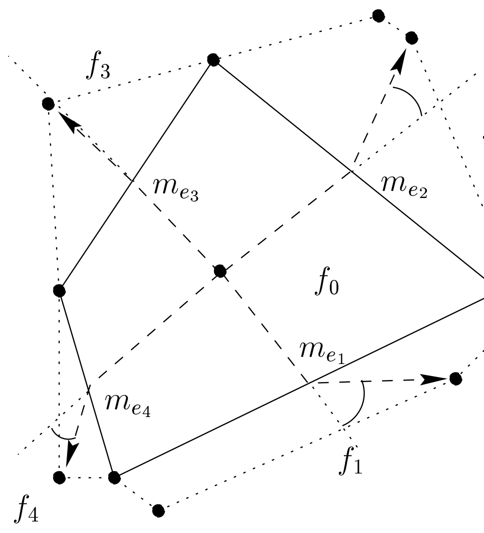


Figure 50: Facet 0 is done; in the next step OUTWARD-POINTING-UNFOLD-I () will take facet f_3, because the absolute turn angle at m_{e_3} is minimal.

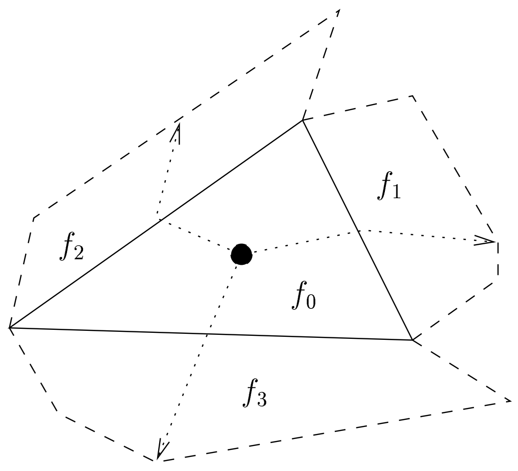


Figure 51: f_3 is the "best" face by the first definition of "outward", f_2 by the second definition.

Rule 15 OUTWARD-POINTING-UNFOLD-I (P)

```
choose a start-facet $f$
$N$ =LAYOUT ($f$)
compute the barycenter $b$ of $f$
while $\partial^+(N) \neq \emptyset$ do
  for all edges $e = (p_1, p_2) \in \partial(N)$ do
    let $m_e = \frac{1}{2}(p_1 + p_2)$
    compute $d_e = \text{opposite}(f, e) - m_e$
  let $e$ be an edge such that the absolute turn angle
        $\left| \arccos \frac{\langle m_e - b, d_e \rangle}{||m_e - b|| \, ||d_e||} \right|$   is minimal
  $N = N \cup$ LAYOUT ($e^+, e^-$)
return $N$
```

is as small as possible.

We omit the pseudo-code for OUTWARD-POINTING-UNFOLD-II here, since it is essentially the same as OUTWARD-POINTING-UNFOLD-I, except for the line where the "outward-pointingness" is computed.

As yet, in the preceding two unfolding rules, we left open how the first facet is chosen. That, however, does not influence the overall performance of the two rules, so we resorted to taking a randomly chosen facet of P as the start facet.

Overlaps: In OUTWARD-POINTING-UNFOLD-I, 42.5% of the unfoldings overlapped; for the second version, overlaps occurred in 36.7% of the test polytopes.

Running time: Again we can use a priority queue to implement $\partial(N)$ and yield a running time of $\mathcal{O}(|E(P)| \log |E(P)|)$.

