Edge-Unfolding Almost-Flat Convex Polyhedral Terrains

by

Yanping Chen

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2013

Author .....
Department of Electrical Engineering and Computer Science
May 18, 2013

Certified by .....
Erik Demaine
Professor
Thesis Supervisor

Accepted by .....
Prof. Dennis M. Freeman
Chairman, Masters of Engineering Thesis Committee

Edge-Unfolding Almost-Flat Convex Polyhedral Terrains

by

Yanping Chen

Submitted to the Department of Electrical Engineering and Computer Science
on May 18, 2013, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

In this thesis we consider the centuries-old question of edge-unfolding convex polyhedra, focusing specifically on edge-unfoldability of convex polyhedral terrain which are "almost flat" in that they have very small height. We demonstrate how to determine whether cut-trees of such almost-flat terrains unfold and prove that, in this context, any partial cut-tree which unfolds without overlap and "opens" at a root edge can be locally extended by a neighboring edge of this root edge. We show that, for certain (but not all) planar graphs G, there are cut-trees which unfold for all almost-flat terrains whose planar projection is G. We also demonstrate a non-cut-tree-based method of unfolding which relies on "slice" operations to build an unfolding of a complicated terrain from a known unfolding of a simpler terrain. Finally, we describe several heuristics for generating cut-forests and provide some computational results of such heuristics on unfolding almost-flat convex polyhedral terrains.

Thesis Supervisor: Erik Demaine
Title: Professor

Acknowledgments

I would like to acknowledge my thesis advisor Prof. Erik Demaine, not only for the suggestion of this interesting yet tangible corner of edge-unfolding, but also for his continual guidance and enthusiasm throughout my research. I would also like to thank my parents for their unending support and care throughout my time here at MIT and my work on this thesis.

Contents

List of Figures

Chapter 1

Introduction

In this thesis, we pursue the age-old question of edge-unfolding convex polyhedra by examining a specific subset of such polyhedra. Specifically, we consider almost-flat convex polyhedral terrain, which can be informally considered as convex liftings of planar graphs with a very small height. First, in this chapter, we review the history of the edge-unfolding problem, as well as give a brief overview of our results and the organization of this thesis.

1.1 History and Background of Edge-Unfolding

For centuries, mathematicians and artists alike have studied and depicted polyhedra in manuscripts. An early example of this was *Underweysung der Messung* [8] (German for "The Painter's Manual"), a book by Albrecht Dürer about the technique of perspective drawing. Throughout the book are many pictures of polyhedral nets, i.e. pictures of unfolded polyhedra where the faces of the polyhedra are laid out in the plane and connected at the appropriate edges. What is interesting is that every such polyhedral net Dürer drew was not only a single connected piece, but also non-intersecting or non-overlapping. While we will cover these concepts more formally in Chapter 3, informally, this process of "edge-unfolding" can be thought of as taking a pair of scissors and cutting along the edges of a polyhedron in such a way as to yield a single connected component of faces which is then flattened in the plane. If this

flattened "net" does not overlap with itself, then we say that the original polyhedron is "edge-unfoldable." Shephard [8] formally conjectured in 1975 that it is possible to edge-unfold in the same manner all convex polyhedra.

Conjecture (Shephard's Conjecture)**.** *All polyhedra are edge-unfoldable.*

This conjecture remains an open problem to this day. Meanwhile, the related problem of whether a general non-convex polyhedron is edge-unfoldable has been solved by Grünbaum [10, 11] and Tarasov [18] in their papers, and perhaps most comprehensively, by Bern et al. in their papers on "ununfoldable polyhedra," where they demonstrated several general non-convex polyhedra [4], including ones with only convex faces [6] or with only triangular faces [5], which cannot be edge-unfolded into non-overlapping nets.

While there have been several studies on the open problem of edge-unfolding convex polyhedra, when given a convex polyhedron or polyhedral surface, still not too much is known about exactly which edge-unfoldings, if any, will yield non-overlapping nets.

One of the first major results in this area come from Schevon's 1989 PhD thesis *Algorithms for Geodesics on Polytopes* [16], where she showed that most unfoldings of convex polyhedra appear to be overlapping by computationally exploring random unfoldings of random convex polyhedra with vertices on the unit sphere. For each value of n between 10 and 80, Schevon generated 5 convex polyhedra of n vertices on the unit sphere. For each such polyhedron, 1000 unfoldings were randomly selected by random generation of glue-trees. Each unfolding was tested for overlap, producing an estimate of the percent of unfoldings which are overlapping. The results of the experiment showed that as n gets larger, the percent of overlapping unfoldings approached 100%. Specifically, almost all of the unfoldings tested of polyhedra of more than 70 vertices were overlapping, implying that a random glue-tree of a large convex polyhedron is almost guaranteed to be overlapping and giving evidence that Shepard's Conjecture might be false.

A more comprehensive study of various algorithms and polyhedra comes from

Schlickenrieder's 1997 master's thesis *Nets of Polyhedra* [17], where he defined and tested several unfolding algorithms against several classes of convex polyhedra which he created. The results were inconclusive for the general problem — every single algorithm had a counterexample convex polyhedron which it could not unfold, while every convex polyhedra generated was successfully edge-unfolded by some algorithm. One of the more promising algorithms was named "STEEPEST-EDGE-UNFOLD," and it selected edges for the cut-tree by picking locally at each vertex the "steepest" edge e which maximized $e \cdot o$ for some objective vector o in $\mathbb{R}^3$. This algorithm unfolded almost all of the polyhedra, and a variation which repeated the algorithm using randomly generated objective vectors until an edge-unfolding was found managed to unfold all of the polyhedra tested after at most 7 objective vectors tested. A promising conjecture of the paper was that this "RANDOMIZED-STEEPEST-EDGE-UNFOLD" could potentially unfold all convex polyhedra.

Unfortunately, this was disproven by Lucier's 2006 article *Local Overlaps in Special Unfoldings of Convex Polyhedra* [14], where he created counterexamples for RANDOMIZED-STEEPEST-EDGE-UNFOLD and another more general algorithm Schlickenrieder conjectured to always produce valid edge-unfoldings. Lucier accomplished this by first constructing a convex polyhedral terrain which had no non-overlapping steepest-edge unfoldings for any objective vector in a cone. This was done by showing that, for every objective vector in the cone, the steepest-edge unfolding would cause a local overlap. Then, embedding the terrain in a triangle, he tiled the faces of a large convex polyhedron with this constructed terrain in such a way that the cones of ununfoldability covered all of space. This guaranteed that any objective vector picked for the algorithm would fall in one such cone and thus fail to edge-unfold that corresponding terrain-embedded face. Lucier [13] also used similar methods to construct an ununfoldable convex polyhedra for normal-order unfoldings, a generalization of steepest-edge unfoldigs proposed by Schlickenrieder.

Another interesting result in the field comes from Benton and O'Rourke's 2007 article *Unfolding Polyhedra via Cut-Tree Truncation* [3], where they presented the technique of "vertex truncation" which takes an unfolding cut-tree T of a convex

polyhedron P and turns it into an unfolding cut-tree T' of a related convex polyhedron P'. In more detail, let P be a convex polyhedron, and let P' be P with a corner cut off. This means that P' has a face where P has a vertex. Benton and O'Rourke showed that if an unfolding cut-tree T of P fulfilled an "empty-sector property," and the newly created face is triangular, then T can be modified to T', an unfolding cut-tree of P', and this new T' also has the empty-sector property. Using this technique, they showed that any convex polyhedron which can be obtained by a series of such operations from an initial convex polyhedron which has a cut-tree with the empty-sector property — for example, a pyramid — is also edge-unfoldable.

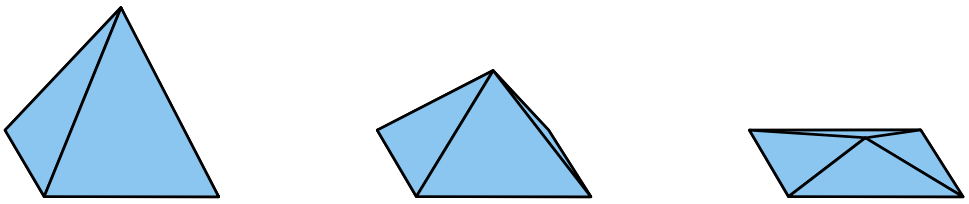


Figure 1-1: Flattening of a triangular pyramid to an almost-flat triangular pyramid

In this thesis we will be focusing on "almost-flat convex polyhedral terrains." As Figure 1-1 shows, an almost-flat convex polyhedra terrain can be thought of as a convex polyhedral surface which is "flattened" by uniformly scaling it down in one direction. The inspiration for studying such constructs came from a chat among Marshall Bern, Erik Demaine, and David Eppstein in 1998 (and continued in discussions with Günter Rote and Greg Price) — the motivation is that since such constructs are almost flat, then they are very close to their planar projections, which are close to non-overlapping nets, and unfolding such constructs will only result in slight shiftings of the faces in their planar projection. Therefore, the only possible overlaps should be between faces which are adjacent in the planar projection, i.e. local overlaps. This will hopefully make it easier to analyze which cut-forests unfold, since we can then consider each cut-tree separately because we only need to worry about local overlaps. We will formally define such constructs in Chapter 2, as well as explore various methods of representing them.

1.2 Our Results

As mentioned, we start in Chapter 2 by defining rigorously various terms we have been using such as polygons, polyhedra, terrains, and almost-flatness. While it is trivial to see any convex terrain as a convex lifting of a planar graph, we show in Section 2.5 an alternative representation based on the angle differences between the faces of the terrain and the faces of the planar projection of the terrain. Such a representation is useful since it tells us how much various angles open when we unfold based on a cut-tree, and we use it throughout the rest of the thesis in our analysis of cut-trees. We also demonstrate how to convert between the convex lifting representation and the angle-delta representation and how to bound the maximum angle-delta.

After some introductory definitions of cut-forests and edge-unfolding, in Section 3.1 we consider the issue of local overlaps. A local overlap is an overlap between neighboring faces, which is a special case of a general overlap between any two faces. We show that, for any almost-flat convex terrain T, there is a positive finite bound BoundLocal(T) such that if T has height at most BoundLocal(T), then for any cut-forest F, any overlaps from the unfolding of T by F will be local overlaps. This proves our intuition that if the convex terrain is flat enough, then it is only possible to have local overlaps on unfolding if we have any overlaps at all. We also show that there are almost-flat convex terrains T for which no matter how flat T is, as long as it has non-zero height, there are cut-forests of T which will always result in overlaps. This shows that overlaps are still possible, and almost-flatness does not necessarily trivialize edge-unfolding by any means.

Next, in Section 3.2, we consider just the planar projections of almost-flat convex terrains. We define several path types, including most notably the Strongly Monotonically Increasing Distance (SMID) Paths, which are based on just the planar projection and not on the associated convex lifting or angle-delta additions. We prove that, for any planar graph G, there is a positive finite bound BoundSMID(G) such that, for any convex lifting T of G, if T has height less than BoundSMID(G), then any SMID paths of G will be a cut-path of T which unfolds without overlap. By

putting together multiple SMID paths to form a tree, we show that such SMID trees share a similar property — any SMID trees of G will unfold without overlap for any almost-flat convex lifting of G. Hence, we can find an unfolding of an almost-flat convex terrain by finding a SMID cut-forest of its planar projection. Unfortunately, we also note that it is not always possible to find a SMID forest for all planar graphs, and give several counterexample planar graphs which have no SMID spanning forests. Furthermore, we show that, for any convex polygon p, we can make a planar graph which has no SMID spanning forests with p as its outer boundary. Therefore, by considering projections alone, it is not possible to unfold all such almost-flat convex terrains.

Then, in Section 3.3, we consider the case of unfolding almost-flat convex terrains while taking into account the relative heights of vertices. We give a first-order approximation of determining whether a given cut-tree unfolds without overlap in Section 3.3.2, and then argue in Section 3.3.3 that considering first-order effects alone is appropriate by showing that any second-order or higher effect can be made insignificant by almost-flatness. We then prove in Section 3.3.4 a result of local extensibility of partial cut-trees. This means that if we have a partial cut-tree C which unfolds without overlap, then there is at least one edge neighboring the root of the tree by which we can "locally" extend our C to give a larger partial cut-tree C' which also unfolds without overlap. Note that the "local" part of this result comes from the fact that we assume we can extend C at its root by any neighboring edge, but this is not true in general since it could be that extending C by some neighboring edge of the root of C will result in a loop.

In Section 3.4, we describe a different unfolding technique called "slice unfolding," which is similar to, and can be considered as an extension of the "cut-tree truncation" methods of Benton et al. [3]. Informally, this method takes a convex polyhedron P (or in our case, an almost-flat convex terrain T) for which one knows E, an edge-unfolding of P, and then tries to create an edge-unfolding of P', the convex polyhedron of P with a section "sliced" off, by modifying E in the neighborhood of the sliced off section. We start by demonstrating how slice unfolding works in general, showing what slices

look like, and giving an example sequence of unfoldings created by a sequence of slices. Then, we focus our attention on vertex-slices, a type of slice which cuts away a section which contains exactly one vertex. We show that, in the context of almost-flat convex terrain, all triangular vertex-slices result in new valid unfoldings, which essentially translates the result of "Cut-Tree Truncation" [3] to the space of almost-flat convex terrains. We then continue on to analyze and categorize which unfoldings of quadrilateral vertex-slices and general vertex-slices result in valid unfoldings.

Next, in Chapter 4, we detail some algorithms and heuristics we developed for computationally searching for edge-unfoldings in almost-flat convex terrain. We start in Section 4.1 with algorithms for generating spherical convex liftings in regular m-gons, general spherical liftings, and general random liftings of convex planar graphs. We also show algorithms for finding unfolding cut-paths and determining, based off of our first-order approximation method from Section 3.3.2, if a cut-tree unfolds without overlap. Then we demonstrate an algorithm for non-repeating brute-force enumeration of cut-forests, which is useful for computationally calculating the total number of cut-forests and the total number of unfolding cut-forests. Finally, we devise a greedy algorithm for constructing a cut-forest and propose several heuristics to use it with.

Finally, in Chapter 5, we give some computational results of edge-unfolding almost-flat convex terrain. We reaffirm Schevon's [16] result by showing that, for almost-flat spherical liftings, the percentage of randomly selected cut-forests which unfold without overlap seems to decrease as the terrain size increases. Similarly, our results suggests that, for general spherical liftings, on average, the total number of cut-forests as well as the total number of unfolding cut-forests increase exponentially as terrain size, while the percentage of unfolding cut-trees decreases exponentially as terrain size. Lastly, we tested the heuristics we devised for our greedy cut-forest generation algorithm, showing that a heuristic which relies on Dijkstra distance and threshold angle values works the best — it is able to maintain an 80% success rate even for general spherical liftings of close to 6,000 vertices.

Chapter 2

Almost-Flat Convex Terrains

In this chapter, we introduce the concept of almost-flat convex terrains, the focus of our study in unfolding. We rigorously define such notions as terrains, convexity, and almost-flatness, as well as examine two main methods of representing such constructs.

2.1 Polygons, Polyhedra, and Terrains

We start with a review of definitions and properties of polygons, polyhedra, and terrains. For each such construct, we give a rigorous definition, but assume knowledge of common Euclidean geometric concepts such as angles, edges, faces, etc. Similarly, we often make use of Cartesian coordinates and base some of our definitions on such coordinates.

Let $x_1, x_2, \ldots, x_n$ be coplanar points in $\mathbb{R}^3$ such that none of the straight line segments $x_1x_2, x_2x_3, \ldots, x_{n-1}x_n, x_nx_1$ intersect except at common endpoints. Then, we say that $x_1x_2\cdots x_n$ is an *n-sided polygon* Q with *vertices* $V_Q = \{x_1, \ldots, x_n\}$ and *edges* $E_Q = \{x_1x_2, \ldots, x_{n-1}x_n, x_nx_1\}$ (see Figure 2-1a). Essentially, a polygon is a planar region bounded by straight line segments — let this bounded region be the *interior* of the polygon, which is separated from the rest of the plane (the *exterior*) by the edges of the polygon (the *boundary*). We say that two polygons are *touching* if they share points only along their boundaries, and two polygons are *intersecting* if they share points in their interiors. Note that polygons do not need to be coplanar

to be touching or intersecting.

(a) Polygon (b) Polyhedron

Figure 2-1: Example polygon and polyhedron

Then, a *polyhedron* P is a union of non-intersecting polygons $\{q\}$ which are connected at edges to form a closed surface, and where no two adjacent polygons are coplanar (see Figure 2-1b). Let us define a polyhedron P by the ordered triplet (V, E, F), where $V \subset \mathbb{R}^3$ is the set of vertices of P, $E \subset V \times V$ is the set of edges of P, and $F \subset V^*$ is the set of faces of P. Similarly to before, let the finite space bounded by the faces of a polyhedron be its interior, and the rest of $\mathbb{R}^3$ be the exterior. With this definition of "inside" and "outside", we can define the *normal* to a given face to be the unit vector perpendicular to the face and pointing away from the interior of the polyhedron. Similar to how polygons have angles at vertices, polyhedra have *dihedral angles* at edges; the dihedral angle at edge e is the interior angle between the two faces which share e. It can also be calculated as $\pi - A$, where A is the signed angle between the face normals of the same two neighboring faces.

Next, an *open polyhedron* O (Figure 2-2a) is a connected subset of faces of P homomorphic to a disk — i.e. there are no holes. Finally, a *terrain* T (Figure 2-2b) is an open polyhedron with a unit vector $\vec{Z}$ such that the projection of T to a plane perpendicular to $\vec{Z}$ is a planar graph. Also, let the vertices of T not surrounded by faces be the *boundary vertices* of T. In essence, T is a "patch" of a polyhedron, which can be "flattened" in the direction of $\vec{Z}$ without any overlap or degeneracy of faces. Note that this means any line parallel to $\vec{Z}$ intersects T at at most one point, so there can be no "vertical" faces with respect to $\vec{Z}$. Continuing, we define the "interior" of T to be the set of points $\{x - t\vec{Z} \mid x \in T, t > 0\}$, where $x \in T$ are points on the

surface of T. Informally, this represents the volume "under" T. Using this definition of the interior, the previous condition for projection without degeneracy or overlap can also be stated as the following: for every face f of T, the dot product of the normal of f with $\vec{Z}$ is strictly positive.

(a) Open polyhedron (b) Terrain

Figure 2-2: Example open polygon and terrain

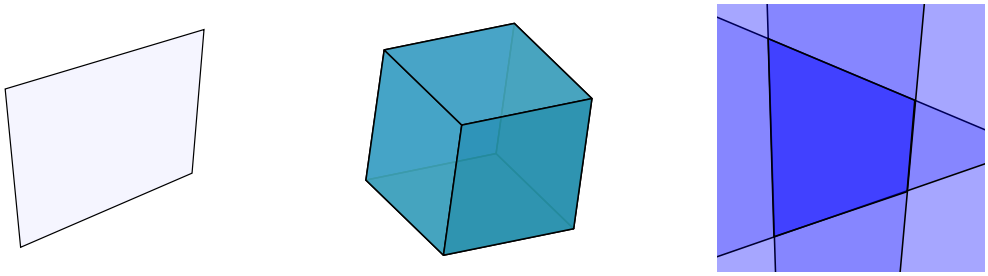
Let us also define a *simple* terrain to be a terrain which contains no non-boundary edge between two boundary vertices. Then let a non-simple terrain be a *complex* terrain. A complex terrain is multiple simple terrains joined at edges.

For convenience, whenever we speak of a terrain T, we will mean a simple terrain with $\vec{Z}$ parallel to the positive z axis, in which case the projection of T to the xy plane will be a planar graph. Similarly, when considering such terrains, we assume that that all z coordinates are non-negative, and that the lowest vertex lies in the $z = 0$ plane.

2.2 Convexity

A polygon or a polyhedron P is *convex* if and only if for every pair of points $a, b \in P$, the line segment $\overline{ab}$ is contained in P as well. Here, $a \in P$ means that a is not in the exterior of P, so a can be either in the interior of P or on the boundary of P. More practically, a polygon is convex if all of its interior angles are less than π, and a polyhedron is convex if all of its dihedral angles are less than π. Also, all the faces of a convex polyhedron are all convex as well: this is easy to see by considering any non-convex face and the faces adjacent to the $> \pi$ angle. For a terrain T, we can use

the same definition for convexity by using the definition of interior as described in the previous section.



(a) Convex polygon (b) Convex polyhedron (c) Convex polygon as half-space intersection

Figure 2-3: Convexity Examples

Another important property of convexity is that we can treat convex shapes as intersection of half-spaces (see Figure 2-3c). For polygons, this means that convex polygons are intersections of half-planes, while for polyhedra, this means that convex polyhedra are intersections of half-spaces. Hence, another way to represent a convex polyhedron would be a list of planar inequalities representing the half-spaces which form the polyhedron. While we won't be using this representation directly, some unfolding techniques we present in Section 3.4 will make use of these ideas.

2.3 Almost-Flatness

We say that a terrain T is *almost flat* with *flatness* ε if the z coordinate of every vertex of T is less than or equal to ε. Note that this definition makes use of the coordinate-based definition of terrains which we mention at the end of Section 2.1. To avoid confusion, we will use the common term of "flatter" to mean a smaller ε, and "less flat" to mean a larger ε. A useful fact to note is that any terrain T can be converted into an almost-flat terrain by merely scaling all the vertices of T in the z axis by an appropriate constant. Also, note that such scaling does not affect the convexity of a terrain.

An almost-flat convex terrain T is, as its name dictates, "almost flat." This means that, for small flatness ε, the faces of T are actually very close in size and shape to

the faces of the xy projection of T. This is the main defining characteristic of the terrains which we will be examining in the rest of this thesis, as it is one which is not well-explored. A big motivation for looking at such T is that, since the faces of T and the corresponding faces of the xy projection of T are very close, an unfolding of T is merely a slight perturbation of the projection of T. These concepts of unfolding will be discussed more in Chapter 3.

2.3.1 Height Bounds on Flatness

In many places we will use the assumption that, since T is almost flat, we can use first-order approximation on functions of quantities involving ε. While we don't explicitly prove this, we assume that we can bound ε for certain functions $f(x)$ such that the difference between $f(a + c\varepsilon)$ and $f(a) + f'(a)c\varepsilon$ for applicable constants a and c is small enough not to matter for our applications. Stated more formally:

Claim (Height Bound for First-Order Application)**.** *For a given almost-flat convex terrain T and a given finite set of strict inequalities involving continuous functions of values involving heights and positions of vertices of T, there exists a flatness value ε such that using the first-order approximation of such functions in regards to ε will not change the result of the inequalities.*

Proof Sketch. For any function f which is continuous in a neighborhood of x, we have a bound on $f(x)$ for x in the neighborhood. By shrinking the neghborhood, and thus shrinking the bound on $f(x)$, we can ensure that whatever strict inequalities which use $f(x)$ will be satisfied despite the inaccuracy brought on by the first-order approximation. □

Where applicable, we will provide arguments for the validity of first-order approximation in our calculations.

2.4 Convex Lifting Representation

Taking advantage of the fact that an almost-flat convex terrain is very close in shape to its projection, we will now detail two methods of representing such a terrain T by its projection, the planar polygonal graph G, along with another piece of information. The first such representation is a convex lifting of G.

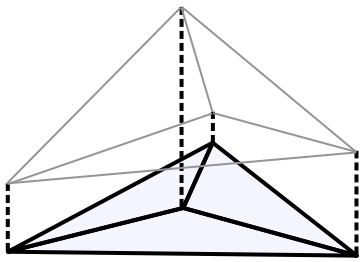


Figure 2-4: Convex lifting

More specifically, the *convex lifting representation* of T is the pair (G, H), where G is the planar projection of T, and H is a function mapping vertices of G to heights as a ratio of ε (see Figure 2-4). In other words, the z coordinate of vertex v of T is $H(v)\varepsilon$. Since T is almost flat with flatness ε, we see that $\forall v, 0 \leq H(v) \leq 1$.

