3.4 Collision Repair

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

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

3.4.1 Terminology

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

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

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

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

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

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

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

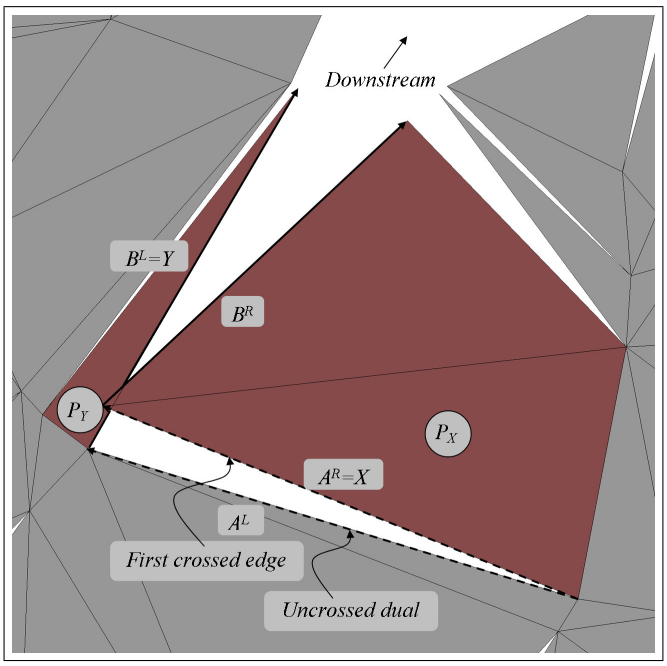


Figure 3.10: Detail of a collision showing labeled terms. There is exactly one *first crossed edge* in every collision.

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

3.4.2 The *first crossed edge*

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

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

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

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

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

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

3.4.3 Repairing a single collision

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

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

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

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

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

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

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

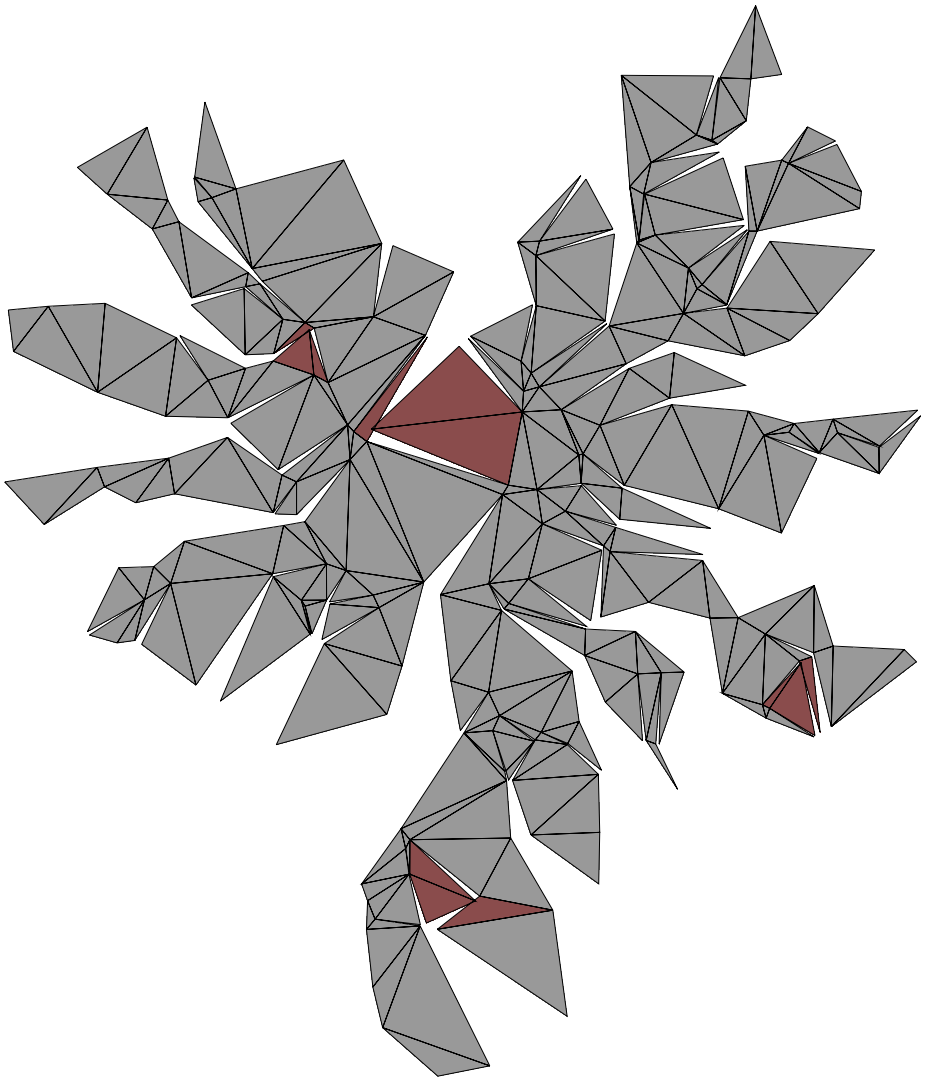
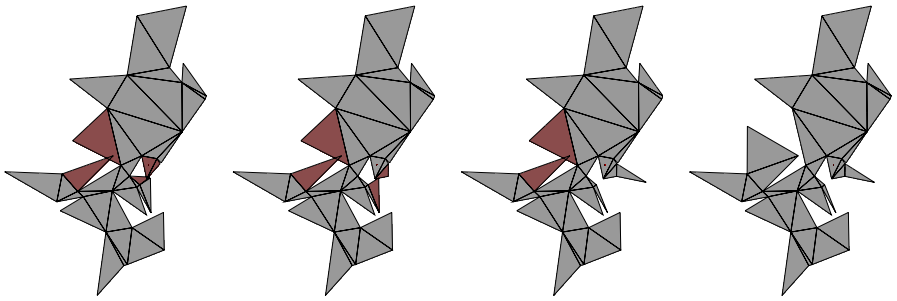