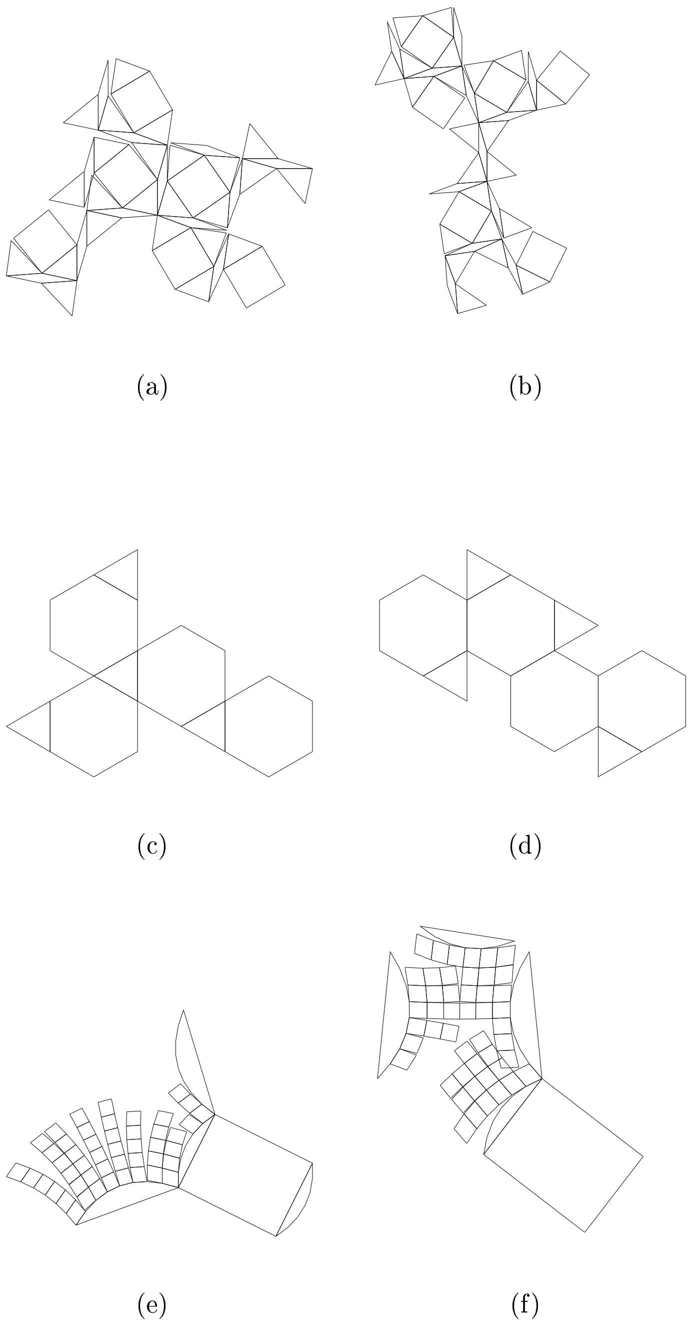


Figure 52: Unfoldings obtained by the first ((a), (c)& (e)) and second ((b), (d) & (f)) version of OUTWARD-POINTING-UNFOLD.

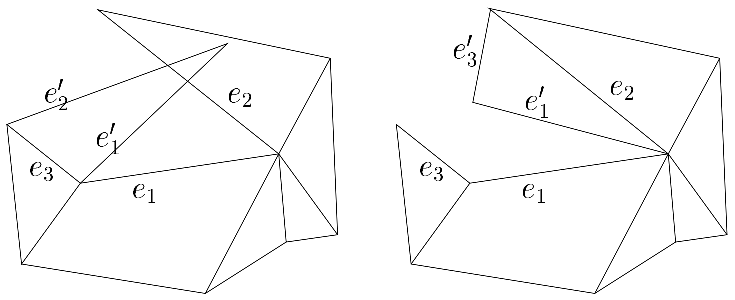


Figure 53: A partial net with an overlap, and a simple modification to "repair" it.

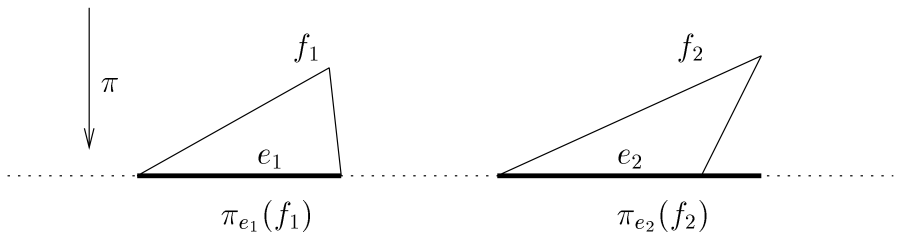


Figure 54: Projection of a facet to the affine hull of one of its edges.

4.9.4 Avoid "hanging facets"

A frequently occurring kind of overlap is a *local* overlap: Two facets overlap which are adjacent on P. Although this can be repaired by the operation shown in Figure 53, the aim of the following rule is to avoid this kind of overlap altogether.

One observation is that local overlaps are often caused by *hanging facets.* If we join a new facet f to an unfolding along an edge e, we say the facet *hangs over* e, if the projection $\pi_e(f)$ of f onto aff(e) is strictly longer than e, i.e.

$$||\pi_e(f)|| > ||e||,$$

or, equivalently, if the angle between e and at least one of the edges incident to e is larger than $\pi/2$ (see Figure 54).

The unfolding rule then works as follows: Starting with a first facet f_0 with barycenter b, in each step,

- if there is at least one edge $e \in \partial(N)$, such that e^+ does not hang over e: let $\tilde{e}$ be the edge where

  $$||b - m_{\tilde{e}}|| \quad \text{is as small as possible.}$$

  (Here m_e is the midpoint of an edge e.) Join $\tilde{e}^+$ along e.

