The Unfolding Problem

Alex Benton

1 The Problem

1.1 Overview

The *Unfolding Problem* can be succinctly described as "How to peel an orange... no matter what shape the orange is." It is the question of how to 'unwrap' a 3D polyhedron, breaking some of its edges or faces so that it can be unfolded into a flat net in the 2D plane. From there the flattened net of faces might be printed out, cut from paper or steel and folded to recreate the virtual model in the real world.

In 1525 the artist *Albrecht Dürer* used the term 'net' to describe a set of polygons linked together edge-to-edge to form the planar unfoldings of some of the platonic solids and their truncations. Dürer used these unfoldings to teach aspiring artists how to construct elemental forms, but today the applications for solutions to the unfolding problem lie in a broad range of fields, from industrial manufacturing and rapid prototyping to sculpture and aeronautics. In the textiles industry, work has already begun in computing digital representations of fabric and trying to flatten those representations to optimize seam and dart placement [MHC05]. There has been similar work in the fields of paper-folding [MS04] and origami [BM04] and even ship and sail manufacturing. Advances in robotics and folding automation [GBKK98] have brought with them a new need for faster, more robust unfolding methods.

If a polyhedron can generate a net which is not self-intersecting, solely by breaking a subset of its edges and flattening the join angles of those which remain, then it is called *edge-unfoldable* or *developable*. At present, it is strongly believed–but not yet proven–that all *convex* surfaces are developable. In counterpoint, examples are easily found of non-convex surfaces which are cannot be edge-unfolded, but no robust solution yet exists for testing whether or not a given mesh will prove to be developable.

1.2 The Goal

This project asks the question,

> Given a polyhedral mesh M, can M be cut along its edges and unfolded flat into a single nonoverlapping piece?
> - How quickly can the question be answered?
> - How quickly can an unfolding be found?
> - What does it mean to do the job 'well'?

Earlier this year, I devised brute-force approaches which will always find the unfolding of any mesh, if such an unfolding exists; unfortunately their running time is exponential in the number of faces on the polyhedron, and a simple mesh of twenty faces can take longer than three months to solve on a state-of-the-art PC. When I first began this research, my goal was simply to find an unfolding for an arbitrary mesh; having succeeded at this already, in a manner which is functionally useless because of its running time, my goal is now to solve the problem again, this time in sub-exponential time.

I would also like to investigate what it might mean to find the 'best' unfolding, to do the job 'well'. Most applications will impose limitations on the form that an unfolding can take. Some might need to minimize the length of the border of the unfolded net (for instance, in ship hull manufacturing, where the goal is to minimize welded seals[1]) while others would seek to minimize the area of the bounding rectangle (steel stamping) or favor long, linear unfolded patterns (woodworking). These constraints add an interesting twist to the problem, imposing new minima requirements on the process.

This project is concerned only with geometry which cannot be modified in any manner before unfolding, and which must be unfolded across its edges to a single connected net of faces. It seems to me that in the real world, any application that had reduced its data to polygons probably did so for a reason; best to preserve them. Limited solutions already exist for cases which relax these restrictions [AAOS93].

[1] Seals are protected under the Arctic Wildlife Treaty. This project does not advocate welding them.

2 What has come before

2.1 Existing research

The 1990's saw a surge of research on the unfoldings of convex polyhedra. O'Rourke et al [AAOS93] described the *star unfolding*, which could unfold any convex polytope by cutting across its faces. Namiki and Fukuda sparked further interest in the field with a Mathematica plugin [NF94] capable of unfolding convex surfaces, and today several pieces of commercial software offer convex unfolding solutions [SS][T3D].

Unfolding solutions break down into three broad classes [O02]:

(1) *general unfoldings* which split the faces of the surface;
(2) *edge unfoldings* which split only the edges of the surface;
(3) recently *vertex unfoldings*, which split edges and retain only connectivity at vertices to create long chains of vertex-linked triangles.

Of these three general unfoldings have been shown to be most powerful [BDEKMS99].

Solutions exist which will unfold a mesh into many separated components. Costly iterative loops can then be applied to re-join the disparate pieces, but the running time of the rejoining processes are invariably poor. An open task remains to be able to determine, for a given undevelopable mesh, the *lower bound* on its number of flattended components. Clearly, the unfolding problem cannot be solved without such a lower bound; without the constraint that separated pieces be limited, the worst- case limit of an unfolding of a mesh of n faces would be n disjoint polygons in a plane. Not a sufficient solution.

To date, it has been shown that all *convex* surfaces may be unfolded through the star unfolding and that this method generates an infinite number of unfoldings [AAOS93]. (The star unfolding is a "general" unfolding.) It has also been demonstrated that even relatively simple non-convex surfaces can not always be edge-unfolded[BDEK99], [BDEKMS99]. Examples have been generated of non-convex surfaces which are unfoldable if closed but which can be rendered undevelopable through the removal of a single polygon [BDEK99]. Several simple algorithms have been advanced for the reasonably quick edge-unfolding of a convex surface, but it appears that no single approach yet devised is guaranteed to be effective on all surfaces [F97].

Relatively little research has been published dealing with analytical approaches to finding the unfoldings of non-convex surfaces. O'Rourke summarizes the current research nicely in [O02].

2.2 What I had done before Cambridge

I first became interested in the unfolding problem in 1995 and began to investigate it seriously on my own time in late 2003. By the time I entered Cambridge in the fall of 2004, I had built a test harness which could load 3D models and try to unfold them. The YAMM project, (which stands for "Yet Another Metaball Modeler"–I liked the surfaces that implicit surface modeling could give me, because they were so well-behaved) was written in OpenGL and C++. My models were free to be non-convex and non-simplicial. To these models I could apply a suite of Unfolders, a set of algorithms implemented in the YAMM framework to do their level best to unfold a developed net.

I decided early on to focus on the edge-unfoldings of non-convex polyhedra. Any solution I could devise for the more difficult cases would trivially solve the simpler questions.

I had asked the questions,

- Can the geometric or topological attribute that makes a non-convex surface undevelopable be determined programmatically?
- Is it possible to determine whether or not an edge-unfolding can exist for a given non-convex surface without actually attempting to find it?
- If a surface cannot be edge-unfolded, can an algorithm be designed to generate only the minimum number of separated unfoldable components?
- What other constraints can be applied to the unfolding?
- What are the concrete applications for this line of research?