For all intents and purposes, when it comes to unfolding, two almost-flat convex terrains T_1, T_2 are the same if they share the same G and have linearly related H, that is:

$$\exists a \in \mathbb{R}, \forall v \in G, H_{T_1}(v) = a \cdot H_{T_2}(v).$$

This is because, as long as they both have flatness ε which satisfies all the bounds mentioned in later sections, being flatter does not change their unfoldability, which depends only on those flatness thresholds and G.

We note that this "representation" is really no different than merely defining T as a graph on points in $\mathbb{R}^3$, where the z coordinates are in terms of ε. While the more useful representation for unfolding is the one detailed in the next section, it is much easier to generate and test for convexity using this representation since we can treat

ε as an arbitrary positive constant to obtain a terrain with real values. This is valid since, as mentioned before, scaling a terrain in the z axis does not affect its convexity.

2.5 Angle-Delta Representation

Instead of height at each vertex, we can instead represent T as the planar projection G along with a function D detailing the angle-delta at each angle of G. In more detail, if A is an angle of a face G, and A' is the corresponding angle of the corresponding face of T, then $D(A)\phi + a = a'$, where a and a' are the magnitude of angles A and A' in radians, and ϕ is a small constant representing the "flatness" for angle-deltas, just like how ε represents "flatness" for the convex lifting representation. Figure 2-5 shows an example angle-delta representation of an equilateral triangular pyramid.

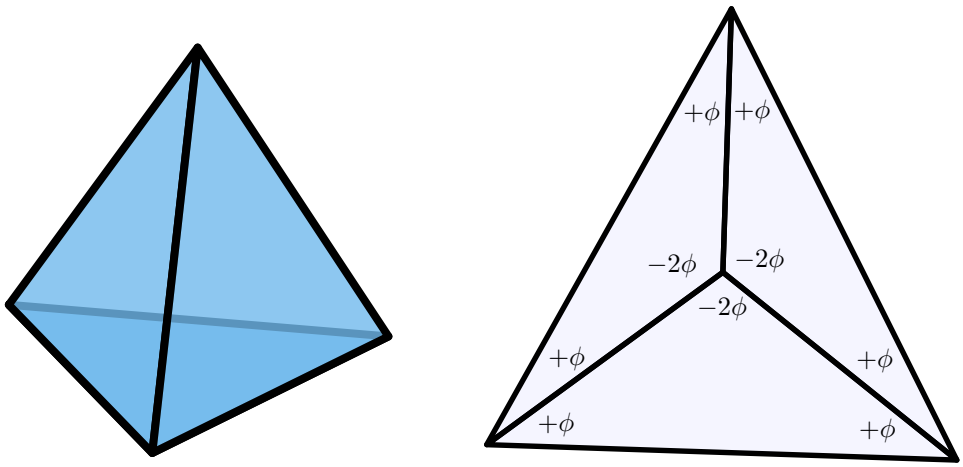


Figure 2-5: Angle-delta representation of a regular triangular pyramidal terrain

This *angle-delta representation* is useful for calculating unfoldings since it means we can just use G with minor angle adjustments. For instance, consider the almost-flat open pyramid T above. This is an almost-flat convex terrain with angle-delta representation (G, D) as shown. By making the approximation that the lengths of edges in G and T are equal and using the first-order approximation of sin, we see that we can calculate approximately what the unfolding of G will look like using just G, D, and simple arithmetic without having to calculate exactly the shapes of the faces

of T and laying them out on the plane. We will explore these concepts of unfolding using the angle-delta representation in more detail in Chapter 3, but this is the main motivation behind this representation.

2.5.1 Height to Angle-Delta First-Order Approximation

Since it is much easier to generate almost-flat convex terrains in the convex lifting representation, but it is easier to reason about unfolding using the angle-delta representation, we detail in this section a way to convert a convex lifting to the first-order approximation of the angle-delta representation.

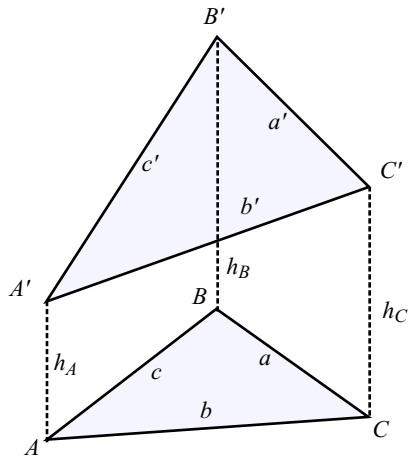


Figure 2-6: Calculating angle-delta from lifting

Referring to Figure 2-6, let A, B, C be three vertices of G such that $\angle ABC$ is an angle of a face of G. Then, let A', B', C' be the corresponding vertices of T. In the convex lifting (G, H) with flatness ε, let the lifting be h_A, h_B, h_C, so the actual heights are $h_A\varepsilon, h_B\varepsilon, h_C\varepsilon$. Now, consider triangle $\triangle ABC$. By the law of cosines, we have

$$b^2 = a^2 + c^2 - 2ac\cos B.$$

Similarly, for triangle $\triangle A'B'C'$ we have

$$b'^2 = a'^2 + c'^2 - 2a'c'\cos B'.$$

Substituting in

$$\begin{aligned} a' &= \sqrt{a^2 + \varepsilon^2(h_B - h_C)^2} \\ b' &= \sqrt{b^2 + \varepsilon^2(h_A - h_C)^2} \\ c' &= \sqrt{c^2 + \varepsilon^2(h_A - h_B)^2} \end{aligned}$$

for the edges, as well as $B' = B + D(B)\phi$ for the angle, where $D(B)\phi$ is the angle-delta for $\angle ABC$ which are we trying to calculate, we get

$$\begin{aligned} b^2 + \varepsilon^2(h_A - h_C)^2 &= a^2 + c^2 + \varepsilon^2(h_B - h_C)^2 + \varepsilon^2(h_A - h_B)^2 \\ &- 2\sqrt{(a^2 + \varepsilon^2(h_B - h_C)^2)(c^2 + \varepsilon^2(h_A - h_B)^2)}\cos(B + D(B)\phi). \end{aligned}$$

Subtracting the law of cosines for $\triangle ABC$ then yields

$$\begin{aligned} \varepsilon^2(h_A - h_C)^2 &= \varepsilon^2(h_B - h_C)^2 + \varepsilon^2(h_A - h_B)^2 \\ &- 2\sqrt{(a^2 + \varepsilon^2(h_B - h_C)^2)(c^2 + \varepsilon^2(h_A - h_B)^2)}\cos(B + D(B)\phi) \\ &+ 2ac\cos B. \end{aligned}$$

Letting

$$\begin{aligned} P &= h_B^2 - h_A h_C + h_B h_C + h_A h_B \\ Q_1 &= 2a^2(h_A - h_B)^2 + 2c^2(h_B - h_C)^2 \\ Q_2 &= (h_B - h_C)^2(h_A - h_B)^2, \end{aligned}$$

we can simplify the equation to be

$$\varepsilon^2 P = ac\cos B - \sqrt{a^2c^2 + \varepsilon^2 Q_1 + \varepsilon^4 Q_2}\cos(B + D(B)\phi).$$

Now, we make the first-order approximation for square root and cos, using the rea-

soning that ε and ϕ are very small, and simplify to get

$$\begin{aligned}
\varepsilon^2 P &\approx ac\cos B - \left(\sqrt{a^2c^2} + \frac{\varepsilon^2 Q_1 + \varepsilon^4 Q_2}{2\sqrt{a^2c^2}}\right)(\cos B - D(B)\phi \sin B) \\
\varepsilon^2 P &\approx ac\cos B - \left(ac + \frac{\varepsilon^2 Q_1 + \varepsilon^4 Q_2}{2ac}\right)(\cos B - D(B)\phi \sin B) \\
\varepsilon^2 P &\approx -\left(\frac{\varepsilon^2 Q_1 + \varepsilon^4 Q_2}{2ac}\right)(\cos B - D(B)\sin B) + acD(B)\phi \sin B
\end{aligned}$$

$$\begin{aligned}
\varepsilon^2 P + \frac{\varepsilon^2 Q_1 + \varepsilon^4 Q_2}{2ac}\cos B &\approx \left(ac + \frac{\varepsilon^2 Q_1 + \varepsilon^4 Q_2}{2ac}\right)(D(B)\phi \sin B) \\
2acP\varepsilon^2 + \varepsilon^2 Q_1 \cos B + \varepsilon^4 Q_2 \cos B &\approx (2a^2c^2 + \varepsilon^2 Q_1 + \varepsilon^4 Q_2)D(B)\phi \sin B \\
\frac{2acP\varepsilon^2 + \varepsilon^2 Q_1 \cos B + \varepsilon^4 Q_2 \cos B}{(2a^2c^2 + \varepsilon^2 Q_1 + \varepsilon^4 Q_2)\sin B} &\approx D(B)\phi \\
\frac{2acP + Q_1 + \varepsilon^2 Q_2}{(2a^2c^2 + \varepsilon^2 Q_1 + \varepsilon^4 Q_2)\sin B}\varepsilon^2 &\approx D(B)\phi.
\end{aligned}$$

Consolidating the constant terms, i.e.

$$c_1 = 2acP + Q_1 \cos B, c_2 = Q_2 \cos B$$

$$d_0 = 2a^2c^2 \sin B, d_1 = Q_1 \sin B, d_2 = Q_2 \sin B,$$

we have the first-order approximation

$$D(B)\phi \approx \left(\frac{c_1 + c_2\varepsilon^2}{d_0 + d_1\varepsilon^2 + d_2\varepsilon^4}\right)\varepsilon^2 \approx \left(\frac{c_1}{d_0} + \frac{c_2 - \frac{d_1 c_1}{d_0}}{d_0}\varepsilon^2\right)\varepsilon^2 \approx \frac{c_1}{d_0}\varepsilon^2$$

for $D(B)$. Making this calculation for every single angle of G, we can convert a convex lifting representation (G, H) with flatness ε of T into an approximate angle-delta representation (G, D) with flatness $\phi = \varepsilon^2$ which should be accurate enough assuming T is flat enough. More specifically, in this calculation we made the assumptions that

$$\sqrt{x + dx} \approx \sqrt{x} + \frac{dx}{2\sqrt{x}}$$

and

$$\cos(x + dx) \approx \cos(x) - \sin(x)dx$$

for small dx. Since cos has no degenerate points, and we only use the approximation for square root for positive constant values, these approximations should be accurate for small ε. Of course, the ε required will depend on G.

Now, by our definition of convex lifting, we limited all the h_i to be in the interval $[0, 1]$, so given that limitation, we can actually bound the absolute value of the angle-deltas:

$$\left|\frac{c_1}{d_0}\right| = \left|\frac{2acP + Q_1}{2a^2c^2}\right| \leq \left|\frac{6ac + 2a^2 + 2c^2}{2a^2c^2}\right| \leq \left|\frac{3}{ac} + \frac{1}{c^2} + \frac{1}{a^2}\right| \leq \frac{5}{\text{MinEdge}^2}.$$

Hence, by scaling G such that its minimum edge has length $\sqrt{5}$, we get a nice bound of 1 on the absolute value of any angle-delta — we assume that all G we consider from now on have this property, which can be easily achieved by uniformly scaling G appropriately.

2.6 Ideal Almost-flatness

We note from the calculations above, assuming our approximations to be accurate, that all non-zero angle-deltas have an ε^2 factor to them, so uniformly scaling the height liftings H by z will also uniformly scale the angle-deltas D by z^2 approximately. So, we can achieve liftings with vertex heights arbitrarily close to 0 which also have angle-deltas arbitrarily close to 0. This is actually what we want to work with: an "ideal" almost-flatness where all heights and angle-deltas are approximately 0, but we know the relative heights and relative angle-deltas. This allows us to freely use first-order approximations, which drastically simplifies the trigonometric calculations of unfolding into simple multiplication. For instance, $a \sin \theta$ can be simplified to be simply $a\theta$.

Indeed, the rest of our calculations will treat an almost-flat convex terrain T as such an "ideal" almost-flat terrains with essentially 0 height. This allows us to freely

use first-order approximations while dropping the ε and ϕ from calculations. At the same time, we will show that this is a valid simplification by providing bounds on ε to show that, for a given projection G, there exist real values of ε such that an almost-flat convex terrain $T = (G, H)$ with flatness ε shares the same properties which we are interested in as an "ideal" almost-flat convex terrain with projection G and relative heights H. So, while we will assume "ideal" almost-flatness, we will also prove ε bounds (i.e. height bounds) to show that such "ideal" almost-flatness is achievable with real values.

Chapter 3

Edge-Unfolding Almost-Flat Convex Terrains

This chapter covers a theoretical exploration of edge-unfolding almost-flat convex terrains through four sections. The first section provides a definition of various ways to specify unfoldings as well as a demonstration of how unfolding affects the faces of an almost-flat convex terrain T in regards to its projection G. The second section explores how much information about the unfoldability of a terrain one can gleam from just its projection. The third section examines what cut-trees unfold without overlapping and proves a result on extending partial cut-trees. The fourth section demonstrates an alternative unfolding technique based on treating the faces of an almost-flat convex terrain as "slices."

3.1 Cut-Forests and Glue-Trees

An *edge-unfolding* of a polyhedron P is a cut-tree or glue-tree of P. A *cut-tree* C is a spanning tree of the vertices of P, and its corresponding *glue-tree* $\overline{C}$ is a spanning tree of the dual of P such that, for every edge e not in C, the dual edge of e is in C'. A cut-tree specifies which edges to "cut" faces apart in order to unfold P, while a glue-tree specifies which edges to "glue" faces together in order to unfold P. Let $\overline{C}$ be an edge-unfolding glue-tree of P, and let $P_{\overline{C}}$ be the faces of P such that $f_i, f_j \in P_{\overline{C}}$

are connected by edge e if the dual of e is in $\overline{C}$. Then, setting all dihedral angles of $P_{\overline{C}}$ to be π gives us the *net* of edge-unfolding P by glue-tree $\overline{C}$. In other words, the net of unfolding P by $\overline{C}$ is a planar embedding $P_{\overline{C}}$ of the faces of P such that two faces of $P_{\overline{C}}$ share edge e if the dual of e is in $\overline{C}$. Similarly, we can define the net of edge-unfolding P by a cut-tree C to be the net yielded by unfolding using the corresponding glue-tree $\overline{C}$. Figure 3-1 gives an example cut-tree and its corresponding glue-tree of a square pyramid, along with the resulting net.

(a) Cut-tree C (red) and glue-tree $\overline{C}$ (blue)

(b) Net from $C/\overline{C}$

Figure 3-1: Cut-trees, glue-trees and unfolded nets

The intuition behind *edge-unfolding* a polyhedron P is that we are "cutting" or "gluing" the faces of a polyhedron along edges and then "flattening" the result to get a net. If two faces of a net intersect, then we say that they *overlap*, and that the net *overlaps*. In addition, an overlap must be caused by a vertex v being in the interior of a face f, so for convenience we say that that v *overlaps with* f and also that v *overlaps with* e, where e is the closest edge(s) which intersects with an adjacent edge to v (Figure 3-2). If no faces overlap in the net of unfolding P by C, then we say that C *unfolds* P, and if some cut-tree of P unfolds P, then P *is (edge) unfoldable*.

The same definitions apply to open polyhedra and terrains, except that instead of a cut-tree of terrain T, we have a *cut-forest*, which is a spanning forest of the vertices of T, where each tree contains exactly one boundary vertex of T. This is required for the net of unfolding T by a cut-forest C to remain a single connected component of faces. For most of the time, we will be dealing with unfolding almost-flat convex terrains and not polyhedra, so let us clarify some terms which we will use: A "cut-forest" will indicate an unfolding of a terrain T as detailed above, while a "cut-path"

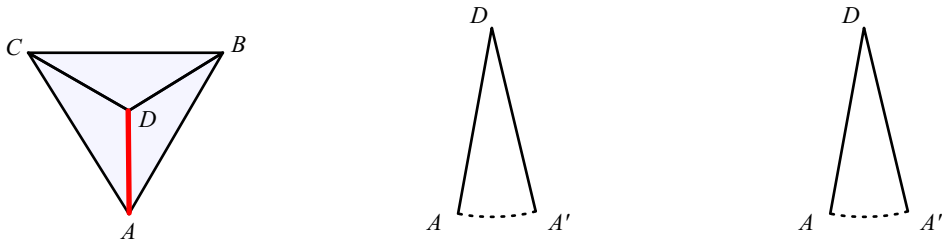
(a) Local overlap example (b) v overlaps with e and f

Figure 3-2: Terminology for overlaps

will indicate a directed path of a subset of the vertices of T ending at a boundary vertex of T and a "cut-tree" will indicate a tree of a subset of vertices of T rooted at a boundary vertex of T. More specifically, for a cut-tree t of a terrain T, let its *root* be the one vertex of t which is on the boundary of T, and let a *leaf* of t be any non-root vertex of t with only one incident edge in t.

3.1.1 Unfolding Motion

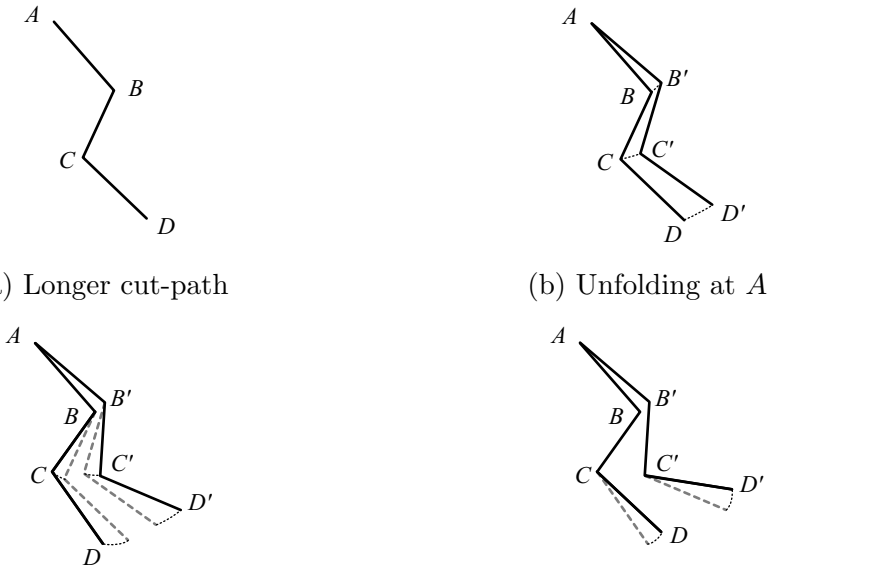
In this section, we will take a look at how faces of an almost-flat convex terrain T move away from their corresponding faces of the planar projection G when unfolded by a cut-tree C. Let us start with the simplest almost-flat convex terrain — a triangular pyramid without the bottom face — and the simplest cut-tree — a single edge. As shown in Figure 3-3, $ABCD$ is a triangular pyramid with angle-deltas as shown, and we are interested in the edge-unfolding by the single cut edge $\overline{DA}$. This results in $ABCA'D$, where $\overline{DA}$ splits into two edges $\overline{DA}$ and $\overline{DA'}$, which opens with angle 3ϕ, the negative of the sum of the angle-deltas at D.



(a) Cut-tree of one edge (b) Unfolded net (c) Just the cut-edges

Figure 3-3: Unfolding motion of a cut-edge

Figure 3-3c shows the results of the unfolding through just the cut-edge $\overline{DA}$: vertex A is first split into A and A', and then fixing $\overline{DA}$, we see that $\overline{DA'}$ rotates by 3ϕ around D to give the final position of A' in the unfolded net. Note that indeed, we only need to look at the cut-edges when considering if a net overlaps, since those are the only edges which can intersect and cause overlaps. Similarly, when considering such cut-edges, we only care about the aggregate angle-delta at each angle of the cut-tree.



(a) Longer cut-path

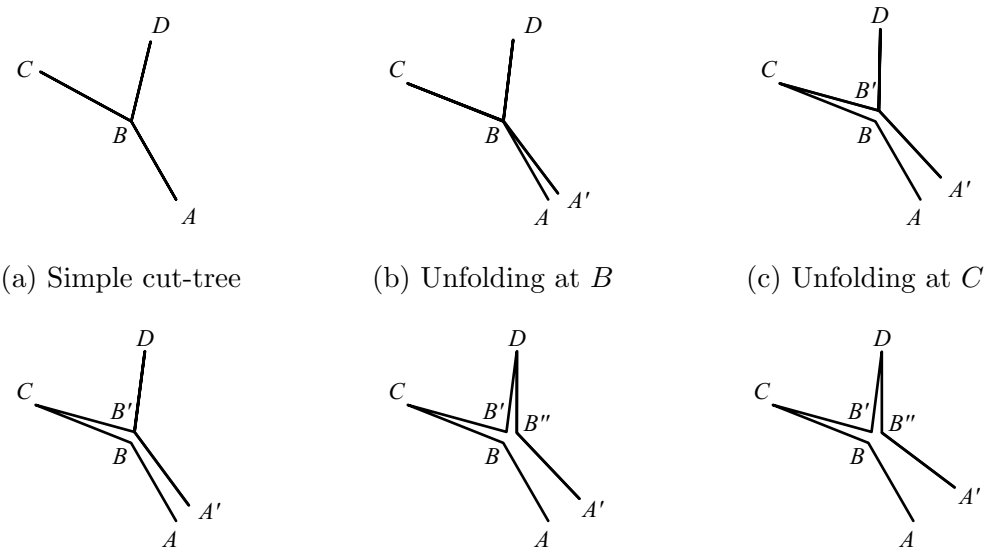
(b) Unfolding at A

(c) Unfolding at B,B'

(d) Unfolding at C,C', final unfolding

Figure 3-4: Unfolding motion of a cut-path

Using this abstraction of considering just the cut-edges, let us observe a more complicated cut-path unfolding. Figure 3-4 shows a cut-path $ABCD$ with aggregate angle-deltas as listed and how the unfolding based on fixing $\overline{AB}$ proceeds. First, unfolding at A rotates the entire subpath $AB'C'D'$ by ϕ_A around A. Next, unfolding at B rotates the subpath BCD by ϕ_B around B and unfolding at B' rotates the subpath $B'C'D'$ by $\phi_{B'}$ around B'. Finally, the unfoldings at C and C' rotate the edges $\overline{CD}$ and $\overline{C'D'}$ by ϕ_D and $\phi_{D'}$ around C and C' respectively. Note that in this example, we fixed the edge $\overline{AB}$, but we could have fixed any edge and still get the same resulting unfolded cut-path in terms of relative vertex locations. Likewise, we can apply the same concepts to a cut-tree, as shown in Figure 3-5.



(a) Simple cut-tree (b) Unfolding at B (c) Unfolding at C

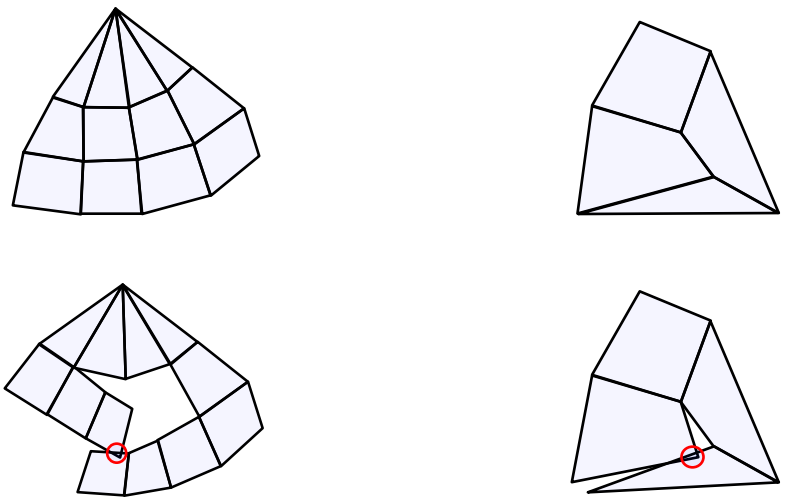
(d) Unfolding at B' (e) Unfolding at D (f) Unfolding at B'', final unfolding

Figure 3-5: Unfolding motion of a cut-tree

Hence, overall we see that we can figure out the unfolded form of T by a cut-tree by considering just the cut-tree itself. We do this by calculating aggregate angle-deltas at each vertex of the unfolding cut-tree and then "unfold" each vertex by rotating the rest of the cut-tree around that vertex by the aggregate angle-delta at that vertex. Then, an overlap would involve an intersection of cut-tree edges. This abstraction is very important since it allows us to focus on the cut-trees themselves while not actually simplifying away any part of the original problem of figuring if an unfolding results in an overlapping net.