Figure 3.11: The unfolded net of a randomly-generated surface with 296 faces, which has four separate conflicts when unfolded by the Steepest Ascent Unfolder

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

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



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

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

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

Corollary 3.11 *Every collision can be repaired.*

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

3.4.4 Application of the repair algorithm

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

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

Algorithm 4: Collision repair for randomized nets

Let U be a randomly-generated unfolding of the polyhedron
Let M be a map from nets to integers; $M[u]$ is zero for any unfolded net u which has not been previously visited
Let F_0 be the first developed face
Let C be the center of F_0
while $\exists$ collisions in U **do**
  Let K be the collision in U closest to C
  if $M[U] \geq 4$ **then**
    Return failure
  else if $(M[U]\,{+}{+})$ is not odd **then**
    Repair K by holding P_X fixed and moving P_Y
  else
    Repair K by holding P_Y fixed and moving P_X
  end if
end while

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

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

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

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

3.4.5 The first approach: repairing a randomly-generated net

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

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

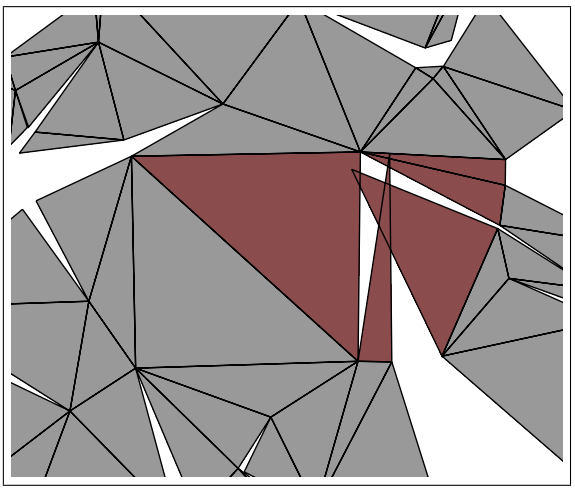


Figure 3.13: This detail from an overlapped random unfolding of a random convex polyhedron cannot be repaired by Algorithm 4. With two branches of the cut-graph competing to determine which direction is 'upstream' from the collision, the algorithm becomes caught in an infinite loop.

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

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

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

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

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

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

Algorithm 5: Collision repair for outward-facing nets

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.4.7 Weaknesses in the argument

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

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

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

The known flaws in the arguments are:

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

3.4.8 Experimental support

All figures in this section were generated by Algorithm 5.

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

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

Figure 3.16 shows the progressive repair of the banded icosahedron.