My first unfolders were uniformly heuristic in nature. Each unfolder treated the source mesh purely as a connectivity graph irrespective of geometry and attempted to find an unfolding through different orderings of its traversal of the graph. The simplest were breadth-first and depth-first traversals: choose a seed polygon arbitrarily (usually the bottom-most face of the mesh, for aesthetic reasons) and then unfold outwards from that face's node in the connectivity graph, ordering the nodes with a breadth- or depth-first search, declaring failure the first time an overlap was generated. Then I tried spiraling through the model, looking at the local geometry just enough to find the next polygon to lay down in a growing spiral pattern, which produced aesthetically pleasant (but not terribly effective) results. These solutions, which I named the BFS (*Breadth First Search*), DFS (*Depth First Search*), and Spiral Unfolders, would often work well on convex models but it was blind luck for them to succeed on non-convex meshes.

In addition to the basic heuristic approaches, I also dabbled with a few constrained solutions. I designed unfolders which attempted to minimize the area of the convex hull of the unfolding net; the cross-section of the net; the length of the longest gore in the net; or the greatest distance from the initial seed face. These side trips were interesting and thought-provoking, but did not in general bring me closer to an actual solution. Generally they used variants of the BFS and would project one move ahead, selecting the next move that minimized the chosen function; since such a stepwise minimization approach is known not to always generate the actual minimal value, I know that their results did not always achieve the minimizing goals even when the unfoldings themselves succeeded.

One of the discoveries that I had already made by 2004 was that the problem itself was actually two different issues layered one atop the other. What I had initially assumed would be purely a question of topology turned out instead to be twin issues of geometry and local behavior. I found that the overall shape of a model seemed to be far less important than local points of extreme behavior; creases, boundaries and saddlepoints all seemed to be far more relevant to success or failure than whether the surface as a whole was topologically equivalent to (for example) a sphere or a torus. In fact, at first glance I was becoming convinced that surface genus was actually irrelevant to the problem. I also believed that mesh density was consistently a real factor in success or failure; certain minutiae, per-vertex behaviors on a face-by-face basis, could determine the success or failure of an entire unfolding. Unfortunately I had had no luck whatsoever in avoiding these special cases; at best I had produced a few heuristics which *tended* to avoid them.

Based on this, I had concluded that the unfolding problem was really two problems in one: what I thought of as the "micro" problem, avoiding per-face collisions on adjacent gores; and the "macro" problem, issues that arose from the overall geometry of the surface.

My early work had very few deterministic, reproducible testing metrics. My test harness was designed to let me try one after another of various unfolding approaches until one worked. My primary metric of success was just that: a binary true or false of "did it work on the given model". Unfortunately, almost all the models that I had were either complete successes or complete failures across the board; only a few of the cases distinguished between the unfolders. Trivial perturbations would often upset that balance, indicating that it had simply been the luck of an ordered generating function that made those shapes so well-behaved.

One model that I did find to be quite useful was a sphere generated from an implicit surface. The sphere was amiably ill-behaved: it had no border and was uniformly convex, but by generating the model with an octree reduction method I had a model whose faces varied wildly in size and angle, not randomly but almost so. This variability proved to be a surprisingly good test for would-be solutions to the "micro" portion of the problem.

3 New Findings 2004-2005

Over the past year, I have attempted to refine my understanding of the Unfolding Problem and to begin to solve it. My approaches have grown from the mindlessly heuristic to the completely mindless, but thence to more geometrically-sensitive tacks; I have begun to identify the numerical bounds on the problem and to find upper and lower bounds to possible solutions. Perhaps most importantly, I can now show that the problem is, without a doubt, difficult.

I have refined my definitions of the key players in my little drama. I have formally defined a *polyhedron* as the connected union of a set of intersections of open half-planes in $\Re_d$ such that no intersection lies in a proper affine subspace of $\Re_d$ [2], and a *mesh* as the surface of a polyhedron, expressed as a union of faces, edges and vertices [3]. These definitions allow me to cleanly distinguish between the mathematical abstraction of a surface, its geometric (and analyzable) expression in mathematics, and its implementation [4] in software.

I have identified the data structures which appear consistently throughout my work and can name and discuss each graph. The source connectivity graph allows me to model initial structure; the cut-tree graph models final output. The manner in which these data structures are traversed determines the running time and complexity of any attempt to solve the Unfolding Problem. In fact, they provide an alternate approach to defining the problem: one could readily say that the Unfolding Problem is the task of finding, in minimal time, the cut-tree of a mesh.

I have made two major additions to my testbed this year, each of which depended on and extended my understanding of the data structures that lurked behind the scenes.

Early in the year I designed three brute-force unfolders, systems which would in one manner or another attempt to evaluate every possible unfolding that a mesh could achieve. These unfolders would iterate across every possible net until they found one with a valid unfolding tree (non-looping, non-self-intersecting.) Unfortunately, the number of possible nets for a model with d edges and n faces is $O(d^n)$ [5]. The brute-force unfolders were clearly suboptimal solutions to the problem.

Today I brush off those early, mindless avenues as though they bore no fruit,

[2] See **Definitions** for a more verbose formulation of this description

[3] See **Definitions** for the rules guiding a Mesh

[4] See **Implementation** for the full set of rules, relaxations and extensions defining the behavior of a PolyMesh

[5] See **Factoids** for proof.

but that wasn't really true. The brute-force approaches lead me to a number of new insights into the problem and highlighted many optimisations which worked for any approach, such as optimising for seed face and not focusing on ordering. The 'mindless' tacks also highlighted the true **scale** of the problem and allowed me, for the first time, to describe the upper bounds of the unfolding problem formulaically.

Brute force had failed me. Undaunted, I faced adversity by making the problem even worse: I asked the question, "Now that I know how many nets a given mesh could possibly generate, how many ways are there to actually construct any one of those nets?" Mindful of the complexity of the problem, I chose to trace my way through the possible nets and their possible assemblies by constructing a tree of all options. I hoped to build a data structure which encapsulated all options without redundantly storing faces that were shared by multiple nets. To this end I devised the MegaTree, a partially-directed graph of options, which drives my Total Unfolder algorithm. One path through the MegaTree equals a single unfolding, valid or otherwise. The running time of the Total Unfolder for a model with d edges and n faces is

$$F(n) \sim O((d^k)((k-1)!)), \text{where} k = (nd-1)/(2d-2) \tag{1}$$

From the Total Unfolder I moved on to write two more brute-force approaches. The Precomputed and Iterative Unfolders were both significant improvements, running in O(d^n) time; the Precomputed Unfolder actually runs a bit faster. The Iterative Unfolder gave rise to a *very* interesting insight, the precise import of which I have as yet been unable to determine, but still, interesting:

> Given an ordering of the faces and vertices of a mesh, one can uniquely identify an unfolding net with a numerical index. This numerical value is formed by using the index of each face as an order index into a base-d number with n digits. The value of the ith digit of the number is the index of the edge across which the ith face was unfolded.

