Strategies for Polyhedral Surface Decomposition: An Experimental Study *

BERNARD CHAZELLE[†] DAVID P. DOBKIN[‡] NADIA SHOURABOURA[§]
AYELLET TAL[¶]

1 Introduction

Convex shapes are easiest to represent, manipulate, and render: Even though they form the building blocks of bottom-up solid modelers, it is more often the case that the convex structure of a geometric shape is lost in its representation. We are then presented, not with the solid-modeling problem of putting together primitive convex objects, but with the reverse problem of *extracting* convexity out of a complex shape.

The classical example is that of cutting up a 3-polyhedron into convex pieces. This is often a useful, sometimes a required, preprocessing step in graphics, manufacturing, and mesh generation. The problem has been exhaustively researched in the last few years [1]—[17]. Despite its practical motivation, however, little of that research has gone beyond the theoretical stage. One possible explanation is that even the most naive solutions are programming challenges. We observe, however, that in practice one often need not partition the polyhedron itself but only its *boundary.* In other words, it often suffices to decompose a polyhedral surface into a small number of convex patches. By abuse of terminology, we call a surface *convex* if it lies entirely on the boundary of its convex hull. We mention some applications briefly.

In rendering, the coherence provided by convex patches can be exploited to speed up radiosity calculations. For example, from the closed-form expressions recently found for the form factors between two polygons [18], it is possible to derive faster iterative methods for handling multiple pairs of facets that are known to lie on one or a few convex patches. Similar speed-ups can be obtained for shading, clipping, hit detection, etc. Snyder et al. [19] point out the importance and difficulty of exploiting polyhedral coherence in collision detection: Because

*Work by Bernard Chazelle, David Dobkin and Ayellet Tal has been supported in part by NSF Grant CCR-93-01254 and The Geometry Center, University of Minnesota, an STC funded by NSF, DOE, and Minnesota Technology, Inc. Work by Nadia Shouraboura has been supported in part by NSF Grant PHY-90-21984.

†Department of Computer Science, Princeton University

‡Department of Computer Science, Princeton University

§Program in Applied and Comput. Math., Princeton University

¶Department of Computer Science, Weizmann Institute

polyhedral hierarchies can be defined on arbitrary convex patches (solid polyhedra are not needed), intersection primitives can be greatly speeded up when convex surface decompositions are available [13].

Another application can be found in multimedia animations. Bitmap animations, such as *flics* and cell animations, are the simplest to play back, but they are tremendously wasteful of storage. They also make it hard to build in interactive user-response features. An alternative is to store background bitmaps and represent moving objects as wire frames [3]. Besides the savings in storage, the main advantage of this approach is to allow for real-time user input. Unfortunately this has an adverse effect on speed, not to mention the added development time associated with more complicated rendering and playback algorithms. We believe that convex surface decomposition techniques can alleviate this problem. Work is underway to test this assumption.

Besides simplicity, decomposing the boundary of a polyhedron into convex patches has other advantages. For example, while a polyhedral solid decomposition can suffer a quadratic blow-up [8], boundary decompositions are always linear in size. Better than that, the number of convex patches can be kept within a constant factor of the number of reflex angles [11]. We have gathered empirical evidence suggesting that even highly complex surfaces typically consist of only a handful of patches. For example, a standard drinking glass, regardless of its description size, might involve no more than a dozen convex patches. Intuitively, one should expect only surfaces of little coherence, such as crumpled, fractal-like sheets or heavily twisted surfaces, to give rise to many patches.

Contributions of this paper: We present a comparative study of simple heuristics for convex surface decomposition. The research reported here is mostly of an experimental nature. The standout exception is the obligatory first step: motivating the search for heuristics by proving that the problem is NP-complete. The proof is somewhat technical, so we give an outline in §2 and provide the details in the Appendix. In §3 we describe the three classes of heuristics investigated: *space partitioning*, *space sweep*, and *flooding*. Within each class we examine several sub-heuristics and compare their relative effectiveness. Experimental results are reported and conclusions are drawn in §4.

Finding meaningful test data was an important component of this work. In the present case, randomly generated data is all but worthless. So, we collected several hundred real-life polyhedral models from manufacturing companies and industrial AutoCAD users. This catalog confirms our working hypothesis that most objects admit of small-size boundary decompositions. We implemented various decomposition schemes from each of the three classes of heuristics; each implementation was tested and benchmarked against a representative sample of objects from our library. From this experimentation it appears that flooding heuristics are the most efficient as well as the easiest to implement. We propose a scheme, called *flood-and-retract*, which seems the method of choice among the heuristics investigated.

Unsurprisingly, space partitioning techniques fared the worst. The main motivation for including them in our investigation was that many users are equipped with space partitioning software, so it was of practical relevance to assess their effectiveness. Space-sweep heuristics were also natural candidates. Their asymptotic performance is guaranteed to be linear [11], but even the simplest ones are quite difficult to implement.

An innovative aspect of this work has been the use of animations to guide our search for good heuristics. Runs and benchmarks produce numbers that tell us how good or how bad a given heuristic is. But they do not help us to design better heuristics. Visualizing the decompositions does. Physicians swear by (and live off) the adage that lab results cannot supplant clinical examination. Likewise, we found that nothing was more useful than the ability to look at a decomposition in three dimensions as though we held it in our hands. Through this means, weaknesses of the heuristics came to light and ways to overcome them suggested themselves. In addition to our in-house geometric animation system GASP, we also used the visualization package GEOMVIEW developed at the Geometry Center. After it became apparent that flooding heuristics were the most competitive, we animated several of them and visually analyzed their most obvious flaws. These animations enabled us to converge rapidly towards the most promising heuristic, ie, flood-and-retract.[1]

2 The Complexity of Convex Surface Decomposition

Let S be a polyhedral surface with n vertices, and let $S_1, \ldots, S_k$ be disjoint convex patches whose union gives S. We show that minimizing the number k is NP-complete. In our terminology, a polyhedral surface is a compact piecewise-linear 2-manifold with boundary. We make no assumption on its orientability or its Euler characteristic. Obviously, convex patches are always orientable, but again their Euler characteristics are let unrestricted; in other words, the patches may be multi-connected. It is natural, however, to require that convex patches be connected.

Note that this is only one of many possible variants of the problem: for example, we could place bounds on the Euler characteristics of the patches; we could require that facets not be split; we could seek a cover and not a partition; we could relax the connectivity requirement; in the case of orientable surfaces, we could distinguish between convexity and concavity, etc.

2.1 Membership in NP

It is easy to see that the problem is a special case of clique partitioning. To begin with, observe that the facets of any patch can always be assumed to originate from the three-dimensional arrangement formed by the planes defined by all triplets of vertices of S. Indeed, for any decomposition that does not satisfy this requirement, we can always extend any facet (if necessary) until its bounding edges lie in one of those planes. This gives us up to $O(n^9)$ candidate facets from which a minimum decomposition can be guessed.