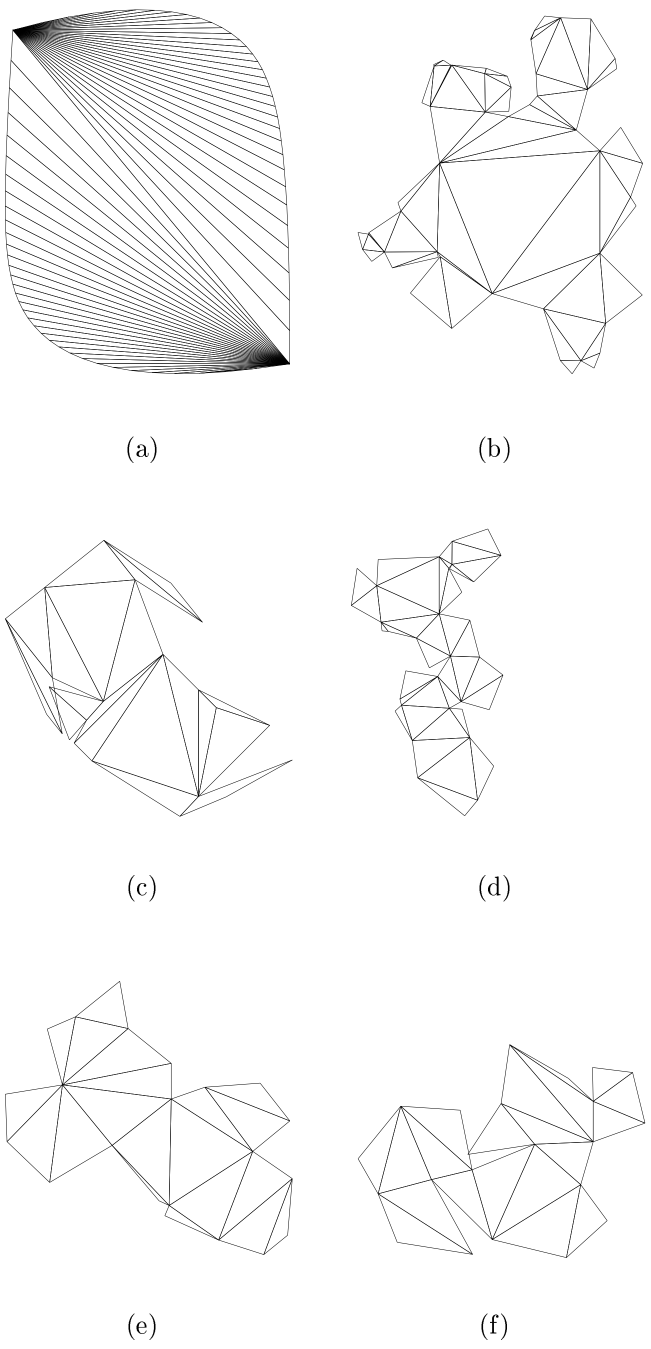
- otherwise, join the next facet along an edge e where

  $$||\pi_e(e+)|| - ||e|| \quad \text{is as small as possible.}$$

Rule 16 AVOID-HANGING-FACETS (P)

```
INITIALIZE   $\ell_{\max} = 0$
             $\phi_{\min} = \infty$
*choose a facet f*
$N$ =LAYOUT ($f$)
**while** $\partial^+(N) \neq \emptyset$ **do**
  **for all edges** $e \in \partial(N)$ **do**
    $f = e^+$
    *let* succ($e$) *and* pred($e$) *be the succeeding and preceding edge*
      *of e along f, respectively.*
    $\phi_e = \max(|\sphericalangle(e_-, e)|, |\sphericalangle(e, e+)|)$.
    **if** $\phi_e < \phi_{\min}$ **then**
      $\phi_{\min} = \phi_e$
      $e_{\min} = e$
    $\ell_e = ||e||$
    **if** $\ell_e > \ell_{\max}$ **then**
      $\ell_{\max} < \ell_e$
      $e_{\max} = e$
  **if** $\phi_{e_{\max}} < \pi/2$ **then**
    $N = N \cup$LAYOUT ($e^+_{\max}, e^-_{\max}$)
  **else**
    $N = N \cup$LAYOUT ($e^+_{\min}, e^-_{\min}$)
**return** $N$
```

Overlaps: 16.5% of the unfoldings overlapped.


(a) (b)

(c) (d)

(e) (f)

Figure 55: Unfoldings obtained by AVOID-HANGING-FACETS.

Running time: This time we use two concurrent priority queues for $\partial(N)$ — one where the sorting criterion for the edges in $\partial(N)$ is the length ℓ_e, and one for which the sorting criterion is ϕ_e, and we can always compute these two values before inserting an edge into $\partial(N)$. This, again, leads to a total running time of $\mathcal{O}(|E(P)| \log |E(P)|)$ if we implement the for-loop of Rule 16 using the priority queues.

4.10 Last resorts

One class of methods which we left out until now are the "last resort" methods: Provided that a given polytope *has* a net, there are a number of methods that will definitely find it. As good as this may sound, they all have a major drawback in common: Their application will most likely (or even most definitely, like the enumeration approach in the next section) spend inordinate computation time even for small polytopes with, say, 50 vertices.

4.10.1 Spanning tree enumeration

A possible way to find out whether a specific polytope has a net is to enumerate *all* spanning trees of its graph, compute $N = \textsc{Cut}(T)$ for each tree and check whether the resulting unfolding is a net.

It is possible to enumerate all spanning trees of a graph $G = (V, E)$ in time $\mathcal{O}(|V| + |E| + |E|N)$, where N is the number of spanning trees; a spanning tree enumeration algorithm is presented in [18]. The number N of spanning trees, however, can be exponential in the size of the graph, which makes this method even more impractical:

Theorem 4.2 (Read & Tarjan, [33])
A connected graph $G = (V, E)$ has at least 2^t spanning trees, where

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

Evaluating this lower bound for even small polytopes with, say, 100 vertices, leads to expected computation times of several years on the computers we used. Therefore, we didn't pursue this approach further.

4.10.2 Backtrack unfolding

The following rule is a "brute force" strategy which also will produce a net for a given polytope, *if the polytope has a net at all.* The basic idea for the algorithm (as it is described in [13]) is *backtracking*:

1. begin by choosing a face f; $N = \textsc{Layout}\,(f)$
2. repeat:

   (a) choose a face f with neighbour g in $\partial^+(N)$

   (b) check whether $p = \textsc{Layout}\,(g, f)$ overlaps any other facet in N

   (c1) if there are no overlaps: $N = N \cup \{p\}$; goto (2 a)

   (c2) if there is an overlap: check whether f has other neighbours g' in $\partial^+(N)$ and go back to (2 b), using g' in the role of g;

   (d) if all neighbouring facets of f have been considered, go back to (2 a) and try to find an alternative choice for f;

   (e) if all facets in $\partial^+(N)$ have been tried without success, announce failure.