3.1.2 Local Overlaps

An overlap which occurs between two edges which share a vertex in T is a *local overlap*, and otherwise it is a *non-local overlap* (Figure 3-6). Local overlaps are easier to consider than general overlaps since they depend mostly on the local geometry. This is the main reason we picked almost-flat convex terrains: because they are almost flat they have small angle-deltas, and hence their nets should be relatively similar to their projections. This then implies that their nets will only have local overlaps if any



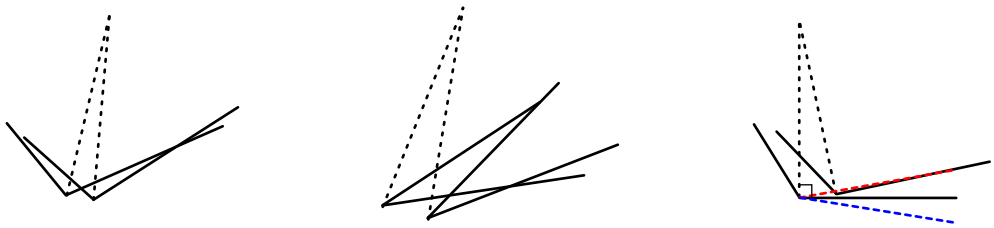
(a) Non-local overlap cut-tree and net (b) Local overlap cut-tree and net

Figure 3-6: Non-local and local overlaps

at all. In Section 3.1.3 we will formally prove this for arbitrary almost-flat convex terrain T by showing a height bound based on T which achieves this.

Meanwhile, let us consider the implications assuming unfoldings of T with projection G can only have local overlaps. First of all, this means that we can consider cut-trees of a cut-forest of T separately, since overlaps between edges of different cut-trees are not local overlaps. Next, note that since the motion of unfolding is rotation, it follows intuitively that a cut-path which is mostly straight in G will have no overlaps regardless of the distribution of angle-deltas — we will explore this idea more in Section 3.2. Similarly, if we assume our analysis of unfolding motions to be accurate, then a local overlap can only be caused by an acute angle between an edge and the vertex it is being rotated around for the unfolding. So, we want to avoid acute angles in cut-paths, but we also want to avoid right angles. The reason is that the angle-deltas might cause a right angle to become an acute or obtuse angle (see Figure 3-7), but it gets more complicated than that — considering angle-deltas when checking whether the motion of unfolding will clear an angle requires dealing with second-order effects, a problem we will discuss more in Section 3.3.3.

Finally, we note that if T is fully flattened (say, by setting $\varepsilon = 0$), then it definitely has no overlapping unfoldings since it becomes the planar graph G. Similarly, as we



(a) Local overlap on obtuse angle

(b) Local overlap on acute angle

(c) Ambiguous right angle case

Figure 3-7: Example local overlaps at various angles

will show soon, for every T there is a certain $\varepsilon_{\text{local}}$ for which all unfoldings of T flatter than $\varepsilon_{\text{local}}$ can only have local unfoldings. Therefore, it becomes a question of whether for arbitrary T there is an $\varepsilon_{\text{no-overlap}}$ for which all unfoldings of T flatter than $\varepsilon_{\text{no-overlap}}$ has no overlaps at all. That is, is there a height bound below which the question of edge-unfoldability is trivial for almost-flat convex terrains? Unfortunately, this is not so — it is easy to see that, for most T, there are cut-trees which always overlap no matter how flat T is. One such example is seen in Figure 3-8: the cut path ABC will always overlap at B' due to the unfolding around A since $\angle ABC$ is acute. Then, because T is almost flat, B' will travel a small distance and $\overline{BC}$ will only "open" a short distance, and hence they will always overlap.

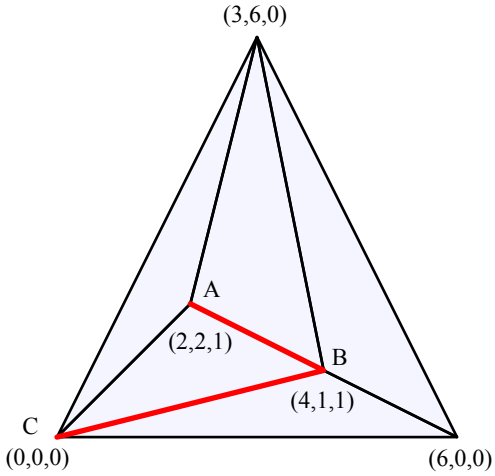


Figure 3-8: Always overlapping cut-tree

If T were not almost flat, it could have been either that ϕ_B is large enough to make $\angle ABC \geq \pi$ or that B' travels far enough that no overlaps happen. In a way, almost-flatness makes it harder for T to unfold since it actually causes more local overlaps to occur.

3.1.3 Height Bound for Local Overlaps

Now, let us show that we can actually achieve the "only-local-overlaps" property with an actual height bound. To start, note that, for a cut-tree of $T = (G, H) = (G, D)$ to cause a non-local overlap, a vertex v must move a relatively large distance. Namely, assuming the target edge e which v overlaps with is fixed, the cut-tree must move v at least the distance between v and e in G. This distance is a strictly positive and finite value which does not depend on ε, but instead only on the geometry of G. For now, let us assume that the convex lifting does not change the length of edges of G appreciably; we will show later in Section 3.3.3 that this minute lengthening of the edges of G only causes a second-order effect which can be ignored when compared to the first-order effect supposing a flat enough ε. Meanwhile, let us prove the following bound:

Theorem 1 (Height Bound for Local Overlaps Only)**.** *Let $T = (G, H) = (G, D)$ be an almost-flat convex terrain with flatness $0 < \varepsilon < 1$. Let* SmallDist(G) *be the smallest distance between a vertex v of G and an edge e of G such that v is not adjacent to e, and let E_G be the edge set of G. Then, we have the bound*

$$\text{BoundLocal}(T) = \frac{\text{SmallDist}(G)}{\left(\sum_{e \in E_G} |e|\right)\left(\sum_{d \in D} |d|\right)}.$$

Then, if $\varepsilon <$ BoundLocal(T)*, every unfolding of T can have only local overlaps if any at all.*

Proof. A cut-tree C can only cause a non-local overlap involving a vertex v if v overlaps with a non-adjacent edge e. Assuming we unfold C with e fixed, we see that the unfolding of C must move v a distance at least that of the distance between e

and v, which must be at least SmallDist(G). What we want to do is to bound the total distance C moves v by bounding the real maximum angle-delta ($\varepsilon^2 \max_D |d|$), which in turn gives a bound on ε. Now, each angle of C with angle-delta d at vertex v_C moves v at most $\text{dist}(v, v_C) \sin d$. Therefore, the total contribution of C in terms of moving v in unfolding is

$$\begin{aligned}
Total &= \sum_{v_C \in C, d \in D(\angle v_C)} \text{dist}(v_C, v) \sin |d\phi| \\
&\leq \sum_{v' \in V_G, d \in D(\angle v')} \text{dist}(v', v) \sin |d\phi| \\
&\leq \sum_{v' \in V_G, d \in D(\angle v')} \text{dist}(v', v) |d\phi| \\
&\leq \sum_{v' \in V_G, d \in D(\angle v')} \left(\sum_{e \in E_G} |e| \right) |d\phi| \\
&\leq \sum_{d \in D} \left(\sum_{e \in E_G} |e| \right) |d\phi| \\
&= \left(\sum_{e \in E_G} |e| \right) \left(\sum_{d \in D} |d| \right) \phi,
\end{aligned}$$

where V_G is the vertex set of G, $D(\angle v)$ is the set of angle-deltas at v, and ϕ is the corresponding "flatness" value for angle-deltas. So, we need

$$\left(\sum_{e \in E_G} |e| \right) \left(\sum_{d \in D} |d| \right) \phi < \text{SmallDist}(G).$$

Solving for ϕ gives

$$\phi < \frac{\text{SmallDist}(G)}{\left(\sum_{e \in E_G} |e| \right) \left(\sum_{d \in D} |d| \right)} = \text{BoundLocal}(T).$$

Now, remembering that $\phi \approx \varepsilon^2$ from our calculations in Section 2.5.1, we see that, if $\varepsilon < K$ for some constant K and $0 < \varepsilon < 1$, then $\phi = \varepsilon^2 < \varepsilon < K$. Hence, if $\varepsilon < \text{BoundLocal}(T)$, then we know that any cut-tree of T, on unfolding with any edge e fixed, will move any vertex v a distance less that SmallDist(G), and so there

can be no non-local overlaps in any unfolding of T. □

This bound is by no means tight. We can achieve a tight bound for a given $T = (G, H) = (G, D)$ by considering all possible cut-trees of G, and then scaling ε enough such that no cut-tree of G causes a non-local overlap. However, this theorem shows that such a bound indeed exists and is non-zero. Hence, we can achieve the "no non-local overlap" property for any convex terrain T by scaling all heights of T by $\frac{\text{BoundLocal}(T)}{\text{MaxHeight}(T)}$, showing that this property of our "ideal" almost-flatness is achievable with real values.

3.2 Projections and Unfolding

In this section we will examine in more detail the projection G of almost-flat convex terrains. We will show that it is possible to find cut-paths which will probably unfold, as well as cut-paths and cut-trees which will definitely unfold, just by looking at G without worrying about H or D, assuming the terrain is flat enough. However, we will also show that there exist projections G for which no such definitely unfolding cut-paths and cut-trees exist.

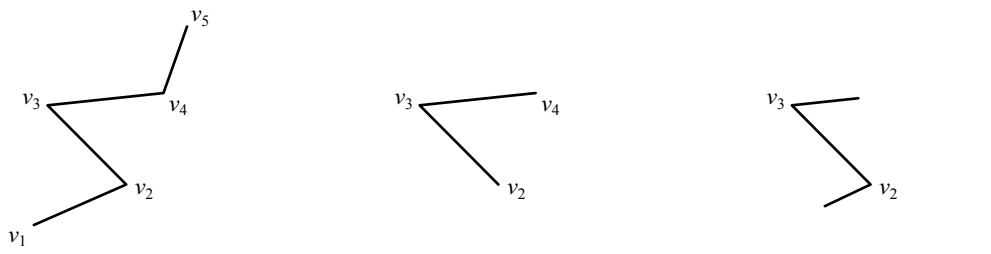
3.2.1 Path Definitions

First, let us define some terminology involving paths (Figure 3-9). A *directed path* p from v_1 to v_n of a graph G is an ordered list of points $p = (v_1, v_2, \ldots, v_n), v_i \in V$, where $(v_1, v_2), \ldots, (v_{n-1}, v_n)$ are all edges of G. For convenience we will often drop the word "directed" and just talk about "paths," and also assume that we are talking about paths of G, even if the G is not specifically mentioned. Similarly, when we talk about "unfolding a path p," we mean to treat p as a cut-path, and then unfold that cut-path. We will also write the shorthand $(v, \ldots)$ to mean a path starting from v, $(\ldots, v)$ to mean path ending at v, and $(\ldots, v, \ldots)$ to mean a path containing v, and only fill in other vertices as necessary when we need them. A lot of times it will also be useful to consider a path $p = (v_1, \ldots, v_n)$ as a function $f_p(x)$ defined for $1 \leq x < n$

where

$$f_p(x) = (1 - (x - \lfloor x \rfloor))\, v_{\lfloor x \rfloor} + (x - \lfloor x \rfloor)\, v_{\lfloor x \rfloor + 1}.$$

For instance, $f_p(1) = v_1$ and $f_p(2.5)$ is the midpoint of the edge between v_2 and v_3. Finally, let us define a *subpath* of $p = (v_1, v_2, \ldots, v_n)$ to be some path $(v_i, \ldots, v_j)$ for integers i, j, and a *partial subpath* to be a path $(f_p(a), v_{\lfloor a \rfloor + 1}, \ldots, v_{\lfloor b \rfloor}, f_p(b))$ for reals a, b, that is, a "subpath" of p which does not necessarily begin or end on vertices of G.



(a) Path $p = (v_1, \ldots, v_5)$ (b) Subpath $(v_2, \ldots, v_4)$ (c) Subpath $(f_p(1.5), \ldots, f_p(3.5))$

Figure 3-9: Path and subpath terminology

3.2.2 Weakly Monotonically Increasing Distance (WMID) Paths

Let us start with a weak approximation of an unfolding cut-path which we think will likely unfold. As discussed in Section 3.1.2, for an unfolding motion centered on v_1 of the path $p = (v_1, \ldots)$, in order to avoid local overlaps we want to avoid making acute angles (and right angles) between the line from v_1 to an edge e of the path and the edge e itself. So, let a *weakly monotonically increasing distance* path, or *WMID* path, be a path $p = (v_1, \ldots, v_n)$ such that, if we let f_p be the function representation of p, then

$$\forall a \in \mathbb{R}, 1 < a < n, \|v_1 - f_p(a)\|^2 + \|f_p(a) - v_{\lfloor a \rfloor + 1}\|^2 < \|v_1 - v_{\lfloor a \rfloor + 1}\|^2.$$

That is, for any point $f_p(a)$ on the path and the vertex $v_{\lfloor a \rfloor+1}$ immediately after it, we have that the triangle $\triangle v_1 f_p(a) v_{\lfloor a \rfloor+1}$ is an obtuse triangle with obtuse angle $\angle v_1 f_p(a) v_{\lfloor a \rfloor+1}$. Also, note that the distance function $g(a) = \|v_1 - f_p(a)\|$ is a strictly monotonically increasing function, which is where the name comes from.

Now, let us look at some properties of WMID paths which let them "somewhat" unfold when used as cut-paths (Figure 3-10). Let $p = (v_1, \ldots, v_n)$ be a WMID path. We see that, by the definition, for any point $x = f_p(a)$ on edge e of p we have that the line $\overline{v_1 x}$ forms an obtuse angle with the rest of e, i.e. $\{y \in e \mid y = f_p(b), b \geq a\}$. We also note that, since the distance from v_1 is a strictly monotonically increasing function, every circle centered on v_1 will intersect p at most once. Both these facts mean that, if we unfold by some small angle ϕ at v_1, assuming the rest of the angles stay fixed, then there will be no local overlaps since the unfolding motion is a rotation.

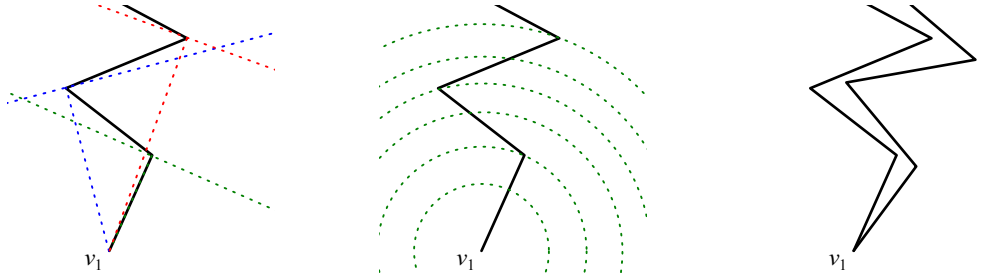


Figure 3-10: WMID path, showing obtuse angles from v_1, single intersection with circles at v_1, and example unfolding

So, a WMID path $p = (v_1, \ldots, v_n)$ of G will unfold without overlap if the aggregate angle-delta at v_1 is some small $-\phi$, and the aggregate angle-deltas at the rest of the v_i is 0.

However, at the same time, this means that many WMID paths of G will not unfold in actuality, since it is impossible for non-boundary vertices of a convex terrain to have 0 aggregate angle-delta, and since with WMID paths we are only concerned that the angle the first vertex makes with the rest of the path is obtuse, the path itself might actually have many acute angles as the above example shows.

3.2.3 Strongly Monotonically Increasing Distance (SMID) Paths

What we really want is a path where every subpath is a WMID path. Let a *Strongly Monotonically Increasing Distance* path or *SMID* path be a path $p = (v_1, \ldots, v_n)$ such that, if we let f_p be the function representation of p, then

$$\forall k \in \mathbb{Z}, \forall a \in \mathbb{R}, 1 \leq k < a < n, \|f_p(k) - f_p(a)\|^2 + \|f_p(a) - v_{\lfloor a \rfloor + 1}\|^2 < \|v_1 - v_{\lfloor a \rfloor + 1}\|^2.$$

Essentially, this says that every subpath $(v_k, \ldots, v_n)$ is a WMID path. Also, by changing the 1 in $1 \leq k < a < n$ to any integer $l < n$, we see that every subpath $(v_l, \ldots, v_n)$ is also a SMID path.

Figure 3-11: SMID paths can be unfolded from any vertex without overlaps

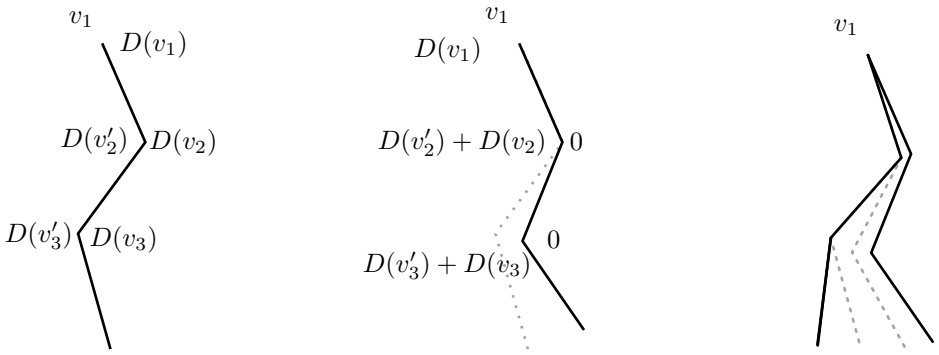
Whereas WMID paths can only be unfolded at v_1 without overlap (assuming the rest have angle-delta 0), since subpaths of SMID paths are WMID paths, SMID paths can be unfolded at any vertex without overlap (again, assuming the rest have angle-delta 0), as seen in Figure 3-11. However, we can do even better than that: we will prove that SMID paths can be always be unfolded without overlap regardless of the angle-deltas.

Theorem 2 (SMID Paths Unfold)**.** *Let $p = (v_1, \ldots, v_n)$ be a SMID path of G, the planar projection of an almost-flat convex terrain T. Then, there is some ε such that, if T is flatter than ε, then the cut-path p unfolds without overlap.*

Proof. First, imagine the path pointing down — conceptually, we can rotate the plane so that $\overrightarrow{v_1 v_2}$ points in the negative y direction — then double the vertices of the path,

so we have $v_1, \ldots, v_n$ on the right and $v'_1, \ldots, v'_n$ on the left. Next, for $1 < i < n$, let $D(v_i)$ be the aggregate angle-delta for the angle $\angle v_{i-1}v_iv_{i+1}$, and similarly for $D(v'_i)$, and let $D(v_1 = v'_1)$ be the aggregate angle-delta around vertex v_1 since this vertex does not split on unfolding. Now, let us consider the act of unfolding in two stages, summarized in Figure 3-12.

In the first stage, we "bend" the path p into $p' = (w_1, \ldots, w_n)$ such that, for $1 < i < n$, we have, on the right side, $\angle w_{i-1}w_iw_{i+1} = \angle v_{i-1}v_iv_{i+1} + D(v_i)\phi$, where ϕ is the angle flatness which depends on ε. This gives angle-deltas $D(w_i) = 0$ on the right, and angle-deltas $D(w'_i) = D(v'_i) + D(v_i)$ on the left. Essentially, we are shifting all the non-zero delta angles from the v_i side to the v'_i side by bending the original path by delta angles.



(a) Initial path (b) Bending (c) Peeling

Figure 3-12: SMID path unfolding

Then, in the second stage, we "peel" off the w'_i side of the path. Due to convexity, we have that at every interior point v, the aggregate angle-delta around that point — the sum of the angle-deltas of all angles at that point — is negative. By our bending in the first stage, the aggregate angle-delta around point w_i is equal to the aggregate angle-delta on the side of w'_i, which then must be negative. So, now we open, or unfold, each angle from w'_1 to w'_{n-1} in order. We assert that this will not result in overlaps as long as certain conditions hold:

Lemma. *If the bent p' is a SMID path and ϕ is small, then unfolding by opening the angles from w'_1 to w'_{n-1} will not result in overlaps.*

Proof. First, note that since we are unfolding by opening each angle from w'_1 to w'_{n-1} in order, then when opening the angle w'_i, the subpath $(w'_i, \ldots, w'_n)$ is a SMID path. This is because p' started as a SMID path, and by opening the angles $w'_1, \ldots, w'_{i-1}$ we have merely translated and rotated the SMID subpath $(w_i, \ldots, w_n)$ to the SMID subpath $(w'_i, \ldots, w'_n)$. Then, using the assumption that ϕ is small and our original description of p pointing down, we see that left side p'_l only moves a small distance from the right side $p'_r = p'$, which does not move. Hence, throughout the unfolding process, p'_l is to the left of p'_r, and on unfolding, we rotate a subpath of p'_l to the left, i.e. clockwise.

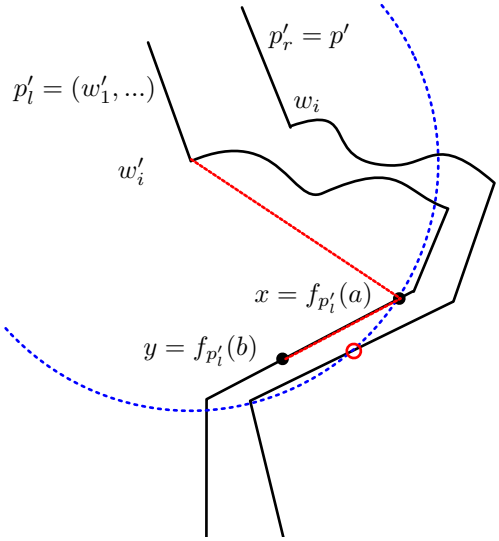


Figure 3-13: Suppose opening at w'_i causes conflict: rotating x around w'_i will cause a conflict at the circled point on p'. However, this means $\angle w'_i x y$ is acute, contradicting $(w'_i, \ldots)$ being SMID.

Let us define the C_{a_1,a_2} to be the circle centered at $f_{p'_l}(a_1)$ = passing through $f_{p'_l}(a_2)$. Then, suppose for the sake of contradiction that on unfolding w'_i, a local conflict is formed. Referring to Figure 3-13, this means that there is some point $x = f_{p'_l}(a), a > i$ later on in the path, such that if we draw the arc clockwise from x of angle $D(w'_i)\phi$ from the circle $C_{i,a}$, then the arc will intersect p' before it intersects p'_l, if it intersects p'_l at all. But we see that, for this to happen, no matter if x is a vertex of p'_l or x is in the middle of an edge, there is some later point $y = f_{p'_l}(b), a < b \leq \lfloor a \rfloor + 1$ inside the circle $C_{i,a}$, which contradicts the fact that $(w'_i, \ldots, w'_n)$ is a SMID path

because then the angle $\angle w_i' xy$ is acute. Therefore, by contradiction, we must have that there are no such local conflicts formed throughout the unfolding process if p' is a SMID path. □

So, p unfolds if the bent p' is a SMID path, and ϕ is small enough that the left side does not move too far from the right side such that they switch sides. This latter condition is almost always upheld and only becomes a problem if v_1 is a very sharp point, which is impossible for the almost-flat case. Meanwhile, for the former condition we see that we need to pick ϕ small enough such that no bending can change a SMID path into a non-SMID path, which we will show below. Hence, for some ε_0 less than the bound demonstrated below, if T is flatter than ε_0, then the cut-path p unfolds without overlap, completing our proof. □

In fact, we will give a stronger bound which holds for all SMID paths of G, not just a specific one.