This implies that one could compress the expression of an unfolding down to an integer of at most n * ceil(log$_2$(d)) bits.

Having gone as far as I felt reasonable with the brute-force unfolders, I moved on to a new tack: curvature-sensitive unfolding. The Curvature Ordering Unfolder is a geometrically-aware heuristic unfolder. Every face is assigned an "average curvature[6]", the average of the curvatures of the vertices of the

[6] After extensive reading on the matter, I've settled on using the *angle deficit* as an approximation to the discrete curvature of a surface at a vertex. The angle deficit is the number of radians by which the sum of the angles of each neighboring face at the vertex approaches 2pi; positive angle deficit implies a rounded curve,

face. The face with the highest curvature is then chosen as the root of the net and all subsequent unfolding flows from there. At every pass, the next face to unfold is chosen from the boundary set of the current unfolding selecting the face with the next greatest curvature. This leads to an unfolding tree whose branches and leaves decrease in curvature steadily as they get further from the original root, with the most negative of leaves at (hopefully) the extremities.

In practice the Curvature Ordering Unfolder was a good step forward but not the end of the road. It assumes that the most negative areas of curvature are as far as possible from the most positive, and that splitting them will give the necessary room in the 2D plane for the unfolding not to overlap. The current implementation addresses only maximal critical points in the curvature field, but should take minima and saddle points into account. There is currently no attempt at handling local perturbation, so while it does well on mathematically-generated surfaces, real-world models remain problematic.

I've noticed that in many of the more regular models, especially those generated from parametric functions, there are often strips or patches of faces with identical (within the limits of floating-point tolerance) curvature. I've also noticed that the COU selects the next face in an unfolding from what can be a pool of faces with identical curvature; the selection is made based on order of addition to the pool, which means that the choice of face is effectively an artifact of the generation algorithm, a side-effect of data entry. This could easily lead to artifacts in the unfolded output, which inspired the Tracing Curvature Unfolder. The TCOU overrides the COU's next-face selection routine: when there is more than one candidate face to be laid down and all of the possible faces have identical, maximal curvature, the TCOU selects the face which is closest to directly along the vector perpendicular to the gradient of the flow of curvature from the last face which was added to the unfolding. This imparts a sort of 'velocity' to the unfolding, but it also guarantees that the selection method is much less an artifact of the heuristic implementation and much more determined by the actual geometry.

One side route that I took with the Curvature Ordering Unfolder was to build the Multipart Curvature Ordering Unfolder.[7] The original Curvature Orderer was based on starting from a single seed point, but that was clearly

zero deficit implies a flat plane, and negative deficit implies a saddlepoint or other locally-non-convex geometry.
The angle deficit method of calculating discrete curvature at a vertex is extremely simple to implement, but not as comprehensive as other methods which have been proposed. For instance, the angle deficit at every point on a ball of crumpled paper would be zero, but obviously the curvature of the paper is not zero.
In my implementation, vertices on the boundaries of the surface are arbitrarily assigned an angle deficit of -2pi.

[7] I suspect that I undertook this task just to see how long the name could get.

not a reasonable goal; many models have many extrema or extremal regions. So the Multipart modification was to support a series of ‘seed’ faces, faces far apart on the model which would all become the germinal centers of unfoldings growing in parallel; when they met they would bond across a shared face. The implementation still has a few bugs to work out, and I do not yet have any performance measures.

To assist my research I have built the beginnings of a library of test cases, models which in one manner or another are difficult to unfold. My favorite so far has been the cow (cow.wrl), a VRML file I found online which combines broad expanses of gentle positive curvature with tight areas of highly positive or negative curvature. The cow is not actually a perfect test case, because its feet and legs make it a little vulnerable to algorithms that benefit unduly from long linear runs, but it’s still an effective (and amusing) test. I’ve found that the number of faces that an unfolder can lay down out of the 5,370 triangles that make up the cow’s simplicial mesh can be an effective index of the method’s effectiveness:

Unfolding Method	Time (P-III 750)	Num faces	Percent of Cow
BFS Unfolder	1m:36s	3509	65.34
DFS Unfolder	1m:36s	4012	74.71
Spiral	2m:16s	3607	67.17
Curvature Ordering	8m:11s	4681	87.17
Curvature Tracer	8m:36s	4691	87.36

In addition to the cow I also test on a set of parametric surfaces, bounded functions from parametric space (u,v) to 3D space (x,y,z):

- Sphere: This surface of universally positive curvature should be unfoldable to every approach.
- Valley: $y = u^2 + v^2$. This surface of universally positive curvature should be unfoldable to every approach.
- Saddlepoint - $y = u * v$. This surface of universally negative curvature cannot be unfolded.
- Torus: The torus induces loops in the cut-tree, unlike most convex surfaces. Many of the mindless heuristics fail here.
- Inner Half-Torus - The inner half of a torus’ ring, this uniformly negative surface does not appear to be unfoldable.
- Spun parabola: The surface of rotation $r = y^2$ spun about the Y axis. Despite the fact that its curvature is negative throughout, the spun parabola is actually unfoldable–but only by a spiral line tracing almost-horizontal loops

around the Y axis.
- ...and various other entertaining models.

An interesting mirror to the timing tests is to count the number of possible *solutions* to an unfolding for a given model. I've modified a few of the unfolders not to stop after success; the Total Unfolders can be run so as to count the total possible number of ways to generate all of the possible unfoldings for a mesh, beginning from a fixed seed face. Obviously, the exponential increase in time to find an unfolding means that I've been unable to apply this method to any but the simplest shapes, but I was intrigued to learn that there are 3,968 different orders in which one can unfold five of the faces of a cube about the sixth.

Before my arrival at Cambridge, I was convinced that the unfolding problem was a two-part thorn: micro and macro. I have made some small progress in resolving the micro problem, using a hash map in my implementation to steer clear of troublesome edges. In the past year I have begun to suspect that there is a third layer to the question, "topo", and that a successful solution will need to address all three layers of the question in parallel. In fact, it is distinctly possible that some meshes exist which could be unfolded were it not for failing some critical criteria in one of these three fields. Where the micro issue lay at the vertex and face level and the macro lay in the angles and lengths of groups of faces, the topo problems I've begun to detect seem to derive from the twisted interconnections of more complex models. I suspect that by using topological knowledge to identify macro portions of the model and then reducing the problem to segments and their interactions, I might be able to gain new traction on the problem.

I've begun to find limited answers to the question of "can a surface be analyzed without unfolding it?". Any surface of uniformly positive curvature is unfoldable, but this is not news; Fukuda had already shown it with [F97]. But I have found no research into surfaces of uniformly *negative* curvature, nor into surface regions of negative curvature *entirely surrounded by positive regions.* I believe that identifying these classes of surface could allow me to identify their unfoldability early on. The spun parabola, the only uniformly-negative surface that I have found that I could unfold, is topologically cylindrical, but so is the inner torus model–which is NOT unfoldable. Clearly, more work is needed here.