A more pseudo-code like, recursive implementation of this algorithm is shown in Rule 17. Here, unlike in the other rules, (N) is a *reference parameter*, such that in every level of recursion, $\textsc{Backtrack-Unfold}\,(P, (N))$ operates on the same instance of N.

This approach actually works, at least with small input polytopes, i.e. polytopes with few facets. For larger polytopes BACKTRACK-UNFOLD will spend inordinate computation time in many cases. Figure 56 below shows the computation time needed[4] to generate a net for a subset of our test polytopes. The major point this diagram illustrates is: In practice BACKTRACK-UNFOLD (as presented in rule 17) *either* finds a net within a few seconds, *or* it takes several hours (or even days) to produce a net (see Figure 56).

There are two points in this rule that considerably influence the running time: One is the order in which the facets are chosen. Another question is, if a facet f has been chosen as the next facet, and f is adjacent to two or more facets in

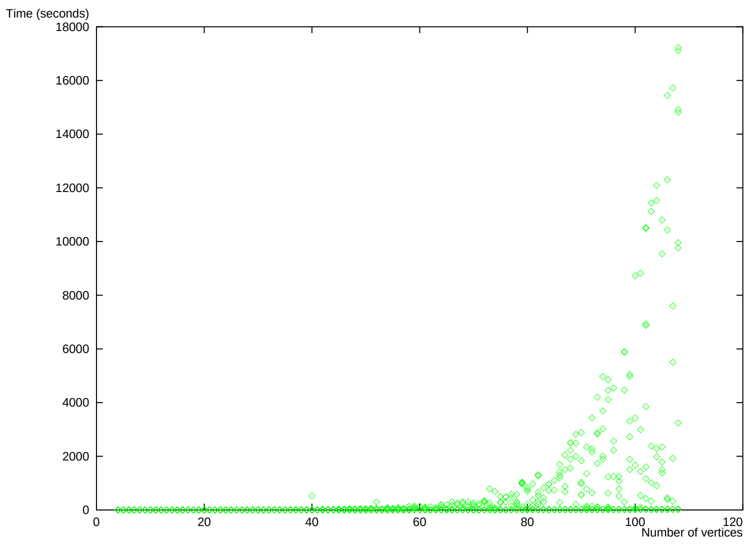
[4]All measurements were made on a Sun 4/20 workstation.

Rule 17 BACKTRACK-UNFOLD $(P, (N))$

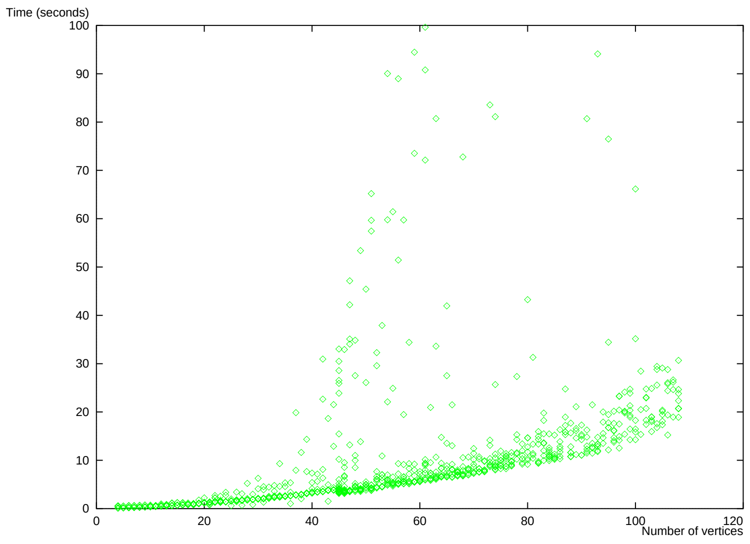
```
if $N = \emptyset$ then
  choose a facet $f$
  $N =$ LAYOUT $(f)$
  return BACKTRACK-UNFOLD $(P, N)$
else
  if $\partial^+(N) = \emptyset$ then
    return true
  else
    forall $f \in \partial^-(N)$ do                                  (*)
      forall $g \in \partial^+(N)$ that share an edge with $f$          (**)
        $p_f =$ LAYOUT $(f, g)$
        if $p$ does not intersect with $N$
          $N = N \cup \{p_f\}$
          if BACKTRACK-UNFOLD $(P, N) =$ true
            return true
return false
```

$\partial^-(N)$, and all possible placements of f would be feasible, which facet should be chosen as the neighbouring facet in the net? To make BACKTRACK-UNFOLD fast, appropriate "tuning" *has* to be done — otherwise this method will perform badly, as the graphs in Figure 56 show.

One application where BACKTRACK-UNFOLD was useful was in *repairing* would-be nets: We implemented a slight modification of the rule, such that given a join tree T, the pairs of facets chosen in steps (*) and (**) of Rule 17 correspond to edges in T. That is, given some join tree T of P which leads to few overlaps, P is unfolded according to T as far as possible; then the remaining facets are joined to the unfolding using normal backtracking. This approach worked well (i.e. fast) for polytopes with up to 150 vertices; beyond that, computation time again increased considerably.