Theorem 3 (Height Bound for SMID Unfolding)**.** *For any convex planar graph G, there exists an $\varepsilon > 0$ such that any convex lifting of G into an almost-flat convex terrain T with flatness ε has the property that any SMID path of G is an unfolding cut-tree for T.*

Proof. First, let AngleMin(G) be the minimum difference between a right angle and an angle formed by a v of a SMID path $p = (v, \dots)$ and a later edge e on p. More formally, this can be expressed as

$$\text{AngleMin}(G) = \min_{p=(v_1,\dots,v_n)\in G_{P_{\text{SMID}}}, 1<a<b<n} \pi - |\pi - \angle v_1 f_p(a) f_p(b)|\,,$$

where $G_{P_{\text{SMID}}}$ is the set of SMID paths of G. This value cannot be 0, because by definition of SMID paths, all such angles must be strictly obtuse angles, and since there are a finite number of paths and angles, such a minimum difference must exist and be positive.

Now note that, if the sum of the absolute values of all the angle-deltas times ϕ is less than AngleMin, then it would be impossible for a SMID path to be bent into a

non-SMID path, since even in the worst case SMID path, even if all the angle-deltas are added together, it would be impossible to change any obtuse angle to a right or acute angle, so all SMID paths will remain SMID paths after bending. Since by Section 2.5.1 we see that the absolute value of each angle-delta is bound by 1, we need

$$|G_A|\,\phi < \text{AngleMin}(G),$$

where G_A is the set of angles of G. Next, we see that $2\,|G_E| > |G_A|$, since each edge is opposite at most two angles. Also, we know from our calculation of angle-deltas from heights that $\phi = \varepsilon^2$. Putting it all together gives us the bound

$$\text{BoundSMID}(G) = \sqrt{\frac{\text{AngleMin}(G)}{2\,|G_E| + 1}}.$$

So, if $\varepsilon < \text{BoundSMID}(G)$, then all convex liftings T flatter than ε have the property that all SMID paths of G unfold without overlaps when used as cut-paths of T. □

Note that the bound we devised does not depend on T, but only on G. This means that we can strengthen our previous theorem to the following:

Theorem 4 (SMID Paths Unfold, Stronger Version)**.** *Let G be a planar graph with convex faces and a convex boundary. Then, any SMID path of G unfolds for any convex lifting of G which is flatter than* BoundSMID(G)*.*

Proof Sketch. Combine the bound we just calculated with the previous proof applied to all SMID paths of G. □

3.2.4 SMID Trees

Similarly to how paths which are mostly straight would intuitively always unfold, trees which have mostly straight branches should also always unfold. Let a cut-tree t be a *Strongly Monotonically Increasing Distance tree* or *SMID tree* if every path from a leaf of t to the root of t is a SMID path. This means that, if t has leaves $l_1, \ldots, l_n$,

and root r, then the unique paths $(l_i, \ldots, r)$ comprised of only the edges of t are all SMID paths.

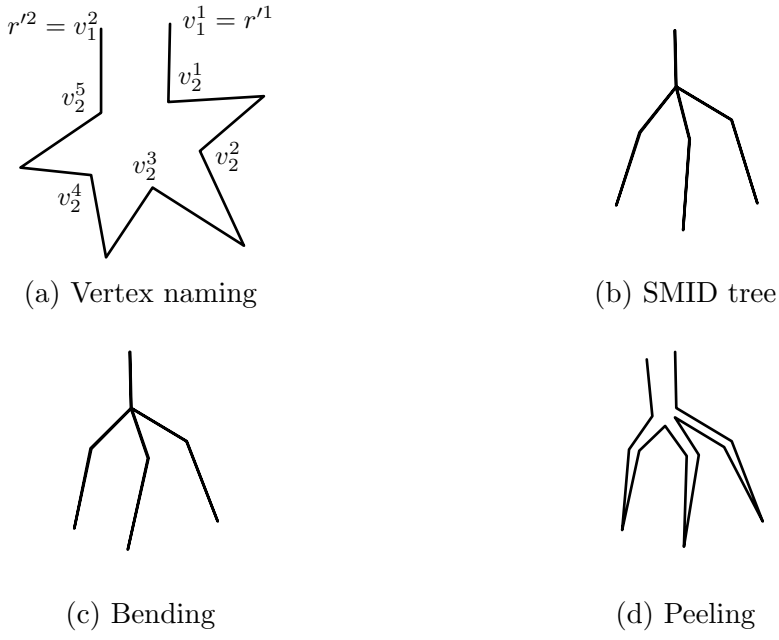
Let us give some intuition as to why SMID trees should unfold. Each sub-path of a SMID tree going from leaf to root is a SMID path, so each part of a SMID tree should unfold by itself. Then, putting it all together should also result in a non-overlapping unfolding. Informally, we can imagine this as the following: we start with a SMID path, which we know to unfold by the previous proofs. Then, we unfold a branch attached to this path. The worry is that unfolding this branch will cause conflicts with the previously unfolded path. However, this will not happen since the branch, along with the rest of the path, form a SMID path as well. Hence, it should not cause conflicts.

Theorem 5 (SMID Trees Unfold)**.** *If t is a SMID cut-tree of G, then there is some $\varepsilon > 0$ such that, if a convex terrain lifting T of G is flatter than ε, then t will unfold.*

Proof. We will again use a two-step unfolding process (Figure 3-14) similar to what we did before with SMID paths. We start by arranging t so that the root of t faces in the negative y direction, so now we have left and right sides of each branch of t. Then, each vertex of t splits into multiple vertices: vertex v splits into k vertices, where k is the degree of v in t, let them be $v^1, \ldots, v^k$ from right to left.

In the first stage of unfolding, we want to bend t into t' such that, for every vertex v split into vertices $v^1, \ldots, v^k$, the aggregate angle-deltas of $v'^1, \ldots, v'^{k-1}$ are all equal to 0, while the aggregate angle-delta of v'^k is equal to the aggregate angle-delta around vertex v. This is done similarly as our previous proofs by bending the tree t by delta angles into tree t'.

Then, in the second stage of the unfolding, we again peel away the left sides by unfolding the angles from right to left. Note that after unfolding, the tree t' becomes a path $r'^1, \ldots, r'^2$, where r' is the root of t' and r'^1 and r'^2 are its right and left images respectively (see Figure 3-14a). So, we unfold the angles in the same order along this path, from r'^1 to r'^2. Note that this is the same order we unfolded the left side when we did SMID path unfolding. By unfolding in this order, when we have unfolded



(a) Vertex naming (b) SMID tree

(c) Bending (d) Peeling

Figure 3-14: SMID tree unfolding

$r'^1, \ldots, v'$, the rest of the tree $w', \ldots, r'^2$ remains a SMID tree, since it is just rotated and moved to its current location. Therefore, by a proof similar to the one we used for SMID paths, we see that peeling a SMID tree will not lead to overlaps assuming a small enough ε.

In fact, we can use the same ε bound as we used for SMID paths. This is because, for a SMID tree to remain a SMID tree after bending, every path from leaf to root must remain a SMID path. This reduces to that, for every SMID path p of G, when each angle v of p is bent by up to the aggregate angle-delta around vertex v in either direction, p remains a SMID path. This results in exactly the same bound as before.

Therefore, we see that the SMID cut-tree t does indeed unfold without overlap assuming the terrain T is flatter than $0 < \varepsilon <$ BoundSMID(G). □

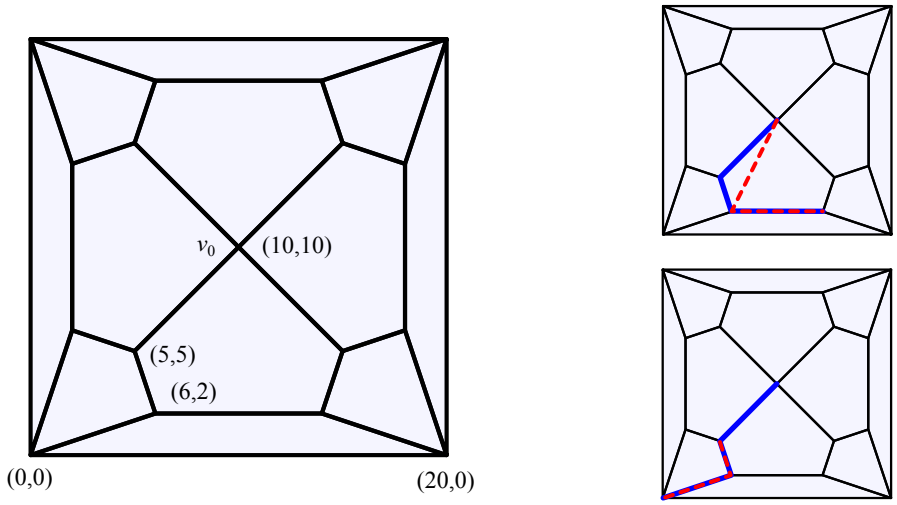
Corollary 6. *Any SMID cut-tree of G unfolds without overlap in any convex-lifting T as long as T is flatter than* BoundSMID(G).

This is a nice result since it means that, in some cases, we do not even need to calculate convex liftings or angle-deltas and instead just need to find SMID cut-trees of G, which we know to always unfold. By combining multiple such SMID cut-trees

together into cut-forests which span all vertices of G, we can find SMID cut-forests which will unfold for any convex lifting, assuming appropriate almost-flatness.

3.2.5 Projections with no SMID Paths

However, it is not always possible to find a cut-forest, or even a cut-path, for all G. For instance, consider Figure 3-15a of a convex planar graph G_{NS1} for which there exists no cut-path from the center vertex v_0 to the boundary. By symmetry, any path from v_0 to the boundary has to contain one of the two subpaths shown. However, as Figure 3-15b shows, neither subpath is a SMID path, and so no paths containing them can be SMID paths either.



(a) G_{NS1} with no SMID paths to center

(b) 2 subpaths with acute angles shown

Figure 3-15: Projection with no SMID paths to center vertex

Hence, G_{NS1} has no SMID paths from v_0 to the boundary, and therefore, by extension, there are no SMID trees which contain v_0, and so there can be no SMID cut-forest spanning all vertices of G_{NS1}. We also note that G_{NS1} does indeed have valid convex liftings, one as shown in Figure 3-16, so this is not a contrived example of a flat terrain.

Next, we construct a convex planar graph G_{NS2} with only triangular faces that also has no SMID paths from one of the center vertices to the outside. As seen in

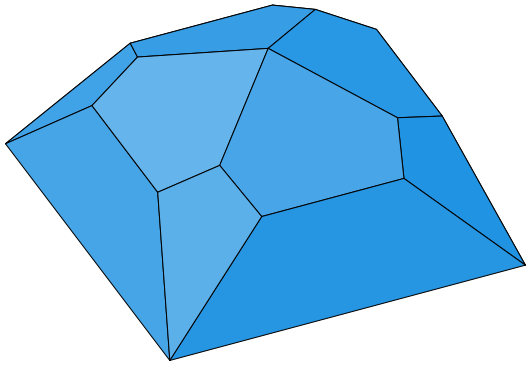
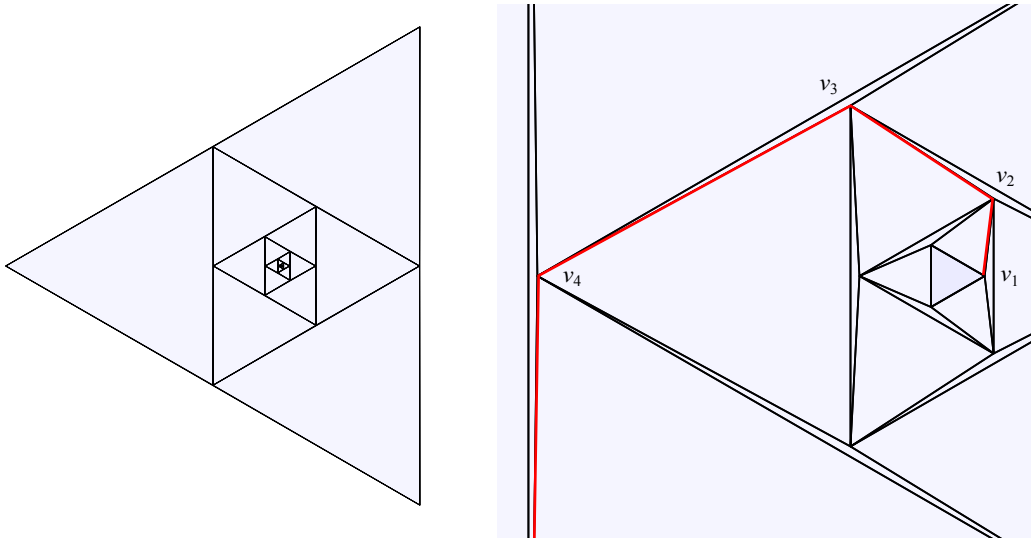


Figure 3-16: Example lifting of G_{NS1}

Figure 3-17, G_{NS2} is constructed by many connected concentric equilateral triangles, where the ith triangle has side length slightly more than twice that of the $(i-1)$th triangle. Then, note that the only path from one of the center three vertices to the boundary which has no acute angles is the one which spirals out, as marked. However, this spiral is not a SMID path since the angle $\angle v_1 v_5 v_6$ is acute.



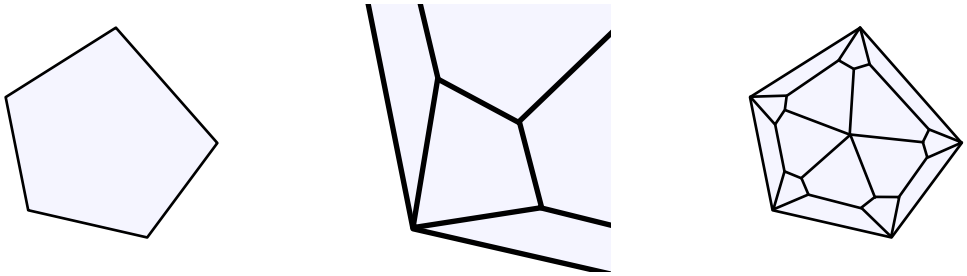
(a) G_{NS2} of only triangular faces, with 7 layers of triangles around the center

(b) Zoomed in on center, spiral path not SMID since $\angle v_1 v_5 v_6$ is acute

Figure 3-17: No SMID path to center projection with only triangular faces

In fact, it is very easy to change any convex polygon p into a convex planar graph

g_{NS} with valid non-zero convex liftings in a similar way to what G_{NS1} looks like by the process shown in Figure 3-18. Essentially, at the interior of each vertex of p, we put a small quadrilateral which is almost a triangle. Then, we connect adjacent such shapes with lines which are parallel or almost parallel to the sides of p. Finally, we connect all such quadrilaterals together at a point in the middle. For reasons similar to why G_{NS1} has no SMID paths from the center vertex to a boundary vertex, neither does this constructed g_{NS}.



(a) Starting convex polygon p

(b) Gadget added at each corner

(c) Resulting g_{NS} with no SMID paths from center

Figure 3-18: Arbitrary convex polygon to no-SMID projection construction

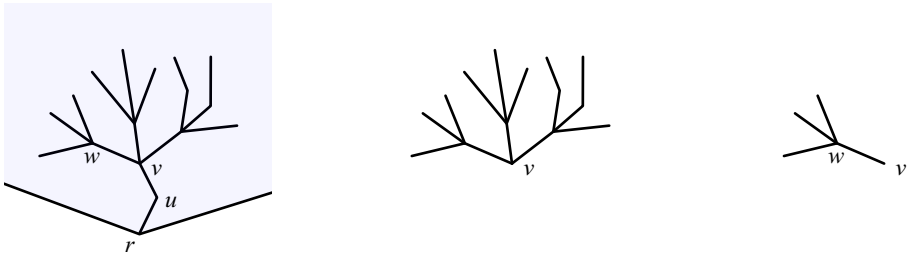
Therefore, this shows that while SMID paths and trees of G represent "universal" unfolding cut-paths and cut-trees of any convex lifting of G, not all projections of almost-flat convex terrains yield such universal unfoldings. For those T, we will need to take into account the convex lifting as well in order to determine whether T contains unfolding cut-paths and cut-trees.

3.3 Unfolding Almost-Flat Convex Terrain

In this section we will look at general unfolding of cut-trees of almost-flat convex terrains. We start with a characterization of what cut-trees unfold by using a first-order approximation and then demonstrate reasons for why such a first-order approximation is appropriate given the almost-flat nature. Finally, we end with a result concerning universal local extensibility of all partial cut-trees.

3.3.1 Tree Definitions

Let us first start by defining some terms which will make it easier to talk about unfolding cut-trees. Let t be a cut-tree of an almost-flat convex terrain T. This means that t has a root r, which is on the boundary of T, and all other vertices of t are interior vertices of T. Now, let a *vertex-subtree* be $t_v = \{w \mid w \in t, v \in (w, \ldots, r)\}$, that is, the vertex-subtree of t defined by the vertex v is the subtree t_v comprised of all vertices w (and connecting edges) such that the path from w to the root r passes through v. Similarly, let a *edge-subtree* or *branch* be $t_e = \{w \mid w \in t, e \in (w, \ldots, r)\}$, where e is an edge of t, and the subtree t_e includes all vertices w where the path from w to the root r passes through e. Note that if a vertex v has neighboring vertices $w_1, \ldots, w_n$ which are farther from the root r by breadth first search than v, then the vertex-subtree t_v is the union of the branches $t_{w_1}, \ldots, t_{w_n}$. Similarly, if a vertex v only has 1 neighboring vertex w farther than v from r, then the vertex-subtree t_v is the same as the edge-subtree $t_{(v,w)}$. Finally, a branch $t_{(v,w)}$ is an edge (v, w) and the connected vertex-subtree t_w. All these terms are summarized in Figure 3-19.



(a) Cut-tree t with root r (b) Vertex-subtree t_v (c) Branch $t_{(w,v)}$

Figure 3-19: Tree, subtree, branch terminology

3.3.2 First-Order Approximation

We will now consider what it means for a cut-tree to unfold in the first-order approximation. As mentioned in previous sections, when a cut-tree is unfolded, it becomes a path, where each angle of the path is increased or decreased slightly depending on the aggregate delta angle at that angle. To make things more clear, let us fix a frame of reference as shown in Figure 3-20: when we talk about unfolding a tree or subtree

t with root r, we assume that r opens "downward," or in the negative y direction. Then, when t is unfolded, let the path be labeled from left to right $(r, v_1, \ldots, v_n, r')$, and we assume the edge (r, v_1) to be fixed, so when an angle $\angle v$ is slightly changed, the side which does not include (r, v_1) is the side which rotates.

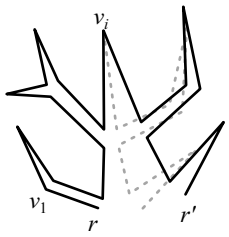
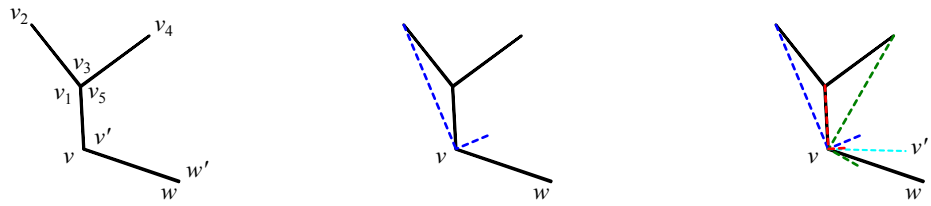


Figure 3-20: Cut-tree orientation, unfolding around v_i rotates side without (r, v_1)

With this in mind, let us look at the effect of the unfolding of a branch $t_{(w,v)}$ on the edge (w, v) (see Figure 3-21). What we want to know is whether unfolding the subtree $t_{(w,v)}$ will cause a local overlap involving the unfolded edges (w, v) and (w', v'). Hence, let us call (w, v) the *outgoing edge*, and we are essentially trying to figure out the effect of unfolding t_v on the right outgoing edge (v', w') assuming we fix the left outgoing edge (v, w).

So, letting the unfolded cut-tree of t_v be the path $(v, v_1, \ldots, v_n, v')$, we unfold by fixing the position of (v, v_1), and then bending, in order, $\angle v_{i-1}v_iv_{i+1} = \angle v_i$ by $D(\angle v_i)\phi$, where $D(\angle v_i)$ is the aggregate angle-delta at $\angle v_i$. Since we are using a first-order approximation, we will not consider the effect of bending $\angle v_i$ on later angles $v_j, j > i$ and consider all the v_i to be fixed — we will show in the next section, Section 3.3.3, why this is valid. Similarly, we assume that all the bending will only make a negligible difference on the orientation of the outgoing edge (v', w'), and instead we are only interested in the direction that all the bending moves the vertex v', to see if it will cause a local overlap with the fixed outgoing edge (v, w).

We will also make a first-order approximation on the opening motion. When $\angle v_i$ is opened by $D(\angle v_i)\phi$, the rest of the path $(v_{i+1}, \ldots, v_n, v')$ rotates clockwise by $D(\angle v_i)\phi$, which means v' rotates clockwise by $D(\angle v_i)\phi$ around v_i. Instead of an actual rotation, we will approximate it by a movement in a straight line in the



(a) Branch $t_{(w,v)}$ (b) First-order unfolding of single vertex (c) Successful unfold since v' is on the left

Figure 3-21: First-order approximation of branch unfolding

direction $\overrightarrow{v_i v'}_\perp$, the direction perpendicular to $\overline{v_i v'}$ in the clockwise direction, by the distance $\sin(D(\angle v_i)\phi)\left|\overline{v_i v'}\right| \approx D(\angle v_i)\left|\overline{v_i v'}\right|\phi$ (Figure 3-21b). This approximation is reasonable since there is definitely a $\phi > 0$ where the difference between $C_0 \sin(C_1\phi)$ and $C_0 C_1 \phi$ is negligible, and similarly for the difference between a rotation and the tangent approximation.

Hence, the overall effect of unfolding t_v on v' is the vector sum

$$\overrightarrow{dv'} = \sum_{v_i \in (v, v_1, \ldots, v_n, v')} \widehat{v_i v'}_\perp D(\angle v_i)\left|\overline{v_i v'}\right|\phi,$$

where $\widehat{v_i v'}_\perp$ is the unit vector perpendicular to $\overrightarrow{v_i v'}$. Then, unfolding t_v does not cause a local overlap on the outgoing edge (v, w) if and only if $\overrightarrow{dv'}$ points away from (v, w) — that is, if (v, w) is parallel to the y axis, that $\overrightarrow{dv'} \cdot \langle 1, 0\rangle > 0$ since we are unfolding with the left side fixed (Figure 3-21c).

One simplification to note here is that instead of calculating the effect of each angle of the tree on v, we can just calculate the effect of each vertex of the tree on v. This is because, each time we consider an angle $\angle wxy$ of vertex x, we add $\widehat{xv'}_\perp D(\angle wxy)\left|\overline{xv'}\right|\phi$ as its effect on v. However, since over the course of traversing the tree in unfolding, we will add every angle of x, we can instead just add, for each vertex x of the tree, $\widehat{xv'}_\perp D(x)\left|\overline{xv'}\right|\phi$ as the aggregate effect of all the angles around x, where $D(x)$ is the aggregate angle-delta at x (which must be negative for interior

vertices because of convexity). Therefore we can also calculate $\overrightarrow{dv'}$ as

$$\overrightarrow{dv'} = \sum_{x \in t_v} \widehat{xv'}_{\perp} D(x) \left|\overline{xv'}\right| \phi.$$

So, the branch $t_{(w,v)}$ unfolds if the subtree t_v unfolds, and the unfolding of t_v does not cause a local overlap in the outgoing edge (v, w).