A few other random facts[8] that I've reasoned out this year include:

- Removing edges reduces unfoldability
- The genus of the source shape is not tied precisely to the genus of the cut-tree

[8] As elaborated upon in the **Factoids** appendix

- The necessity of cutting vertices with negative curvature
- Ordering doesn't really matter
- Starting face doesn't matter

4 Where to go from here

In the next two years, I will continue to wrestle with the unfolding problem. I can organize the work ahead of me into two broad categories: new approaches which call for investigation and the underlying questions which drive those approaches. A very rough estimate of how long it might take me to examine the points below, based on my performance to date, leads me to suspect that to address every concept below should take me about two and a half years. Hopefully past performance is not indicative of future results; I would like to lay these questions to rest in at most the next year to a year and a half, especially because I want to move on to the more interesting issues that I am sure will arise while investigating those discussed here.

In general, my methodology has been to brainstorm new approaches; do a literature search to see whether they've been tried before; extend the search to techniques of implementation; and then if they still seems to bear examination, write a testbed unfolder to examine the techniques in action. I then catalogue each approach in an evolving document listing all of my attempts and move on to the next. Often, I devise new variations on an approach in the course of implementing it, which I then test in turn.

4.1 New approaches to try

The Curvature Ordering Unfolder sorts faces in favor of greatest curvature. I'd like to try sorting by greatest distance from areas of negative curvature instead. If I do, I'll have two tacks to try out: measuring distance by the length of the shortest path between the face and the regions of negative curvature, or measuring distance by the number of edges crossed along that shortest path. While the former might seem to be the more intuitive, the latter addresses the real concern: that the distance is a first-pass approximation to the number of faces which can be laid down between this polygon and those of negative curvature. So I'll want to try both concepts of 'distance' and see what I get from each.

One update that I do want to make to my system is my curvature implementation. The angle deficit description of curvature is really insufficient for my needs. Meyer and Desbrun give a more evolved form of the angle deficit calculations in [MDSB03], but I'm tempted to try Polthier and Hildebrandt's method [HP04] or Lyuba Alboul's approach [A02] if I can calculate them reasonably quickly for local surface features.

I also intend to refine my concept of a "shortest path". Instead of just a geometric or edge-counting weight, I could use these weights in conjunction with a function which maximizes curvature along a path–solve for the shortest path on a surface where the weight of each step in the path is a function not only of distance but of curvature. The resulting path will not be physically the shortest but it would cross the greatest amount of positive curvature.

As an entirely new approach, I'd like to try segmenting the surface into regions and then trying to work with each region separately. I have a number of different ideas on how to segment the source surface:

- Using discrete curvature, separate regions of positive curvature from regions of negative curvature. I could then discard the former immediately, knowing already that they are unfoldable[9].
- Using discrete curvature or, perhaps, through patch-fitting, identify gradients in the curvature flow upon the surface. Unfold along these lines.
- Trace the shortest paths between points of maximal curvature. Use these paths of faces as initial unfoldings joining multiple seed points, then work outwards from the seed points.
- Trace the shortest paths, but instead of linking points of maximal curvature, link those points furthest from regions of negative curvature.

Another version of a reduce-unfold-expand approach would be to reduce the model itself to its topological minimum (ie., merging polygons and collapsing vertices to reduce the mesh to the smallest set of polygons such that the removal of one more face would alter the topology of the surface [HA03].) If the last surviving facets in such a reduction retain some subset of the original edges, then I should be able to treat each facet as enclosing a limited region to unfold. I can foresee problems with this approach–the reduction will lose local non-topological features, such as warps in the surface, which are relevant at the "macro" level–but it could still offer me interesting insight. A similar approach is taken in Mitani and Suzuki's paper on reducing a model to strips to simplify unfolding [MS04].

One side track I'd like to look into some day, especially as I begin to isolate regions of the surface and work with the regions independently, is where I could benefit from parallel processing. It seems to me that the Multipart Unfolder, with its many seeds all unfolding at once, is a natural candidate for a parallelized implementation. But could this be done in a manner which would significantly improve performance?

Beyond working with each region separately, I would like to look into expressing in some clear manner the relationships between regions. Consider the inner torus model, merged at its base onto a sphere. How might I express, and react

[9] Is this *always* true? Or can I construct a model where two adjacent regions of positive curvature are positioned in such a way that they prevent each other from unfolding?

to, this negative region ‘surrounded’ by a positive one, which is large enough to unfold the negative? In fact, any closed surface at all at the base of the inner torus would be enough to make it solvable. Can I express this interrelationship of regions of the surface in such a way that I could determine a priori whether a region could ever be unfolded?

4.2 Open questions

Some, but not all, non-convex surfaces can be edge-unfolded. Can the geometric or topological attribute that makes a non-convex surface undevelopable be determined programmatically? Is it possible to determine whether or not an edge-unfolding can exist for a given non-convex surface without actually attempting to find it? Can one design a robust algorithm which can find an edge-unfolding for any surface (if that unfolding exists) and do so in less time than would be required by brute force? If a surface is not developable, can one find the minimium number of components into which it must be split?

If a surface cannot be edge-unfolded, can an algorithm be designed to generate only the minimum number of separated unfoldable components? Solutions exist for broadly detecting features, usually by analyzing lines of sudden changes in curvature. What is the correlation between features and unfoldability of patches?

Beyond preserving the critical subset of connectivity and preventing interpenetration of flattened polygons, what other constraints can be applied to the unfolding?

In my twin test cases of the inner torus vs. the spun parabola–when does it all go wrong? At what point does the torus lose what the parabola has, and what is it that is lost?

How should I handle vertices with non-zero curvature but with zero angle deficit?

Unfolding a mesh of infinitely-small faces would really be a general unfolding, as though one were willing to cut faces. What is the relationship between granularity of faces, edge lengths, incidence angles at vertices and angle deficits? How do these factors play out in a mesh’s unfoldability?

Thinking about the inner half-torus model, I see that it boils down to one of two problems: one must either split the rings at the top or bottom, or somehow fit all of the intervening geometry into the plane inside the ring. Can I express the latter in the form of some sort of ‘area deficit’, akin to the angle deficit metric, where the sum of the areas of the faces linked to the inside of a closed ring must not exceed the area of the plane in the ring? Perhaps, the sum of the areas of the faces, multiplied by the sign of their curvature? This expression needs further work but I think it could dramatically help to identify meshes which cannot unfolded.

