Modeling Developable Surfaces from Arbitrary Boundary Curves

by

Kenneth Lloyd Patrick Rose

B.Math., University of Waterloo, 2005

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

The Faculty of Graduate Studies

(Computer Science)

The University Of British Columbia

August 2007

Abstract

Developable surfaces are surfaces that can be unfolded into the plane with no distortion. Although ubiquitous in our everyday surroundings, there is currently no easy way to model them on a computer. This thesis fills this void by presenting a general method for creating developable geometry that utilizes the connection between developable surfaces and the convex hulls of their boundaries. Given an arbitrary, user-specified 3D polyline boundary, our system generates a smooth discrete developable surface that interpolates this boundary. We identify desirable properties of such surfaces, present a practical algorithm to compute them, and extend it to handle darts and internal singular points. We demonstrate the effectiveness of our method through a series of examples, from architectural design to garments, using a sketch-based interface to quickly create the boundaries.

Table of Contents

List of Tables

List of Figures

Preface

Part of this thesis has been published in the following paper:

- Rose, K., Sheffer, A., Wither, J., Cani, M.-P., Thibert, B. (2007). Developable Surfaces from Arbitrary Sketched Boundaries. Eurographics Symposium on Geometry Processing 2007. Pages 163 - 172.

Acknowledgements

First and forefost, I would like to thank my supervisor Alla Sheffer, without whom this work would not have been possible. There are countless things that I learned with her over these past two years which I am grateful for. I also would like to thank Michiel van de Panne, my second reader, for all of his valuable feedback.

Thank you to everyone in the Imager lab that I had the pleasure of working with: Tibi, Vlady, Dan, Llach, Michael, Ian, Steve, Chris, Hagit, Mike, David, Abhi, Derek, Cheryl, Brad, Gordon, Aaron, Stellian, Yoel, and anyone else that I've forgotten. As well, thank you to everyone at Hillel. I take with me many great memories.

Thank you to my father and sister for understanding my decision to pursue graduate work three time zones away. To my housemate Izzet, thank you for many enlightening scientific discussions. To Kira, thank you for all of the great times and for making me feel like a part of your family. I also owe a great deal of gratitude to Candice Chatz, who continuously supported me and prevented me from fulfilling the stereotype of a starving grad student. Finally, thank you to my late mother, who taught me that everything about life leaves answers. Undeniable, the pursuit of this degree was no exception.

KENNETH ROSE

The University of British Columbia
August 2007

Statement of Co-Authorship

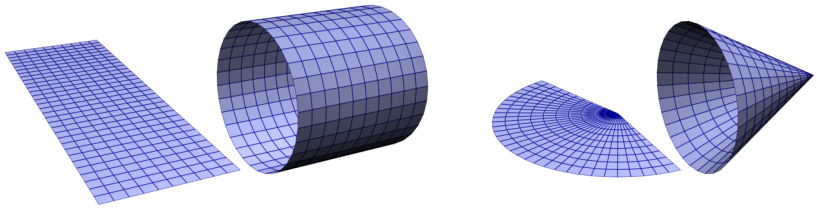
The algorithm described in Chapter 4 was developed together with Dr. Alla Sheffer. Dr. Sheffer supervised the project while I performed research and implementation. The algorithm was combined with a sketch based user interface developed by Jamie Wither and Prof. Marie-Paule Cani. A paper describing a system unifying the user interface and the algorithm was cooperatively prepared [32].

Chapter 1

Introduction

1.1 Developable Surfaces

Developable surfaces are surfaces that can be unfolded into the plane without distortion. As shown in Figure 1.1, cylinders and cones are examples of developable surfaces since each can be unfolded to the plane without stretching or shearing the surface. The unfolded, planar surface is referred to as the *pattern* or *development* of the original surface, depending on the application. An example of a non-developable surface is a sphere. To illustrate, consider the cartographical problem of representing the 2D curved surface of the earth on a flat map (i.e., a plane). Figure 1.2(b) shows one possible projection, though it exhibits gross stretching at the north and south poles (e.g., Greenland in white). As explained in Chapter 2, there is in fact no way to map a sphere to a plane without introducing distortion. The ability of a surface to map to the plane with little or no distortion is extremly practical, making developable surfaces useful in several applications.



(a) Cylinder (b) Cone

Figure 1.1: Examples of developable surfaces and their corresponding patterns.

(a) Spherical Earth (b) Mercator Projection

Figure 1.2: Mapping a sphere to the plane. Greenland is highlighted in white to indicate distortion.

1.2 Applications

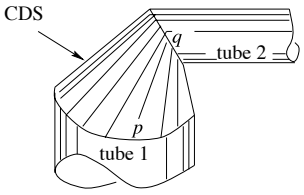
Developable surfaces are commonly used when manufacturing with materials that do not stretch or tear. Any process manipulating fabric, paper, leather, sheet metal or plywood will benefit from developable surface modeling techniques since these materials admit little distortion. In typical setups, the patterns of the product are first designed by trained individuals, with a computer performing a bending simulation to help forecast the manufactured result. The product is then fabricated by cutting out the patterns of the surface from a flat sheet of the respective material and bending these planar patterns to form the desired shape. Applications include modeling ship hulls, buildings, airplane wings, garments, ducts, automobile parts.

In ship design, the hull is usually constructed by segmenting it into pieces fitted with large metal sheets. On singly curved parts of the hull, sheets can be fitted by a process called *rolling*, which simply bends the sheets to the desired shape. The result of rolling is a developable surface which is typically cylindrical. Rolling alone is normally not sufficient to construct an entire hull since some parts of the hull may be doubly curved and thus non-developable. For example, the bow usually includes a bulbous piece (Figure 1.3(a)) which moves water around the hull in the direction of least resistance, ultimately reducing fuel consumption. In these non-developable regions, a *heating* process is used to deform a sheet after rolling in order to

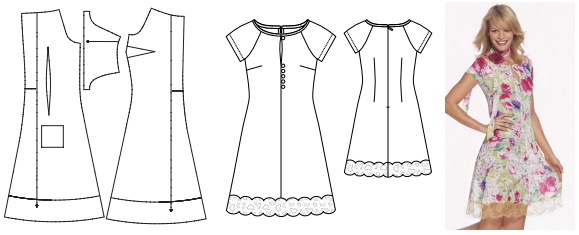
introduce the additional curvature direction. The heating process is usually performed manually by an experienced individual who heuristically determines the parameters required to achieve the correct amount of bending. The heating process is time consuming, labour intensive and error prone. Thus, modeling a hull primarily with developable surfaces minimizes the use of heating and simplifies and improves the fabrication process [11, 12].

(a) Ship Hull [38].

(b) Peter B. Lewis building [7].



(c) Connecting tubes [37].



(d) Garment Design [9].

Figure 1.3: Applications of developable surfaces.

The benefits of developable surfaces in ship hull design are also pertinent when designing buildings from sheets of material. The prolific and contemporary architect Frank Gehry extensively uses developable surfaces in his structures. Shelden notes that a major accomplishment of Gehry's work is the ability to create innovative designs within the context of conventional construction processes [36]. Though any free form shape can be constructed by the digital CNC fabrication techniques commonly used by the aerospace and computer animation industries, the costs of these methods are frequently prohibitive when applied in an architectural setting. By

utilizing developable constraints and staying within the realm of existing fabrication practices, Gehry is able to reduce costs and fabrication error, ensuring the final form is tightly correlated to the original design. Figure 1.3(b) shows an image of the Peter B. Lewis building at Case Western Reserve University in Cleveland, USA. The brick and steel portions of the building are both modeled as piecewise developable surfaces [36].

In duct construction, a common problem is fabricating a metal surface that connects two tubes of different shapes. Specifically, given two space curves p and q, the problem is finding a surface that interpolates both p and q (Figure 1.3(c)) [37]. Though there are an infinite number of such connecting surfaces, using a connecting *developable* surface (CDS) is desirable since a CDS is most easily fabricated from sheet metal. This problem is straightforwardly solved by the algorithm presented in Chapter 4.

In fashion design, developable surfaces have great utility. Garments are constructed by sewing together panels that are cut out as patterns from flat sheets of fabric. For example, in Figure 1.3(d), the dress on the right is assembled by cutting out and sewing together the patterns on the left. When designing garments, a core requirement is constructibility and ensuring that each panel of the garment can be developed from a flat sheet. Although certain materials stretch slightly due to gravity after assembly (e.g., cotton weave), garments are traditionally modeled as developable surfaces since the effect of this stretch is negligable when assembling the panels together. Designing sewing patterns is a challenging task, requiring significant training to understand the many ways that panels can join together to correctly form around the geometry of humans. The next section introduces a modeling paradigm that greatly simplifies the task of pattern creation and allows interesting garments to be designed.

1.3 Developables from Boundaries

Despite their ubiquity, developable surfaces remain difficult to model, particularly for non-expert users. This thesis focuses on the problem of easily modeling developable surfaces and presents a method in which users simply

specify the boundaries of each surface patch as a 3D curve. 3D boundary curves are a natural modeling choice since they can be easily specified and manipulated through a variety of interfaces (Chapter 5) and provide intuitive shape control for the underlying surface.

To enable this modeling paradigm, we introduce a method for creating developable surfaces which interpolate *arbitrary* boundaries (Chapter 4). We observe the correlation between a developable surface interpolating a boundary curve and the convex hull of that curve. This linkage is the basis for a novel algorithm that generates interpolating developable surfaces for any given smooth input boundary. The method explores the space of possible interpolating surfaces searching for solutions which have a desired set of shape properties. It allows the user to rank the importance of the different properties in order to control the shape of the resulting surface and supports exploration of alternative solutions.

Chapters 2 and 3 review the relevant mathematical background and survey related work. Chapter 4 describes a novel algorithm operating in a discrete setup for computing a developable surface interpolating an arbitrary polyline boundary. Results obtained from this algorithm are showcased in Chapter 5. Finally, Chapters 6 and 7 summarize and discuss future work.

Chapter 2

Background

Developable surfaces have a lengthy mathematical history originating in differential geometry. This section reviews their main properties, focusing on those used by the algorithm presented in Chapter 4.