To test if a set of facets forms a convex patch, we compute its convex hull and verify that all the facets lie on the boundary. An equivalent certificate can be formed by expressing the fact that for each pair of facets, f, g, all the vertices of f lie on the same side of the plane supporting g, and vice-versa. Let G be the graph whose nodes represent the facets, and whose edges connect pairs of nodes whose corresponding facets satisfy the certificate in question. Let H be the graph with the same node-set, but whose edges connect nodes with adjacent associated facets. A minimum decomposition is simply a min-size partition of G into cliques whose induced subgraphs in H are connected. Checking whether a given decomposition size k is achievable can thus be done nondeterministically in polynomial time.

[1]To illustrate this process of iterative improvements we have produced a video: this supplement plays the same role as traditional figures but with motion and color added.

2.2 NP-Hardness

Let $x_1, \ldots, x_n$ be n Boolean variables; an instance of SAT consists of n clauses $c_1, \ldots, c_n$, each of them a disjunction of literals x_k or $\bar{x}_k$. Here is a brief overview of our geometric model: Each variable x_k is associated with a closed polygonal curve L_k zigzagging in planes parallel to xy. There are exactly two optimal ways of cutting L_k into convex pieces: each of them corresponds to a different truth assignment of x_k. A clause is modeled by a vertical line segment that connects the curves corresponding to its literals. The contact between the vertical segment and L_k takes place at a local y-maximum or y-minimum depending on whether x_k is negated or not in the clause in question. The remainder of the proof involves transforming the curves into thin strips and then arguing that the formula is satisfiable if and only if the minimum decomposition is below a certain size.

3 Three Classes of Heuristics

All the surfaces in our library are orientable. The reason is that they usually originate from the boundary of some polyhedron. This suggested distinguishing between convex and concave patches; the latter being convex patches (in the old sense) all of whose edges exhibit reflex angles. We shall specify below which sub-heuristics make this distinction.

Space Partitioning. The strategy is to use binary space partitioning [14] to split up the surface into convex patches. Recall that the method builds a tree by recursively dividing space by a (well-chosen) cutting plane. Each node v of the tree is associated with a convex polyhedron P_v. The idea is to explore the two children of v if and only if the portion of the surface within P_v is not convex. The advantage of this method is that space partitioning code is widely available, so implementing it is by and large effortless. One drawback is that facets get split up in the middle: this produces Steiner points, which cause roundoff errors and can be undesirable. Furthermore the efficiency of the heuristic is highly sensitive to the input surface. An obvious improvement we use is to cut along reflex edges only. In the worst case the number of patches is quadratic. In practice it appears that such a blowup is unlikely, especially in view of our edge selection rule.

Space Sweep. We implemented a simplified version of the method in [11]. In particular, we modified the algorithm to prevent the creation of Steiner points. Although the linearity of the output size is no longer (theoretically) guaranteed, the simplification seems to have no adverse effect; on the other hand, it makes coding much easier. The idea is to sweep space with a plane: at any given time, the cross-section of the surface consists of simple polygonal curves which are decomposed into convex pieces. The heuristic attempts to maintain each curve as long as possible while moving the plane, thus producing convex patches in the process. Each time a convexity violation is found, we relent from including the violating facet and start up a new convex curve (and hence, a new patch).

Flooding. Let H be the dual graph of the surface, where nodes represent facets and arcs join nodes associated with adjacent facets. The class of *flooding* heuristics refers to the incremental strategy of starting from some node and traversing the graph H, collecting facets along the way as long as they form a convex patch. We distinguish between two sub-heuristics: *Greedy flooding* involves collecting facets until no adjacent facets can be found that does not violate the convexity of the current patch. A new patch must then be started, at which point the traversal can resume. *Controlled flooding* includes other stopping rules besides convexity violation. Our animations have revealed cases where flooding in a greedy fashion is arbitrarily bad: off the optimal by a factor proportional to the input size! On the other hand, producing smaller patches by controlled flooding can sometimes lead to near-optimal decompositions.

Given a convex patch, consider adding a new facet to it. The new patch may fail to be convex for one of two reasons:

1. **Local failure:** The edge at which the facet is attached to the patch exhibits non-convexity.
2. **Global failure:** The new patch is locally convex everywhere, but some facet fails to be on the boundary the new convex hull.

Global failures are rare in practice. Geometrically, they are associated with twisted shapes, as in a spiral or a drill. Note that global failures are the only reason the surface decomposition problem is hard. Without global failures, *any* greedy flooding heuristic produces an optimal decomposition (in the version of the problem where concave patches are ruled out). Furthermore, covers and partitions are indistinguishable. In other words, without global failures, trying to cover the surface into convex patches instead of partitioning it cannot produce fewer pieces. In the presence of global failures, however, just the opposite is true. This suggests producing controlled-flooding partitions in two steps. First, flood the surface by covers: this means that when restarting a new patch, allow the traversal of old facets as well as new ones. Then, in a second pass, transform the covers into partition. We pursue this approach below.

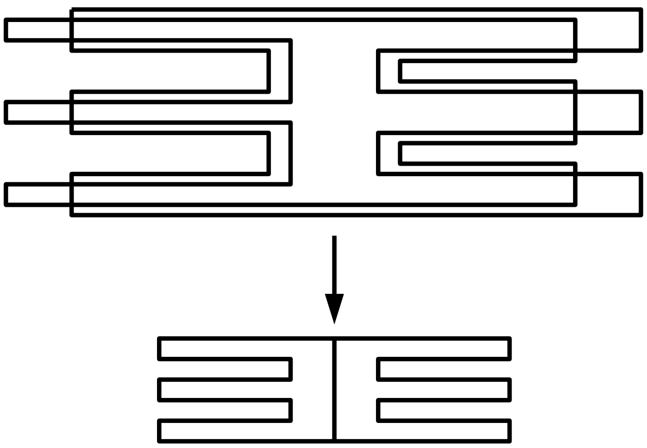


Figure 1:

Flood-and-Retract. The first phase, flooding the surface, produces convex patches which might overlap. The purpose of the second phase is to remove the overlapping by "retracting" each patch. The main pitfall to avoid is breaking up the patch into several connected components. In Fig. 1, for example, two patches cover the surface and can be easily modified into a 2-patch partition. However, a naive approach might end up producing up to $\Omega(n)$ convex patches. Plate 3 gives a three-dimensional picture of this phenomenon. To avoid this trap, we adopt the following strategy: for each patch in turn, retract it as much as possible as long as it remains connected. The retraction takes place along the portions of the boundary of the patch that lie inside other patches. The data structure is quite simple: we keep a queue of facets to be retracted. We iterate on the following process: remove the top of the queue and remove the facet from the patch, unless it disconnects it. After the removal, check which adjacent facets should be inserted into the queue, and insert them in arbitrary order. Once each patch has been retracted in this manner, transform the resulting cover into a partition in arbitrary manner.