The cut-tree of a sphere has no loops; the cut-tree of a torus has two loops; the cut-tree of the top half of a torus, closed across the bottom, can have one or two loops. What is the relationship between genus of the surface and genus of the cut-tree?

The Multipart Curvature Unfolder uses a multi-seed scheme that should translate well into other unfolders. Would it work anywhere? Does it solve the geometric ("macro") question, leaving only the topo to be addressed? If so, what is its running time? Is it at all reasonable, or have I just traded one impossible problem for another? (Although, if I have, perhaps the new impossible problem would be one that other researchers have already addressed?)

What **new** metrics can I find to describe a surface, what new numerical analyses of geometry and topology? What means exist beyond simple curvature for describing the behavior of a discrete surface?

And lastly, what haven't I even begun to think of?

A Terminology

A.0.1 The abstract mathematical entities

A *convex polyhedron M_c of dimension d* is the intersection of a set of open half-spaces in $\Re_d$ such that M_c does not lie in a proper affine subspace of $\Re_d$. The surface of such a union is the set of all points which are the limits of convergent sequences of points within the open half-spaces, yet not in the half- spaces themselves. Convex polyhedra have the interesting attribute that their surface is exactly their convex hull.

A *polyhedron M of dimension d*[10] is the union of a set of convex polyhedra M_c, a connected union such that there exists a path between any two points in the polyhedron which lies entirely within the union.

A great deal of the existing research has focused on the unfolding of *closed*, polyhedra. A polyhedron is described as *closed* if the surface is without boundary. It is an open question at this time whether solving the unfolding problem will require that polyhedra be closed. Hopefully such a requirement will prove to be overly strong and unnecessary. Examples exist of both un-unfoldable closed surfaces and easily unfoldable open surfaces.

A.0.2 The concrete geometric primitives

A *mesh* describes the discrete representation of the surface of a polyhedron. Meshes are $\Re_2$ manifolds in $\Re 3$. A mesh is a set of *faces*[11], planar regions bounded by *edges* linking *vertices.*

- Meshes are bounded. Any mesh may be enclosed within a sufficiently-large finite convex hull
- The set of faces defining a mesh must form a single connected set
- Faces must be planar
- No more than two faces may share an edge
- Faces may only meet at edges
- No more than two edges of a single face may meet at any point.

[10] In the literature, the term *polytope* is sometimes used instead of 'polyhedron'. In 3D graphics the two terms are more-or-less identical, although some researchers use the two words to distinguish the abstract mathematical forms (as polytopes) from their concrete counterparts (as polyhedra).
[11] The term 'face' or 'facet' is often used interchangeably with 'polygon' in computer graphics. 'Facet' stems from the field of solid modeling, in which many facets would make up a single face. Despite the more feature-sensitive meaning of the solid modeling usage, the computer graphics usage of 'face' is assumed.

- The set of edges defining a single face must be a single connected set. Faces may not have more than one boundary.

In my work to date, I have not constrained faces to be triangles. This may change. Simple examples have already been found of polyhedra with triangular faces which cannot be unfolded [BDEKOS99] and polyhedra with non-triangular faces which unfold easily (any of the platonic solids, for example.)

I do require that faces be convex.

A.0.3 Classes of polyhedra

Some interesting subclasses of polyhedra:

- *Simplicial polyhedra*: Polyhedra whose faces are the minimum degree to not fall into proper affine subspaces of $\Re_d - 1$. In $\Re 3$, the faces of a simplicial polyhedron are triangles.
- *Orthogonal polyhedra*: Polyhedra whose faces meet only at right angles.
- *Polyhedral bands*: Polyhedra bordered by two paths in parallel planes, where the projection of one path fits entirely within the other.

A.0.4 The Forest for the Trees

Alongside the mathematical entities, several graphs and tree appear throughout the unfolding problem. Some of these data structures describe the connectivity or associativity of the original or final meshes; other describe possible manipulations and alternative routes to finding a valid unfolding.

The *source connectivity graph* of the original mesh is the data structure which encodes connectivity. Such an encoding is implicit in the concept of a mesh, but almost any implementation will have to make it explicit. Any unfolding into a single piece must retain a spanning subset of the source connectivity graph.

The *Cut-tree* is the tree of broken edges after unfolding is complete: each node corresponds to a vertex of the original mesh and each link corresponds to an edge that has been split. There may be loops in the cut-tree (ex: the torus at right has one loop around its innermost ring and one loop around a cross-section of the tube; the sphere has none), so technically the cut-tree is not actually a tree; it is an undirected graph. But in picturing the cut-tree on a surface it is often useful to think of it as a branching, spreading, tree-like pattern. The term cut-tree was first coined in early research in the field, so I have chosen to retain it.

The *Unfolding tree*, AKA the 'Net' or 'Unfolding', is the tree of joined faces that results from the unfolding process: each node corresponds to a face of the original mesh and each link is the dual of an original edge. These nets are undirected acyclic graphs, subsets of the original connectivity graph.

A quick validity test of an unfolding tree takes two steps. Firstly, any loops in the graph or any disconnected nodes mean that it's immediately invalid. Secondly, the projection of the net down into the plane must never overlap one polygon atop another; such a conflict invalidates the unfolding.

The *MegaTree* (AKA 'Wayne' in my own work)is the tree of ALL possible orderings of all possible unfoldings from a fixed root. Each node in Wayne corresponds to a face of the source mesh; each edge corresponds to an edge/face pair, representing the edge across which that face will be added to the unfolding net. Each path from the root of the tree to a leaf node will visit each face once and thus defines a unique ordering of the generation of a single net.

B Factoids

B.0.5 Scope of the problem

On a mesh with d edges and n faces, the time to find a valid unfolding is at most O(d^n):

Assume a mesh with one face. It has precisely one unfolding; F(1) = 1

Assume a mesh with n faces. Every face is connected to at most d other faces. Therefore when the other n-1 faces of the mesh have been unfolded, the nth face will have d places where it could be attached to the net; F(n) = d * F(n-1)

Thus,

$$F(n) \sim O(d^n) \tag{B.1}$$

B.0.6 Enumerating unfoldings

An interesting insight from the implementation of the Iterative Tree Unfolder is that a unique index can be assigned to every unfolding net that could be derived from a mesh. The ITU builds the index as a more-than-binary integer by assigning a numerical base to each face; the base is the total product of the number of edges of every face before it in any (arbitrarily-chosen, probably implementation-dependent) ordered listing of the faces of the mesh. The index of a net is then the number formed by summing the products of the base of each face with the index of the edge that connects that face to its parent in the net graph. For example, unfolding a cube would assign to each of the six faces the bases 1, 4, 16, 64, 256, 1024; the ITU could then uniquely identify each net as the sum across all faces of the index of the edge that joined a face to its parent (a number 0-3) times the base of the face.