2.1 Ruled and Developable Surfaces

A *ruled surface* is a surface containing (at least) one one-parameter family of straight lines [22]. A ruled surface $S \subset \mathbb{R}^3$ may be represented in the form

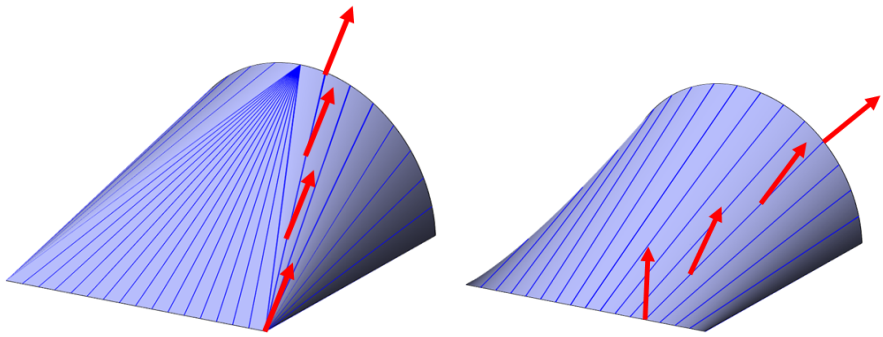
$$\mathbf{x}(s,t) = \mathbf{b}(s) + t\boldsymbol{\delta}(s),$$

where $\mathbf{b}(s)$ is called the *directrix* or the *base curve* of the surface and $\boldsymbol{\delta}(s)$ is called a *generator*. Intuitively, a ruled surface can be thought of as being constructed by continuously sweeping out a line in space (i.e., the generator) with some marked point of the line following the path of the directrix. Indeed, for a given fixed value of s, the above formula reduces to the equation of a straight line. This line is called a *ruling* of the surface.

A *developable surface* is a ruled surface with the additional property that the tangent plane is constant at all points along a given ruling [22]. Since the tangent plane at a point can be described by the surface normal at that point, an equivlent requirement is that the surface normal at all points along a given ruling is constant [31]. A ruled surface which is not developable has normal variation along its rulings and is thus called a *warped ruled surface*. A ruled surface is a developable surface if and only if

$$\dot{\mathbf{b}} \cdot (\boldsymbol{\delta} \times \dot{\boldsymbol{\delta}}) = 0$$

where the dot above denotes the derivative with respect to s (Theorem 58.1 of [22]). Figure 2.1 shows a developable surface and warped ruled surface sharing the same boundary.



(a) Developable Surface (b) Warped Ruled Surface

Figure 2.1: Developable and warped ruled surfaces. In (a), the normals are constant along the specified ruling while in (b), the normals vary along the ruling.

2.2 Gaussian Curvature under Isometric Mapping

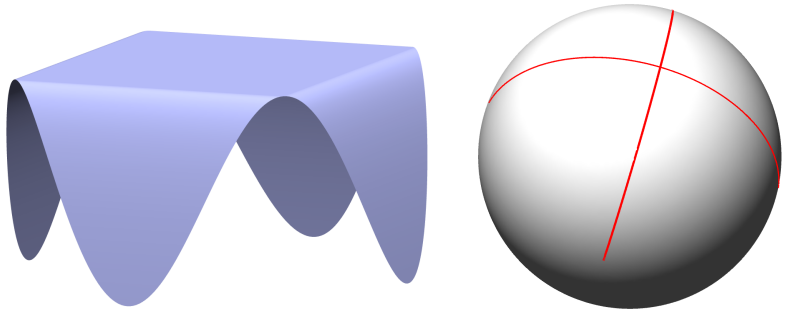
A bijective function $f : S \subseteq \mathbb{R}^3 \rightarrow T \subseteq \mathbb{R}^3$ is an *isometric* or *length preserving* mapping if the length of any arc on S is the same as that of its image on T under f. For example, rotation around a given axis is an isometric mapping. An extremely useful property is that the intrinsic properties of a surface, those depending only on the first fundamental form, are invariant under isometric mapping (Theorem 57.1 of [22]). The principal curvatures at a point on a surface measure the minimum and maximum bending of the surface at that point. Gauss' *theorema egregium* states that the Gaussian curvature K, the product of the principal curvatures, is an intrinsic property of a surface [19]. Therefore, K is invariant under isometric deformation.

2.3 Properties of Developable Surfaces

A portion of a surface is developable if and only if $K = 0$ everywhere on the portion (Theorem 59.2 of [22]). Since a plane has $K = 0$ everywhere, a plane is an example of a developable surface. Furthermore, since K is invariant under isometric deformation and isometries are bijective, any surface that can be isometrically mapped to the plane is developable. The converse of this statement, that any developable surface can be isometrically mapped to the plane, is also true and is proved in Theorem 59.3 of [22]. Thus, developable surfaces are the only surfaces that can be isometrically deformed to the plane. This property explains why it is impossible to map a sphere to the plane without distortion (Figure 1.2). Since a sphere has $K > 0$ everywhere, it is not developable and thus cannot be isometrically mapped to the plane.

The Gauss map is a function that maps every point p of an oriented surface in $\mathbb{R}^3$ to the point on the unit sphere that is parallel to the normal at p. In general, the Gauss map of a surface is another surface (right side of Figure 2.5(d)). However, in the case of developable surfaces, since the normals are constant along a given ruling, the Gauss map degenerates into a curve (right side of Figure 2.5(c)) or possibly a network of curves (Figure 2.2(b)). If the Gauss map is a single curve, then the directrix of the surface is a single continuous curve. Pottmann and Wallner [31] refer to these surfaces as *developable ruled surfaces* or *torsal ruled developable surfaces*. To avoid ambiguity with ruled surfaces as defined in Section 2.1, these surfaces will be referred to as *torsal* developable surfaces, in contrast to *composite* developable surfaces whose Gauss map is a network of curves. A composite developable surface is thus made of a union of torsal developable surfaces joined together by transition planar regions [16], where the latter correspond to the branching points on the Gauss map. In either case, another useful property of developable surfaces is that their image under the Gauss map is one dimensional.

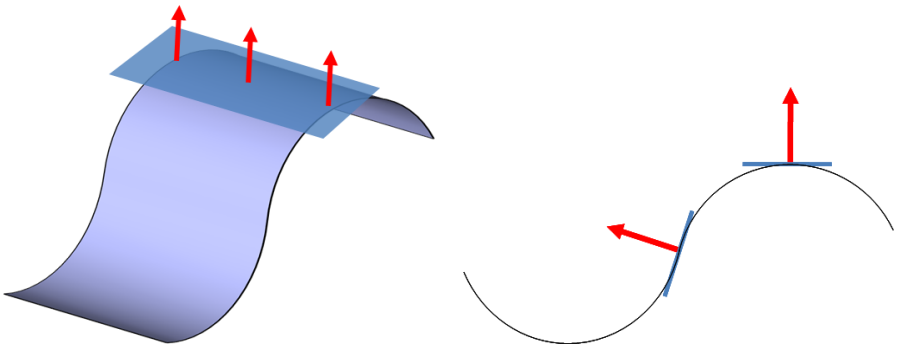
On a developable surface, the tangent planes of most rulings bound some local neighbourhood of the ruling in one of the two closed half-spaces induced by the plane (Figure 2.3(a)). Therefore, most tangent planes of rulings are



(a) (b)

Figure 2.2: A composite developable surface and its Gauss map.

supporting planes [23], the exception being rulings where the surface has an inflection (Figure 2.3(b)). When the tangent plane of a ruling is a supporting plane, since all rulings in the local neighbourhood lie on one side of the plane, the given ruling lies *on* the convex hull of the neighbourhood (i.e., the local convex hull) [17]. In contrast, on a warped ruled surface, most rulings lie *inside* their local convex hull.

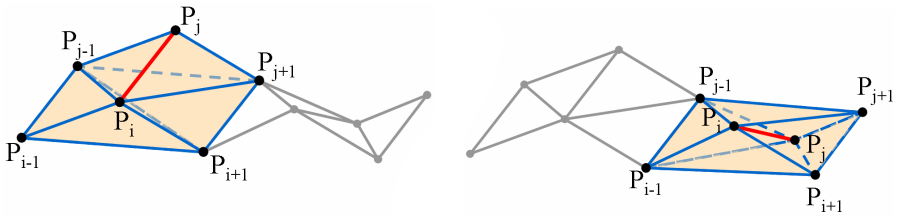


(a) (b)

Figure 2.3: (a) Tangent plane is a supporting plane on a developable surface; (b) Profile view of (a). Only on the inflection ruling is the tangent plane not a supporting plane.

2.4 Developable Boundary Triangulations

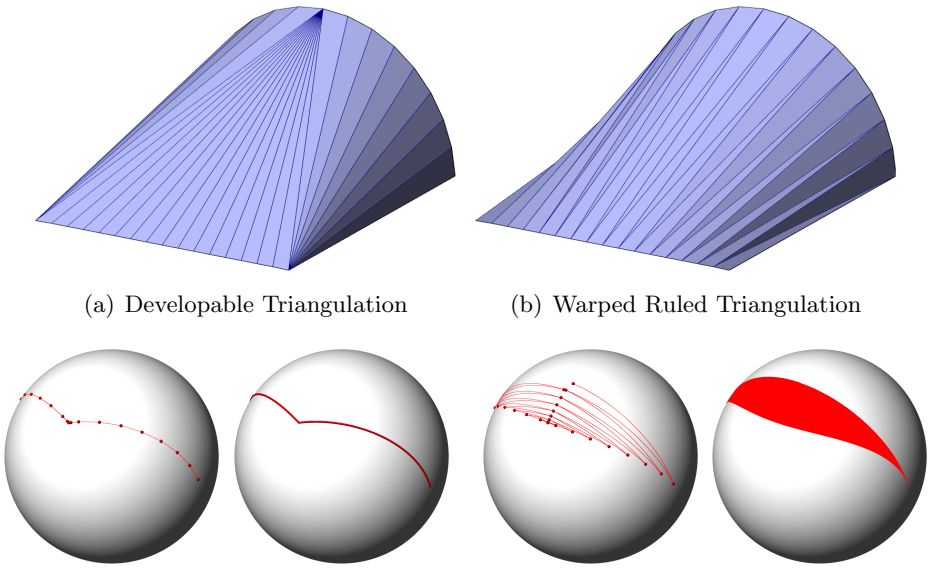
Given a polyline with vertices sampled from an input piecewise smooth curve, a boundary triangulation is a manifold triangulation with no interior vertices whose boundary is the polyline. By construction, any boundary triangulation is developable, as the triangles can be unfolded into the plane with no distortion. In the limit however, as the sampling density of the polyline increases, not every triangulation will approximate a smooth developable surface. Specifically, the limiting surface of a triangulation is a developable surface if and only if the majority of the interior edges of the triangulation are *locally convex* [17]. An interior edge is defined as *locally convex* if it lies on the convex hull of its end vertices and the four adjacent polyline vertices [17] (Figure 2.4(a)). An interior edge is *non-convex* if it lies inside this convex hull (Figure 2.4(b)).