4 Experimental Results

The tables below present the results of our experiments. We implemented 8 heuristics and ran them on the objects of the library. For the sake of concreteness, we limit our discussion to a sample of 12 representative objects. See Plates 1,2. Next to each object, we indicate the number of vertices V, the number of *convex* edges and *reflex* edges. Some edges are incident to coplanar facets: their count is *par*. Such edges originate in one of two ways: either the dihedral angle was too close to π to decide on convexity, or the edge was introduced to triangulate a facet, in which case the angle was exactly π. The heuristics chosen were:

- **BSP:** We adapted existing Binary Space Partitioning code by pruning the tree as explained earlier. One difficulty was to handle the facets that lay within cutting planes. For this reason we found it easiest to allow disconnected convex patches. No distinction is made between convex and concave patches.
- **Space-Sweep:** Both convex and concave patches are admissible in *sweep_cc*. On the other hand only convex patches are allowed in *sweep_c*.
- **Flooding:** We consider greedy flooding produced by breadth-first search with convex-concave patches (*bfs_cc*) and convex patches only (*bfs_c*). We also present results on depth-first search with convex-concave patches (*dfs_cc*) and depth-first search with convex patches only (*dfs_c*). Plates 1,2 show the *bfs_cc* decompositions of the 12 sample objects.

	name	V	*convex*	*reflex*	*par*	*bsp*
1.	epcot	195	384	192	0	344
2.	mushroom	227	365	142	165	567
3.	glass	81	124	34	49	98
4.	pawn	155	216	96	144	303
5.	bishop	251	352	118	274	671
6.	rook	131	152	80	152	381
7.	spring	910	1626	875	220	2144
8.	flashlight	388	549	145	415	532
9.	eyeball	966	1683	24	934	642
10.	torso	321	619	234	162	547
11.	book	88	118	34	88	131
12.	hand	309	525	234	162	931

Table 1.

	name	V	*sweep_cc*	*sweep_c*
1.	epcot	195	129	129
2.	mushroom	227	66	64
3.	glass	81	21	21
4.	pawn	155	41	40
5.	bishop	251	39	33
6.	rook	131	34	31
7.	spring	910	389	420
8.	flashlight	388	62	78
9.	eyeball	966	45	27
10.	torso	321	99	89
11.	book	88	48	57
12.	hand	309	146	91

Table 2

	name	V	*bfs_cc*	*dfs_cc*	*bfs_c*	*dfs_c*
1.	epcot	195	129	129	129	129
2.	mushroom	227	56	48	56	49
3.	glass	81	12	12	13	12
4.	pawn	155	28	28	31	29
5.	bishop	251	20	19	20	19
6.	rook	131	18	18	18	18
7.	spring	910	277	247	256	234
8.	flashlight	388	18	18	35	35
9.	eyeball	966	4	4	4	4
10.	torso	321	69	67	70	69
11.	book	88	13	13	12	12
12.	hand	309	89	86	93	93

Table 3

There is a remarkable consistency in the way flooding outperforms its competitors. Binary space partitioning fares so poorly it probably is worse than doing nothing. Space sweep, on the other hand, provides flooding a fairly close race. On *epcot* it produces the optimal decomposition, as does any kind of flooding. But this is an exception. On average space sweep loses to flooding by at least 50% and sometimes much more. On *book* it loses by a factor of 4. In general, the differences between convex-only and convex-concave are surprisingly small. Note that (unlike what one might have expected) convex-concave does not always outperform convex-only: this is owed to the nondeterministic nature of the heuristics.

Flooding performs uniformly well. The differences between depth-first search and breadth-first search are small enough to be negligible, although DFS appears to have a slight edge. Note that visually it might seem that an inordinately high number of patches are produced: but recall that from a computational standpoint, coplanarity is an elusive property. We chose to make the program err on the side of robustness. This means that the patches produced are guaranteed to be convex, but it could be that near-flat convex edges are classified as reflex because of roundoff errors. Flood-and-retract is not included in the tables because the results essentially do not differ from breadth-first-search, except on a few objects, eg, *spring*. The reason is that the retract part of the heuristic "kicks in" only when global failure is detected, something which happens rarely.

Plate 3 shows an object, a bracelet, where greedy flooding (and all the previous heuristics for that matter) performs very poorly. Flood-and-retract avoids the obvious pitfalls and provides a near-optimal decomposition. The upper-left figure displays the input surface. BFS and DFS decompositions are shown right below. Note that both produce an additional (unnecessary) patch for half the fingers of the bracelet. The right-hand side figures show flood-and-retract in action. First, the algorithm floods the bracelet with a BFS run (dark patch). Then it repeats the same operation starting outside that patch. Since the dark patch is ignored, we end up with two overlapping patches. Finally the retraction moves back one of the patches to produce the final two-patch partition of the bracelet.

It would be a mistake to think this is an isolated case. The strategy of covering first and then retracting is meant to deal with a general shortcoming of greedy flooding, ie, *fragmentation*. This phenomenon is similar to what happens when partitioning a tree with fixed-size subtrees. If we start at the root and proceed top-down we run into the risk of ending up with a linear number of fragments near the leaves, each requiring a distinct tree. Of course, in that case, we know better not to start at the root but instead to proceed bottom-up. A similar phenomenon occurs with flooding. The difference, however, is that unlike tree partitioning flooding has no natural starting place. The idea of covering first and retracting later is meant to overcome that intrinsic difficulty by trying to make the starting place immaterial.

5 Conclusions

This paper has investigated the practical issues behind convex surface decomposition, a problem shown to be NP-complete. We have not addressed issues of implementation, robustness, and computation time. Robustness is (as always) a thorny problem. Computation time, however, is not a serious bottleneck, because all our heuristics run either in $O(n \log n)$ time or in time linear in the input/output size.

Our conclusion is that the flood-and-retract heuristic should be the method of choice in practice. Of course, as we observed, in many cases the covering produced in the first phase of the algorithm will actually be a partition (in which case flood-and-retract reduces to greedy flooding). In such cases, the decomposition is optimal or near-optimal and there is no need to pursue the heuristic further, ie, DFS does the job. It remains to be seen if more complex variants can produce significantly better decompositions.

Acknowledgments: We thank Herb Voelcker and Jovan Zagajac of Cornell University for kindly making their BSP code available to us, and generally bringing us the benefit of an engineering perspective on the issues discussed in this paper.

References

[1] Aronov, B., Sharir, M. *Triangles in space or building (and analyzing) castles in the air*, Combinatorica, 10 (1990), 137–173.

[2] Aronov, B., Sharir, M. *Castles in the air revisited*, Proc. 8th Ann. ACM Symp. Comput. Geom. (1992), 146–156. To appear in Disc. Comput. Geom.

[3] *AutoCAD Reference Manual*, Autodesk, Inc. Publication 100625 (1992).