(This will also enumerate invalid nets, which do not meet the requirement that they be unlooping and fully-connected.)

This means that if a remote system already had a copy of the geometry and ordered connectivity of a mesh, then one could compress the expression of an unfolding down to an integer of at most n * ceil($\log_2$(d)) bits.

B.0.7 Removing edges reduces unfoldability

Because the set of edges that form the cut-tree is a subset of the set of edges in the source tree, cutting any edges of a surface before the unfolding process begins can only reduce its unfoldability.

For example, consider the spun parabola test model, which can only be unfolded in a spiraling strip. If one were to break two vertical edges parallel to one another on the same horizontal row of the model, the strip would be broken and the model would be un-unfoldable.

B.0.8 The genus of the source shape is not tied precisely to the genus of the cut-tree

While intuitively one would expect there to be a relationship between the genus of a surface and the loop count of its cut-trees, this is pretty easily disproved. A sphere's cut-tree can have no loops; a torus' cut-tree will generally have two, and a simple deformation of the torus–chopping off the bottom half and closing the open face, giving a model which is still topologically a torus but looks more like the top of a donut–will have only one.

B.0.9 The necessity of cutting vertices with negative curvature

A single vertex whose curvature is negative must lie on at least *two* edges in the cut-tree. This is because if it were only on one, i.e., only one of its connecting edges were cut, then all of its neighboring ring of faces would continue to share it as a vertex. If their curvature is negative then the sum of the angles of each of the neighbor faces at the vertex would be greater than 2pi and in flattening, two of the faces would necessarily interpenetrate.

B.0.10 Ordering doesn't really matter

The order in which an unfolding is constructed is irrelevant to the ultimate validity of the unfolding. This is because the validity test, which tests for connectivity and interpenetrating faces, can be implemented in a manner which can be run once on the entire set of unfolded faces without relying on any inherent ordering. The existence of such an implementation shows that order is irrelevant.

That said, many implementations of solutions to the Unfolding Problem are implementations which internally *do* rely on order. That said, even among such implementations there are almost always independent paths and sub-branches of the cut-tree which could be visited in any order.

B.0.11 Starting face doesn't matter

One might be tempted to believe that an unfolding which was built by beginning at one face must perforce be different from any unfolding that was built beginning somewhere else. But the connectivity of an unfolding forms an undirected acyclic graph (i.e. a tree) in which the edges of the new graph are a subset of the shared edges of the original mesh. And any two undirected acyclic graphs which are topologically identical, even if they're drawn on paper (or in a software implementation) with different roots, really are identical. So two unfoldings that retain precisely the same sets of unbroken edges from the original mesh, no matter what order they were generated in, will produce the same unfolding and thus can be tested for validity with the same test.

C A few words about Implementation

C.1 Implementing a Mesh in software - the YAMM classes

From the geometric definition it is possible to derive a concrete software implementation of a polyhedral mesh. This implementation will be called a *PolyMesh.* A PolyMesh will contain a list of *Poly*'s, which will represent the faces of the polyhedron; these in turn will access a list of *Vertex*'s, which represent the endpoints of edges on a single face.

To distinguish between geometric forms and their software implementations, the term 'polyhedron' (the mathematical entity) and the terms 'mesh', 'face' and 'vertex' (the geometric forms) will never be capitalized. The capitalized names 'PolyMesh', 'Poly' and 'Vertex' are reserved for implementations. Bearing this distinction in mind, the terms for each will be used interchangeably except where the distinction is actually pertinent to the discussion at hand.

The PolyMesh, Poly and Vertex class, regardless of implementation specifics, may be defined by the queries which they can support:

- Ability to iterate over the faces of the PolyMesh. Order is not imposed or implied but the iteration mechanism must traverse each Poly precisely once.
- Ability to iterate over the vertices of a single face of a PolyMesh. Although any iteration implies a 'starting' vertex, no special relevance is assigned to that position and vertex lists in a face should be seen as a closed loop, modulus index. The order is given to allow algorithms to iterate over neighboring vertices.
- Ability to iterate over the polygons sharing a vertex, in a counterclockwise order rotating around the surface normal at the vertex. The order is given to allow algorithms to iterate over neighboring polygons.
- Ability to query for the Poly that shares an edge with another Poly.

A strong restriction imposed on the Poly class is that its vertices must all be coplanar. (This is in support of a key rule of the geometric form, that every face must be flat.) Since this may not always be possible, it may be necessary to include a triangulating routine in the data entry stage, so that a Poly whose vertices lie too far off a plane can be split into multiple triangles. This raises the larger question of data entry in general: many forms of data entry generate complex meshes which do not necessarily conform to the polyhedral definition given here. It is tempting to wave ones hands and lay the burden of data conversion at the feet of the user, and in a purely mathematical discussion of the problem it is acceptable to do so. In any software implementation it will probably be necessary to internally ensure the consistency of the data.

Unfortunately, floating point drift remains the bugbear of any scheme which moves polygons around and transforms them. Mechanisms will be suggested throughout the implementation which try to minimize the impact of floating point error, but this must be recognized a priori as a source of potential difficulty.

It is also worth noting that any implementation of the generation of an unfolding will create new Vertex instances, as vertex sharing is no longer possible when two faces are separated and unfolded to different parts of the plane. In other words, while the number of faces will remain constant, the number of edges and vertices will grow.

C.2 Supporting Algorithms

C.2.1 Finding the shortest path on a polygonal surface

Several of my unfolding approaches rely on the concept of a *shortest path*, where path length is in some manner a summation of a local attribute of the surface (geometric distance; edges crossed; positive curvature.) I have found a number of sources which detail excellent shortest-path solutions. Having found them, I have decided not to use them: Djikstra wrote a very, very simple O(n^3) shortest-path algorithm in 1967 and I plan to use it. I'll take it as writ that anybody bringing this research to industry would replace the shortest-path finder (and all the other algorithms not directly tied to unfolding, like convex hulls and some of the accelerations available to calculating discrete curvature) with something more efficient. In my timing calculations, I will treat all of these sub-algorithms as black boxes running in unknown time.

C.2.2 Testing for inter-polygon intersections in the plane

I use a *quadtree* data structure to detect interpenetrating polygons during the unfolding process. I believe that the time to evaluate a single intersection is no more than log(n) to identify the quadtree boxes that contain a face and then a quick short list loop to check for overlap.