(a) Locally Convex (b) Non-Convex

Figure 2.4: Locally convex and non-convex interior triangulation edges P_iP_j.

For a triangulation to approximate a smooth developable surface, the number of non-convex edges should not depend on the sampling density of the polyline. Figure 2.5 shows two triangulations of the same polyline, one of which approximates a developable surface, while the other approximates a warped ruled surface. In the first case, all the interior edges are locally convex (Figure 2.5(a)). In the second case, the majority of edges are non-convex (Figure 2.5(b)).



(a) Developable Triangulation (b) Warped Ruled Triangulation

(c) Gauss map of 2.5(a) and of limit surface (d) Gauss map of 2.5(b) and of limit surface

Figure 2.5: Developable and warped ruled triangulations interpolating the same polyline and corresponding Gauss maps.

Chapter 3

Related Work

In computer graphics and modeling, developable surfaces have raised interest in several different contexts including reconstruction from point clouds [12, 29] and mesh segmentation into nearly developable charts for parameterization and pattern design [20, 35, 43]. The following review only covers methods for modeling developables either via developable approximation or directly.

3.1 Developable Approximation

Given an existing non-developable surface, a large number of methods aim at approximating it with one or more developable surfaces. Some of the methods operate on triangle meshes in a discrete setup [15, 25, 28, 41] while others operate in a continuous setup for incorporation into NURBS based modeling systems.

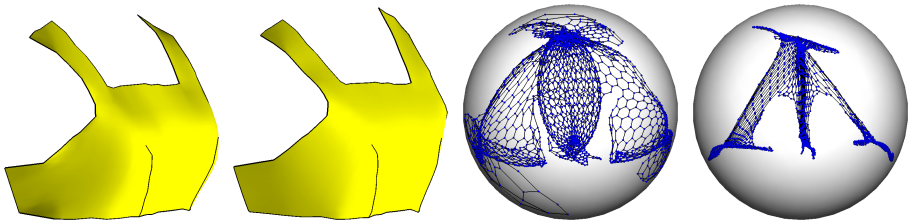
3.1.1 Discrete Methods

Wang and Tang [41] increase the developability of a mesh surface by minimizing its Gaussian curvature. Using a penalty based function, they solve a global constrained optimization problem that accounts for Gaussian curvature, the amount of deformation, and continuity between patches. Since solving the global optimization may be slow, they additionally formulate an iterative local optimization scheme. The authors note that if a high degree of developability is required, large discrepancies result on the final surface. Since the surface normals are not constrained in the optimization, these discrepancies are often manifested as wrinkles, a possibly undesirable

effect.

Similar to Wang and Tang [41], Frey [18] also attempts to increase the developability of a mesh by minimizing its Gaussian curvature. Frey attempts to introduce singular vertices into a 2.5D developable triangulation in order to model buckled developable surfaces. Each singular vertex is iteratively moved in the z direction until the sum of angles around the vertex equals 2π. Like many others, Frey translates the property of a developable surface having zero Gaussian curvature everywhere into a requirement that the sum of angles around each vertex equals 2π.

Decaudin et al. [15] describe a system for designing virtual garments where a user first constructs a non-developable garment that is segmented into overlapping mesh patches. For each mesh patch, the locally best approximating developable surface is computed and the mesh is deformed towards this surface. As evident in Figure 3.1, the resulting surfaces are "more" developable in the sense that their Gauss map covers less area.



(a) Typical Input (b) Approximation Output (c) Gauss Map of 3.1(a) (d) Gauss Map of 3.1(b)

Figure 3.1: Virtual garments [15].

In their conical meshes paper, Liu et al. [25] address approximation in the context of architectural design. Given a quadrilateral tiling of an input model, the method iteratively perturbs vertices to create a tiling with planar faces and the same connectivity as the input. Alternating the pertubation with subdivision induces a method for modeling developable strips.

3.1.2 Continuous Methods

Pottmann and Wallner [30] approximate an input NURBS surface with certain types of developable NURBS surfaces. Their technique finds a developable surface that approximates a set of tangent planes sampled from the input surface. The fidelity of the approximation is defined in terms of a distance metric defined between planes. This distance metric is optimized to construct the approximating developable surface.

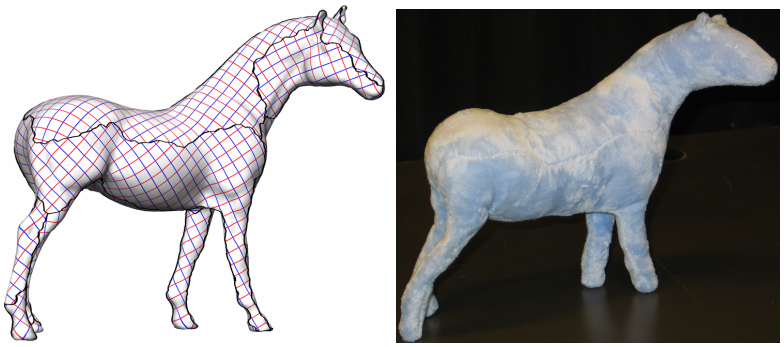
Chen et al. [12] focus on approximating ship hulls. They initially segment the input surface using a region growing approach and each segment is approximated individually by a cone or cylinder of revolution. These pieces can then be joined together with G^1 continutity using [24], or a G^r $(r \geq 2)$ approximating developable can be found using [30].

Wang et al. [39] increase the developability of a trimmed NURBS surface by minimizing its Gaussian curvature. Analagous to their approach for 3D meshes described in [41], the optimization function accounts for both the overall Gaussian curvature and the amount of deformation. The optimization process adjusts the positions and weights of the control points of the original trimmed surface. The authors note that the running time may be significant and that the Gaussian curvature may actually *increase* locally.

3.1.3 Benefits and Limitations of Approximation

Modeling developable surfaces through approximation is attractive as designers do not have to concern themselves with developability constraints during the modeling process. Ideally, they can freely utilize all sorts of modeling tools (e.g., blends, fillets) and then rely on an approximation algorithm to yield a developable result. In practice though, the approximation approach is highly restricted since the methods can only succeed if the original input surfaces already have fairly small Gaussian curvature. Moreover, in most cases the final result is not analytically developable. While this is not a problem for applications such as texture-mapping, it can be problematic for manufacturing, where the surfaces need to be realised from planar patterns (e.g., sewing). In these setups the distortion caused by using un-

folded patterns from approximate developables can be quite significant, as demonstrated in Figure 3.2. In this example from [20], the horse model was segmented into nearly developable charts unfolded into the plane with L^2 stretch of less than 1.01 [20]. This can be visually confirmed by observing that the isolines in Figure 3.2(a) are perpendicular and define squares of approximately equal area. However, when the patterns created from the unfolding were sewn back together, the resulting toy horse had significantly different proportions from the initial model (e.g., the front legs). Though minor surface details are expectedly lost due to the resilience of the fabric and possible sewing and cutting errors, a contributing factor to the discrepancy in proportions is the fact that the charts are not completely developable.



(a) (b)

Figure 3.2: Artefacts in using approximate developables [20] for manufacturing. (a) approximate developable segmentation (L^2 stretch 1.01); (b) reassembled model.

3.2 Direct Modeling

As opposed to modeling by approximation, another class of techniques directly model developable surfaces, ensuring that the user has an analytically developable surface at all times. Most existing methods for modeling developable surfaces consider only torsal developable surfaces, surfaces whose normal map is a single curve, and are restricted to modeling four sided

patches. In the continuous setup, these surfaces are often represented using ruled Bézier or B-Spline patches and developability is enforced using non-linear constraints [4, 5, 13].

Aumann [4] proposes a general condition required for a developable Bézier surface to interpolate two given Bézier curves. To compute such a Bézier surface, the presented method restricts the input boundary curves to lie in parallel planes. This requirement greatly simplifies the non-linear system of equations, though at the expense of the modeling capability of the developable patches.

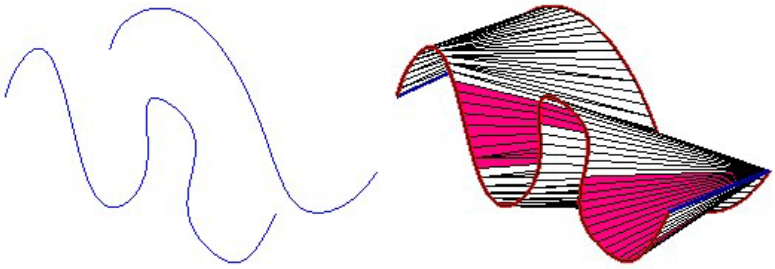
Chu and Séquin [13] derive Aumann's developability condition [4] geometrically from the de Casteljau algorithm. This formulation permits them to work with boundary Bézier curves lying in non-parallel control planes. In their method, given one freely specified boundary curve, the second boundary curve has five available degrees of freedom.

In a later work, Aumann [5] extends the de Casteljau style approach of [13] by increasing the number of available design parameters using degree elevation. The resulting algorithm generates a developable Bézier surface interpolating two given Bézier curves of arbitrary degree and shape.

A common requirement of these continuous methods is that users must clearly specify ruling directions for the final surface. This type of interaction assumes that users have sufficient geometric knowledge to know what ruling directions actually are, preventing these methods from being adopted by non-experts.

Wang and Tang [40] use a discrete setup for modeling torsal developable surfaces. The input to their method is two polyline directrices for the ruling and the output is a developable triangle strip where each interior edge approximates a ruling connecting the two directrices (Figure 3.3). They cast the problem as a Dijkstra's shortest path search [14] on a weighted solution graph whose vertices represent potential edges in the final triangulation. The shortest path corresponds to the optimal triangulation. Different optimization objectives, such as minimal area or maximal convexity, can be realized by varying the weights used in the solution graph.

Pottmann and Wallner [31] use a dual space approach to define a plane-



(a) Polyline Directrices (b) Torsal Boundary Triangulation

Figure 3.3: Torsal boundary triangulation interpolating polyline directrices [40].

based control interface for modeling developable patches. Controlling such an interface requires significant geometric expertise.

For garment design, a highly time consuming approach presented by some commercial modeling tools [10, 27] is to first design a planar pattern for the surface and then deform it into the desired shape using bending and physical simulation. Since designing sewing patterns is a challenging task, this approach is limited to the realm of expert users.