Next, consider a subtree t_v which is not a branch. This means that t_v is composed of multiple branches $t_{(v,w_1)}, \ldots, t_{(v,w_n)}$, and the unfolding of t_v is the path

$$(v = v^0, w_1, \ldots, w_1', v^1, w_2, \ldots, w_2', v^2, w_3, \ldots, w_n', v^n = v').$$

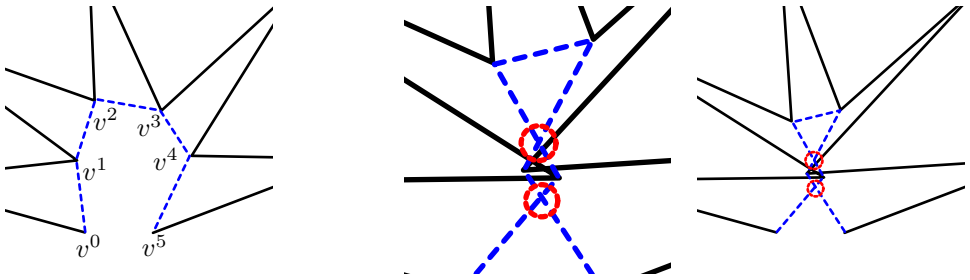
Now, each branch $t_{(v,w_i)}$ causes a displacement $\langle v^{i-1}, v^i \rangle$, so, fixing v^0 at the origin, the unfolding places each v^i at a place in the plane. Each such point v^i represents the tip of an angle $\angle w_i' v^i w_{i+1}$, and what we are worried about is whether two such tips might overlap.

(a) Example overlap (b) Close up of overlap

Figure 3-22: Possible local overlap of tips in a vertex-subtree

Assuming that the subtrees of t_v unfold, the neighboring tips v^i and v^{i+1} will not overlap. However, two non-neighboring tips might still overlap, as shown in Figure 3-22. Note that such non-neighboring tip overlaps are local overlaps since the edges involved are all incident on the vertex v in G. In this case, consider the edges $\overline{v^i v^{i+1}}$: if tip v^i overlaps with v^j, then we see that $\overline{v^{i-1}v^i}$ or $\overline{v^i v^{i+1}}$ intersects with $\overline{v^{j-1}v^j}$ or $\overline{v^j v^{j+1}}$. So, to check for overlaps, we just need to check all of the edges $\overline{v^{j-1}v^j}$ against one another for intersections, as shown in Figure 3-23.

Unfortunately, this misses out on possible overlaps between the extreme tips v^0



(a) Non-overlapping case with vertex labels

(b) Overlapping case, v^4 overlaps with v^1, causing overlaps between $\overline{v^0v^1}, \overline{v^4v^5}$ and $\overline{v^1v^2}, \overline{v^3v^4}$

Figure 3-23: Checking for tip overlaps

and v^n. However, in this case, suppose this subtree t_v is part of a branch $t_{(w,v)}$. Then, if the tips v^0 and v^n overlap, then this means the outgoing edge (w, v) will overlap. So, we will still catch this overlap when we consider the branch $t_{(w,v)}$, which contains t_v.

This leaves the final case of v being the root vertex r, so t_v is actually the entire cut-tree. In this case, since the boundary of G is a convex polygon, it is impossible for the tips v^0 and v^n to intersect in a way that we will not detect with our previous method of checking for intersections of $\overline{v^{j-1}v^j}$.

Hence, to check if a vertex-subtree t_v is valid, we need to check whether each of its branches $t_{(v,w_1)}, \ldots, t_{(v,w_n)}$ are valid by themselves, and that none of the edges $\overline{v^{i-1}v^i}$ intersect, where $\langle v^{i-1}, v^i \rangle$ represents the displacement caused by the unfolding of the branch $t_{(v,w_i)}$. Then, if in addition t_v is part of a branch, then as long as the branch it is part of has no overlaps with the outgoing edge, t_v will be valid as well.

3.3.3 Insignificance of Second-Order Effects

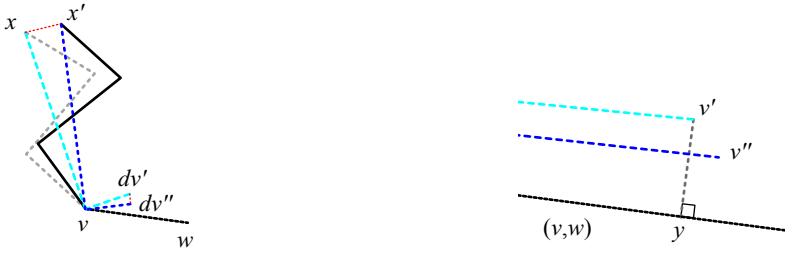
In the previous section we ignored a lot of second-order effects which we will show in this section to be insignificant. Here, we take "second-order effect" to be anything beyond a first-order effect.

First, let us consider the effect of opening one angle by a delta angle on the rest of the tree. So, let the path of vertices we will unfold be $(v_1, \ldots, v_n)$. This means that the angles we are unfolding are $\angle v_1v_2v_3 = \angle v_2, \ldots, \angle v_{n-1}$. Starting by unfolding v_2,

this shifts vertex $v_i, i > 2$, by

$$|v_2 - v_i| \sin(D(\angle v_2)\phi)\widehat{v_2 v_{i\perp}} \approx |v_2 - v_i|\, D(\angle v_2)\phi\widehat{v_2 v_{i\perp}},$$

where $\widehat{v_2 v_{i\perp}}$ is the unit vector perpendicular in the clockwise direction to the vector $\overrightarrow{v_2 v_i}$. This moves $v_3 = (x_3, y_3)$ to $v_3' = (x_3 + C_1\phi, y_3 + D_1\phi)$ for some constants C_1, D_1 depending on the geometry of G and angle-delta D, which means when opening the angle at $\angle v_3'$, later angles will be moved by $dv_i \approx \langle C_1\phi + C_2\phi^2, D_1\phi + D_2\phi^2\rangle$ for appropriate constants. Overall, after unfolding all angles, the position of v_i will be affected by some amount $dv_i = \langle \sum_{j>0} C_j\phi^j, \sum_{j>0} D_j\phi^j\rangle$ for some appropriate constants C_j, D_j (different for each v_i) depending on the geometry of G and angle-delta D. Then, this dv_i in turn causes the contribution of $\angle v_i$ on the tree root v_r to change by some $\left(\sum_{j>0} C_j\phi^j\right)\phi$. Hence, this second-order effect will cause an aggregate change of approximately $C'\phi^2$ in some direction for some constant C'. Hence, by picking a ϕ small enough that this $C'\phi^2$ term is dominated by the $C\phi$ value from the first-order approximation, we do not need to worry about this second-order effect affecting the result of the unfoldability test. Figure 3-24 below shows a simplified version of this concerning a single vertex being affected by second-order effects.



(a) Unfolding branch at v moves x to x', causing deviation in dv'

(b) However, $|v'y| = C\phi$, but $|v'v''| = C'\phi^2$, so pick small ϕ to compensate

Figure 3-24: Second-order effect of shifting vertex positions

Second, on unfolding a vertex-subtree t_v with outgoing edge (v, w), this edge itself might slightly bend due to the delta angle at $\angle v$. This might cause a conflict, since if

(v, w) bends towards the unfolded v', that could cause a local overlap. However, we see that this is a second-order effect since v' itself moves a distance of $C\phi$, for which some amount $C_\perp\phi$ is away from (v, w). This means that the maximum distance (v, w) can move is $\sin(D(\angle v)\phi)C\phi \approx CD(\angle v)\phi^2$ (see Figure 3-25). So, by picking a ϕ small enough, the amount (v, w) moves will be dominated by the amount $C_\perp\phi$ that v' moves away from (or towards) (v, w), so the result of whether an unfolding is valid will not change.

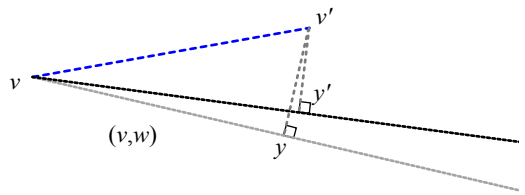


Figure 3-25: Angle-delta at v rotates (v, w). But, since $|vy'| = A\phi$, then $|yy'| = A\phi\sin(D(v)\phi) \approx A'\phi^2$, which is dominated by $|v'y| = C\phi$

These two problems are the reason why, when considering whether an outgoing edge unfolds, we want to make sure that the vertex v' moves strictly away from the fixed outgoing edge (v, w). This is because if v' merely slides along (v, w) — say, if we are unfolding around the point x, and $\angle xvw = \frac{\pi}{2}$ is a right angle — then we do actually need to calculate the second-order effects in order to figure out whether an overlap occurs. By using a more strict validity criteria, we avoid this case by assuming that such cases result in overlaps. Hence, perhaps by calculating the actual unfolding, some unfoldings which we regard as overlapping will actually be non-overlapping, but such results will be quite rare, and this way we make sure that any tree counted as non-overlapping by our first-order approximation will definitely be non-overlapping when fully calculated.

3.3.4 All Partial Edge Cut-Trees Locally Extensible

Up until now, we have only considered "proper" cut-trees — trees with exactly one boundary vertex — and subtrees of such cut-trees. However, we can also think of

subtrees as partial cut-trees. So, let us define a *partial edge cut-tree* of T to be a tree t of only interior vertices of T with a specified root edge (w, v) such that w is a leaf of t and every path from every vertex of t to w passes through the edge (w, v). Now, let us prove our main result in unfoldability:

Theorem 7 (All Partial Edge Cut-Trees Locally Extensible)**.** *Let t be an unfoldable partial edge cut-tree of T with root edge (w, v). Then, t can be locally extended by an outgoing edge from w to become either an unfoldable proper cut-tree of T or a larger unfoldable partial edge cut-tree.*

Proof. Let us start by orienting (w, v) with w at the origin and v on the positive y axis. Then, let unfolding t move w' to (w'_x, w'_y). Since t is unfoldable, it unfolds without overlap, so w' cannot conflict with edge (w, v). This means that $w'_x > 0$, so w' is in the positive x half-plane.

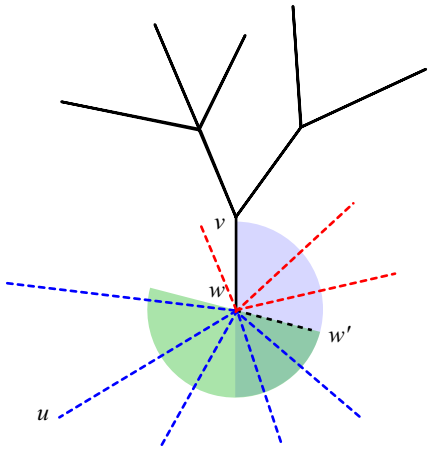


Figure 3-26: Example partial tree t, w' must be in the blue half-space. Extensions (blue) in the green half-space clockwise from w' unfold without overlap while counter-clockwise extensions (red) conflict. By convexity, there must be some valid unfolding extension u

By Section 3.3.2, we see that t with outgoing edge (w, u) unfolds if the unfolded w' does not conflict with this outgoing edge. So, we need an edge (w, u) such that $\langle w, u\rangle \cdot \langle w, w'\rangle > 0$, which means that (w, u) is in the half-space clockwise from (w, w'). As Figure 3-26 shows, since we know that w' is in the positive x half-plane, this means

that the half-space clockwise from (w, w') does not intersect with (v, w). Now, since T is a convex terrain, every face of its projection G is also convex. This means that there must be some neighbor u of w in G where $\angle uww' < \pi$, so w has an adjacent vertex u which is in the half-space clockwise from (w, w').

Finally, we can extend t by this outgoing edge (w, u) to get t', which must unfold without overlap since w' does not conflict with (w, u), and t already unfolds. Therefore, we can extend t locally to t'. If u is another interior vertex, then t' is another unfoldable partial edge cut-tree. Else if u is a boundary vertex, then t' is a proper cut-tree. □

The "locally" criteria in the proof serves to mean that we assume we can extend t at w by any edge other than (v, w) while keeping t a tree. The proof would not work if we remove this condition since it may be the case that all neighbors of w which are in the negative x half-plane are already vertices of t, so we have essentially "trapped" ourselves in a corner, and would not be able to extend t.

3.4 Slice Unfolding

In this section we will consider a more ground-up approach to unfolding, and see what sort of constructions can be shown to be unfoldable using this method. We build on and extend the result of "Cut-Tree Truncation" by Benton et al. [3] for the case of almost-flat convex terrain.

3.4.1 Definitions and Examples

Recall back to Section 2.2 that we can also think of a convex terrain as an intersection of half-spaces, where each face is a linear constraint. In this manner, we can "add" a face by adding such a linear constraint, and "remove" a face by removing such a linear constraint, as Figure 3-27 shows.

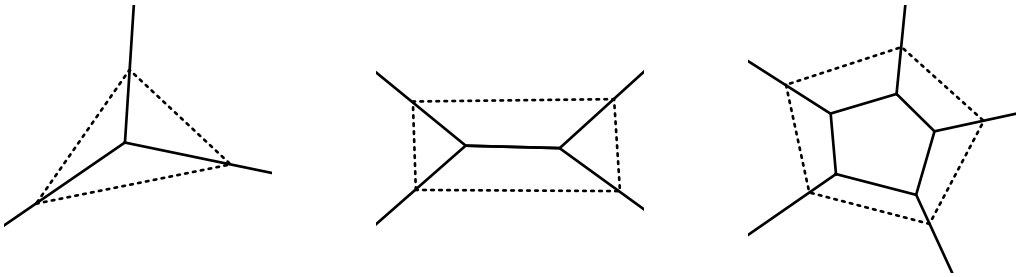
Let a *slice* be the addition of a face by the addition of such a linear constraint. Note that a slice cuts away part of the terrain, and if the original terrain T is almost

(a) Simple pyramid (b) Top sliced off

Figure 3-27: Example slice operation

flat, then this removed portion R will also be an almost-flat terrain.

Figure 3-28 shows a few of the many possibilities for the removed terrain R: R can contain just one vertex, one or more edges, or one or more faces. While we will give a general analysis on slice unfolding, later on we will be focusing on the first case, when R contains exactly one interior vertex. Let such a slice be a *vertex-slice* — it is the same as a degree-n truncation as mentioned in [3], where n is the degree of the interior vertex.



(a) Vertex-slice (b) Edge slice (c) Face-slice

Figure 3-28: Different types of slices

3.4.2 General Slice Unfolding

The general method of slice unfolding is to start with a simple terrain with a simple unfolding cut-tree, and then using various slices paired with modifications to the cut-tree, achieve an unfolding cut-tree for a more complicated terrain. In more detail, say we have a terrain T with unfolding glue-tree C and unfolded net N. Then, we make the slice s which yields T' and removed portion R when creating the new face

f. Now, each face of R is also a part (or all) of a face of T. Suppose we remove all such parts of faces from T, which creates some free space and new edges E_R. If we can attach f to an edge in E_R without causing any overlaps, then we have created an unfolding glue-tree of T'. Figure 3-29 shows three such steps in slice unfolding starting from a triangular pyramid.

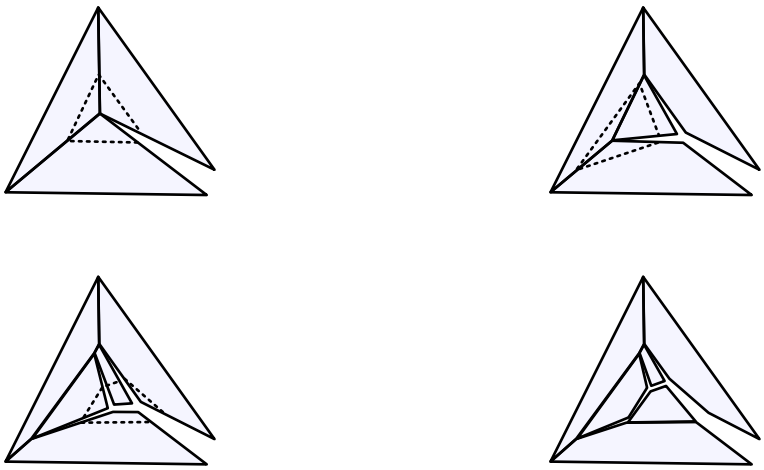


Figure 3-29: Example general slice unfolding from triangular pyramid with two vertex-slices and an edge-slice

Let us also define f_G to be the projection of f in T' (and hence T), and f_N to be the polygon formed by connecting the edges of f in N. Then, if we consider an almost-flat T, then the question of overlap becomes a question of whether f will intersect with f_N when f is attached to some appropriate edge of f_N. Now, there are two ways the edges of f_G can move to becomes the edges of f_N. First, an edge can remain connected, but rotate, that is, an angle of f_G can change based on the angle-deltas of T. Second, an edge of f_G can disconnect from neighboring edge(s) and move slightly (both translation and rotation). Unfortunately, this movement in the second case is not purely expansive: as shown in Figure 3-30, depending on the cut-tree and slice, it is possible for edges and vertices of f_G to move closer to one another when moving to their positions in f_N.

In the end, it becomes a matter of whether the movement of f_G to f_N based on the cut-tree producing N causes an overlap with where f is attached in the unfolding

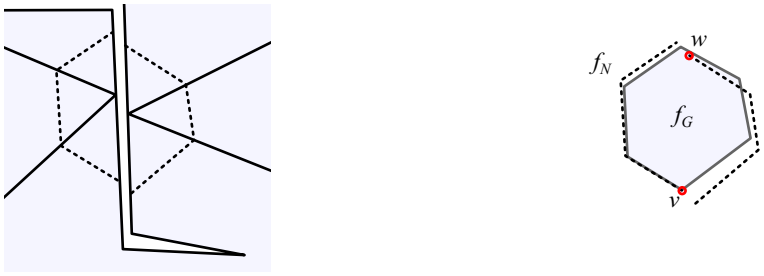


Figure 3-30: Example vertex-slice with non-expansive f_N motions. The marked vertices v and w of f_N actually get closer, so there is nowhere to attach f_G without overlap

of T'.

The approach mentioned above is a very constructive method, and is also the one used in [3]. However, given a terrain T, we can also use principles of slice unfolding by simplifying T to a simpler terrain T' through removing faces. As shown in the definitions, removing a face results in another convex terrain with one less face. So, to find an unfolding of T, instead of trying to find a cut-tree, we can instead try to find a subset S of the faces of T and a numbering of the rest of the faces of T such that, if we start from a trivially unfoldable terrain constructed of just the faces of S, and we slice unfold the faces of T as numbered, then each slice of face f will have a valid non-overlapping attachment for f. While it is not clear whether looking for unfoldings by this method is easier than just looking for unfolding cut-trees, at least the prospect of simplifying T by removal of faces seems to be likely useful.

3.4.3 Empty Sector Property

For slice unfolding to work, Benton and O'Rourke [3] made use of an "empty sector property" for a cut-tree C of a polyhedron P. This is the property that, for every leaf edge e of C, the circular sector between the unfolded edges e, e' in the net of P is empty. While not all cut-trees have this property for general polyhedra, when considering almost-flat convex terrain, all cut-trees which we recognize as unfolding based on our first-order approximation from Section 3.3.2 have the empty sector property.

First, consider a leaf edge e which is adjacent to only one other edge d. In this

case, for this cut-tree to unfold, we need that the angle between e and d be obtuse (see Figure 3-31). Hence, it would be impossible for the edges d, d' to overlap the arc from e to e'.

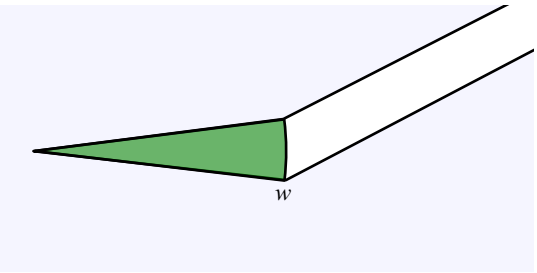


Figure 3-31: Simple leaf trivially has empty sector property since $\angle w$ must be obtuse

Then, consider a leaf edge e which is adjacent to several other edges. Referring to Figure 3-32, let $e = (w, v)$, and let v be split into $v_0, \ldots, v_n$ on unfolding, where $e, e' = (w, v_i), (w, v_{i+1})$. Now, let x be the point of intersection of the perpendiculars to e and e'. Then, the triangle $\triangle v_i x v_{i+1}$ contains the arc from e to e', and is almost degenerate since $\angle v_i v_{i+1} x = \angle v_{i+1} v_i x = \frac{1}{2}\angle v_i w v_{i+1} \approx \frac{1}{2} A\phi$ where $A < 1$ is the aggregate angle-delta at w. This means that the height of this triangle is

$$\left(\frac{1}{2}|e| A\phi\right) \sin\left(\frac{1}{2}A\phi\right) \approx \frac{1}{4}|e| A^2\phi^2.$$

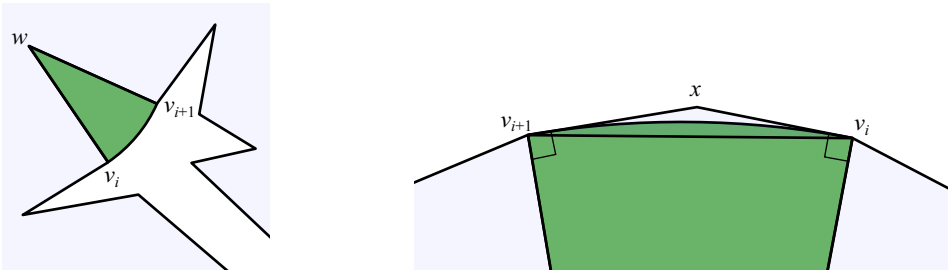


Figure 3-32: Complex leaf case: $\angle v_i x v_{i+1} = \frac{1}{2} D(w)\phi$, so triangle $\triangle v_i x v_{i+1}$ has height $C\phi^2$, which is second-order

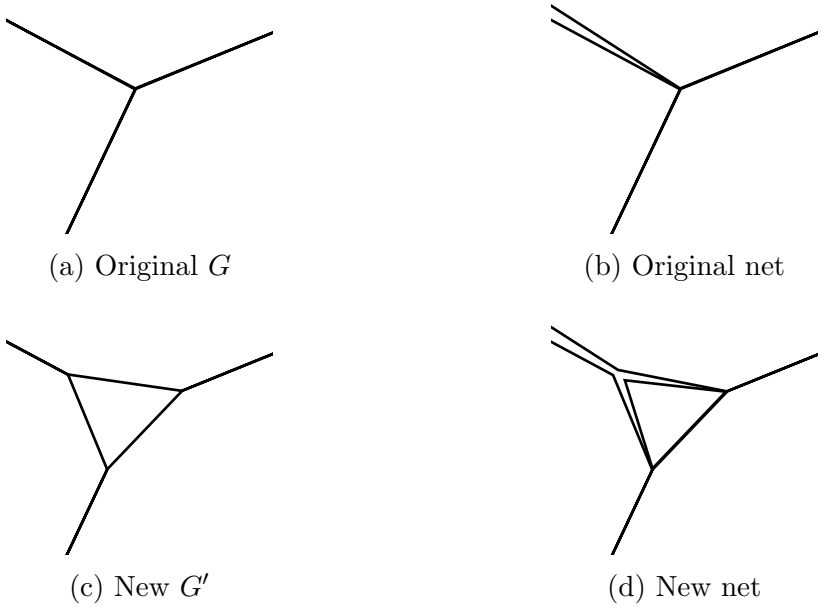
Now note that, by our first-order definition for non-branch subtrees, for this cut-tree to unfold, $\overline{v_i v_{i+1}}$ must not intersect with any other line $\overline{v_j v_{j+1}}$. However, since the height of triangle $\triangle v_i x v_{i+1}$ has a ϕ^2 factor, if none of the lines $\overline{v_j v_{j+1}}$ intersect $\overline{v_i v_{i+1}}$, then we can find a $\phi_0 > 0$ such that none of them intersect the triangle $\triangle v_i x v_{i+1}$

either if $\phi < \phi_0$. Since the triangle contains the arc between e and e', again we see that the sector is empty.