(a) Computation time needed by BACKTRACK-UNFOLD for 1000 random spherical polytopes with 4–108 vertices.



(b) Computation time shown only if it is less than 100 seconds.

Figure 56

4.11 Statistics

We have summarized the performance data for the unfolding rules in Table 1 on page 93.

Above all else, Steepest-Edge-Unfold — one of the simplest rules we presented — produces the least overlapping unfoldings. The second and third row of Table 1 show the performance of two variations of Steepest-Edge-Unfold that differ in the choice of the cut direction: Steepest-Edge-Diam computes two diametral vertices v and w of P, and uses $w - v$ as the cut direction; Steepest-Edge-Rnd-Vertex selects two *random* vertices v and w, and also uses $w - v$ as the cut direction. We implemented several more variations, and the result remained the same: For every deterministic or non-deterministic choice for the cut direction we came up with, 1–2% of the resulting unfoldings overlapped.

Contrary to our expectation that the *flat* polytopes will turn out to be the toughest input for many unfolding rules, in fact the *turtle* polytopes lead to the most would-be nets.

Even Steepest-Edge-Unfold produces overlapping unfoldings for almost half of them. It is easy to see why the turtles have many overlapping unfoldings. Consider the sketch of a (part of a) turtle surface in Figure 57(a). Figure 57(b) shows how to cut the boundary in order to obtain an overlap. In fact, this kind of cut is possible at many vertices of a turtle polytope.

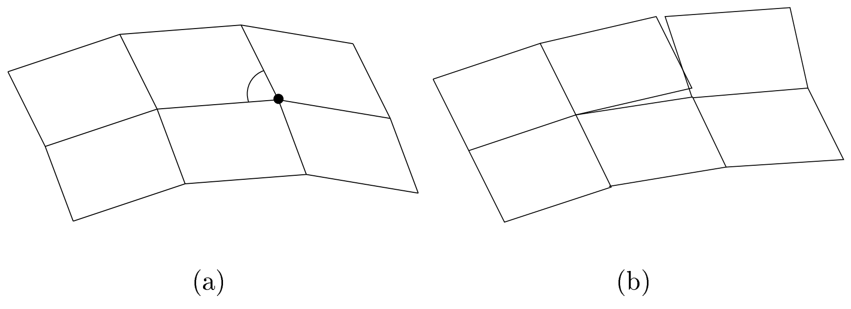


Figure 57: How to construct an overlap in a turtle unfolding.

A further (rather subjective) observation one can make in Table 1 is that the "less complicated" an unfolding rule is, the less overlaps it produces.

4.12 Finding an "unfolding rule cover"

After implementing and testing all unfolding rules described up to this point, not having found a *single* rule that sufficed to unfold *all* polytopes, we looked for a minimal *set* of unfolding rules, such that given a polytope, at least one of the rules produces a net for it.

For this purpose, let $P_1, \ldots, P_{9703}$ be the test polytopes, and let the unfolding rules be $R_1, \ldots, R_{34}$. Define

$$u_{i,j} = \left\{ \begin{array}{ll} 1 & \text{if } R_j \text{ produced a net for } P_i, \\ 0 & \text{otherwise.} \end{array} \right\}, \quad 1 \le i \le 9703, q \le j \le 34,$$

and

$$U = \begin{pmatrix} u_{1,1} & \cdots & u_{1,9703} \\ \vdots & & \vdots \\ u_{34,1} & \cdots & u_{9703,34} \end{pmatrix}$$

Finding such a minimal set of unfolding rules, such that at least one of them unfolds every given polytope from our test set, amounts to finding a minimal number of columns of U, such that in each row of this sub-matrix there is at least one 1. This can be done by solving the *integer program*

$$\text{minimize} \sum_{i=1}^{34} x_i \quad \text{such that} \quad Ux \ge \mathbb{1}, \quad x_i \in \{0, 1\}.$$

Although finding a solution for the integer program above is known to be $\mathcal{NP}$-hard in general [20], `cplex` — a linear and integer program solver [8] — found a solution within a few seconds. It turned out that for our test polytopes there are several *triples* of unfolding rules such that at least one of them produces a net. In Table 2, each column describes one such triple. Two observations can be made here: First, STEEPEST-EDGE-UNFOLD is in every such triple. Second, in every unfolding rule triple except for the first one, there is a variation of SHORTEST-PATHS-UNFOLD involved.