Bo and Wang [8] introduce a modeling system for developable surfaces with an emphasis on paper bending. The system utilizes the relationship between a torsal developable surface and a geodesic curve lying on that surface. Users isometrically manipulate a smooth 3D curve representing a geodesic and the system finds the unique torsal developable surface containing the user's curve. The input curve is reparameterized numerically at each time step to ensure an isometric deformation. Though composite developable surfaces are supported by the system, users must manually specify each individual torsal piece with a separate geodesic curve.

Frey [17] describes a method for computing discrete height-field developable surfaces that interpolate a given polyline (Figure 3.4). Given a user-provided projection plane, the method first computes all of the possible interior edges in the polygon formed by projecting the polyline to the plane. It then classifies edges in terms of their likelihood of being part of a developable surface, giving a higher priority to locally convex edges. Finally,

it selects a subset of the edges that forms a valid triangulation by simulating the bending caused by closing a *blankholder*. A blankholder is the component of a press machine that holds the *punch surface*, the surface that is pressed over an initially flat sheet (the blank) to cause the blank to bend. This algorithm operates under the assumption that the projection to the plane of the desired triangulation contains no self intersections, restricting the method to height field (2.5D) surfaces. While the method is well suited for predicting the bending of a metal sheet under the closing of a blankholder, using the technique for *modeling* developable surfaces is difficult since the only available control over the final surface is the user's choice of projection direction.

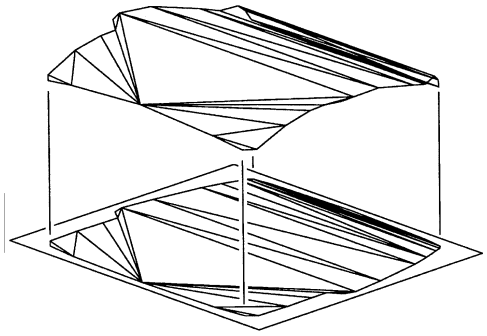


Figure 3.4: Modeling height field developable surfaces [17].

The following chapter introduces a novel algorithm to compute a developable boundary triangulation interpolating an arbitrary smooth input boundary.

Chapter 4

Constructing Developable Triangulations

As mentioned in Section 1.3, developable surfaces are difficult to model. Chapter 3 pointed out that approximation techniques may admit too much distortion to be practical for manufacturing setups. Direct modeling approaches, though guaranteeing analytically developable surfaces, are often difficult to control and require users to have significant geometric expertise. Additionally, most direct modeling approaches are limited to only modeling torsal developable surfaces. In order to model composite developable surfaces with these approaches, users must manually segment their desired composite surface into individual torsal pieces, a non-intuitive operation.

This chapter describes an algorithm for computing a developable boundary triangulation interpolating an input polyline. Combining this algorithm with a user interface for specifying 3D curves (Section 4.9) creates an easy-to-use system for modeling developable surfaces. Users simply specify a closed boundary and the algorithm returns a developable surface with desirable surface properties (Section 4.2) that interpolates the boundary. The system is easily accessible to non-experts since users are not required to have significant knowledge of geometry or the properties of developable surfaces.

The algorithm is sufficiently robust and can handle complex boundaries, including boundaries with darts (Section 4.5) and tangential discontinuities. The next section introduces the linkage between a developable surface interpolating a boundary curve and the convex hull of that curve and explains how this can be used as the basis for the algorithm.

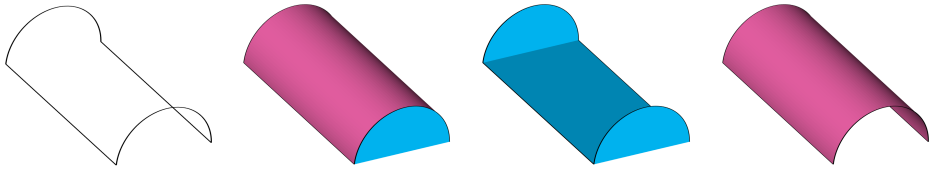
4.1 Developability and Convexity

Section 2.4 discussed the potential of boundary triangulations to represent developable surfaces that interpolate a given boundary polyline. As explained, triangulations that approximate smooth developable surfaces have the property that the majority of their edges are locally convex. A general method for obtaining such triangulations is now described. Section 4.3 refines this method to efficiently search for triangulations which satisfy additional requirements.

We make the following observation which forms the basis for our method: *Since most edges of a desirable triangulation must be locally convex, a natural place to identify developable regions interpolating a boundary polyline is the convex hull of the boundary, where* ***every*** *edge is locally convex.* This observation is well motivated by the continuous case where the convex hull of almost every sufficiently smooth closed space curve consists of planar regions and torsal developable surfaces, with the torsal developable surfaces interpolating parts of the curve [34]. We rely on this observation in order to significantly narrow the search space when looking for desirable triangulations.

If a smooth space curve lies entirely on its convex hull (i.e., the curve is convex), the hull is separated into two developable envelopes [34]. In the discrete case, the convex hull of a closed polyline is a triangular mesh containing a subset of the polyline's vertices. If the polyline is convex, the hull is separated into two developable triangulations. These triangulations are the left and right hull envelopes and are defined with respect to the orientation of the boundary (Figure 4.1). If the polyline is planar, then these envelopes are identical. By construction, both triangulations interpolate the polyline. Moreover, as desired, every interior edge in each triangulation is locally convex since it is an edge on the convex hull.

If the polyline is not convex, it will not separate its convex hull into two envelopes and a more sophisticated modeling strategy is required. As mentioned previously, the convex hull of almost every closed sufficiently smooth space curve consists of planar regions and torsal developable sur-



(a) Polyline (b) Convex Hull with Envelopes (c) Left Envelope (d) Right Envelope

Figure 4.1: Envelope triangulations for a convex polyline. The algorithm in Section 4.3 selects (d).

faces [34], where each of these torsal developable surfaces interpolates parts of the curve. If a polyline is sampled sufficiently densely from a smooth space curve, its convex hull will closely approximate the convex hull of the curve. We observe that the torsal developable surfaces on the hull of the continuous curve correspond to regions, or *charts*, of consecutive triangles on the polyline's hull having edges on the polyline (Figure 4.2(b,c,d,e)). Such charts are formally defined as sequences of hull triangles, such that:

1. each triangle shares at least one edge with another triangle in the same chart;

2. each triangle shares at least one edge with the input polyline;

3. all of the triangles are oriented consistently with respect to the polyline.

The second requirement implies that charts are separated from each other by *interior* triangles: triangles of the convex hull with no edges on the polyline (shown in brown in Figure 4.2(b)). The last requirement ensures that the triangulation constructed by the algorithm is manifold and orientable.

Subtracting each chart from the polyline by removing the portions of the polyline inside the chart and replacing them with the interior boundaries of the chart results in one or two smaller closed polyline subloops (Figure 4.2(c), (d), (e)). If the subloops lie on *their* convex hulls, their left and right envelopes will provide triangulations, which together with the removed chart

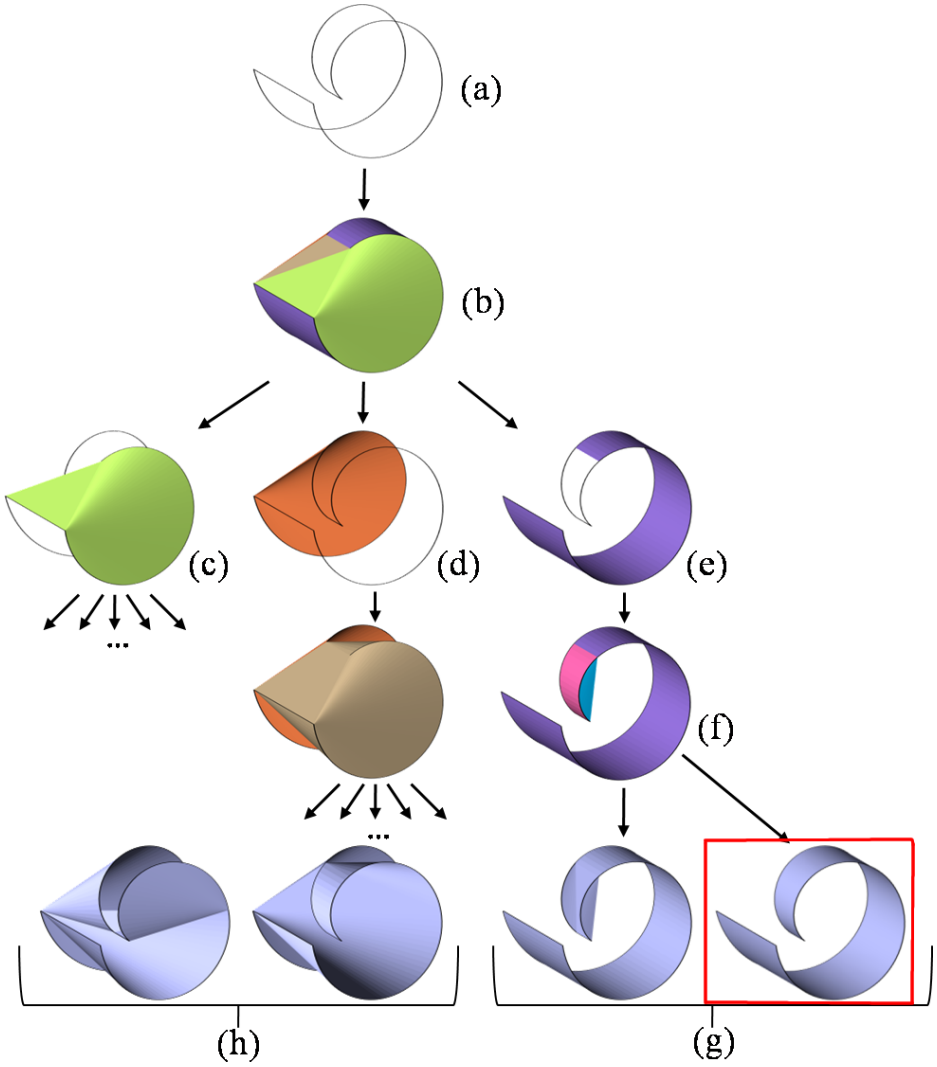


Figure 4.2: Extracting a locally convex triangulation: (a) boundary; (b) convex hull with extracted charts (interior triangle shown in black); (c), (d), (e) individual charts and remaining subloops after subtraction; (f) recursing on the subloop formed by removing the purple chart; (g) resulting triangulations (the framed triangulation is the one returned by the algorithm in Section 4.3); (h) two of the triangulations created with different chart choices.

will interpolate the original polyline (Figure 4.2(f)). If a subloop does not lie on its convex hull, we can identify charts on this convex hull and proceed recursively. By construction, charts on the subloop hulls will also correspond to torsal developable surfaces interpolating the original polyline.