3.4.4 Triangular and Quadrilateral Vertex-Slices

The aforementioned reference [3] demonstrated that, for any unfolding convex polyhedron with the open sector property, if the slice is a vertex-slice on a leaf vertex of the cut-tree such that the removed portion is a triangular pyramid, then the resulting modified polyhedron also have the property and unfolds. Let us call a vertex-slice of a leaf vertex of an unfolding cut-tree C of a convex terrain T *valid* if it produces a new terrain T' and unfolding cut-tree C'. Let us now examine all possible vertex-slices and see which ones are valid in the case of almost-flat convex terrains.

Since we are dealing with vertex-slices of leaf nodes of cut-trees, the removed portion R is a pyramid and it is unfolding along exactly one edge, so the movement of f_G to f_N is strictly a bending motion of the edges. Furthermore, since T is convex, all edges are strictly opening.



(a) Original G (b) Original net

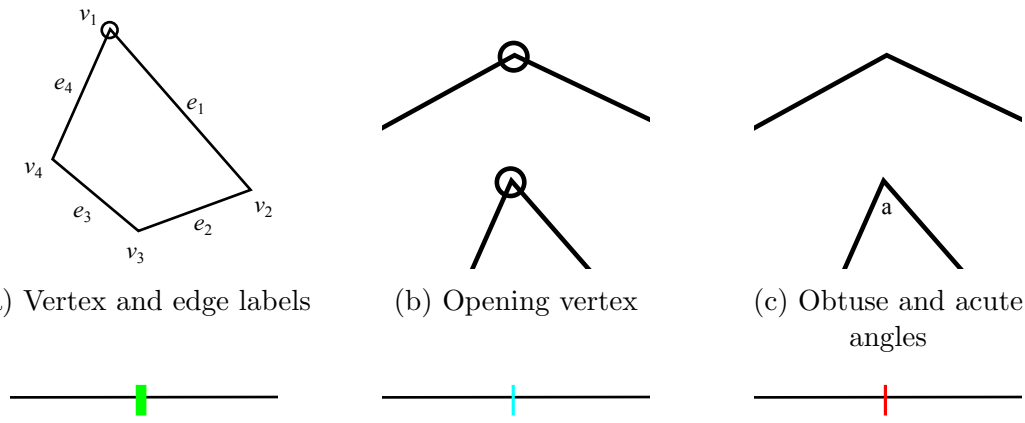
(c) New G' (d) New net

Figure 3-33: A triangular vertex-slice

We start by demonstrating the result from [3] in this context: if f is a triangle

$\triangle ABC$, and the cut-tree C used to be through point A, then we can simply glue the triangle f to $\overline{BC}$ opposite $\angle A$, and add the edges $\overline{AB}$ and $\overline{AC}$ to the cut-tree (see Figure 3-33). Since we know that the rest of the cut-tree, which connects to A, unfolds, and each of the new edges unfold trivially, this new cut-tree unfolds.

Now let us extend this result by considering all possible cases of quadrilateral vertex-slices. We break the casework for the quadrilateral f into three sections since f can have one, two, or three acute angles. Also, let use define $f = v_1 \cdots v_4 v_1 = e_1 e_2 e_3 e_4$ and that the original cut-tree connects to v_1, which we call the *opening*. We want to see whether it is always possible to form a new cut-tree C' of T' by adding all of the edges of f but one, which is the edge we "attach" f to.



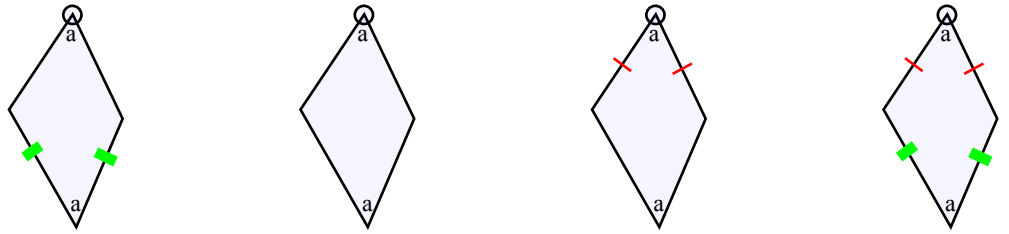
(a) Vertex and edge labels (b) Opening vertex (c) Obtuse and acute angles

(d) Definitely unfolds (e) Possibly unfolds (f) Never unfolds

Figure 3-34: Key for picture categorizations of quadrilateral vertex-slices

Each of these cases will be further split into subcases for where the acute and obtuse angles are in relation to the opening v_1, and for each subcase, referring to the key provided in Figure 3-34, we will create a diagram as in Figure 3-35 for organizing which attachment edges "definitely unfold," "possibly unfold," or "never unfold," labeled respectively as "unfold," "possible," and "overlap." This diagram shows the case for when f has two nonadjacent acute angles, where v_1 is acute. Then, attaching to e_2 or e_3 will always yield an unfolding, since this produces two new cut-paths: a cut-path of a single edge, which always unfolds, and a cut-path of two edges joined by an obtuse angle, which also always unfolds. On the other hand, attaching to e_1 or

e_4 will always cause an overlap, since a cut-path containing v_3, which is acute, will be created. Then, there are no edges marked as "possibly unfolding" since attaching at each edge either definitely unfolds or never unfolds. Finally, for simplicity, we will merge it all into one figure under "merged."



(a) Unfolds (b) Possible (c) Overlaps (d) Merged

Figure 3-35: Example picture characterization of attachment edges for a quadrilateral vertex-slice

We will not give as indepth an analysis beyond the merged picture categorization for the cases, but they all follow the following three rules:

1. A cut-path of one edge, or two edges joined by an obtuse angle, always unfolds.
2. A cut-path containing an acute angle never unfolds.
3. A cut-path containing three or more obtuse angles possibly unfolds.

The reason for the third rule is that even if a cut-path has all obtuse angles, it is not necessarily even a WMID path, as Figure 3-36 shows.

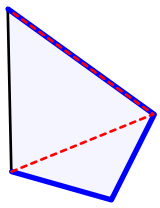
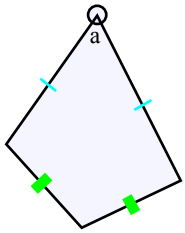
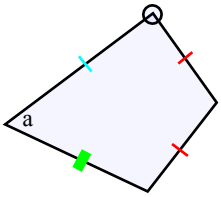


Figure 3-36: Non-WMID path (blue) with only obtuse angles — marked angle (red) is acute

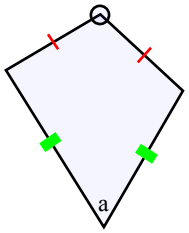
For the one-acute-angle case, as shown in Figure 3-37, there are three subcases: the acute angle can be either v_1, v_2, or v_3 (since v_2 and v_4 are the same by symmetry).



(a) Acute v_1



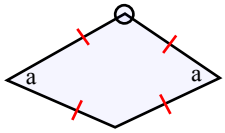
(b) Acute v_2



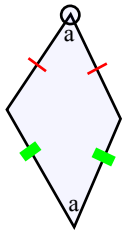
(c) Acute v_3

Figure 3-37: Characterizations of quadrilateral vertex-slice with 1 acute angle

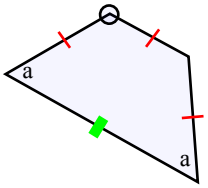
For the two-acute-angle case, as shown in Figure 3-38, there are four subcases: the acute angles can be neighboring or not, and v_1 can be acute or not.



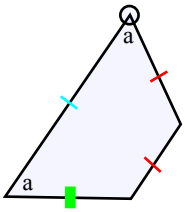
(a) Acutes apart, obtuse opening



(b) Acutes apart, acute opening



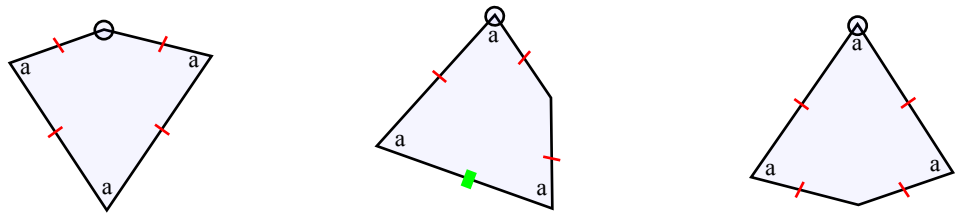
(c) Neighboring, obtuse opening



(d) Neighboring, acute opening

Figure 3-38: Characterizations of quadrilateral vertex-slice with 2 acute angles

For the three-acute-angle case, as shown in Figure 3-39, there are three subcases: the obtuse angle can be either v_1, v_2, or v_3.



(a) Obtuse v_1 (b) Obtuse v_2 (c) Obtuse v_3

Figure 3-39: Characterizations of quadrilateral vertex-slice with 3 acute angles

3.4.5 General Vertex-Slices

Now let us consider the general cases by casework on the number of acute angles in f. Since f has to be convex, we see that it can have anywhere from 0 to 3 convex angles. Again, let use define $f = v_1 \cdots v_n v_1$ and that the original cut-tree connects to v_1, and our goal is to find a valid attachment edge for f. We will use a similar picture categorization as before, where we represent general slices as shown by Figure 3-40 while referring to the key in Figure 3-41.

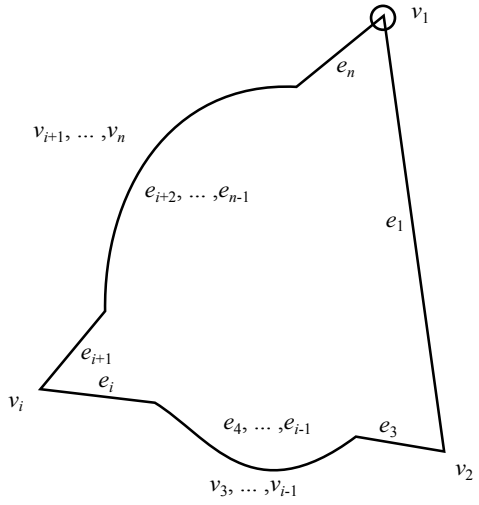


Figure 3-40: General slice with three acute angles

While we are only considering convex faces, the pictures will be much easier to understand if we sometimes draw them as non-convex, and we may do so for clarity. In addition, since we are always dealing with an unknown number of obtuse angles, we can only categorize them into "possibly unfold" and "never unfold," labeled re-

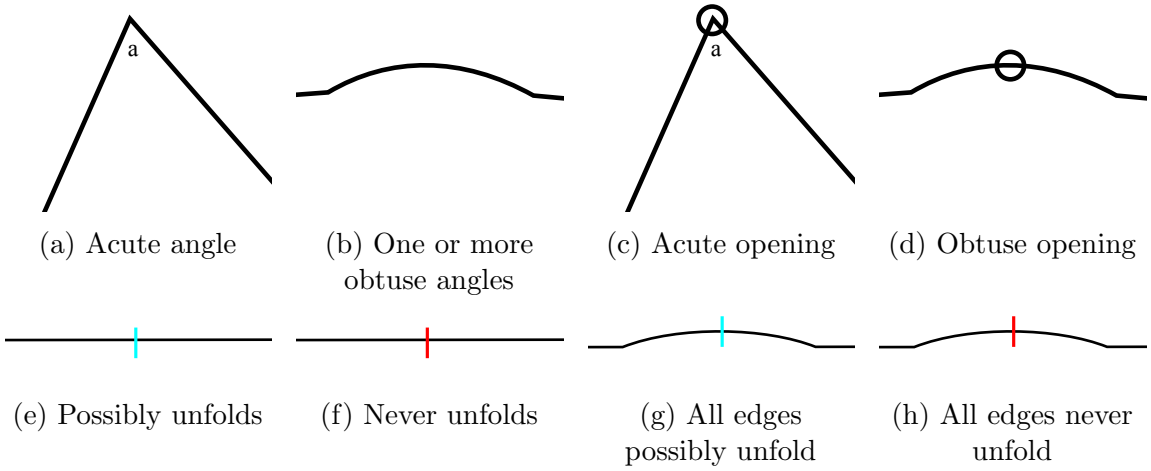


Figure 3-41: Key for picture categorizations of general vertex-slices

spectively as "unfold" and "overlap."

For the zero-acute-angle case, every angle of f is obtuse, thus there is only one subcase, as shown in Figure 3-42, where attaching to any edge yields a possible unfolding.

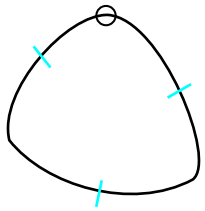
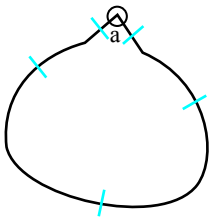


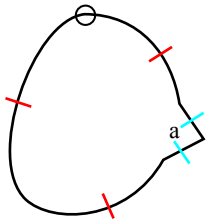
Figure 3-42: Characterizations of general vertex-slice with 0 acute angles

For the one-acute-angle case, as shown in Figure 3-43, there are two subcases: the opening can be either at the acute angle or at an obtuse angle.

For the two-acute-angle case, as shown in Figure 3-44, there are four subcases depending on whether the acute angles are adjacent or not and whether the opening is at an acute angle or not.

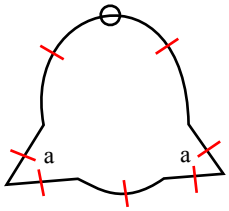


(a) Acute opening

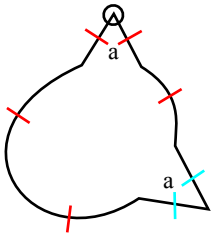


(b) Obtuse opening

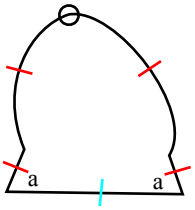
Figure 3-43: Characterizations of general vertex-slice with 1 acute angle



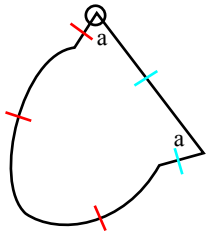
(a) Acutes apart, obtuse opening



(b) Acutes apart, acute opening



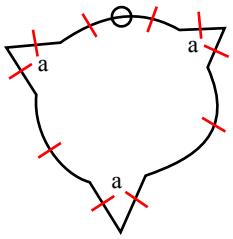
(c) Neighboring, obtuse opening



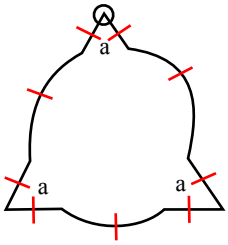
(d) Neighboring, acute opening

Figure 3-44: Characterizations of general vertex-slice with 2 acute angles

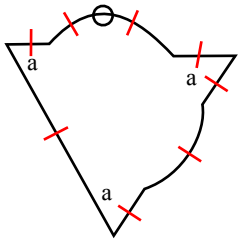
Finally, for the three-acute-angle case, as shown in Figure 3-45, there are three groups of subcases: either none of the acute angles are adjacent, two of the acute angles are adjacent, or all of the acute angles are adjacent. If none of the acute angles are adjacent, the opening can be either at an acute angle or an obtuse angle. Then, if two of the acute angles are adjacent, the opening can be either at the single acute angle, the pair of acute angles, or an obtuse angle. Lastly, if all of the acute angles are adjacent, the opening can be either at the middle acute angle, a side acute angle, or an obtuse angle.



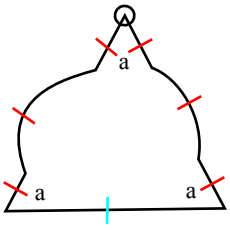
(a) Acutes apart, obtuse opening



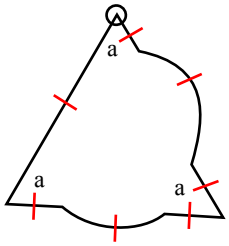
(b) Acutes apart, acute opening



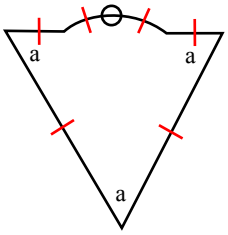
(c) Two neighboring, obtuse opening



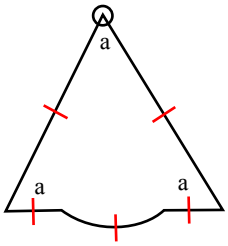
(d) Two neighboring, single acute opening



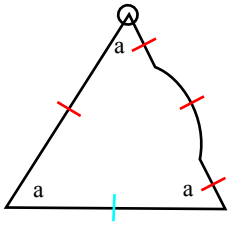
(e) Two neighboring, pair acute opening



(f) All neighboring, obtuse opening



(g) All neighboring, center acute opening



(h) All neighboring, side acute opening

Figure 3-45: Characterizations of general vertex-slice with 3 acute angles

Overall, we see that depending on the shape of the sliced face f, the picture categorizations above show which f can possibly be attached at which edges to yield a new non-overlapping unfolding.

Chapter 4

Computational Search Techniques

In this chapter we will go over a few algorithms we used in our computational experiments, the results of which are in the next chapter.

4.1 Generating Convex Terrain

We start with some simple algorithms for generating convex terrain. While it is easy to get a convex polyhedron by generating random points in space and using a 3D convex hull algorithm, this is suboptimal for convex terrains for a few reasons. First, there is no control over just how many points are actually on the hull. If more points are needed, adding more random points does not always help — a new random point may well either lie in the current convex hull or actually reduce the number of points in the convex hull. Second, a convex polyhedron is not easily convertible to a convex terrain. It is not immediately obvious which faces are on the "bottom" and hence need to be removed from a polyhedron to get a terrain.

4.1.1 Spherical Liftings

First, let us consider spherical liftings. A spherical lifting is a lifting of planar points to a sphere. Suppose we have planar points in the unit circle around the origin, then we raise the point (x, y) to the point $\left(x, y, \sqrt{1-(x^2+y^2)}\right)$. Note here that we do

not necessarily have to lift to a half-sphere: we can also lift a point in the unit circle to a sphere with radius $r > 1$ by lifting to $\left(x, y, \sqrt{r^2 - (x^2 + y^2)} - \sqrt{r^2 - 1}\right)$ — while we will not add this parameter to the algorithms below, keep in mind that it is very simple and possible to do so. A nice property of a spherical lifting is that, since each point on a sphere is the farthest point in that direction from the center of the circle, the convex hull of a spherical lifting contains every single point.

However, even so, it is still not that simple to figure out which faces are on the bottom, so let us start with Algorithm 1, a simpler algorithm which generates points inside a regular m-gon inscribed in a unit circle raised to a unit sphere. Then, we add the m-gon and take the convex hull. Finally, to get the convex terrain, we merely remove all faces which consist of only points of the m-gon.

Algorithm 1 n point spherical lifting in m-gon

SphereLiftMGon n, m

```
 1  Seed random number generator
 2  l ← empty list
 3  for i = 1 to n
 4        (x, y) ← point in circle of radius cos (1/m)π around origin
 5        Append (x, y, √(1 − (x² + y²))) to l
 6  for j = 0 to m − 1
 7        Append (cos (2j/m)π, sin (2j/m)π, 0) to l
 8  C ← ConvexHull3D(l)
 9  for face f in C
10        if all vertices of f have z-coordinate 0
11              Remove f from C
12  return C
```

This algorithm works very well and always produces convex terrains T of exactly n interior points and m boundary points, with the additional property that every boundary point of T is in the xy plane, and some examples can be seen in Figure 4-1. However, the result is not truly random. For small m, since we took the shortcut of generating points in the inscribing circle of the m-gon for simplicity, the points do not fill the m-gon. Meanwhile, for large m compared to n, the boundary m-gon

greatly affect the structure of the resulting terrain. Despite these issues, this is a very fast algorithm since it only requires one application of the 3D convex hull algorithm, and assuming we pick a large n and not too large m, the center of the terrain should contain all the complexities and randomness of any spherical lifting.

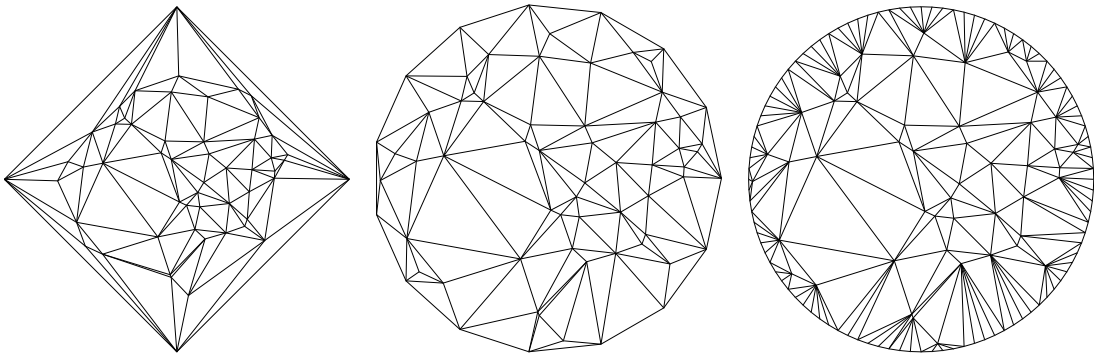
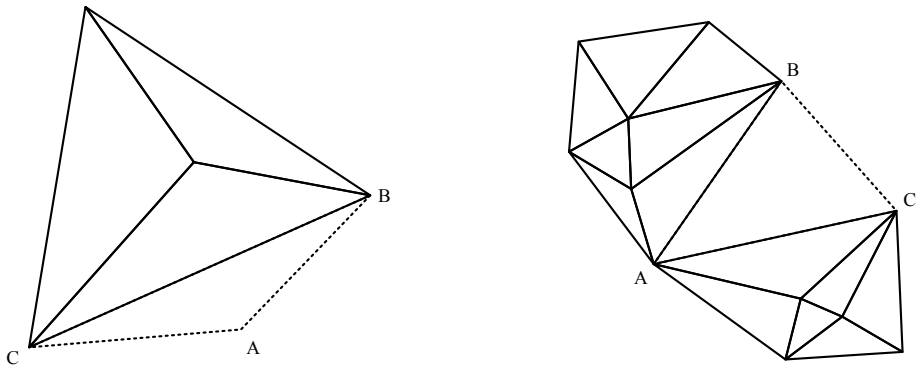


Figure 4-1: Spherical liftings of the same 50 random points in a 4-gon, 15-gon, and 100-gon

To do better, let us look at how we can create a terrain out of a spherical lifting of random points $P = \{p\}$ in the unit circle without a predefined boundary. First, note that the vertices in the 2D convex hull of P are also the boundary vertices of the convex terrain of the "top" faces of the 3D convex hull of P. Then, also note that every "bottom" face of the 3D convex hull of P has to contain only vertices from the 2D convex hull of P. So, we can proceed as before by generating the 3D convex hull C of P, then removing all faces of C composed of only points from the 2D convex hull of P to give us the convex terrain T. However, there are a few side-cases we need to deal with when a "top" face of the 3D convex hull is composed of only vertices from the 2D convex hull. One problem with such P is that our algorithm will remove such faces and leave either disconnected points or multiple terrains joined at vertices, as shown in Figure 4-2. But a bigger problem is that, even if we were to not remove such faces, we see that the resulting terrain is not a simple terrain. We really want simple terrains instead of complex terrains since unfolding a complex terrain is really just unfolding multiple simple terrains. In such cases, depending on how disconnected the graph is, we may either add additional points or just start over with n new points without reseeding the random number generator, as shown in the

pseudocode of Algorithm 2.