Unfolding rule	*Special*	*Prisms*	*Cyclic*	*Spherical*	*Cubical*	*Flat*	*Turtle*	*Half-Spherical*	*Truncated Tetrahedra*	∅
Steepest-Edge	—	—	—	0.6	—	—	**45.0**	—	1.5	1.2
Steepest-Edge-Diam	—	—	—	0.6	0.7	0.9	**16.6**	—	5.2	1.2
Steepest-Edge-Rnd-Vertex	—	—	—	0.3	0.7	1.5	**33.3**	0.5	5.7	1.5
Long-Shortest-Path	—	—	—	0.6	0.6	2.7	**56.2**	2.2	6.2	2.3
Random-Shortest-Path	—	—	—	0.5	0.7	5.4	**64.5**	—	10.4	3.1
Shortest-Join-Maxdegree	—	—	—	0.4	0.3	8.4	**54.1**	—	8.3	3.1
Shortest-Join-Mindegree	—	—	—	0.4	1.0	7.5	**58.3**	1.1	10.4	3.4
Short-Shortest-Path	—	—	—	0.6	1.0	12.3	**33.3**	—	8.3	3.6
Greatest-Increase	—	—	—	10.6	2.2	9.8	**58.3**	13.9	2.6	8.6
Long-Shortest-Join	—	—	**45.6**	10.0	6.1	18.4	—	13.9	11.4	11.0
Rightmost-Edge	4.0	—	—	15.4	5.7	16.2	4.1	**22.9**	11.9	12.8
Normal-Order-Long	—	—	—	20.5	1.9	13.2	**22.9**	20.6	3.6	13.1
Min-Perimeter	4.0	—	—	20.5	1.9	13.2	**31.2**	20.6	5.2	13.4
Shortest-Path-Mindegree	—	—	17.3	11.5	8.8	**31.2**	—	21.2	16.6	14.7
Shortest-Path-Maxdegree	—	—	17.3	11.5	9.4	**37.4**	—	21.2	15.1	15.7
Breadth-First	—	—	—	17.1	7.5	26.7	**52.0**	21.7	12.5	16.4
Avoid-Hanging	4.0	—	—	17.5	8.8	26.9	29.1	**33.5**	4.6	16.5
Left-First-BFS	—	—	—	17.9	8.1	25.3	**64.5**	22.9	13.0	16.9
Short-Shortest-Join	—	—	2.1	12.1	10.9	**49.7**	—	26.8	27.0	19.2
Shelling-Min	—	—	41.3	20.1	13.9	45.6	—	**49.1**	28.6	24.9
Normal-Order-Max	—	—	39.1	30.3	14.4	46.8	**50.0**	23.4	18.7	27.9
Normal-Order-Min	—	—	39.1	30.5	14.4	47.0	**50.0**	23.4	18.2	28.0
Normal-Order-Flat	—	—	30.4	33.4	14.5	36.0	14.5	**58.6**	19.7	29.1
Shelling-Flat	4.0	—	29.1	30.5	17.7	49.7	2.0	**65.9**	32.2	31.0
Shelling-Long	4.0	—	4.3	44.8	12.0	29.9	27.0	**70.3**	23.4	32.1
Nearest-Facet	8.0	—	—	47.5	18.7	27.8	50.0	**66.4**	19.2	32.2
Outward-II	4.0	7.1	30.4	49.3	19.8	38.5	68.7	**75.9**	16.1	36.7
Highest-Node-Far	8.0	—	8.6	54.4	25.1	45.4	35.4	**77.0**	19.7	40.2
Outward-I	8.0	—	6.5	59.8	25.1	43.1	64.5	**76.5**	28.1	42.5
Shelling-Diff	4.0	—	41.3	61.1	18.9	38.8	27.0	**77.0**	25.1	44.2
Shelling-Max	4.0	—	26.0	67.1	18.5	39.4	52.0	**83.7**	20.8	47.1
Depth-First	12.0	—	82.6	60.4	30.3	57.3	70.8	**81.5**	27.6	49.2
Flat-Tree	16.0	21.4	—	66.2	29.6	46.1	**85.4**	70.9	40.1	51.3
Left-First-DFS	12.0	—	91.3	78.3	28.1	46.5	**83.3**	79.8	31.2	57.3
∅	2.6	0.9	16.1	23.7	10.5	25.8	**37.5**	34.3	15.5	20.7

Table 1: Ranking of the unfolding rules w.r.t. the rate of overlapping unfoldings they produce (in %). The rightmost column and bottom row show the overall rate of overlapping unfoldings per rule and per polytope kind, respectively.

Unfolding rule	Unfolding “cover”							
Steepest-Edge-Unfold	●	●	●	●	●	●	●	●
Greatest-Increase-Unfold	●	●						
Shelling-Min-Unfold	●		●	●				
Min-Perimeter-Unfold					●			
Shelling-Flat-Unfold								
Shortest-Path-Min-Unfold						●	●	
Shortest-Path-Max-Unfold								●
Shortest-Path-Join-Min-Unfold			●		●	●		●
Shortest-Path-Join-Max-Unfold								
Short-Shortest-Path-Unfold				●			●	
Long-Shortest-Path-Unfold		●						

Table 2: Each column describes a triple of unfolding rules, such that for each test polytope there is at least one rule that unfolds it.

4.13 Steepest edge unfolding revisited

As far as the overall performance is concerned, a surprising fact about Steepest-Edge-Unfold (P, c) is that it does not seem to matter how c is chosen, as long as it is in general position. In particular, selecting a random vector on the unit sphere leads to similar results. Furthermore, if W_c is the set of would-be nets produced by Steepest-Edge-Unfold (P, c), for different random c and c', the size of the intersection $|W_c \cap W_c'|$ was always less than 40, and very often even empty.