It is theoretically possible, though unlikely, for a hull to contain no valid charts. In this pathological case, the algorithm treats each hull triangle as a separate chart.

The recursion is guaranteed to terminate as the number of polyline vertices decreases at each iteration and a polyline with three vertices always lies on its hull. In any resultant triangulation, the only potentially non-convex edges will bound adjacent triangles computed at different levels of the recursion. All other edges are necessarily locally convex as they originated from within a convex hull, either that of the original polyline or of one of the subloops. As desired, the number of non-convex edges is very small and is related to the boundary complexity and not to the number of boundary vertices. However, as shown in Figure 4.2(g) and (h), the choice of different charts to proceed from leads to drastically different triangulations, raising the question of which choice the user would prefer. The subsequent sections analyse the desirable shape characteristics of discrete developable surfaces and describe an algorithm which guides the selection to efficiently obtain a good interpolating surface.

4.2 Desirable Triangulation Properties

When considering triangulations which approximate a smooth developable surface, we require the majority of triangulation edges to be locally convex. An additional constraint, ignored in Section 4.1, is *smoothness*: requiring the dihedral angles between adjacent triangles to be low. Even with these two restrictions, there may exist multiple boundary triangulations providing a valid solution (see Figure 4.2 (g),(h)), raising the question which of these is expected when a particular boundary is specified. Clearly, when designing a modeling tool, *predictability* is a desirable property. Human perception studies indicate that "simplicity is a principle that guides our perception..."

[6]. This principle is well known in Gestalt theory and is commonly used in sketch interpretation [21]. In our work, it implies that the surface the user expects is the simplest developable surface fitting a given boundary. Based on numerous examples, we hypothesize that a surface is considered simpler and hence more predictable if its normal map has fewer branches, or equivalently, if its directrix has fewer discontinuities.

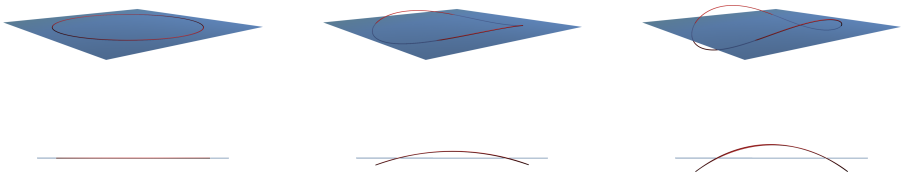
In addition to predictability, or instead of it, we can consider the *fairness* of the created surface. Frey [17] and Wang and Tang [40] describe a large set of measures of surface fairness, including metrics of mean curvature and bending energy. We found that minimizing the integral l^2 mean-curvature described as the sum of squared dihedral angles across interior edges results in visually fair triangulations. The advantage of this metric is that it can be extended to provide a lower bound on the fairness of a boundary triangulation given only a subset of its triangles (Section 4.4.1).

The next section presents a practical method for computing boundary triangulations that satisfy all of these requirements, and thus define which developable surfaces to output.

4.3 Branch-and-Bound Search Algorithm

We now extend the basic methodology described in Section 4.1 to search specifically for smooth triangulations and describe a procedure to efficiently navigate the search space to obtain triangulations that are predictable and fair.

We observe that for convex polylines, the two hull envelopes mentioned above are not necessarily the best solutions with respect to smoothness (see the pink and blue envelopes at iteration one in Figure 4.4). Therefore, our algorithm extracts not only these envelopes, but also the separate charts that are part of the convex hull. It then proceeds to explore possible interpolating triangulations that contain one or more of the identified charts. Fragmenting the envelopes into charts can increase the number of non-convex edges in the final triangulation. However, their number remains a function of boundary complexity and does not depend on the number of boundary vertices, as



(a) $\frac{\lambda_3}{\lambda_1} = \infty$ (b) $\frac{\lambda_3}{\lambda_1} = 132$ (c) $\frac{\lambda_3}{\lambda_1} = 33.5$

Figure 4.3: Planarity metric. The second row is a profile view of the first row. A higher value indicates a more planar polyline. The completely planar polyline in (a) has a value of ∞.

desired.

As mentioned previously, if the polyline is planar, then the two hull envelopes are identical. Clearly, if a polyline is planar then *any* manifold triangulation of it is developable. To measure the planarity of a polyline, we fit an orthogonal distance regression plane to it and consider the ratio $\frac{\lambda_3}{\lambda_1}$ of the largest and smallest eigenvalues of the covariance matrix [2]. This metric is positive, scale independent, and approaches infinity as the polyline becomes more planar. To illustrate, Figure 4.3 shows increasingly non-planar polylines superimposed with their regression planes and the values of their corresponding planarity metric. Polylines whose planarity metric is larger than a user provided threshold (we use 100,000 in our examples) are considered planar and triangulated by a single triangle fan. To prevent this specific choice of triangulation from influencing the fairness computation (Section 4.2), internal edges of the triangulation are marked as having a dihedral angle of zero across their adjacent faces.

To obtain smooth triangulations we require that any interior edge in a chart has a dihedral angle below a specified threshold. Charts with larger dihedral angles are not considered for future processing. For instance, in the first iteration of the algorithm in Figure 4.4, this invalidates the light and dark green charts. We also require the angles on edges between any chart and the adjacent interior triangles to lie below the threshold. We observe that since these edges are on the convex hull, the dihedral angle between

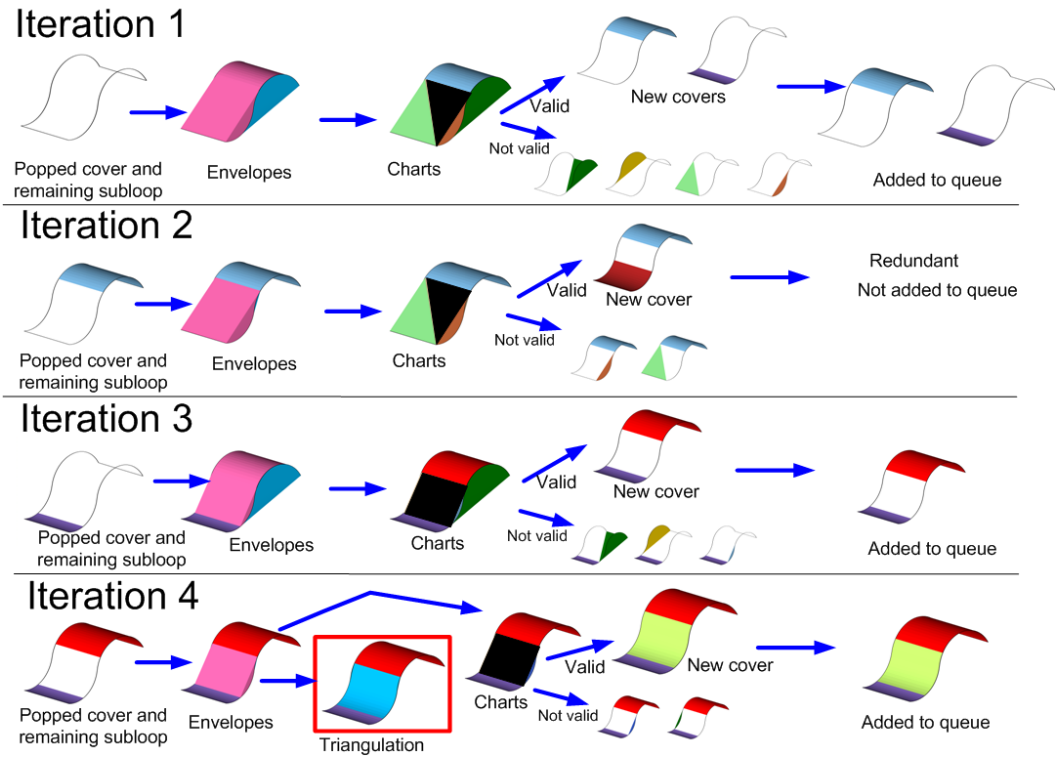


Figure 4.4: Algorithm stages on a simple example. Interior triangles are shown in black. The framed triangulation is the output. The cover pushed into the queue in iteration four will be discarded at iteration five in stage 1 (it is not better than the best triangulation).

the chart and any other triangle formed using these edges is bounded from below by the current angle. Charts which violate this property are also eliminated. In the first iteration in Figure 4.4, this invalidates the orange and dark yellow charts.

To reduce the number of non-convex edges and to speed up processing, we only consider charts larger than a certain percentage of the convex hull area (we use 1% - 3% in our examples). Both the angle and size thresholds can be adjusted depending on the input. If both are completely relaxed, our method will find a solution for practically any input.

Given these definitions of valid charts, our algorithm computes boundary triangulations that are unions of charts and envelopes. The algorithm uses a variation of the branch-and-bound approach [14], which helps drive the search towards a good solution while avoiding the exploration of non-promising ones. Similar to A* search [33], the method uses a priority queue of sets of charts, or *covers*, that interpolate segments of the polyline (Figure 4.4). The queue is initialized with the empty cover. The priority function of the queue is based on a potential metric (Section 4.4.2) and orders covers such that the next popped cover is expected to lead to an acceptable boundary triangulation fastest.

During processing, the method maintains the best boundary triangulation found to that point. The quality of a triangulation is measured with respect to the desired triangulation properties (Section 4.2). The same metric is used to measure the quality of a cover, as a lower bound on the quality of any possible triangulation containing this cover. At each iteration of the algorithm the following sequence of operations is performed as visualized in Figure 4.4.

1. **Pop Cover:** The algorithm pops a cover C from the priority queue, based on the potential metric. If a boundary triangulation was already found, the method compares the quality of the best triangulation found to the quality of C. If the quality of C is worse, it is immediately discarded. Otherwise, the method obtains the set of polyline subloops S formed by subtracting (as defined in Section 4.1) the cover charts

from the original boundary and computes their convex hulls.