[4] Bajaj, C.L., Dey, T.K. *Convex decompositions of polyhedra and robustness*, SIAM J. Comput., 21 (1992), 339–364.

[5] Bern, M. *Compatible tetrahedralizations*, Proc. 9th Ann. ACM Symp. Comput. Geom. (1993), 281–288.

[6] Bern, M., Eppstein, D. *Mesh generation and optimal triangulation*, in: Computing in Euclidean Geometry, 1, World Scientific, ed. D. Z. Du and F. K. Hwang (1992), 23–90.

[7] Bern, M., Eppstein, D., Gilbert, J. *Provably good mesh generation*, Proc. 31st Ann. IEEE Symp. Foundat. Comput. Sci. (1990), 231–241.

[8] Chazelle, B. *Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm*, SIAM J. Comput., 13 (1984), 488–507.

[9] Chazelle, B., Palios, L. *Triangulating a nonconvex polytope*, Disc. Comput. Geom., 5 (1990), 505–526.

[10] Chazelle, B., Palios, L. *Decomposition algorithms in geometry*, in Algebraic Geometry and its Applications, C. Bajaj, Ed., Chap.27, Springer-Verlag, 1994, pp. 419–447.

[11] Chazelle, B., Palios, L. *Decomposing the boundary of a nonconvex polytope*, Proc. 3rd Scandinavian Workshop on Algorithm Theory (1992), 364–375.

[12] Chazelle, B., Shouraboura, N. *Bounds on the size of tetrahedralizations*, Proc. 10th Ann. ACM Symp. Comput. Geom. (1994), 231–239.

[13] Dobkin, D.P., Kirkpatrick, D.G. *Fast detection of polyhedral intersection*, Theoret. Comput. Sci., 27 (1983), 241–253.

[14] Fuchs, H., Kedem, Z.M., Naylor, B. *On visible surface generation by a priori tree structures*, Proc. SIGGRAPH '80, Comput. Graph. 14 (1980), 124–133.

[15] O'Rourke, J. *Art Gallery Theorems and Algorithms*, Oxford Univ. Press, New York, NY (1987).

[16] Mitchell, S., Vavasis, S. *Quality mesh generation in three dimensions*, Proc. 8th Ann. ACM Symp. Comput. Geom. (1992), 212–221.

[17] Ruppert, J., Seidel, R. *On the difficulty of triangulating three-dimensional non-convex polyhedra*, Disc. Comput. Geom., 7 (1992), 227–253.

[18] Schröder, P., Hanrahan, P. *On the form factor between two polygons*, Proc. SIGGRAPH 93, ACM Press (1993), 163–164.

[19] Snyder, J.M., Woodbury, A.R., Fleischer, K., Currin, B., Barr, A.H. *Interval methods for multi-point collisions between time-dependent curved surfaces*, 321–334.

Appendix

A Geometric Model for SAT. Let $x_1, \ldots, x_n$ be n Boolean variables; an instance of SAT consists of n clauses $c_1, \ldots, c_n$, each of them a disjunction of literals x_k or $\bar{x}_k$. Each clause c_i is represented by a vertical rectangular strip C_i of width ε, for some small $\varepsilon > 0$. We position C_i so that it lies in the xz-plane and intersects the x-axis in the interval $[i, i+\varepsilon]$ (Fig.2). There is no need to specify the exact length of the strip: it should simply be "long enough."

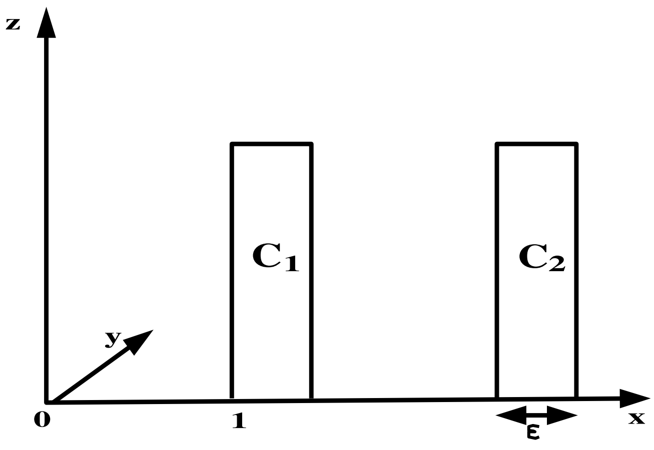


Figure 2:

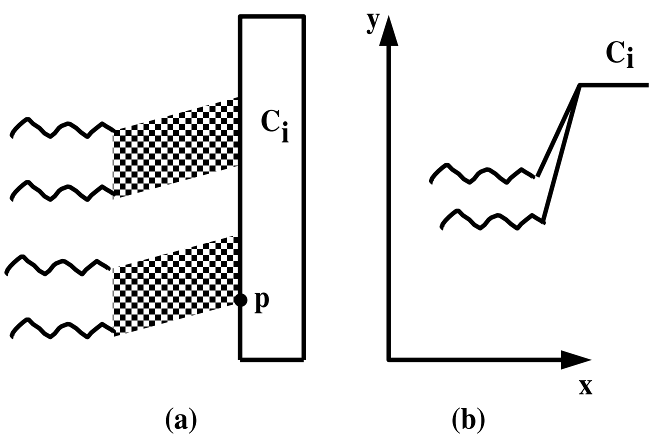


Figure 3:

Clauses are connected together by closed accordeon-like strips that represent the variables. Each variable has its own small z-interval within which its strip lives. We begin by discussing how the strip X_k for variable x_k is locally attached to a clause strip. Suppose that clause c_i contains the literal x_k. Then the strip X_k is attached to C_i around the point $(i, 0, k)$ in the following fashion: two coplanar (congruent) rectangles parallel to the z-axis abut C_i at a 45-degree angle. Both of these *attachment rectangles* are of width ε and are separated from each other by ε: they are the dashed regions in Fig.3a; point p has coordinates $(i, 0, k)$. In projection on the xy-plane, the two rectangles form two coinciding edges: as in Fig.3b, it is convenient to think of these two edges as forming a small but nonzero angle (even though strictly speaking the angle is zero). One final word about attachment rectangles: we slightly perturb them so that the contact angles of any two attachments should be distinct (but very near $\pi/4$ nevertheless). Within a given attachment pair, however, the two rectangles should be kept coplanar.