This led us to a rule Steepest-Edge-Loop (P) that repeatedly calls Steepest-Edge-Unfold (P, c) with a random vector of length 1, until the unfolding is a net (see Rule 18. This rule produced a net for all test polytopes, and, in the sequel, for more than 50 000 randomly generated spherical and cubical polytopes with $4 \ldots 1000$ vertices:

Although especially the random choice of c does not seem a promising approach to finding a proof that this rule even terminates eventually, we conjecture that it does, and have convincing empirical evidence supporting this conjecture: The main loop in Steepest-Edge-Loop (P) was executed

Rule 18 STEEPEST-EDGE-LOOP (P)

```
do
    let $c = (c_1, c_2, c_3)$ be uniformly distributed on the unit sphere.

    Ensure that c is in general position
    $N$ =STEEPEST-EDGE-UNFOLD $(P, c)$
until $N$ is a net
return $N$
```

once	in $\approx 98.8\%$ of the runs
twice	in $\approx 0.79\%$ of the runs,
three times	in $\approx 0.39\%$ of the runs,
four times	in $\approx 0.19\%$ of the runs
five times	in $\approx 0.015\%$ of the runs,
six times	in $\approx 0.0025\%$ of the runs (i.e. 124 times out of 50 000), and
seven times:	never (!)

Thus, the above observation gave rise to the following conjecture:

Conjecture:
For every 3-dimensional polytope P there exists a vector $c \in I\!R^3$, such that STEEPEST-EDGE-UNFOLD (P, c) produces a net of P.

4.14 Shortest paths unfolding revisited

Since the variations of SHORTEST-PATHS-UNFOLD also produced few overlaps, we also implemented a SHORTEST-PATHS-LOOP rule, that considers all vertices of the polytope in random order, computes the corresponding shortest-paths tree, and unfolds it according to this tree — until the unfolding is a net.

Rule 19 SHORTEST-PATHS-LOOP (P)

```
for all vertices $v \in V(P)$ do
  $N$ =SHORTEST-PATHS-UNFOLD $(P, v)$
  if $N$ is a net
    return $N$
return $\emptyset$     // return $\emptyset$ to indicate failure
```

Using this rule (see the pseudo-code in Rule 19), we we obtained nets for all our test polytopes, too. It is, however, much more expensive (in terms of computation time) than STEEPEST-EDGE-LOOP: The turtle polytope $T(6, 6)$ turned out to be the toughest input here — 163 out of 169 shortest-paths trees lead to overlaps (see Figure 58).

We conjecture that there are polytopes for which no shortest paths tree leads to a net.

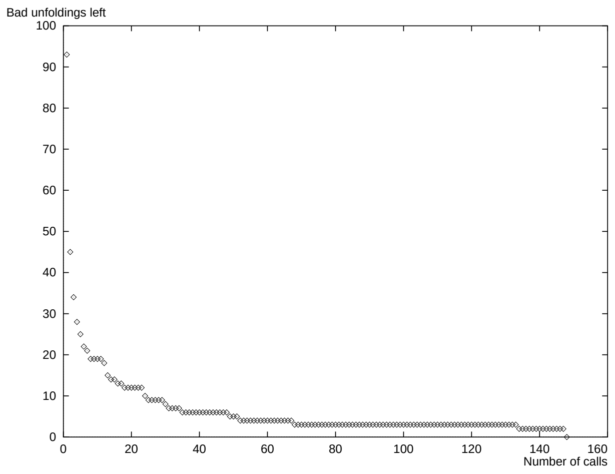


Figure 58: Number of calls to SHORTEST-PATHS-UNFOLD (P, v) needed (each call with a different choice of v) to unfold all 9703 test polytopes; the diagram shows only those for which the first unfolding did not succeed.

5 Conclusion

The open questions and conjectures mentioned in Section 3 (except for the shortest-paths tree conjecture) are still open. We still believe the answer to the main question to be positive, namely, that every 3-polytope has a net.

We found one unfolding strategy (Steepest-Edge-Loop) that always succeeded for 60 000 polytopes. This of course does not prove that it *always* works, but leads us to the conjecture that for each polytope P there exists a cut direction c such that Steepest-Edge-Unfold (see Section 4.7.1) produces a net for P.

The problem turned out to be harder than suspected: we tried numerous other unfolding strategies and all of them failed for at least a few polytopes.

References

[1] A.D. Aleksandrov: *Konvexe Polyeder.* Akademie-Verlag, Berlin, 1958.

[2] B. Aronov & J. O'Rourke: *Nonoverlap of the star unfolding.* Discrete Comput. Geom., 8:219–250, 1992.

[3] T.F. Banchoff: *Dimensionen: Figuren und Körper in geometrischen Räumen.* Spektrum Akademischer Verlag, Heidelberg, 1991. (Engl.: *Beyond the Third Dimension.* Scientific American Library, 1990).

[4] D.W. Barnette: *Trees in polyhedral graphs.* Canad. J. Math., 18, pp. 731–736, 1966.

[5] S. Bouzette, F. Buekenhout, D. Edmond & A. Gottcheiner: *A theory of nets for polyhedra and polytopes related to incidence geometries.* Université Libre de Bruxelles, Bruxelles, 1985.

[6] H. Bruggesser & P. Mani: *Shellable decompositions of cells and spheres.* Math. Scand. 29, 1971.

[7] T.H. Cormen, C.E. Leiserson, R.L. Rivest: *Introduction to Algorithms.* McGraw-Hill, New York, 1990.

[8] Cplex 4.08, Linear & Mixed Integer Program Solver. CPLEX Optimization Inc., 1989.

[9] H.T. Croft, K.J. Falconer, R.K. Guy: *Unsolved Problems in Geometry.* Springer-Verlag, New York, 1991.

[10] H.M. Cundy, A.P. Rollett: *Mathematical Models*, 2nd edition. Clarendon, Oxford, 1961.