2. **Explore Possible Triangulations:** The method checks if the convex hulls of each of the subloops are separable into two envelopes. If the envelopes exist for all the subloops, then each permutation of them combined with the cover's triangles defines a triangulation of the original boundary. Triangulations having interior dihedral angles above the smoothness threshold are discarded. In Figure 4.4, this discards all the boundary triangulations in iterations one through three. If there are multiple possible triangulations satisfying the smoothness constraint, the algorithm selects the highest quality one among them (Section 4.4.1). If this is the first triangulation found or if the new triangulation is better than the best triangulation found so far, then the best triangulation is appropriately updated.

3. **Form New Covers:** The method then extracts valid charts from the convex hulls of all the subloops in S. If a chart shares a boundary with the cover C, it tests if the dihedral angle across the shared edge satisfies the smoothness threshold. Charts which fail the test are discarded. For each of the remaining charts the method forms a new cover N combining C and the new chart.

4. **Add to Queue:** We observe that a subset of a new cover N may already be present in the priority queue. In this case, naively adding N to the queue can lead to repeated computations. To avoid this redundancy, the method checks if N contains a cover already in the queue. If this is not the case then N is added to the queue. If a subset of N is in the queue and the quality of N is better than that of the subset one, the subset cover is removed from the queue and N is inserted. If it is worse, then N is discarded. In Figure 4.4, iteration two, the blue-red cover is discarded since a better subset of it (the purple cover) was added to the queue at iteration one, and was not yet processed.

5. **Termination:** The algorithm terminates if the queue is empty or if

the best computed triangulation is deemed to be acceptable, using the measures described in Section 4.4.1. Otherwise, the algorithm goes back to Stage 1.

As mentioned previously, the algorithm uses a variation of the branch-and-bound approach [14] and operates similarly to A* search [33]. Like all branch-and-bound methods, the algorithm maintains the best solution found so far and uses it as an upper bound against which the current cover is compared. The algorithm is similar to A* in the sense that it uses a priority queue of partial solutions (i.e., covers) and processes them in the order most likely to lead to a solution the fastest. However, unlike classical A*, there is no a priori goal state. As described in Stage 5, the algorithm only terminates when the queue is empty or if the best solution found so far is acceptable. Additionally, to avoid repeated computation, the algorithm does not maintain an explicit *closed list* of previously processed covers. Rather, as described in Stage 4, the method avoids repeated computation by checking if the current cover entirely contains a cover already in the priority queue, and if so appropriately removes one of these.

The pseudocode for the algorithm is presented in Figure 4.5.

4.4 Metrics

4.4.1 Triangulation Quality

When evaluating triangulation quality, we consider two of the criteria discussed in Section 4.2: predictability and fairness. We do not need to take smoothness into account as the algorithm automatically discards non-smooth triangulations. To evaluate predictability, we compute the number of branching points on the surface normal map. In a discrete setup, these correspond to interior triangles in the triangulation and hence can be easily counted. Fairness is measured as the sum of squared dihedral angles across interior triangulation edges. Note that the optimum is zero for both metrics. In our setup, we consider predictability as more important than

```
Input: Polyline orig
best ← Null ;
PriorityQueue pq ;
pq.Insert(EmptyCover);
while pq not empty and best not good enough do
    C ← pq.Pop();
    if best is better quality than C then continue ;
    S ← orig.Subtract(C);
    ComputeConvexHulls(S);
    if every subloop ∈ S has envelopes then
        foreach permutation P of envelopes do
            if P + C is smooth then
                if P + C is better quality than best then
                    best ← P + C ;
                end
            end
        end
    end
    foreach subloop ∈ S do
        Charts ← ComputeCharts(hull of subloop);
        foreach chart ∈ Charts do
            N ← chart + C ;
            if N is not smooth then continue ;
            if N ⊇ some other cover R ∈ pq then
                if N is better quality than R then
                    pq.Remove(R);
                    pq.Insert(N);
                end
            else
                pq.Insert(N);
            end
        end
    end
end
return best
```

Figure 4.5: Pseudocode of main loop.

fairness. Thus, to compare two triangulations, we first compare predictability and only if the predictability is the same compare fairness.

When determining if a triangulation is acceptable (Stage 5), the two criteria can be compared against lower bounds set by the user. Using such lower bounds can speed up the processing, as the algorithm will terminate once an acceptable triangulation is found.

4.4.2 Cover Quality

We consider the same two criteria when evaluating a cover, wherein a cover evaluation aims to provide a lower bound on the quality of any triangulation that contains it. The lower bound on predictability measures the minimal number of interior triangles in any triangulation containing the cover. To compute this value, we consider the set of subloops S formed by subtracting the cover charts from the original boundary. We observe that if a subloop shares edges with more than two cover charts, any triangulation of it will contain at least one interior triangle[1]. A subloop which is adjacent to one or two cover charts can potentially be triangulated without any interior triangles. Thus the predictability metric of a cover is the number of subloops adjacent to more than two cover charts.

To measure the fairness of a cover we first compute the sum of squared dihedral angles within the cover charts and then add to it a lower bound on the sum of angles for the subloops in S computed as follows. If a subloop has two adjacent cover charts, we first fit an orthogonal distance regression plane to the subloop and compute the dihedral angles α_1 and α_2 between the plane and the chart triangles adjacent to the subloop (Figure 4.6). The sum of the two angles gives us a lower bound on the sum of angles on any interpolating triangulation of the subloop and between this triangulation and the adjacent charts. To bound the sum of squared angles, we assume equal distribution on all the $n-1$ edges involved, where n is the number of vertices on the subloop[2]. Thus for each such subloop we add to the fairness

[1]The triangulation has $n-2$ triangles and less than $n-3$ edges on the original boundary, where n is the polyline size. Hence at least one triangle has no boundary edges.

[2]We arrive at $n-1$ as the number of interior edges in the triangulation $n-3$ plus the

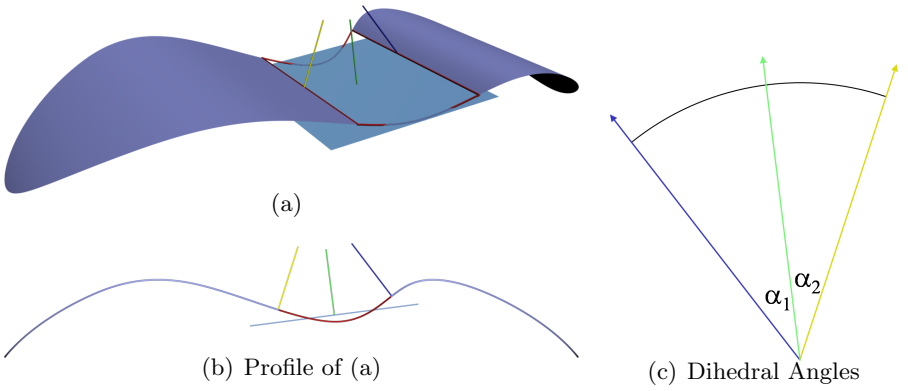


Figure 4.6: Computing a lower bound on fairness. (a) An orthogonal distance regression plane (blue) is fit to the red subloop and normals are calculated. (b) Profile view; (c) Notation used for the dihedral angles.

metric $(\alpha_1 + \alpha_2)^2/(n-1)$. If a subloop has more than two adjacent cover charts, we pick a random pair and do the same computation. If a subloop has only one adjacent chart, we return zero as an estimated lower bound for that subloop.

A cover and a triangulation or two covers are compared in the same way as two triangulations, by first considering predictability and then fairness. Since the cover quality is a lower bound, it can be safely used when deciding to discard a cover if it cannot lead to a triangulation better than the current one (Stage 1).

4.4.3 Cover Potential

The purpose of this metric is to prioritize covers based on their potential to be part of the expected final triangulation. The final triangulation is expected to have a very small number of interior triangles. Thus a cover is more likely to lead to an acceptable triangulation if it contains a small number of charts, where at least one of the charts is quite large. We first order the covers in ascending order based on the number of charts, and then

two edges adjacent to the charts.

(a) (b)

Figure 4.7: Modeling a cone by a boundary with a single dart.

in descending order based on the largest consecutive chart area.

4.5 Darts and Multiple Boundaries

Our method is the first to our knowledge to seamlessly handle darts as well as multiple boundary loops. Darts are duplicate edges on the boundary and are frequently used in design setups such as garment making to introduce points or lines of non-zero curvature onto the surface. To illustrate, Figure 4.7 shows how a cone can be modeled by a boundary with a single dart. The processing of darts is straightforward and requires only minor data-structure modifications to support coincident polyline vertices. When processing boundaries with multiple loops the method prioritizes processing of charts which connect separate loops before processing any other chart. If such charts are unavailable, the method connects the loops by the shortest tree of edges, treating those as interior edges for processing purposes.

4.6 Additional Modeling Control

The algorithm, as described, returns the best boundary triangulation computed, based on user indicated preferences in terms of quality metrics. Clearly, there might be cases when a user has additional constraints in mind. For instance, for the gazebo in Figure 5.6 we had a particular orientation in mind. We provide two mechanisms for users to explicitly control the final surface: specifying rulings and overriding optimal selection.

4.6.1 Specifying Rulings

To influence the final surface, users can specify a few of the rulings they expect to see on the result. These rulings are treated as triangulation edges which are constrained to be part of the final surface. For the purse example (Figure 5.11) we used this option to specify a ruling to the right of the handle, causing the purse to bulge outwards instead of curving inside (Figure 4.8). The specified edges segment the boundary into several separate subloops and the algorithm is run separately on each subloop, considering only the original polyline edges as boundary edges for chart extraction.

4.6.2 Overriding Optimal Selection

In addition to user drawn rulings, we provide another mechanism for obtaining alternative triangulations. Each time the algorithm computes a triangulation, it is immediately visualised and stored while the rest of the processing continues. The user thus has the option to interrupt the algorithm when they see a triangulation that they like, and they may also browse all the computed triangulations at any point during or after processing. The gazebo (Figure 5.6) was selected this way. Alternatives found by the method are shown in Figure 4.9.