How do we connect all these attachment rectangles together? Rather than giving long formal definitions, we illustrate the construction on a representative example. Suppose that C_1, C_2 are the only clauses containing x_k and that $\bar{x}_k$ does not appear in any clause. We join the four corresponding attachment rectangles together by merging them into a single simple closed strip X_k of width ε, which lies entirely within $\{ k \leq z \leq k + 3\varepsilon \}$. The projection of X_k on the xy-plane is a simple closed polygonal line L_k (Fig.5). We impose two requirements:

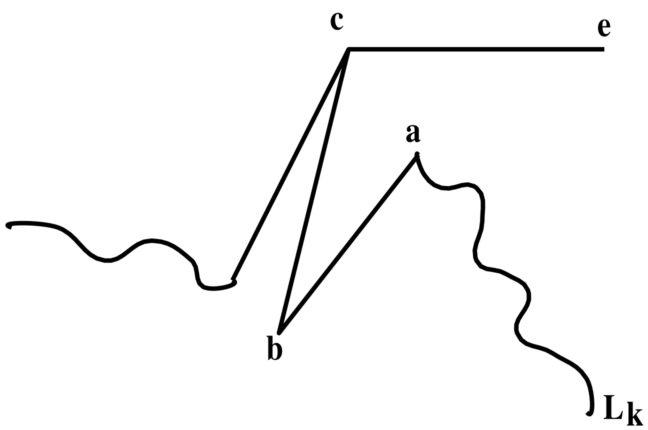


Figure 4:

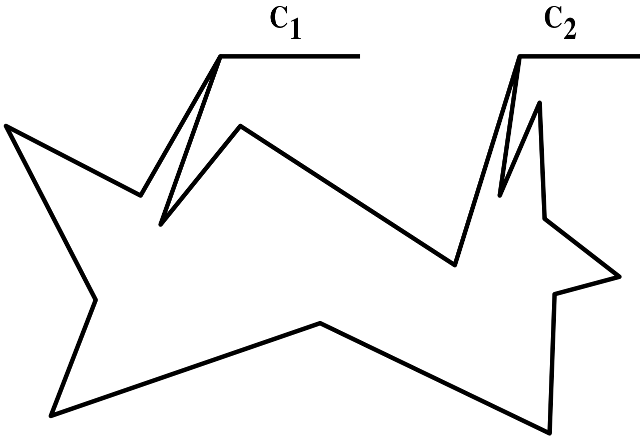


Figure 5:

1. A walk around L_k should constantly alternate between left and right turns.

2. Let $abcd$ be the portion of L_k abutting the (projected) clause strip ce, with $\angle(bce) \approx 3\pi/4$ (Fig.4). We require that the angle $\angle(abc)$ should be small enough that the quadrilateral $abce$ is *not* convex.

3. The complexity of L_k should be linear in the number of clauses containing x_k or $\bar{x}_k$ (here, the number is 2).

Instead of pursuing the construction to treat the other cases, it might be useful at this point to pause and motivate the three requirements on L_k. The last one indicates that we do not care to minimize the size of L_k (as long as it remains linear or even polynomial): this way, the strip X_k has plenty of room to wiggle leisurely between its attachements to clause strips without self-intersecting or intersecting other strips (Fig.5).

The first requirement implies that there are exactly two optimal ways of decomposing L_k into convex polygonal curves. One is called *convex-cut* and the other one *concave-cut*: a small, self-explanatory example is given in Fig.6. The convex-cut decomposition of L_k induces a convex patch decomposition of the strip X_k that can be made to include the whole clause strips C_1, C_2 as well. Note that the 3-facet patches formed around the clause strips are convex because the two abutting facets are –despite appearances to the contrary– coplanar;

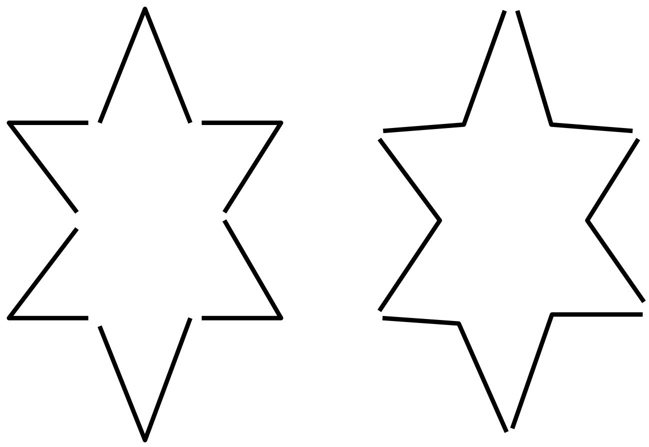


Figure 6:

Similarly, the concave-cut decomposition of L_k induces a convex patch decomposition of X_k, but now because of the second requirement, it is impossible to include any clause strips (one might be able to attach a little chunk of it but not the whole strip). Logically, the convex-cut (resp. concave-cut) decomposition corresponds to a true (resp. false) assignment of x_k: By including the strips C_1, C_2 the decomposition is able to "satisfy" the corresponding clauses. On the other hand, the concave-cut decomposition is unable to satisfy any of them.

How do we handle the case of negated literals? Suppose that C_2 contains $\bar{x}_k$. (Obviously we can assume that no clause contains both x_k and $\bar{x}_k$.) We must now ensure that convex-cut cannot include C_2 but that concave-cut can. For this we need to place the attachment to C_2 on a concave vertex of X_k, which is achieved in straightforward fashion (Fig.7). We call this kind of attachment *negative*; the other kind is called *positive*.

This completes the construction of the geometric model of the boolean formula. We claim that it is satisfiable if and only if there exists a convex patch decomposition of the geometric model of size $N = \sum_k |L_k|/2$, where $|L_k$ is the number of edges in L_k.

Suppose that the formula is satisfiable. Then, apply to each X_k the particular decomposition corresponding to its truth assignment. For each clause c_i at least one pair of attachement rectangles is covered in a single patch (pairs whose literals satisfy c_i). If the pair is unique we easily extend the patch to cover all of C_i. If there are several, we must be careful not

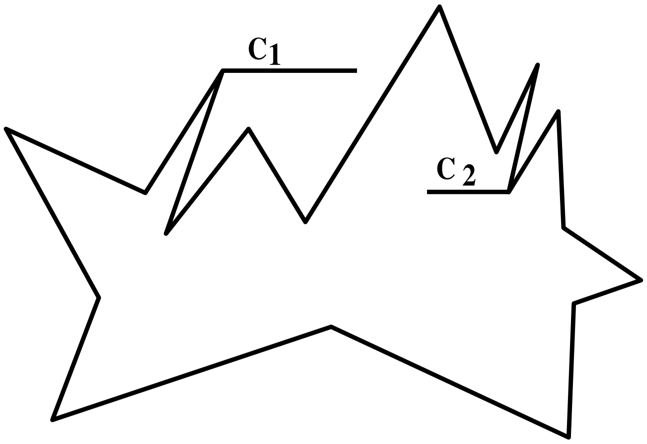


Figure 7:

to disconnect existing patches. We create fictitious border lines across C_i to separate the "satisfying" attachment pairs from each other, and we flood each corresponding patch within its borders. In Fig.8 the clause is satisfied by x_1, x_3, and $\bar{x}_4$, but not by $\bar{x}_2$: we need two borders; one to separate the x_1 patch from the x_2 patch, and the other one to keep the other satisfying patches (for x_3 and $\bar{x}_4$) separated.