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

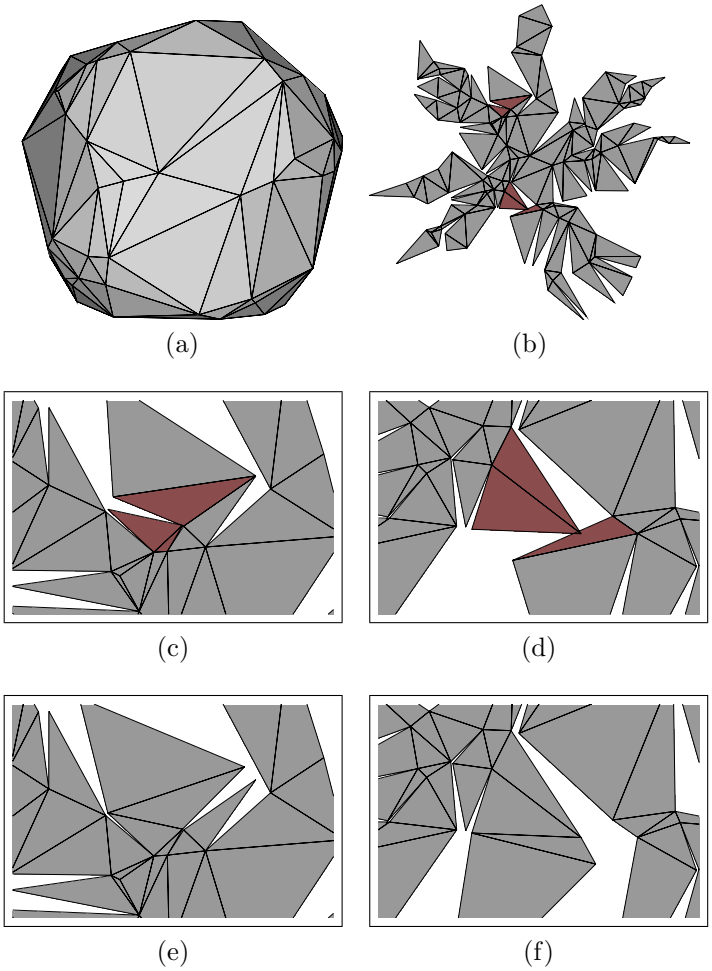


Figure 3.14: Repairing two simple conflicts. (a) The polyhedron (b) The unfolding, with collisions (c) Detail of the first collision (d) Detail of the second (e) The first collision, repaired (f) The second collision, repaired