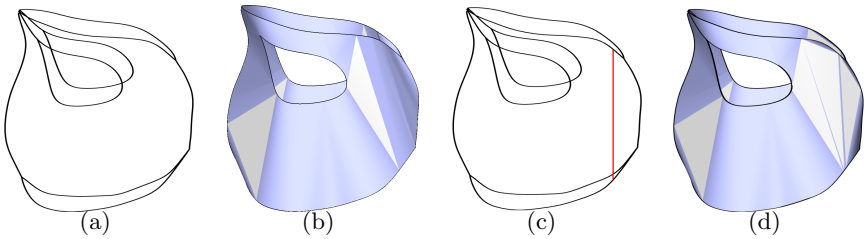


Figure 4.8: Specifying Rulings. (a) Original input boundary network; (b) Surface structure of (a); (c) Boundary network and specified ruling; (d) Surface structure of (c)

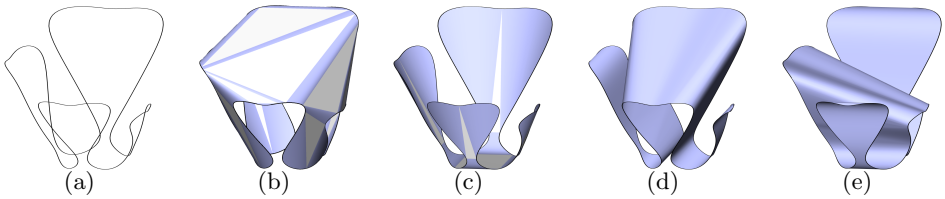


Figure 4.9: Alternative triangulations for the gazebo example (Figure 5.6) found by our method. (a) Original input boundary; (b) Solution used to make gazebo; (c), (d), (e) Alternative solutions.

4.7 Runtime

The search space for the algorithm is exponential in the number of charts found. However, using the priority queue combined with the potential and quality estimations, the method typically performs only a small number of iterations (less than two thousand for all the models shown in Chapter 5). At each iteration the dominant component of the runtime is the convex hull computation, which takes $O(n \log n)$ time in the number of vertices on the input boundaries. Thus, in practice, the overall runtime varies from a few seconds for simple models such as the Opera House (Figure 5.5), to a few minutes for more complex models. We observe that the total runtime strongly depends on the number of charts formed at each iteration, which is directly linked to the complexity of the input boundary rather than to the number of vertices on it. Table 4.1 lists running time statistics for several of the examples presented in Chapter 5. The total time is the amount of time passed until the priority queue of covers is empty, at which point the algorithm terminates since it has exhausted its search space. Since we permit overriding the optimal selection (Section 4.6.2), we also show the solution time which is the amount of time for the algorithm to first arrive at the presented surface as a solution. For some of the examples such as the skyscraper roof or the paper lamp leaf, the presented surface was found almost immediately, though the algorithm continued to find alternative solutions until its search space was exhausted. The running times were collected on a computer with an AMD Opteron 2218 processor and 4 GB of

Model	# Vertices	Solution Time (s)	Total Time (s)
Skirt (back)	737	17.8	19.9
Skirt (front)	736	31.2	34.7
Tanktop (back)	354	4.2	4.6
Tanktop (front)*	358	38.7	156.8
Opera House	245	2.8	2.8
Skyscraper Body	300 + 400	8.9	11.2
Skyscraper Roof	400	0.9	82.7
Paper Lamp Petal†	509	13.2	243.2
Paper Lamp Leaf†	300	1.4	476.0
Helmet (back middle)	214	3.8	4.0
Helmet (back left/right)	466	4.1	4.3
Helmet (front middle)	310	4.3	9.2
Helmet (front left/right)	835	15.7	17.8
Brown Chair	244	4.5	5.3
Red Chair	232	1.5	1.6

*1.5% †2%

Table 4.1: Running times for the algorithm on several examples. Solution time is the amount of time for the algorithm to first arrive at the presented surface as a solution. Total time is the amount of time passed until the priority queue of covers is empty and the algorithm has exhausted its search space. All of the examples were run with an area threshold of 3%, unless otherwise specified.

main memory.

4.8 Robustness

We observe that the topology of a convex hull is easily affected by noise in the input polyline. This can drastically affect the algorithm runtime as it leads to chart fragmentation and can sometimes also influence the resulting surface. To ensure a robust convex hull calculation, the algorithm expects smooth and sufficiently well sampled polylines as input.

To further increase the robustness of the hull calculation, the algorithm computes the center of mass C of the polyline boundary and slightly offsets each vertex radially from it. This offsetting effectively makes the curve more

"convex". This pre-processing drastically reduces the number of interior triangles on the hull and improves stability. The offsetting also bends nearly all planar portions of the boundary, which would otherwise allow for ambiguous triangulations and possible numerical issues when computing the planarity metric. Additional offsetting from a slightly shifted center is performed in the rare cases where C is in the same plane as part of the boundary.

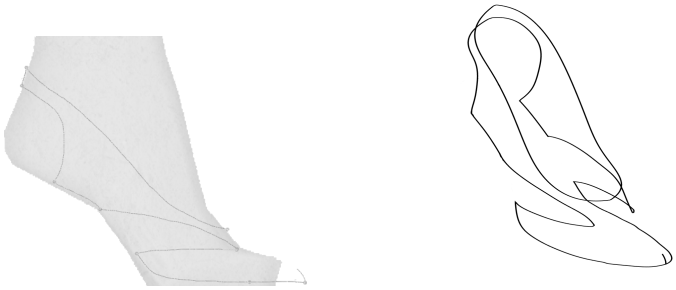
4.9 User Interface[3]

The algorithm requires as input a polyline boundary which is assumed to be sampled from an underlying smooth 3D space curve. There are several ways of specifying such a 3D space curve. For example, support for editing NURBS curves is common in most commercial modeling packages (e.g., [1], [3]). However, to make our modeling metaphor feasible for non-expert users, a fast, sketch-based interface, based on [15], is available. Though easy to use, this interface is sufficiently powerful to generate rich and complex examples. Indeed, the majority of the examples in Chapter 5 were created using this sketch-based interface.

In the interface, users can create the 3D boundary curves by first sketching them in one plane and then deforming them from a different viewpoint. Additionally, similar to [15], the sketching system infers depth information from a single sketch when the polyline is drawn over an existing model (Figures 4.10). The polyline is then set at a frontal distance to the model that interpolates the two distances at the extremities. This feature is especially useful for our garment examples (Section 5.1), where we drew the desired boundaries on top of a 3D mannequin automatically keeping the boundaries at the desired distance from the body.

The sketching system identifies darts as polyline sections that start from a closed boundary loop. When a dart is detected, this section is duplicated and added twice to the parent polyline while its orientation is switched, forming a single closed boundary. Lastly, when the tip of a dart reaches

[3]The user interface for specifying boundaries is not a contribution of this thesis and is discussed only for completeness.



(a) User's Sketch (b) Extracted Boundary

Figure 4.10: Sketching a shoe overtop of an underlying model of a foot.

the same boundary again, the latter is split into two loops, enabling easy generation of a boundary network.

Since a user's sketch may have small amounts of noise due to jitter in their hands, the interface smooths the sketch strokes by fitting a piecewise B-spline curve. This curve is then sufficiently sampled to create a polyline that is satisfactory for the algorithm.

4.10 Limitations

The theoretical setup of our algorithm assumes that the polyline boundaries are sampled from a sufficiently smooth curve. As shown by the examples, the algorithm remains robust even when this is not the case. As noted earlier, though it is possible that a convex hull may not contain any valid charts, such situations are extremely rare. If such a situation occurs the runtime is significantly increased, but the method is still guaranteed to find a solution.

We also observe that there may exist smooth developable surfaces where no ruling of the surface appears on the convex hull of its boundary. One such example is provided by Wang [42]. Consider a non-intersecting smooth curve C lying on the unit sphere centered at origin. C should be sufficiently long so that its convex hull contains the origin in its interior. An offset curve C' is then constructed by radially extruding C inwards towards the origin (Figure 4.11(a)). A developable surface D with boundaries C and C'

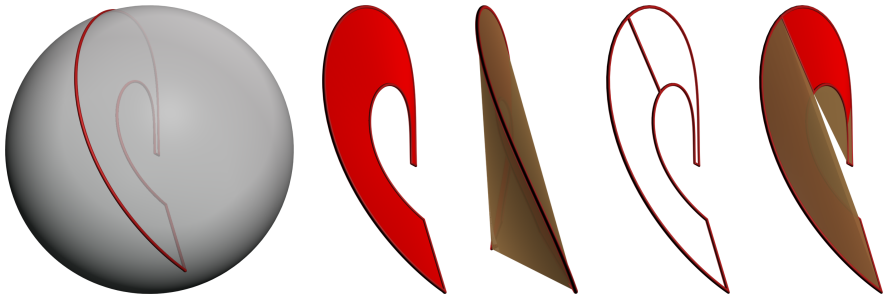


Figure 4.11: Boundary illustrating limitations. (a) Boundary curve defined with its outer portion lying on the unit sphere centered at the origin. The inward portion is created by extruding the outer portion radially towards the origin; (b) A developable surface is defined by connecting corresponding points on the outer and inner portions with straight lines; (c) The developable surface in (b) is entirely contained inside the convex hull of its boundary. None of the rulings of the surface lie on the convex hull; (d) The boundary is split into two parts by specifying a single ruling; (e) Each part how has a convex hull containing rulings from the surface in (b).

is then defined by connecting together corresponding points on C and C' (Figure 4.11(b)). Since all rulings pass through the origin and the origin lies inside the convex hull, no rulings lie on the convex hull. Therefore, D is entirely contained inside the convex hull of its boundary (Figure 4.11(c)). In cases like this, our method will not find the desired developable surface D. However, adding one or two extra rulings (Figure 4.11(d)) would typically break such surface into parts that partially lie on the respective boundary hulls and are thus computable by the method (Figure 4.11(e)).

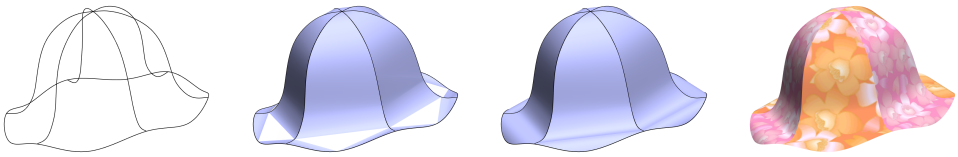
Chapter 5

Results

This chapter demonstrates the application of our method on a variety of inputs coming from different application areas where developables are used. The purple and white coloring of the surfaces shows the surface structure with torsal developable surfaces shown in purple and interior triangles, corresponding to planar transition regions, in white.