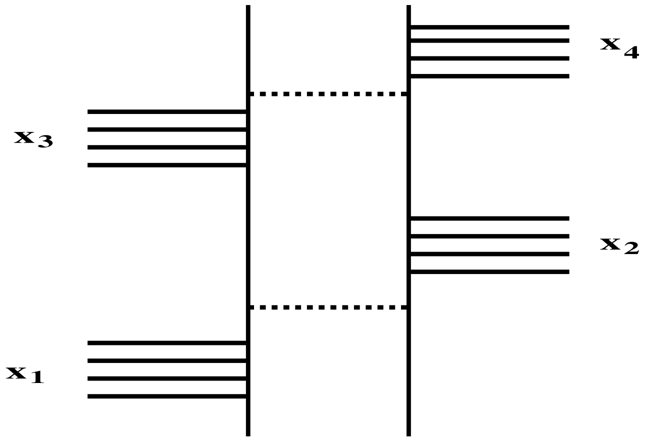


Figure 8:

We now show the converse, ie, that if the surface can be decomposed into N convex patches then the formula is necessarily satisfiable. From now on, all convex patches are understood to be part of such an N-patch decomposition $\mathcal{D}$. We identify a number of special points, which we call *sites*.

- For each clause strip C_i, declare the point $(i+\varepsilon, 0, 0)$ to be a *clause site* (this is the point on the bottom right side of the strip).

- For each variable strip X_k, declare the center of gravity of each facet to be a *variable site*.

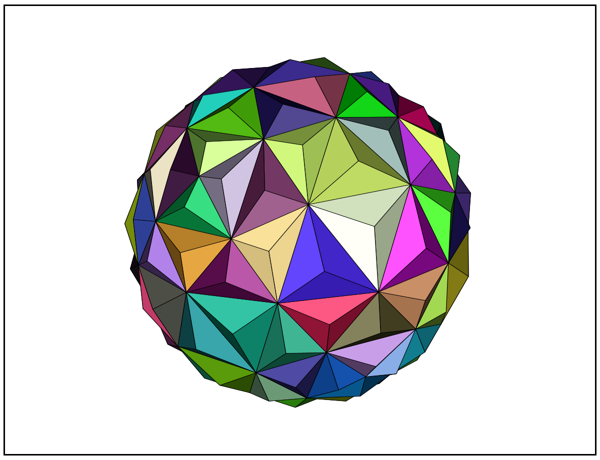
Note that the total number of variable sites is $2N$ and the number of clause sites is n. Let G be the graph whose vertices are the variable sites and whose edges connect vertices whose associated sites are covered by the same patch in the decomposition $\mathcal{D}$. The following result establishes the connection between minimum decompositions and truth assignments.

Lemma 5.1 *The graph G is such that: (i) no edge connects two variable sites on different variable strips; (ii) the set of edges forms a perfect matching.*

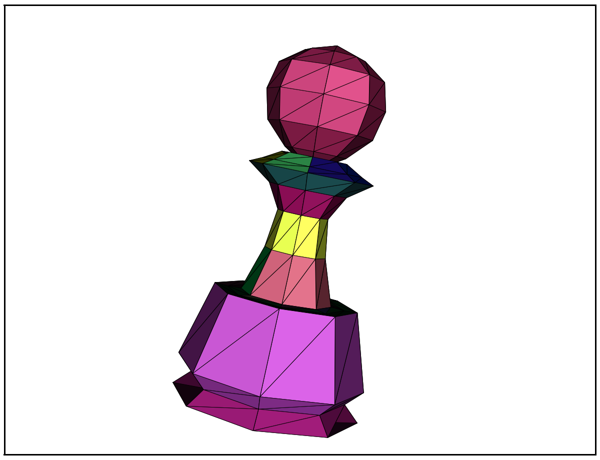
Proof: To begin with, observe that no convex patch can cover more than two variable sites. This is even true of a patch that contains the two variable sites associated with a given pair of attachment rectangles of X_k. Indeed, because the patch must be connected, it must also include at least part of the incident clause strip: this prevents it from including any part of other facets of X_k. For the same reason, the perturbation of attachment rectangles makes it impossible for a patch to contain two variable sites on distinct variable strips. It follows that a patch can contain no more than two variable sites, both of which must belong to the same variable strip. If the decomposition is N, however, then each patch must, indeed, cover two variable sites. The graph thus forms a perfect matching. $\Box$

There are exactly two ways of forming a perfect matching among the variable sites of the strip X_k. From this we derive a natural truth assignment for x_k, as explained earlier. We declare the variable to be true (resp. false) if and only if there exists a matched pair of variable sites on a positive (resp. negative) attachment pair. Note that the perfect matching ensures the consistency of the assignment, ie, if a pair of variable sites on a positive attachment pair is matched, then all others are, and none of those on negative attachments are (and vice versa).

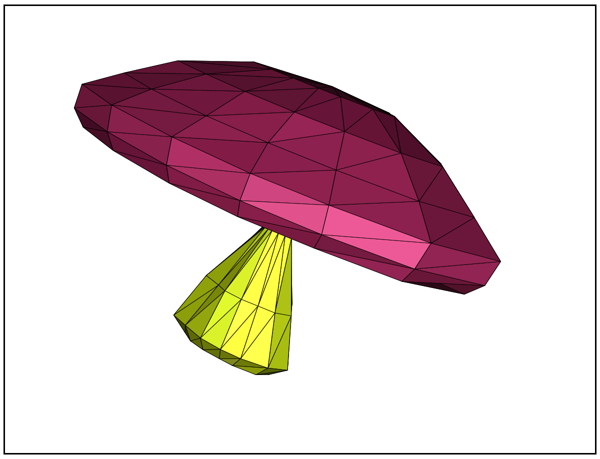
Given C_i, its clause site must be covered by some patch used in the perfect matching. By convexity, this patch must be one that covers the two variable sites on some attachment pair for some X_k. This patch specifies an assignment of x_k that satisfies c_i. It follows that every clause is satisfied, which proves that the formula is satisfiable. This completes the proof of NP-completeness. $\Box$