(a) Since $\triangle ABC$ is composed of boundary vertices, it is removed, but that disconnects vertex A

(b) Removing $\triangle ABC$ causes several problems, including non-convexity

Figure 4-2: Problematic side-cases for general spherical liftings

Algorithm 2 General n point spherical lifting

SphereLiftMGon n

```
1  Seed random number generator
2  l ← empty list
3  C ← empty polyhedron
4  while |l| < n
5       for i = |l| to n − 1
6            (x, y) ← point in circle of radius cos (1/m)π around origin
7            Append (x, y, √(1 − (x² + y²))) to l
8       C ← ConvexHull3D(l)
9       C2 ← ConvexHull2D(l)
10      for face f in C
11           if all vertices of f in C2
12                Remove f from C
13      for v ∈ C2
14           if |AdjacentEdges(v)| > |AdjacentFaces(v)| + 1
15                l ← empty list // start over without reseeding random
16                Continue while loop from line 4
17           elseif |AdjacentEdgesv)| = 0
18                Remove v from l
19 return C
```

Figure 4-3 shows some example terrain from this method. While this produces more random convex terrain without the m-gon of the previous algorithm, it requires finding multiple 3D and 2D convex hulls, and might loop an unbounded number of times depending on the random points selected. However, in practice it works decently fast even for large n.

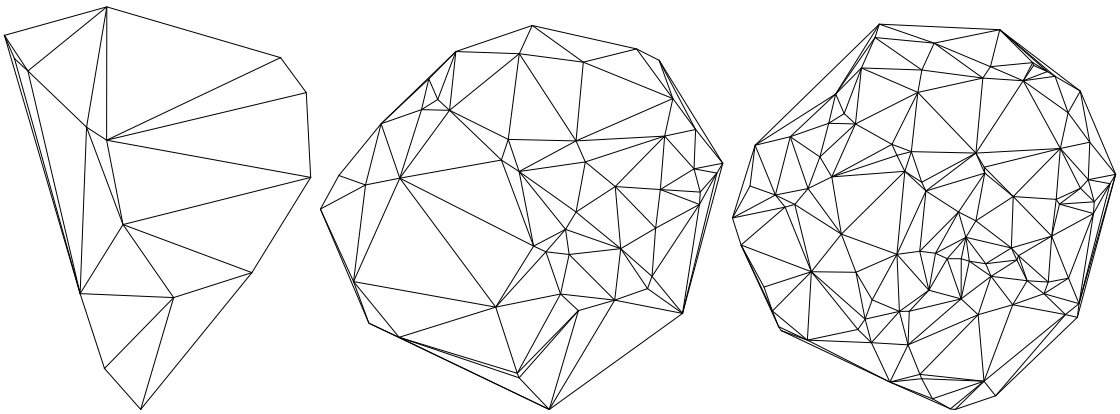


Figure 4-3: General spherical liftings of 15,50, and 100 points from the same random seed

4.1.2 Convex Functional Liftings

Generalizing from spherical liftings from a circle, we can generate points in a convex shape S and lift them to a convex function $f(x, y)$. The procedure for doing this is similar to Algorithm 2, except that instead of generating random points in a unit circle, we generate random points in S, and instead of lifting to $\left(x, y, \sqrt{1-(x^2+y^2)}\right)$, we lift to $(x, y, f(x, y))$. However, similar to the circle case, we still need to calculate convex hulls and we still have no control of exactly what faces and graph we will get.

4.1.3 General Convex Liftings

Another way to generate a convex terrain is to start with a convex planar graph G and take it to a convex lifting. Note first that a convex terrain has the property that every interior edge is a "mountain edge" by origami terms. Then by the Maxwell-Cremona Theorem [7], if we treat the planar graph as a tensegrity, this corresponds

to an equilibrium stress where all the interior edges have positive stress. Such a stress can be converted into a lifting, one which we know to be convex and also valid in terms of planarity of each face.

How we actually go about doing this is by constructing and solving a LP, which has the following variables:

Heights — v_z, the height of vertex v

Face Variables — f_x, f_y, f_z, representing the face f as all points (x, y, z) which satisfy

$$f_x x + f_y y + f_z = z$$

Stress — w_e, representing the stress on the edges e

Using these variables, we make the following constraints:

Face Equations — a constraint for each vertex of each face to ensure planarity of vertices of each face

$$\forall f, \forall v \in f, f_x v_x + f_y v_y + f_z = v_z$$

Stress Equations — a constraint to calculate stress from the faces f_1, f_2 incident on interior edge e, where $e_\perp$ is the 2D unit vector pointing from f_1 to f_2

$$\forall e, f_1, f_2, \langle f_{1x}, f_{1y} \rangle - \langle f_{2x}, f_{2y} \rangle = w_e e_\perp$$

Convexity — a constraint to maintain convexity by making all stresses positive

$$\forall e, w_e > 0$$

Height Limitation — a constraint to limit the heights to the range $[0, 1]$, to make sure the solution is not unbounded

$$\forall v, 0 \leq v_z \leq 1$$

However, this LP is very much underdetermined: every planar graph G which has valid convex liftings have infinitely many planar liftings. Conceptually, we want all maximal solutions of this LP, that is, every vertex of the convex polytope which makes up the solution space of this LP, which gives us a set of convex liftings of G which span the space of all convex liftings of G. However, this will probably be an exponential time algorithm, since there will likely be exponentially many such maximal solutions. Instead, to generate a random lifting, we can solve the LP multiple times, each time with a randomly generated objective function seeking to maximize $\sum_v v_z r_v$ for random r_v. Then, we can take a random mix of the resulting convex liftings to produce a random convex lifting of G.

4.2 Testing Tree Validity

In Section 3.3.2 we detailed how to test, under a first-order approximation, whether subtrees unfold without overlap. We can directly convert the process to an algorithm for checking unfoldability of cut-trees. First, however, let us define a tree datastructure: Let t_v be a tree (or subtree) rooted at v. Then, $\text{Child}(t_v, e) = \{t_{w_1}, \ldots, t_{w_k}\}$ is the set of vertex-subtrees of t_v of vertices connected to v in t_v in clockwise order from edge e. Similarly, $\text{Child}(t_v)$ for a boundary vertex v returns the same result as $\text{Child}(t_v, (v, w))$, where w is the boundary vertex clockwise of v. Also, if (v, w) is an edge of t_v, then let $\text{Tree}(v, t_w)$ be the branch or edge-subtree $t_{(v,w)}$. Also, if T represents the terrain, then let $T = (G, H) = (G, A)$ be the convex lifting and angle-delta representations of T - A can be easily calculated from $T = (G, H)$ using the formula presented in Section 2.5.1. Finally, let us define two helper functions: $\text{OutgoingEdgeValid}(t_v, e, T)$ calculates if unfolding t_v will cause an overlap on the outgoing edge $e = (v, w)$, and $\text{Displacement}(t_w, v)$ calculates the 2D displacement of v from unfolding t_w. Using these, Algorithm 3 gives the pseudocode for determining whether a cut-tree unfolds without overlap under a first-order approximation

Algorithm 3 Recursive algorithm for testing tree validity. The input t_v is a tree rooted at v, and the optional input e is the parent edge of v if t_v is a subtree from a recursive call.

TreeValidRecursive $t_v, T, e = null$

```
 1  if |Child(t_v)| = 1, Child(t_v) = t_w // t_v is a branch
 2      result ← TreeValidRecursive(t_w, T, (w, v))
 3      if result not valid
 4          return not valid
 5      elseif e = null // v is a boundary vertex
 6          return valid
 7      else // v is an interior vertex
 8          return OutgoingEdgeValid(t_v, e, T)
 9  else // t_v is not a branch
10      // list of positions of v^j in the unfolded tree
11      vpos ← list of points, size |Child(t_v)| + 1, 0 indexed
12      vpos[0] ← (0, 0)
13      for i = 1 to |Child(t_v)|
14          if TreeValidRecursive(t_{w_i}, T, (w_i, v)) not valid
15              return not valid
16          else
17              vpos[i] ← vpos[i − 1] + Displacement(t_{w_i}, v)
18          // now, test for conflicts between branches
19          for i = 0 to |vpos| − 1
20              for j = i + 2 to |vpos| − 1
21                  if (vpos[i], vpos[i + 1]) intersects (vpos[j], vpos[j + 1])
22                      return not valid
23              if e = null // v is a boundary vertex
24                  return valid
25              else // v is an interior vertex
26                  return OutgoingEdgeValid(t_v, e, T)
```

4.3 Simple Path Unfolding Algorithm

While Section 3.2.5 showed that not every G contains a SMID path from ever vertex to the boundary, it remains an interesting question whether it is always possible to find an unfolding cut-path from ever vertex of an almost-flat convex terrain T to the boundary. So, building off of SMID paths, we made an algorithm which tries to find paths for each vertex by adding edges in order of straightness from the already constructed paths, and backtracking when the partial cut-path fails to unfold (see

Algorithm 4).

Algorithm 4 Algorithm for finding an unfolding cut-path of T from v

FindCutPath v, T

```
1  // this array of arrays stores possible adjacent edges from each vertex
2  vp ← empty array of arrays
3  Append the array of all adjacent vertices of v to vp
4  lvl ← 0
5  path ← (v) // 0 indexed list of vertices in path
6  while lvl > 0 or |vp[0]| > 0
7      if |vp[lvl]| = 0
8          Pop vp[lvl]
9          Pop path[lvl]
10         lvl −−
11         Continue while loop from line 6
12     if lvl = 0
13         Remove random vertex w from vp[0]
14     else
15         Remove w from vp[lvl] such that the
   angle ∠path[lvl − 1]path[lvl]w is closest to π
16     if w ∈ path
17         Continue while loop from line 6
18     if OutgoingEdgeValid(path, (path[lvl], w)) valid
19         Append w to path
20         Append array of all adjacent vertices of w to vp
21         lvl ++
22 return path if it is a valid path to boundary
   otherwise return null or failure
```

While it is true that this algorithm could possibly enumerate all paths from v to boundary vertices, in practice it is quite fast and does not backtrack often when used on random spherical liftings.

4.4 Cut Forest Generation

Finally, we will give some algorithms for generating spanning cut-forests to test for unfoldability of a given terrain T. After introducing a basic brute-force enumeration algorithm, we will give a few heuristics creating cut-forests which we think will likely

unfold.

4.4.1 Brute-force Enumeration of all Forests

Let us start with an algorithm for a brute-force enumeration of all spanning forests. Let $T = (G, H)$ be an almost-flat convex terrain, then let Interior(G) be the set of interior vertices of G, and Exterior(G) be the set of exterior or boundary vertices of G. Now, if $|\text{Interior}(G)| = n$, then any spanning cut-forest of G has exactly n edges. Also, let Neighbor(v) be the set of all neighbors of vertex v.

The intuition behind Algorithm 5 is to remember the order edges are added onto the partial forest, and keep track of the pool of possible edges at each each part of the partial forest. This means that if the array *forest* contains the edges of the forest added in that order, then *pedges* will be an array such that *pedges*$[i]$ contains a list of possible edges to try for the forest consisting of just the edges *forest*$[0], \ldots,$ *forest*$[i]$. So, we extend a partial forest of i edges by removing an edge from the possible edge pool *pedges*$[i]$; this adds new possible edges, so we add those new possible edges to a copy of *pedges*$[i]$ and set it as *pedges*$[i+1]$. Then, when *pedges*$[i]$ is empty, we backtrack to *pedges*$[i-1]$ and proceed from the possible edges there.

There are a few things to note about this algorithm. First, every cut-forest is enumerated exactly once. This can be seen from line 12 — when an edge is added to a partially complete forest f_{partial}, it is removed from the pool of possible edges from then on. So, after all cut-forests containing f_{partial} and that edge are enumerated, then all cut-forests containing f_{partial} and not containing that edge will be enumerated. Hence, no cut-forest is enumerated twice.

This algorithm is presented as the simplest spanning forest enumerator, but it can be inefficient. For instance, it is possible for all edges connecting to a vertex to be removed from the possible edge pool. If this happens, no spanning forests can be constructed from the remaining possible edges, though this algorithm will continue checking all of them. This, however, can be tested for between lines 7 and 8.

Another issue is that when using this enumeration for the sake of finding an unfolding cut-tree of $T = (G, H)$ through brute-force search. In this case, *forest*

Algorithm 5 Algorithm for enumerating all cut-forests of G

CutforestEnumerate G

```
 1  // all arrays are 0 indexed
 2  forest ← empty set of forest edges
 3  pedges ← empty array of arrays containing possible edges to be tried
 4  for v ∈ Exterior(G)
 5      Append all edges (v, w), w ∈ Interior(G) to pedges[0]
 6  while |pedges[0]| > 0 or |forest| > 0
 7      numedges ← |forest|
 8      if |pedges[numedges]| = 0
 9          Pop pedges[numedges]
10          Pop forest[numedges − 1]
11          Continue while loop from line 6
12      Remove an edge (v, w) from pedges[numedges]
13      if w ∈ forest
14          Continue while loop from line 6
15      Append (v, w) to forest
16      newedges ← empty array
17      for u ∈ Neighbor(w)
18          if u ∉ forest
19              Append (w, u) to newedges
20      Make a copy of pedges[numedges] as the array copy
21      Append all edges of newedges to copy
22      Append the array copy to pedges
23      if numedges = |Interior(G)|
24          // forest represents a spanning cut-forest
25          Output forest
26          Continue while loop from line 6
```

is unfoldable if and only if each rooted tree of *forest* is unfoldable. However, this enumeration will yield many *forests* with the same unfoldable tree. This, however, can be remedied by checking for it after line 25, and backtracking by removing edges from *forest* until at least one edge is removed from each tree of *forest* which fails to unfold. This is a very powerful heuristic, as will be shown later in Section 5.2, but it also misses some valid cut-forests. The reason is, even if a cut-tree t is not unfoldable, it might be possible to add edges to t to make it unfoldable. Hence, by pruning until at least one edge of t is gone, we miss out on the possible unfolding cut-trees which include t as a subtree. So, while this heuristic drastically speeds up the search for a

single unfolding cut-forest, it fails to accurately find all unfolding cut-forests.

4.4.2 Random

Instead of bruteforcing, it is also useful to be able to just output a random spanning cut-forest. This can be done by modifying Algorithm 5 to return the first tree created at line 25, and changing line 12 to remove a random edge from the pool of possible edges. While this will not pick each spanning cut-forest with equal probability, it is very simple to implement and should give decent results.

4.4.3 BFS Limitation

A heuristic we can place on cut-forest generation is to only allow BFS edges. In more detail, assign each vertex v of G a "BFS index" $\text{BFSI}(v)$, which is the fewest number of edges one must traverse starting from v to reach a boundary vertex of G. Then, we limit the enumeration or random generation algorithm to only allow the addition of edges (v, w) where $\text{BFSI}(v) + 1 = \text{BFSI}(w)$. That is, if v is part of a tree, we can only add (v, w) if w is strictly farther from boundary vertices than v.

While this heuristic does not take into account the geometric location of vertices, it instead uses the intuition that we want our cuts to be, in terms of the graph structure, the "shortest" paths from interior vertices to boundary vertices.

4.4.4 Greedy Heuristics

For a more systematic method, we can use a greedy algorithm by creating a heuristic to rank potential edges and always append the "best" edge at any point in time to our partial forest. Here we will list several potential heuristics along with reasonings for why we considered that heuristic. The results of these heuristics used on general spherical liftings can be found in Section 5.5.

Dijkstra — rank by smallest Dijkstra distance from the new vertex the edge connects to the boundary point that tree is rooted at. This aims to make the smallest,

simplest trees possible, which should be more likely to unfold than larger, more complicated trees.

Dihedral Angle — rank by either the smallest or largest dihedral angle of the edge leading to the new vertex. If an edge has a small dihedral angles that means it is between faces which are almost coplanar. It is unclear whether it would be better to cut along such edges or edges with large dihedral angles, so we decided to test both.

Angle — rank by smallest absolute angle between the new edge and the extension of the parent edge it is to be attached to. This seeks to make cut-trees be as straight as possible, somewhat like finding SMID-trees, which are more likely to unfold than more angular trees.

Chapter 5

Computational Results

In this chapter we show the results of a few computational experiments we ran based on the algorithms we described in the previous chapter.

5.1 Test System and Implementation Details

All code is written in Python for ease and speed of scripting, using Scipy 0.12 [12] for their Python wrapping of Qhull [1] for calculating convex hulls. Other than that, the rest of the datatypes are all implemented in native python lists and dictionaries. To keep things simple, we kept everything single threaded, though there are definitely many parts which could have been multithreaded to improve performance. Everything is run on a CSAIL cloud cluster machine, which has a 6-core Xeon X5650 at 2.67GHz.

For the dataset, we mainly used spherical liftings, both general spherical liftings as well as spherical liftings in an m-gon. This is because these are the easiest and fastest forms to generate, and we could get a lot of variety by merely varying the random seed used for generation. Similarly, we figured that if the number of points is large enough, the randomness should ensure that most structures of interest should appear somewhere in the midst of the spherical lifting.

5.2 Time to First Unfolding

First, we tested how far our simple brute-force search algorithm can get in unfolding randomly generated general spherical liftings, and how much this can be improved by the pruning heuristic detailed in Section 4.4.1. So, we ran the brute-force cut-forest enumeration and testing algorithm both with and without pruning for a fixed period of 8 days on the same dataset. This dataset consisted of general spherical liftings generated using the fixed seed of 0, with increasing vertex counts — starting with 5-vertex liftings, after the algorithm successfully unfolds the lifting of n vertices, it is given the lifting of $n + 1$ vertices.

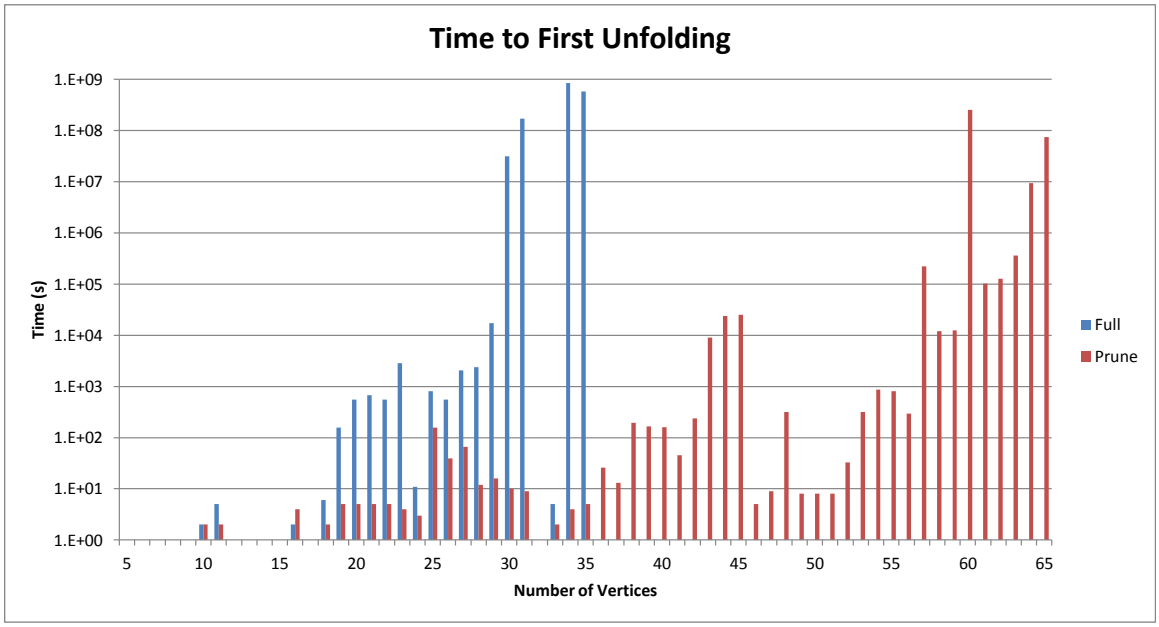


Figure 5-1: Time to first unfolding versus size of graph for brute-force enumeration with and without pruning heuristic

As the graph in Figure 5-1 shows, the pruning heuristic made a huge difference, allowing the simple brute-force algorithm to unfold up to 65-vertex liftings, while the original brute-force algorithm only managed to unfold up to 35-vertex liftings. There are two interesting facts to note about this graph and experiment.

Firstly, since all the liftings are generated using the same seed of 0, this means that in most cases, barring any sidecases as mentioned in Section 4.1.1, the $n + 1$-vertex lifting has n of the same vertices as the n-vertex lifting. Since each time the brute-

force search began anew without keeping information from previous searches, it is interesting to see that sometimes increasing the number of vertices actually dropped the search time. This means that by adding a vertex to spherical lifting, we can actually make it much easier (that is, several orders of magnitude lower search time by brute-force) to unfold the lifting.

Secondly, note the data at the 16-vertex unfolding times. This is the single case where pruning took much longer than not-pruning, and searched many more trees than not-pruning. This is likely because of the fact that it is possible for pruning to skip many valid cut-forests, which is also mentioned in Section 4.4.1, but here we see that it is actually possible for this to cause pruning to miss enough valid forests that it becomes slower than brute-force search. This also arises the possibility that, while very unlikely, this pruning heuristic might possible result in not finding any valid unfolding cut-forests even if ones definitely exist.

5.3 Percent Random Edge-Unfoldings

Similar to Schevon's [16] exploration of percent of random cut-trees of convex hull of spherical points which unfold, we did a similar study in percent of random cut-forests of almost-flat spherical liftings which unfold.

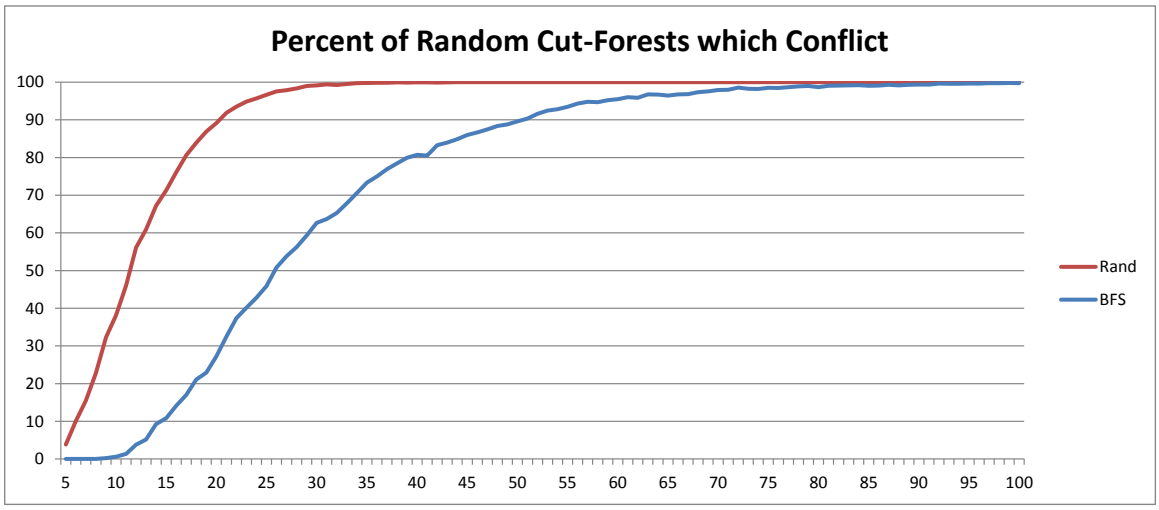


Figure 5-2: Percent of random cut-trees and random BFS cut-trees which conflict

As the graph in Figure 5-2 shows, similar to Schevon's results, as the number of vertices increase, the percent of random cut-forests which unfold without overlap goes to 0. The additional line corresponds to random cut-forests generated only from BFS edges, as described in Section 4.4.3. While using this heuristic helped shift the results to a slightly higher percentage, the eventual outcome is the same. Hence, we arrive at the same conclusion that as the number of vertices increase, the chance a randomly generated cut-forests unfolds without overlap decreases to 0.