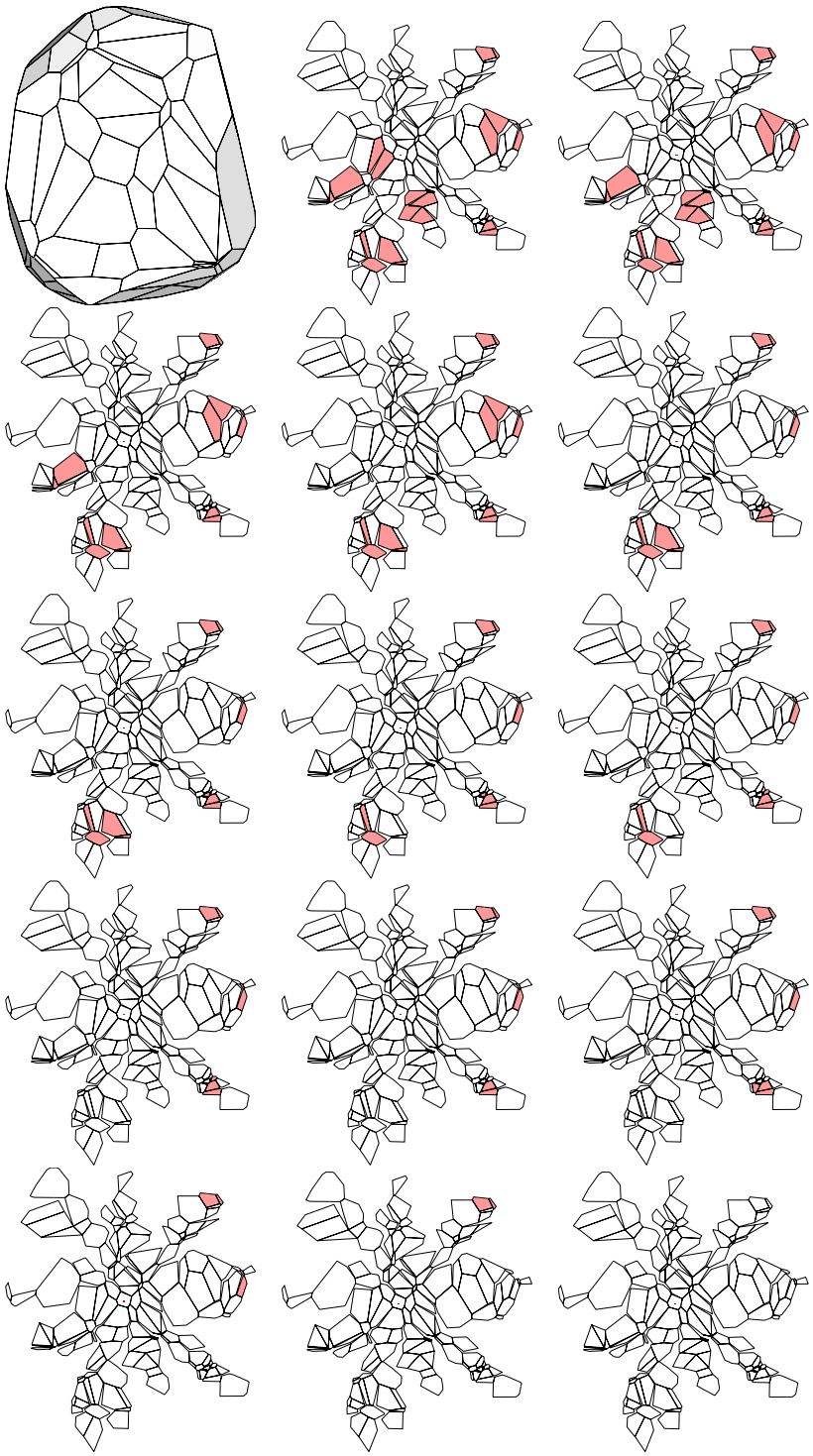


Figure 3.15: Repairing a randomly-generated convex polyhedron

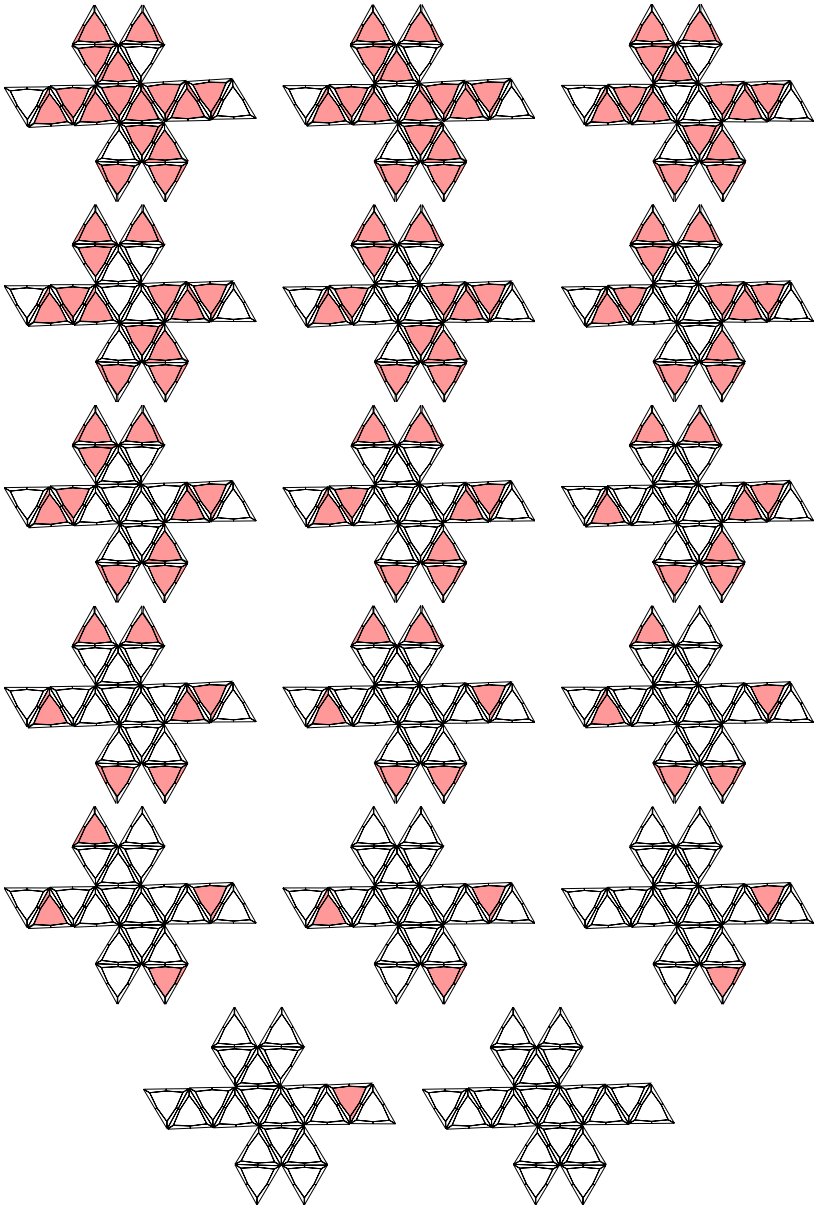


Figure 3.16: Repairing the banded icosahedron

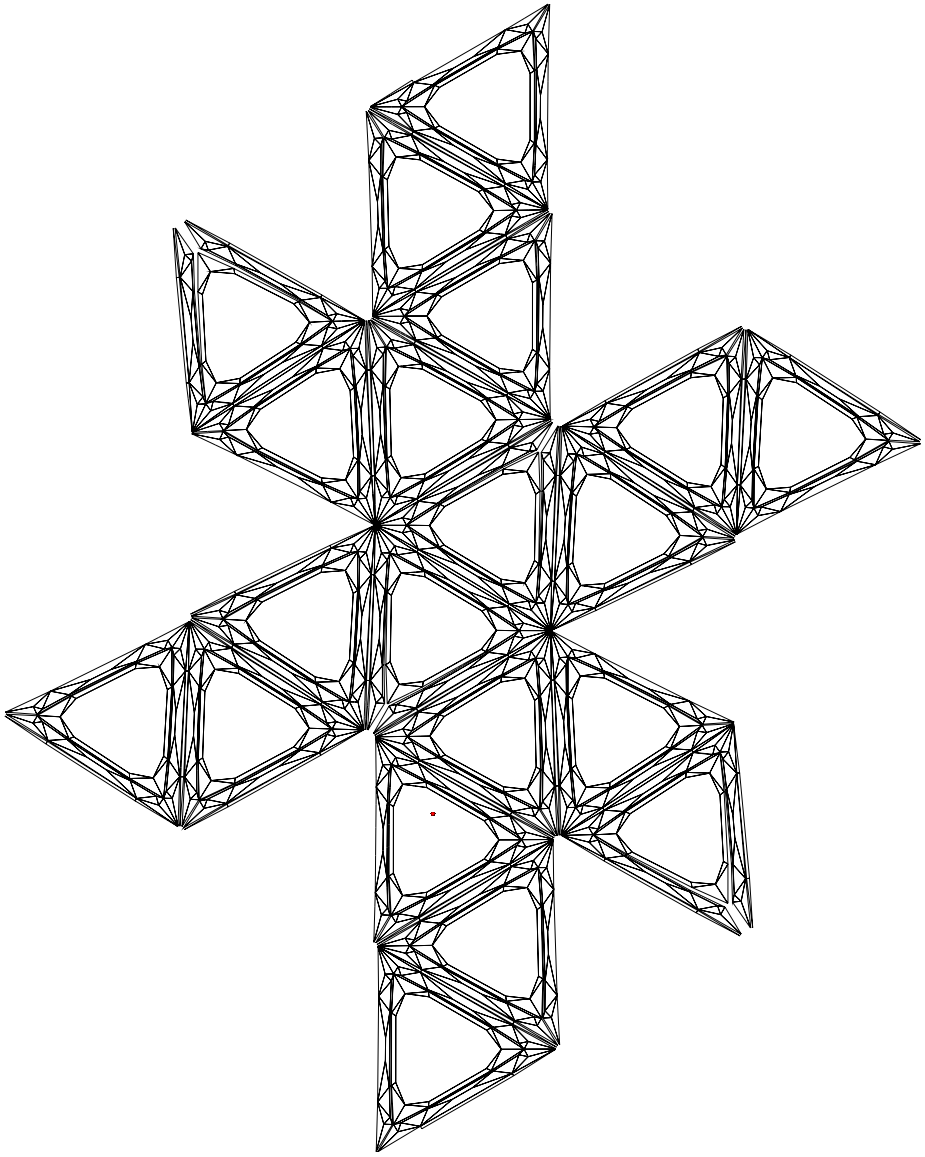


Figure 3.17: The doubly-banded icosahedron

3.5 Conclusions

The lower bound of undevelopable

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

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

The alpha-beta rules

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

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

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

Collision repair

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

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

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

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

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

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

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