[11] A. Dürer: *Unterweysung der Messung mit dem Zyrkel und Rychtscheyd.* Nürnberg, 1525. (English translation with commentary by W.L. Strauss, *The Painter's Manual*, New York, 1977.)

[12] E. Gamma, R. Heml, R. Johnson, J. Vlissides. *Design Patterns — Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional Computing Series, 1995.

[13] M. Eisenberg: *The thin glass line: Designing interfaces to algorithms.* CHI 96 Conference Proceedings, 1996.

[14] M. Eisenberg & A. Nishioka: *Creating polyhedral models by computer.* To appear in: Journal of Computers of Mathematics and Science Teaching, 1997.

[15] D. Epstein, S. Levy, R. de la Llave: *Experimental Mathematics; Statement of Philosophy and Publishability Criteria*; see `http://www.expmath.com/expmath/philosophy.html`.

[16] K. Fukuda & G.M. Ziegler, personal communication, 1996/97.

[17] D. Avis & K. Fukuda: *Reverse search for enumeration.* Tech. Report No. 92-5, Graduate School of Systems Management, University of Tsukuba, Tokyo, Japan, 1992. (To appear in Discrete Applied Mathematics, ps file available from `ifor13.ethz.ch` (129.132.154.13), directory `/pub/fukuda/cdd`.

[18] G.N. Gabow & E.W. Myers: *Finding all spanning trees of directed and undirected graphs.* SIAM J. Computation 7, 1978; pp. 280–287.

[19] E. Gawrilow, M. Joswig, G.M. Ziegler, et al.: *POLYMAKE — a framework for analyzing convex polytopes,* TU Berlin 1996;
Information: `{gawrilow,joswig,ziegler}@math.tu-berlin.de`.

[20] M.R. Garey & D.S. Johnson: *Computers and Intractability: a Guide to the Theory of NP-Completeness.* W.H. Freeman, San Francisco, 1979.

[21] Geomview, Version 1.6: The Geometry Center, University of Minnesota, 1993; Information: `software@geom.umn.edu`.

[22] B. Grünbaum: *Convex Polytopes.* Pure and applied Mathematics, J. Wiley and Sons, 1967.

[23] B. Grünbaum: *Nets of polyhedra.* Geombinatorics 1, No. 2, 5–9, 1991.

[24] B. Grünbaum: *Nets of polyhedra II.* Geombinatorics 1, No. 3, 5–10, 1991.

[25] M. Jeger: *Über die Anzahl der inkongruenten Netze des Würfels und des regulären Oktaeders.* Elemente der Mathematik, Bd. 30:4, pp. 73–96, 1975.

[26] L. Lovász: *An Algorithmic Theory of Numbers, Graphs and Convexity.* CMBS-NSF Regional Conference Series in Applied Mathematics 50, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1986.

[27] A. Lubiw, J. O'Rourke: *When can a polygon fold to a polytope?* Smith College, Northampton, USA, 1996.

[28] L.A. Lyusternik: *Convex figures and Polyhedra.* D.C. Heath, Boston, MA, 1966.

[29] P. McMullen & G.C. Shephard: *Convex Polytopes and the Upper Bound Conjecture.* Cambridge University Press, Cambridge, England, 1971.

[30] K. Mehlhorn & S. Näher: *LEDA: a library of efficient data structures and algorithms.* Communications of the ACM, 38:96–102, 1995.

[31] M. Namiki and K. Fukuda: *UnfoldPolytope: A package for* MATHEMATICA *1.2 or 2.0.* Mathematica Notebook, Univ. of Tokyo, 1993; cf. `http://www.ifor.math.ethz.ch/staff/fukuda/fukuda.html`

[32] F.P. Preparata, M.I. Shamos: *Computational Geometry, An Introduction.* Texts and Monographs in Computer Science, Springer-Verlag, New-York, 1985.

[33] R.C. Read & R.E. Tarjan: *Bounds on backtrack algorithms for listing cycles, paths, and spanning trees.* Networks 5, 1975; pp. 237–252.

[34] H.C. Reggini: *Regular polyhedra: random generation, Hamiltonian paths and single chain nets.* Monografias de la Academia Nacional de Ciencias Exactas, Fiscas y Naturales; No. 6. Buenos Aires, 1991.

[35] C. Schevon: *Algorithms for Geodesics on Polytopes.* Ph.D. Thesis, John Hopkins University, 1989.

[36] R. Sedgewick: *Algorithms in C++*, Addison Wesley Publishing COmpany, Inc., 1992.

[37] M. Senechal, G. Fleck: *Shaping Space, A Polyhedral Approach.* Design Science Collection, Birkhäuser, 1988.

[38] G.C. Shephard: *Convex polytopes with convex nets.* Math. Proc. Cambridge Philological Society, Vol. 78, pp. 389–403, 1975.

[39] E. Steinitz: *Über isoperimetrische Probleme bei konvexen Polyedern.* J. angewandte Mathematik, 159, 1928.

[40] L. Wall & R.L. Schwartz: *Programming Perl.* O'Reilly & Associates Inc., 1990.

[41] M.J. Wenninger: *Polyhedral Models.* Cambridge University Press, 1970.

[42] G.M. Ziegler: *Lectures on Polytopes.* Graduate Texts in Mathematics, Springer-Verlag, New York.

List of unfolding rules