5.4 Total Cut-Forests and Unfolding Cut-Forests

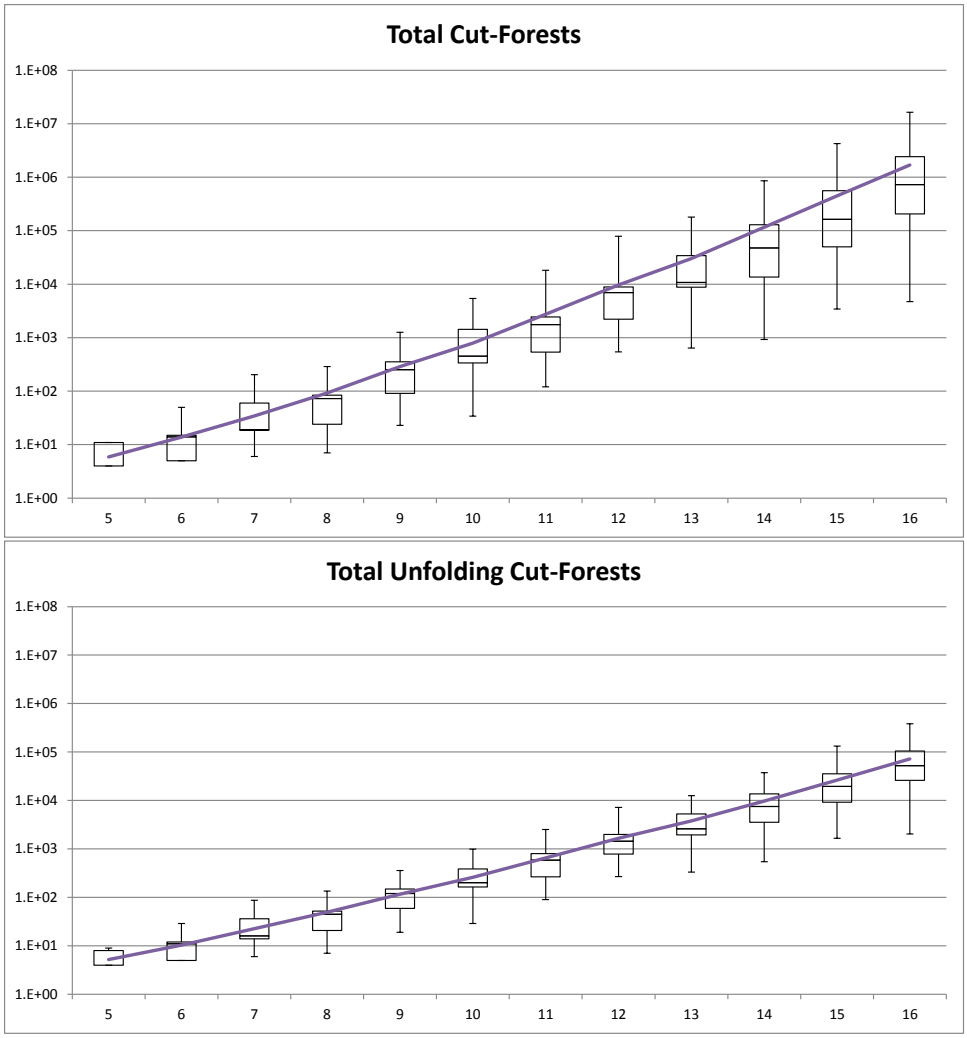


Figure 5-3: At each size, 100 general spherical liftings are generated. The graphs show a standard box and whisker plot while the line represents the average counts.

Another computational test we did was to check the total number of cut-forests and unfolding cut-forests for general spherical liftings. It is well known that the total number of spanning trees of a graph grows exponentially in the number of vertices, which means that the number of cut-forests grows exponentially, since cut-forests of the graph G can be put in a bijection with the spanning forests of a graph G' which is the same as G, except that the boundary vertices G are all merged into a single vertex. Instead, we wanted to see if the total number of unfolding cut-forests also grows exponentially in the number of vertices.

As the graphs in Figure 5-3 shows, the total number of cut-forests is definitely increasing at an exponential rate, and it definitely appears that the number of unfolding cut-trees of general spherical liftings is also increasing exponentially with the number of vertices. Also, there is quite a lot of variation, as the counts vary by up to 3 orders of magnitude. Furthermore, consider the graph of the ratio of the average number of unfolding cut-forests to total cut-forests in Figure 5-4.

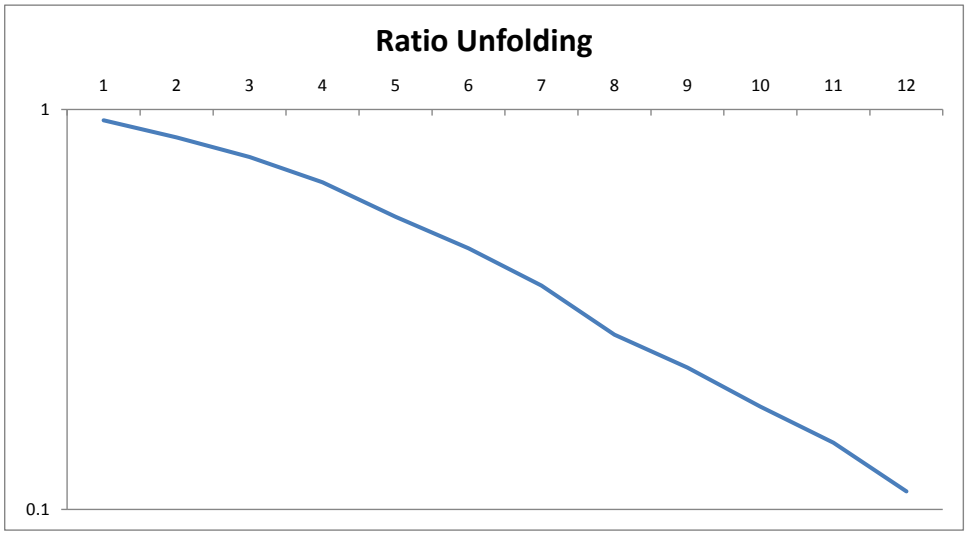


Figure 5-4: Ratio of cut-forests which unfold at different graph sizes

While the ratio of unfolding cut-forests to total cut-forests decreases at an exponential rate, experimentally the rate of this decay is much smaller, at least in the small numbers which we tested, than the rate of growth of the total number of cut-forests. Hence, we are lead to believe by experimental results that the number of unfolding cut-forests increases at an exponential rate.

5.5 Cut Forest Algorithm Comparison

In Section 4.4.4, we listed several heuristics for use with a greedy algorithm for cut-forest generation. We tested each heuristic on general spherical liftings of 10 to 490 points. At each size, every heuristic is tested using the same 100 terrains, generated using 100 different random seeds.

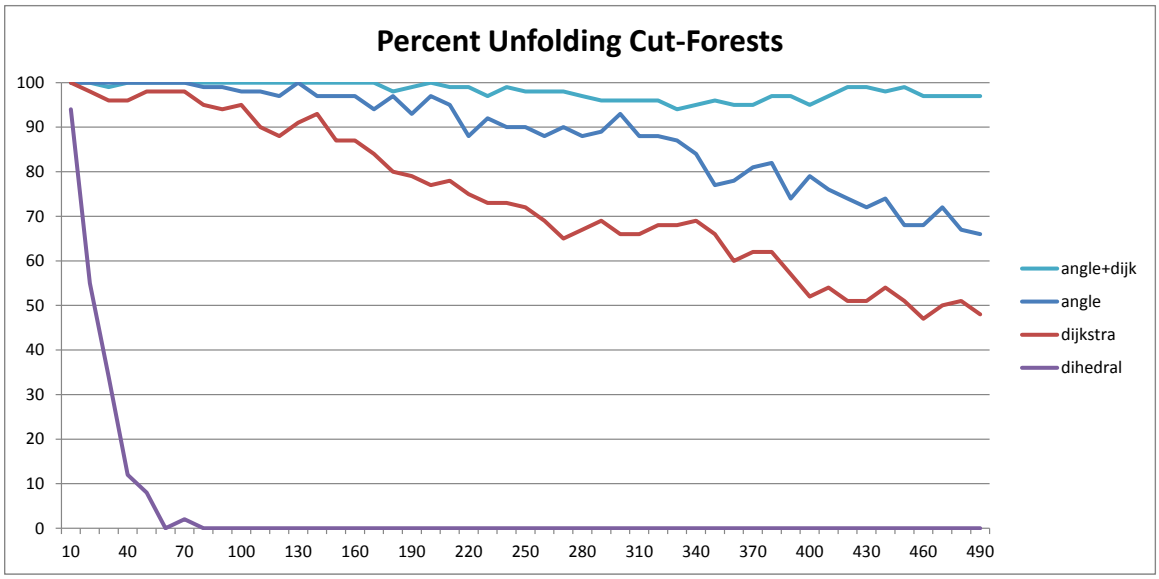


Figure 5-5: Percent of cut-forests which unfold for various greedy heuristics

As the graph in Figure 5-5 shows, the "min-angle" and "Dijkstra" heuristics did extremely well. However, even better was the "angle+dijkstra" heuristic, which mixed the two while using a cut-off which added a large penalty for edges which caused acute angles with their parent edge. The "dihedral" heuristic actually did very poorly: the one shown in the graph was for adding edges which had the smallest dihedral angle first, which meant cutting along the edges which "bent" the most. The heuristic which added the largest dihedral angle edges first did even more poorly than the one shown.

Since the "angle+dijkstra" heuristic did so well in the range of sizes we tried, we expanded the test to a larger range from 100 to 5900 vertices. We also increased the number of trials to 1000 to see if that will smooth out the curve.

Unfortunately, the graph in Figure 5-6 shows that, like any of the other heuristics,

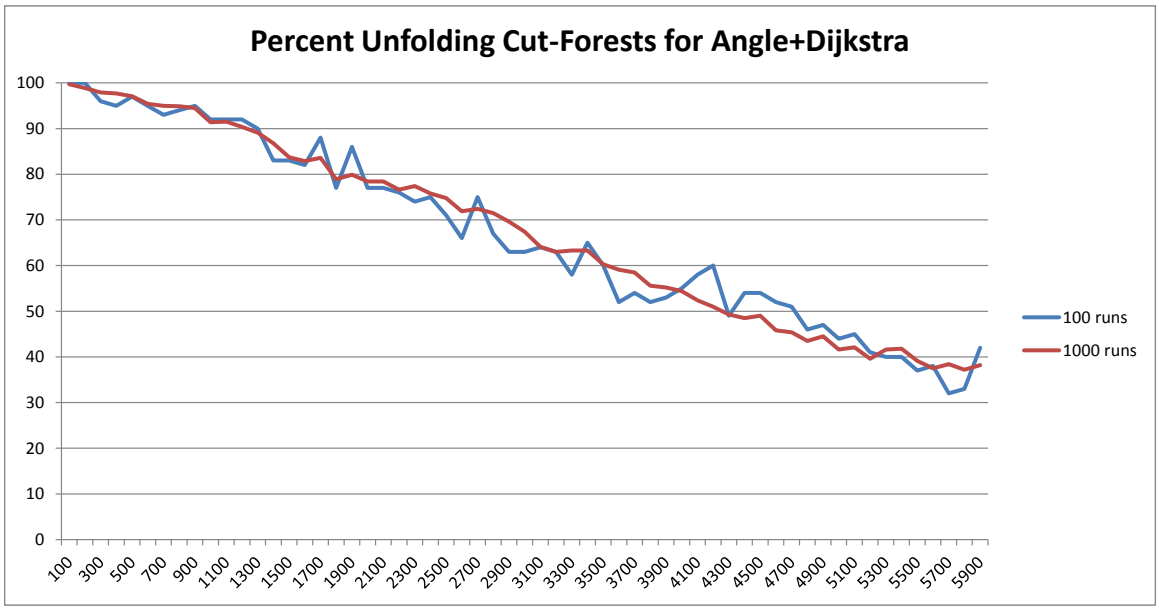


Figure 5-6: Percent unfolding cut-forests for 100 and 1000 runs of "angle+dijkstra" heuristic

at much larger vertex counts, the success of this heuristic also decreases, and as expected, 1000 trials did indeed smooth out the graph. So, then we decided to try varying the parameters — for this heuristic, we had four parameters A, B, C, D: if the angle between the edge and the parent is acute or right, we add a large penalty and rank by $\frac{\text{Dist}}{A} + \frac{\text{Ang}}{B}$, else if the angle is obtuse, we rank by $\frac{\text{Dist}}{C} + \frac{\text{Ang}}{D}$, where Dist is the Dijkstra distance including the new edge, and Ang is the absolute difference in angle between the new edge and a continuation of the parent edge. For the above graphs, we used the values $A = 1, B = 1, C = 1, D = 5$ chosen arbitrarily. In order to see the impact of the parameters on performance of this heuristic, we tested again with a wider range of parameters.

As the graphs in Figure 5-7 show, the parameters A and B had negligible effect on the result — the data for $AXBYC1D1$ almost perfectly follows the data for $A1B1C1D1$. Most likely, rarely is the algorithm ever forced to add an acute angle, and when it does, that likely causes a conflict regardless of how acute the angle is. Then, judging from the graphs of varying C and D, we see that they have opposite effects — this is expected, since if the contribution of Ang is scaled down, this is the same as the contribution of Dist being scaled up, and vice versa. Also, it seemed

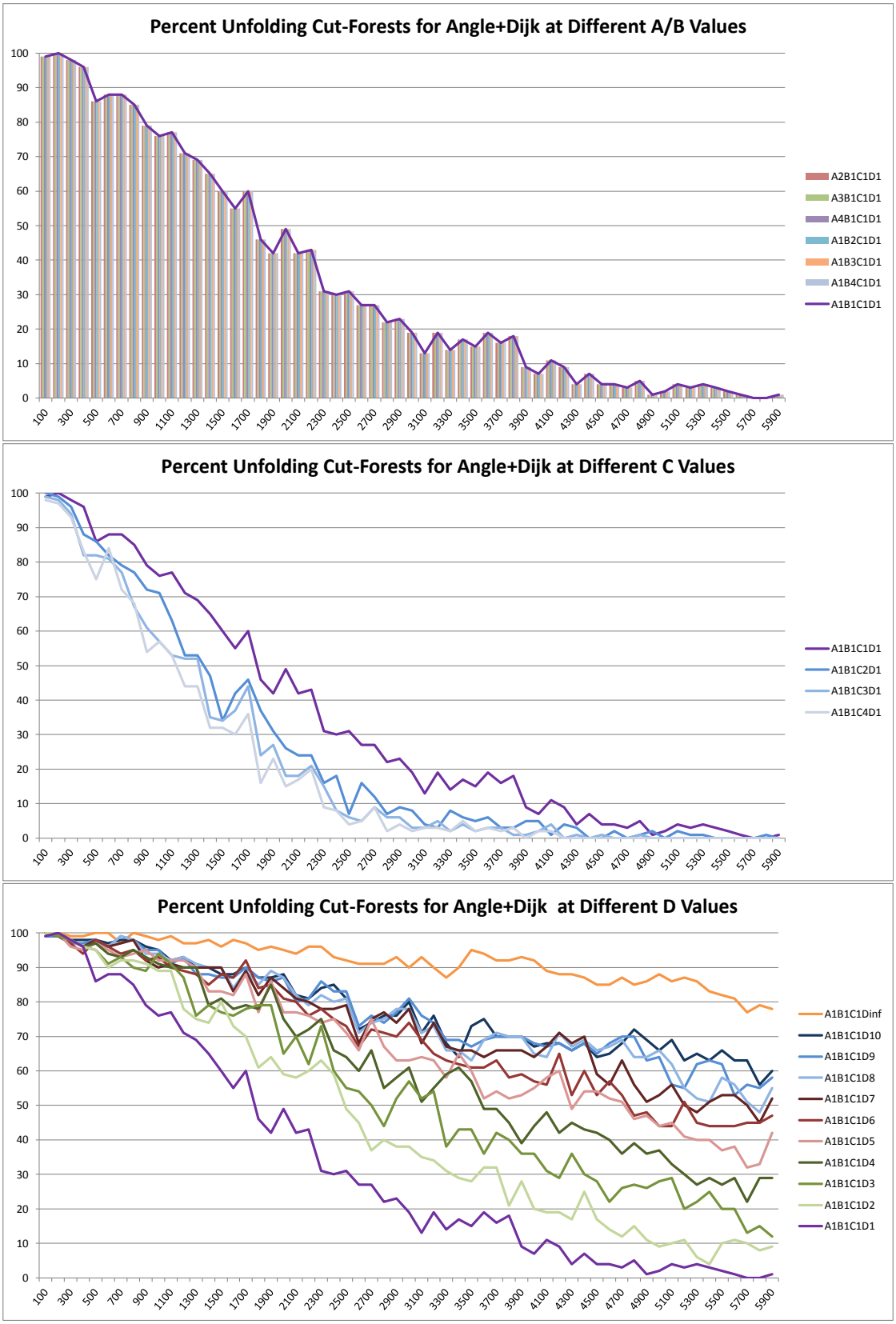


Figure 5-7: Percent unfolding cut-forests for various A,B,C, and D parameter values

that the Ang term in the obtuse angle case had a negative effect on the outcome, hence we added the additional test case shown, which effectively set $D = \infty$ by only ranking by Dijkstra distance in the obtuse angle case. Overall, it appears that the best heuristic for a greedy algorithm is this last heuristic, which ranks new edges by Dijkstra distance, considering first the edges which make an obtuse angle with their parent edge.

Chapter 6

Conclusions and Future Work

Overall, this thesis presented an exploration of almost-flat convex polyhedral terrain along several avenues. On the theoretical side, we showed that almost-flat convex terrains can only have local overlaps on unfolding and gave a method for determining whether a given cut-tree unfolds without overlap. We also demonstrated that any partial unfolding cut-tree can be locally extended into a larger partial unfolding cut-tree. Then, considering only the planar projections, we showed that SMID-paths and SMID-trees always unfold regardless of the convex lifting applied to the projection. Next, we presented a more constructive method of unfolding based around "slicing," which is similar to and can be seen as a generalization of Benton and O'Rourke's vertex-truncation technique. We showed that all unfolding cut-trees of almost-flat convex terrain have the "open-sector property" of Benton et al. and also characterized the unfoldability of various vertex-slices. Finally, for all these properties, we gave proofs and arguments to show that there exist positive and finite height bounds which achieve these properties, showing that almost-flat convex terrains are still three dimensional constructs and not flat.

Then on the computational side, we gave several algorithms for generating convex terrain, checking cut-tree unfoldability, and enumerating cut-forests. We also provided several heuristics for use with a greedy algorithm for creating cut-forests to test unfoldability. The computational results showed that as the number of vertices increased, the probability of a random cut-tree unfolding fell to zero. Also, through

brute-force enumeration, our data shows that both the total number of cut-forests and number of unfolding cut-forests increased exponentially as the size of the terrain. Finally, our results show that the "angle+dijkstra" heuristic does a very good job of producing unfolding cut-forests even for very large terrains.

6.1 Future Work

One interesting thing to note is that as mentioned at the end of Section 3.1.2, almost-flatness at times actually makes it harder for cut-trees to unfold without overlap. This is because, if the cut-tree contains a leaf which ends in an acute angle, then this by itself guarantees a conflict in the unfolding. There are several ways this could be useful for future study. First, it may be possible to use this fact along with other properties of almost-flat convex terrains to create an almost-flat convex terrain which has no unfolding cut-forests. Tiling such a construction on the faces of a convex polyhedron would potentially lead to a convex polyhedron which is ununfoldable.

Second, this might be a reason to study the opposite of almost-flat convex terrain — since almost-flatness actually results in more overlaps, it might be that the opposite constructs would result in less overlaps. Since almost-flatness involves shrinking a terrain in a direction, the opposite of almost-flatness, perhaps "almost-cylindrical," would involve stretching a terrain in a direction. Such terrain will be more cylindrical in nature, and it may be interesting to consider what the nets from unfolding such constructs would look like. It may be that since such forms are more tube-like, it may be more possible to "unroll" them. Unlike unfoldings of almost-flat terrain which are very close to their planar projection, unfoldings of such almost-cylindrical terrain will be very different from their planar projection, but this may give them the space they need to unfold without overlap.

A third use of the ununfoldability of cut-trees ending in acute angles in the context of almost-flat convex terrain would be to prove that the probability of a random cut-forest of a random almost-flat convex terrain unfolding goes to 0 as the size of the terrain increases. Our computational results definitely show this for spherical liftings,

but a rigorous proof of the more general case might be more enlightening.

Finally, the work on vertex-slices can definitely be extended through considerings of edge-slices and face-slices, and showing when such slices result in valid unfoldings.

Bibliography

[1] C. Bradford Barber and Hannu Huhdanpaa. Qhull. *The Geometry Center, University of Minnesota, http://www.qhull.org/*, 1995.

[2] Nadia Benbernou, Patricia Cahn, and Joseph O'Rourke. Unfolding smooth prismatoids. *arXiv preprint cs/0407063*, 2004.

[3] Alex Benton and Joseph O'Rourke. Unfolding polyhedra via cut-tree truncation. In *Proc. of CCCG*, pages 77–80, 2007.

[4] Marshall Bern, Erik D. Demaine, David Eppstein, and Eric Kuo. Ununfoldable polyhedra. In *Proceedings of the 11th Canadian Conference on Computational Geometry (CCCG'99)*, pages 13–16, Vancouver, British Columbia, Canada, August 15–18 1999.

[5] Marshall Bern, Erik D. Demaine, David Eppstein, Eric Kuo, Andrea Mantler, and Jack Snoeyink. Ununfoldable polyhedra with triangular faces. In *Abstracts from the 4th CGC Workshop on Computational Geometry (CGC'99)*, Baltimore, Maryland, October 15–16 1999.

[6] Marshall Bern, Erik D. Demaine, David Eppstein, Eric Kuo, Andrea Mantler, and Jack Snoeyink. Ununfoldable polyhedra with convex faces. *Computational Geometry: Theory and Applications*, 24(2):51–62, February 2003. Special issue of selected papers from the 4th CGC Workshop on Computational Geometry, 1999.

[7] Robert Connelly, Erik D. Demaine, and Günter Rote. Straightening polygonal arcs and convexifying polygonal cycles. *Discrete & Computational Geometry*, 30(2):205–239, September 2003.

[8] Erik D. Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, New York, NY, 2007.

[9] Julie DiBiase. *Polytope Unfolding*. PhD thesis, Smith College, Northampton, Mass., 1990.

[10] Branko Grünbaum. A starshaped polyhedron with no net. *Geombinatorics*, 11:43–48, 2001.

[11] Branko Grünbaum. No-net polyhedra. *Geombinatorics*, 11:111–114, 2002.

[12] Eric Jones, Travis Oliphant, and Pearu Peterson. Scipy: Open source scientific tools for python. *http://www.scipy.org/*, 2001.

[13] Brendan Lucier. Unfolding and reconstructing polyhedra. Master's thesis, University of Waterloo, 2006.

[14] Brendan Lucier. Local overlaps in special unfoldings of convex polyhedra. *Computational Geometry*, 42(5):495–504, 2009.

[15] Travis Oliphant et al. Numpy. *http://www.numpy.org/*, 2007.

[16] Catherine Schevon. *Algorithms for Geodesics on Polytopes*. PhD thesis, Johns Hopkins University, 1989.

[17] Wolfram Schlickenrieder. Nets of polyhedra. *Master's Thesis, Technische Universität Berlin*, 1997.

[18] A. S. Tarasov. Polyhedra that do not admit natural unfoldings. *[In Russian] Uspekhi Mat. Nauk*, (54), 1999.