C.2.3 Convex hull in 2D

I maintain a 2D convex hull for some of the unfolders. Adding a vertex to the hull is a quick operation: the vertex's position in the ordered list of vertices (sorted by compass angle) is a quick O(n) search, then it's just simple maths and a little searching to determine whether to extend the hull or not. Konrad

PolthierKlaus Hildebrandt I met Polthier, and then Hildebrandt, for an afternoon in Berlin. I demo'd my software and they showed me theirs. Their stuff wasn't geared towards unfolding, it was just a side note added feature; they would split the model up into a 'reasonable' number of separated pieces, and then at the user's discretion could try to merge the separated bits.

We talked about using curvature to guide calculations. Polthier agreed that my idea of starting cutting at the negative curvature points was a good one. He also suggested that perhaps broad intersections could be avoided by testing for inter-gore intersections right up front [which brings to mind the thought that perhaps if my approach emphasises gore generation (as opposed to long spirals) then I can actually identify and optimise for interesections between them by making them real entities in the code?] by *integrating curvature along the paths of the gores* and seeing if it exceeded that of a closed circle. Something along those lines. Lubya Alboul –need to fill this entry in–

References

[C75] G. C Shephard - *Convex polytopes with convex nets*

Math. Proc. Camb. Phil. Soc, 1975

[W] E. W. Weisstein - *'Unfolding' From MathWorld–A Wolfram Web Resource*

http://mathworld.wolfram.com/Unfolding.html

[P03] K. Polthier - *Imaging maths - Unfolding polyhedra*

http://plus.maths.org/issue27/features/mathart/index.html

[L77] W. L. Strauss - *The Painter's Manual by Albrecht Durer (1525)*

Abaris Books (June, 1977) ISBN: 0913870528

[DDLO02] E. Demaine, M. Demaine, A. Lubiw, J. O'Rourke - *Enumerating Foldings and Unfoldings between Polygons and Polytopes*

Graphs and Combinatorics, v.18 n.1 2002, pp.93-104

[O98] J. O'Rourke - *Folding and Unfolding in Computational Geometry*

Lecture Notes Comput. Sci.. Vol. 1763, Springer-Verlag, Berlin, 2000, pp. 258-266

[DEEHO02] E. Demaine, D. Eppstein, J. Erickson, G. Hart, J. O'Rourke - *Vertex-Unfoldings of Simplicial Manifolds*

Discrete Geometry: In Honor of W. Kuperberg's 60th Birthday (Andras Bezdek, editor), Marcel Dekker, 2003, pp. 215-228.

[BDEK99] M. Bern, E. Demaine, D. Eppstein, E. Kuo - *Ununfoldable Polyhedra*

Proceedings of the 11th Canadian Conference on Computational Geometry (CCCG'99), Vancouver, British Columbia, Canada, August 15-18, 1999, pp.13-16

[BDEKMS99] M. Bern, E. Demaine, D. Eppstein, E. Kuo, A. Mantler, J. Snoeyink - *Ununfoldable Polyhedra with Triangular Faces*

Proceedings of the 4th CGC Workshop on Computational Geometry (CGC'99), Baltimore, Maryland, October 15-16, 1999.

[BDEKMS99] M. Bern, E. D. Demaine, D. Eppstein, E. Kuo, A. Mantler, J. Snoeyink - *Ununfoldable Polyhedra with Convex Faces*

Computational Geometry: Theory and Applications, volume 24, number 2, February 2003, pp.51-62. Special issue of selected papers from the 4th CGC Workshop on Computational Geometry, 1999.

[BDDLOORW98] T. Biedl, E. Demaine, M. Demaine, A. Lubiw, M. Overmars, J. O'Rourke, S. Robbins, S. Whitesides - *Unfolding Some Classes of Orthogonal Polyhedra*

Proceedings of the 10th Canadian Conference on Computational Geometry (CCCG'98), Montreal, Quebec, Canada, August 10-12, 1998

[ADLMOST04] G. Aloupis, E. D. Demaine, S. Langerman, P. Morin, J. O'Rourke, I. Streinu, G. T. Toussaint - *Unfolding Polyhedral Bands*

Proceedings of the 16th Canadian Conference on Computational Geometry (CCCG 2004), Montreal, Quebec, Canada, August 9-11, 2004, pp.60-63

[DLS99] E. D. Demaine, M. L. Demaine, J. S. B. Mitchell - *Folding Flat Silhouettes and Wrapping Polyhedral Packages*

Proceedings of the 15th Annual ACM Symposium on Computational Geometry (SoCG'99), Miami Beach, Florida, June 13-16, 1999, pp.105-114

[LO96] A. Lubiw, J. O'Rourke - *When Can a Polygon Fold to a Polytope?*

Smith Tech. Rep. 048, June 1996. AMS Conference, Oct. 1996, Lawrenceville, NJ

[AAOS93] P. Agarwal, B. Aaronov, J. O'Rourke, C. Schevon - *Star Unfolding of a Polytope with Applications*

SIAM. J. on Computing, 26(6) 1689-1713, December 1997

[OD02] J. O'Rourke, E. Demaine - *[Problem 43] General Unfoldings of Nonconvex Polyhedra*

Proc. of the 15th Canadian Conference on Computational Geometry, Halifax, August 2003, pp. 178-181

[BCO04] N. Benbernou, P. Cahn, J. O'Rourke - *Unfolding Smooth Prismatoids*

Smith College Computer Science Technical Report 078, July 2004

[F97] K. Fukuda - *Strange Unfoldings of Convex Polytopes*

http://www.ifor.math.ethz.ch/∼fukuda/unfold_home/unfold_open.html

[NF94] M. Namiki, K. Fukuda - *Unfolding 3-dimensional convex polytopes: A package for mathematica 1.2 or 2.0*

ftp://ftp.ifor.math.ethz.ch/pub/fukuda/mathematica/UnfoldPolytope.tar.Z

[MP03] E. Miller, I. Pak - *Geodesic Flow on Convex Polyhedra and Nonoverlapping Unfolding*

http://www-math.mit.edu/∼pak/fold63.pdf

[BLS99] T. Biedl, A. Lubiw, J. Sun - *When Can a Net Fold to a Polyhedron*

Eleventh Canadian Conference on Computational Geometry, U. British Columbia, 1999, 1-4

[BG04] T. Biedl, B. Genc - *When Can A Graph Form An Orthogonal Polyhedron*

Proceedings of the 16th Canadian Conference on Computational Geometry (CCCG'04), pp. 53–56

[SC02] A. Sheffer, J. C Hart - *Seamster - Inconspicuous Low-Distortion Texture Seam Layout*

IEEE Visualization (Vis02), 291-298, 2002

[S02] A. Sheffer - *Spanning Tree Seams for Reducing Parameterization Distortion of Triangulated Surfaces*