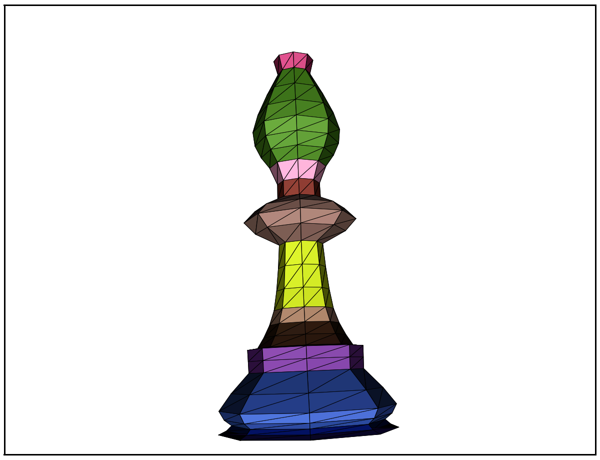
Epcot



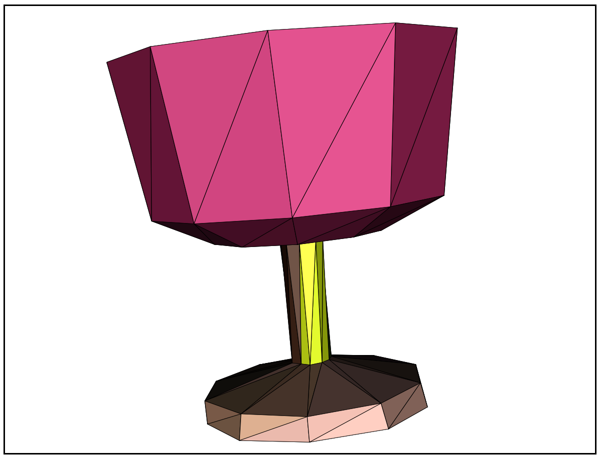
Pawn



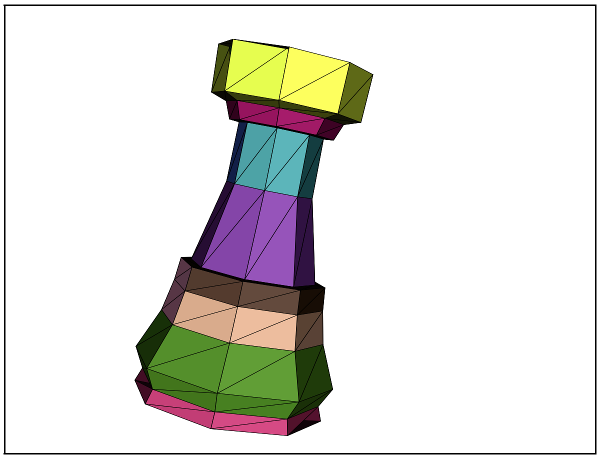
Mushroom



Bishop

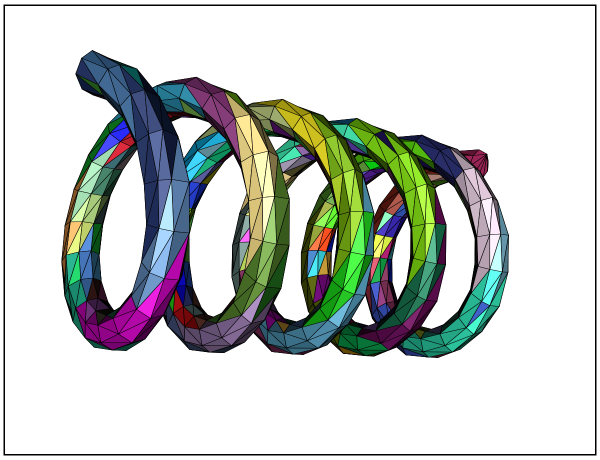


Glass

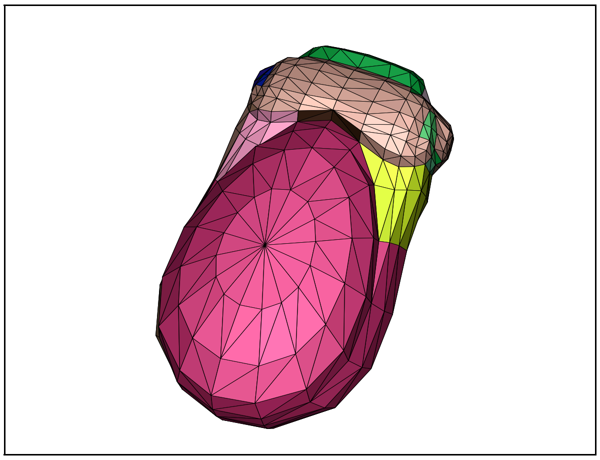


Rook

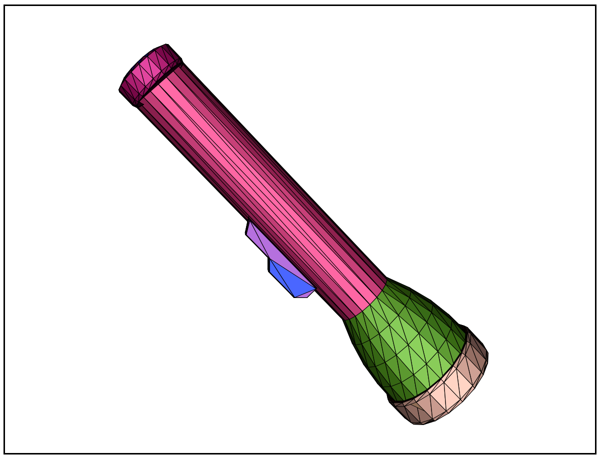
Plate 1



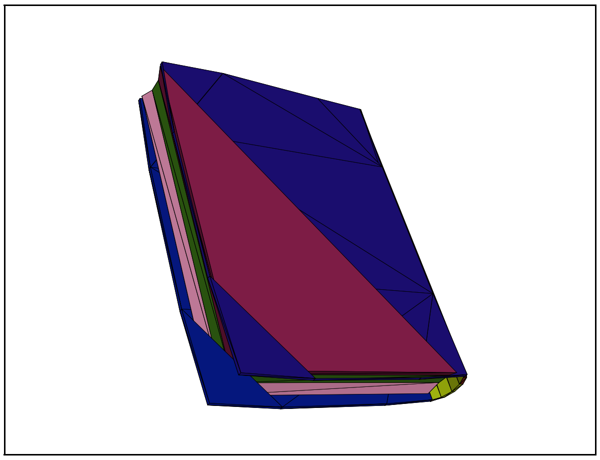
Spring



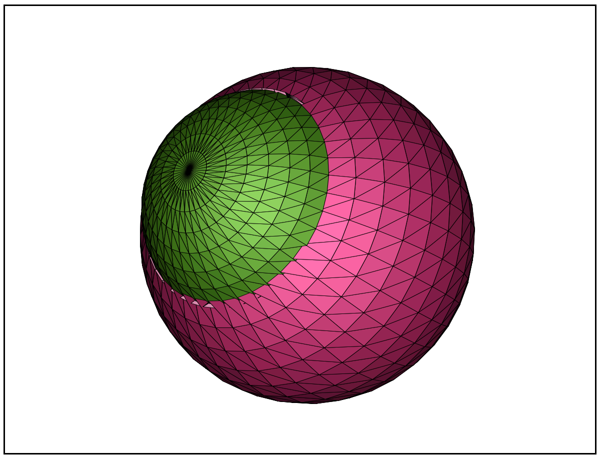
Torso