Shape Modelling International, 61-66, 2002

[NGC04] X. Ni, M. Garland, J. C Hart - *Fair Morse Functions for Extracting the Topological Structure of a Surface Mesh*

ACM Trans. Graph. 23(3): 613-622 (2004)

[C99] J. C Hart - *Using the CW-Complex to Represent the Topological Structure of Implicit Surfaces and Solids*

Proc. Implicit Surfaces '99, Sept. 1999, pp. 107-112

[PW99] H. Pottman, J. Wallner - *Approximation Algorithms for Developable Surfaces*

Computer Aided Geometric Design 16 (1999), 539-556

[DBGPC05] S. Dong, P. Bremer, M. Garland, V. Pascucci, J. C Hart - *Quadrangulating a Mesh using Laplacian Eigenvectors*

Technical Report UIUCDCS-R-2005-2583, June 2005

[G02] B. Grunbaum - *No-Net Polyhedra*

Geombinatorics 11:111-114, 2002

[S97] W. Schlickenrieder - *Nets of Polyhedra*

Ph.D. thesis. Berlin: Technische Universitat Berlin, 1997

[P02] K. Polthier - *Polyhedral Surfaces of Constant Mean Curvature (chapters 1, 3, 4)*

abilitationsschrift, TU-Berlin (Febr. 2002), 1-212

[MDSB03] M. Meyer, M. Desbrun, P. Schroder, A. Barr - *Discrete Differential Geometry Operators for Triangulated 2-Manifolds*

VisMath '02 Proceedings

[A02] L. Alboul - *Curvature Criteria in Surface Reconstruction*

Communications on Applied Mathematics, Russian Academy of Sciences, Computing Centre, Moscow, 2002 ISBN: 5-201-09784-7

[SMSER03] T. Surazhsky, E. Megid, O. Soldea, G. Elber, E. Rivlin - *A Comparison of Gaussian and Mean Curvatures Estimation Methods on Triangular Meshes*

International Conference on Robotics and Automation, Taipei, Taiwan, pp. 1021-1026, 14-19 September, 2003

[DHKL00] N. Dyn, K. Hormann, S. Kim, D. Levin - *Optimizing 3D Triangulations Using Discrete Curvature Analysis*

Mathematical Methods for Curves and Surfaces: Oslo 2000 archive, pp.135 - 146, 2001 ISBN: 0-8265-1378-6

[SJ00] D. S. Meek, D. J. Walton - *On Surface Normal and Gaussian Curvature Approximations Given Data Sampled from a Smooth Surface*

Computer Aided Geometric Design archive, vol.17 Issue 6 (July 2000), pp.521-543

[HA03] F. Hetroy, D. Attali - *Detection of Constrictions on Closed Polyhedral Surfaces*

Proceedings of the symposium on Data visualisation 2003, Grenoble, France, pp.67 - 74 ISBN: 1-58113-698-6

[GG04] T. Gazke, C. Grimm - *Improved curvature Estimation on Triangular Meshes*

Technical Report, CSE, Washington University in St Louis, 2004

[HP04] K. Hildebrandt, K. Polthier - *Anisotropic Filtering of Non-Linear Surface Features*

ZIB Preprint, 04-25, in: Computer Graphics Forum, 23 (3), 2004

[PP93] U. Pinkall, K. Polthier - *Computing Discrete Minimal Surfaces and Their Conjugates*

Experimental Mathematics, Vol 2 (1), 1993, Page 15-36

[F62] R. Floyd - *Communications of the ACM - ALGORITHM 97 - SHORTEST PATH*

[KO00] B. Kaneva, J. O'Rourke - *An Implementation of Chen and Han's Shortest Paths Algorithm*

Proc. of the 12th Canadian Conference on Computational Geometry, New Brunswick, August 2000, pp. 139-146

[O99] J. O'Rourke - *Computational Geometry Column 35 [The subquadratic algorithm of Kapoor for finding shortest paths on a polyhedron]*

SIGACT News 30(2), Issue #111 (1999) 31-32.Int. J. Comp. Geom. Appl., 9(4-5) (1999) 513-515

[HP04] M. Hofer, H. Pottman - *Energy-Minimizing Splines in Manifolds*

Transactions on Graphics 23(3):284-293, 2004. (Proceedings of ACM SIGGRAPH 2004)

[GE04] C. Gray, W. Evans - *Optimistic Shortest Paths on Uncertain Terrains*

CCCG 2004, pp.68-71

[AMOT90] R. Ahuja, K. Mehlhorn, J. Orlin, R. Tarjan - *Faster Algorithms for the Shortest Path Problem*

Journal of the Association for Computing Machinery 37 (1990), 213–223

[CG94] B. Cherkassky, A. Goldberg - *Chapter 57: Shortest Paths Algorithms: Theory and Experimental Evaluation*

SODA: ACM-SIAM Symposium on Discrete Algorithms, 1993

[CH90] J. Chen, Y. Han - *Shortest Paths on a Polyhedron*

Proceedings of the sixth annual symposium on Computational geometry table of contents, Berkley, California, United States, pp.360-369, 1990 ISBN: 0-89791-362-0

[GMS] R. Graham, H. McCabe, S. Sheridan - *Pathfinding in Computer Games*

ITB Journal Issue Number 8, Institute of Technology Blanchardstown

[JT04] D. J. Balkcom, M. T. Mason - *Introducing Robotic Origami Folding*

IEEE International Conference on Robotics and Automation, pp.3245-3250, April 2004

[MS04] J. Mitani, H. Suzuki - *Making Papercraft Toys from Meshes using Strip-Based Approximated Unfolding*

SIGGRAPH 2004

[DLL98] E. D. Demaine, M. L. Demaine, A. Lubiw - *Folding and Cutting Paper*

Revised Papers from the Japan Conference on Discrete and Computational Geometry (JCDCG'98), Lecture Notes in Computer Science, volume 1763, Tokyo, Japan, December 9-12, 1998, pp.104-117

[MKW05] J. McCartney, B. K. Hinds, K. W. Chong - *Pattern Flattening for Orthotropic Materials*

CAD, Volume 37, Number 6, May 2005, pp. 631-644

[KAHS98] S. K. Gupta, D. A. Bourne, K. H. Kim, S. S. Krishnan - *Automated Process Planning for Sheet Metal Bending Operations*

Journal of Manufacturing Systems, No. 5, September, 1998

[MM01] P. Morin, J. Morrison - *The Geometry of Carpentry and Joinery*
Discrete Applied Mathematics, 144(3):374-380, 2004

[V94] B. Van Loon - *Geodesic Domes*
Tarquin Press (February, 1994) ISBN: 0906212928

[L99] D. Lovell - *TeXML: Typesetting XML with TeX*