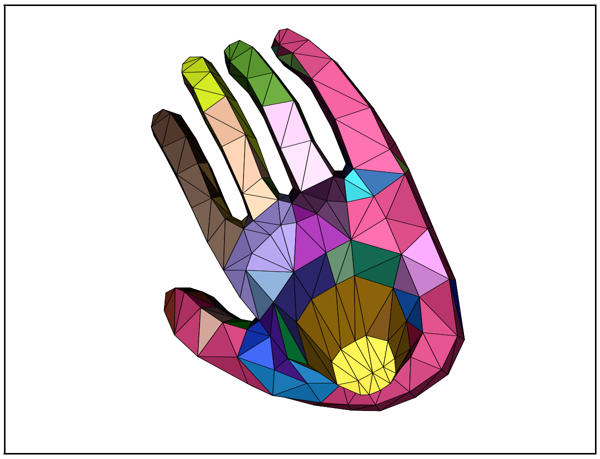
Flashlight



Book

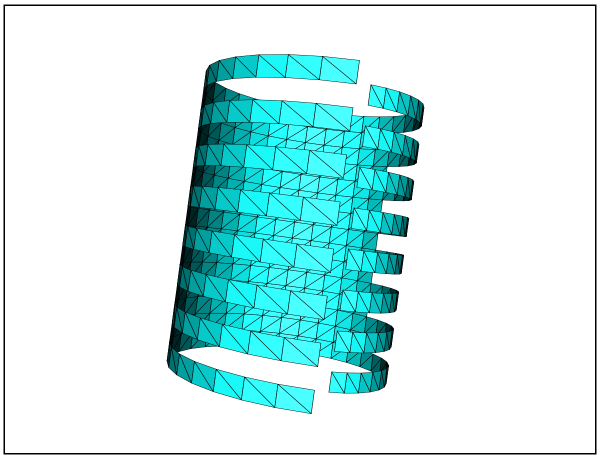


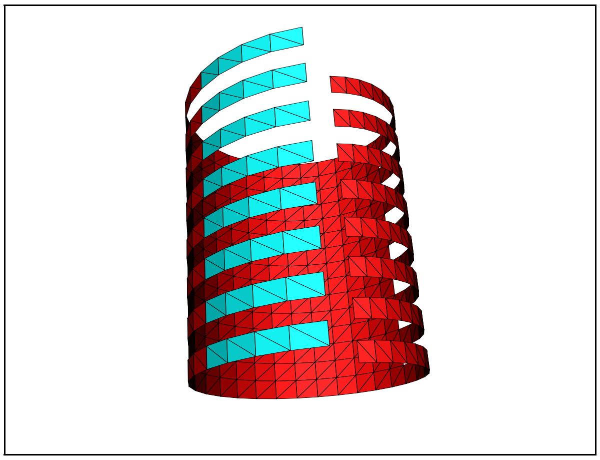
Eyeball

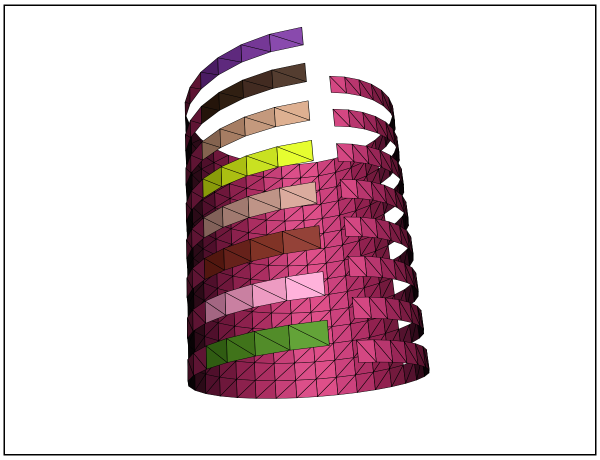


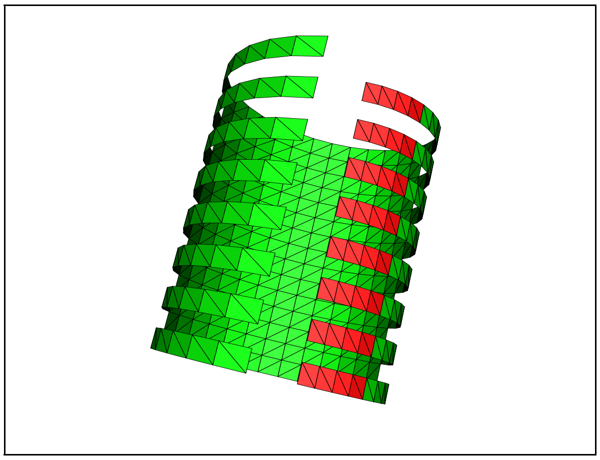
Hand

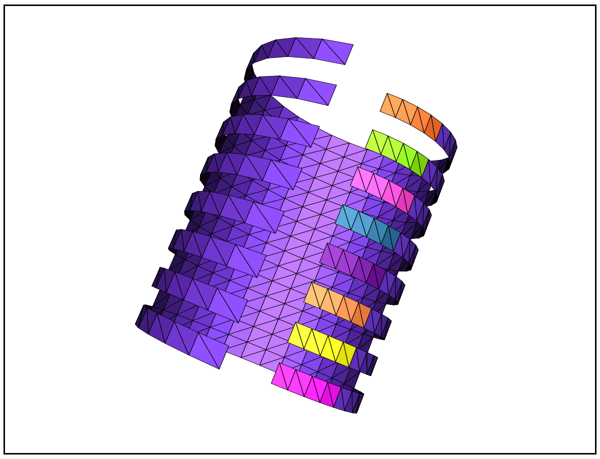
Plate 2


Bracelet


Flood-and-Retract 1


BFS


Flood-and-Retract 2


DFS

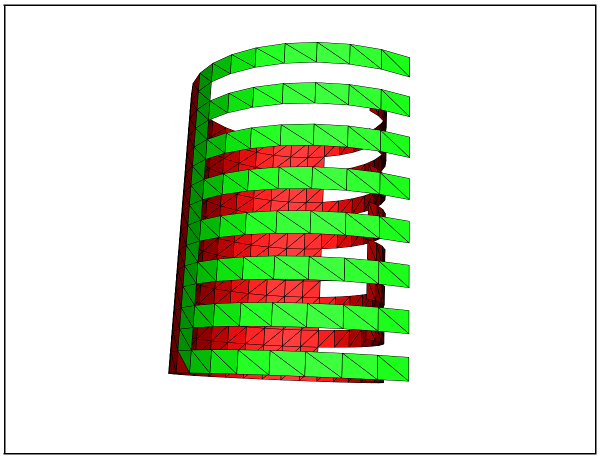

Flood-and-Retract 3

Plate 3