5.1 Garments

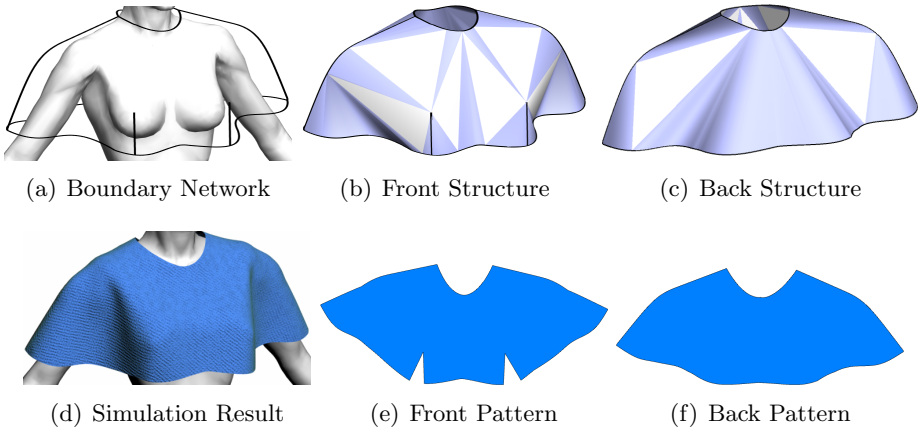
Figures 5.1, 5.2, 5.3, and 5.4 show several garments generated from simple sketches using our system. The modeling of each of the garments took only a few minutes compared to hours using traditional garment modeling tools such as [27] where the user is required to manually specify the 2D patterns for the garment. Real garments at rest are always piecewise developable since they are assembled from flat fabric pieces. Once worn by a character or a mannequin they stretch slightly due to gravity and collisions. The main challenge when modeling garments is obtaining the rest shape and the corresponding 2D patterns. Once these exist, standard simulation or procedural techniques can be applied to account for collisions and gravity [1, 15, 27]. In the examples in this thesis, we focused on obtaining the rest shapes. We then used a standard simulation tool, Autodesk 3ds Max with Reactor [1], to visualise the garment behaviour for the poncho, skirt and tanktop, and dress subject to the physical forces involved. As expected, the results after simulation appear less stiff but remain very similar to the developable rest shapes. We note that in all of the examples the garments are generated using a network of seams. Each individual panel surrounded by seams is a developable surface, but the surfaces are not developable across



(a) Boundary Network (b) Composite Solution (c) Torsal Solution (d) Rendering of (c)

Figure 5.1: Hat (modeled from six panels)

seams. Most of the examples in this section utilize darts as part of the input polylines which are robustly handled by our method. In most cases, the created surfaces are composite developable surfaces, each containing several torsal developable surfaces connected by transition planar regions. Since the created surfaces are analytically developable, the patterns (e.g., Figure 5.3(e)) can be used as-is to create reliable real-life replicas of the garments and the garment texture exhibits no distortion.



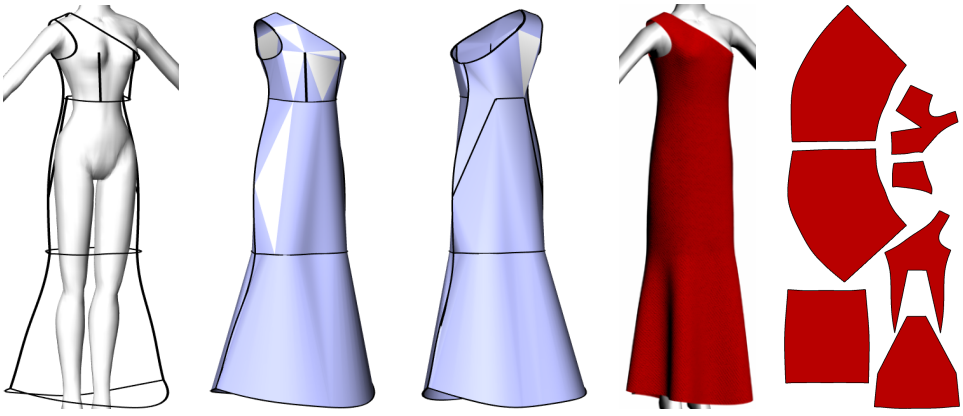
(a) Boundary Network (b) Front Structure (c) Back Structure

(d) Simulation Result (e) Front Pattern (f) Back Pattern

Figure 5.2: Poncho (modeled from two panels, front and back)

(a) Boundary Network (b) Front Structure (c) Back Structure (d) Simulation Result (e) Patterns

Figure 5.3: Tanktop (modeled from four panels) and Skirt (modeled from two panels)

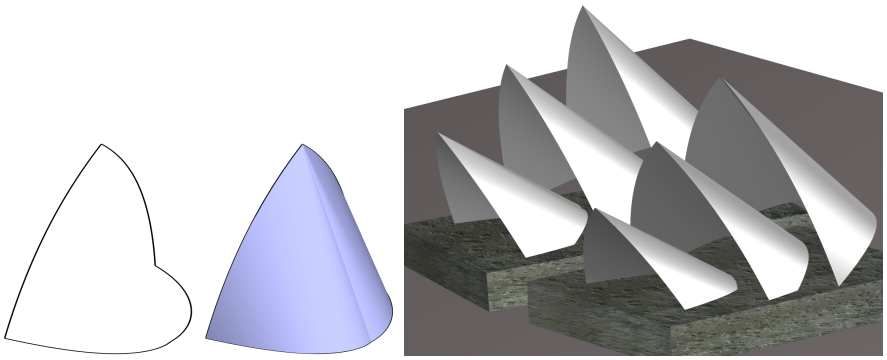


(a) Boundary Network (b) Front Structure (c) Back Structure (d) Simulation Result (e) Patterns

Figure 5.4: Dress (modeled from seven panels)

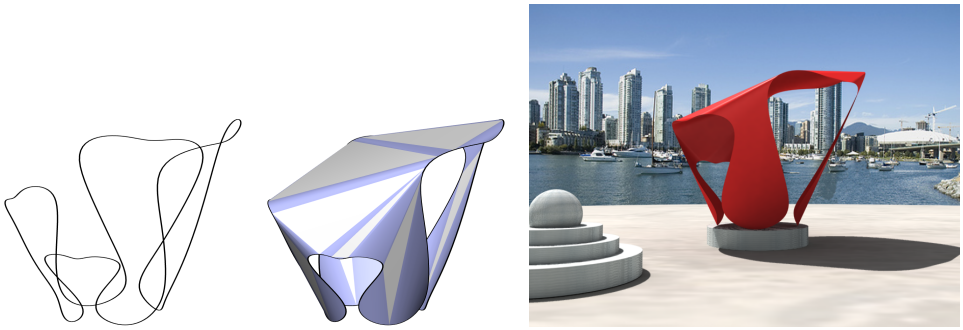
5.2 Architecture

Figures 5.5, 5.6, and 5.7 show examples of architectural structures generated using our method. The Opera House model (Figure 5.5) was inspired by the Sydney Opera House and created by duplicating a single developable surface six times at different scales. The boundary for the developable surface was generated using the sketch based system. The gazebo (Figure 5.6) is an example of a complex composite surface which cannot be projected to a plane without intersection and hence could not be easily generated by any previous method for modeling developables. It was modeled by sampling a B-spline curve with control vertices lying on the surface of a cone. The skyscraper (Figure 5.7) was created from two surfaces, one for the body of the building and one for the roof. The body surface is an example of a multi-loop boundary that is easily handled by our method. Similar to the gazebo, the roof of the skyscraper is a complex composite developable surface that cannot be projected to a plane without intersection.



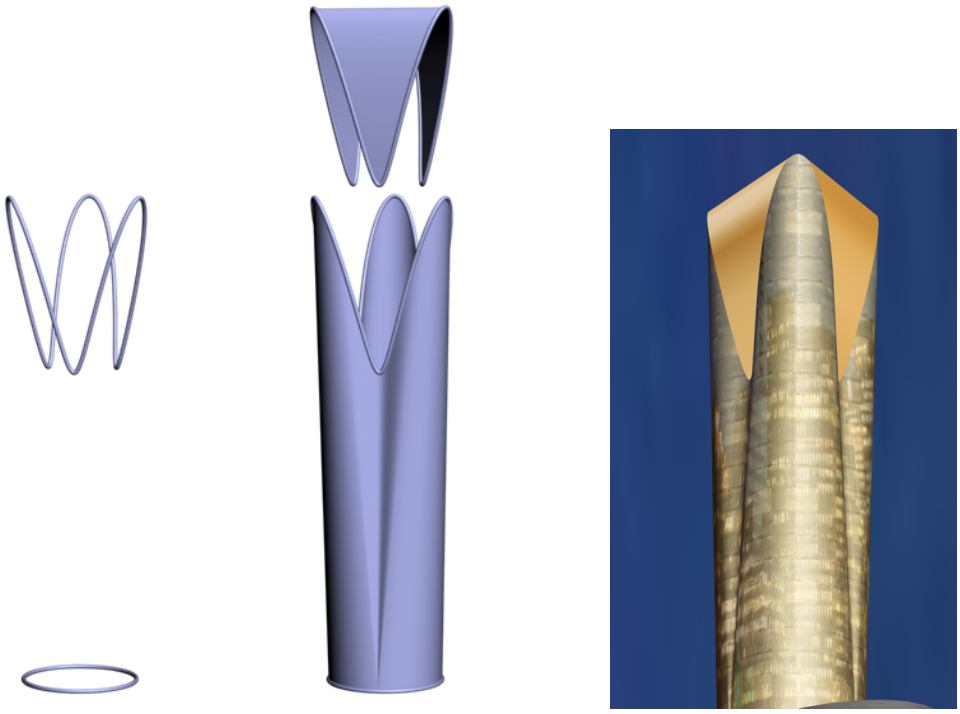
(a) Boundary (b) Surface Structure (c) Rendering

Figure 5.5: Opera House



(a) Boundary (b) Surface Structure (c) Rendering

Figure 5.6: Gazebo

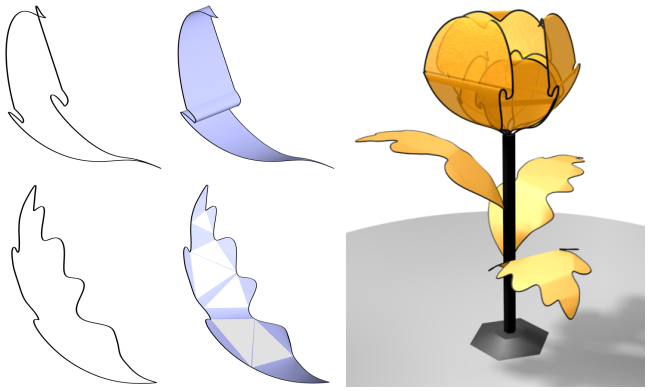


(a) Boundary (b) Surface Structure (c) Rendering

Figure 5.7: Skyscraper

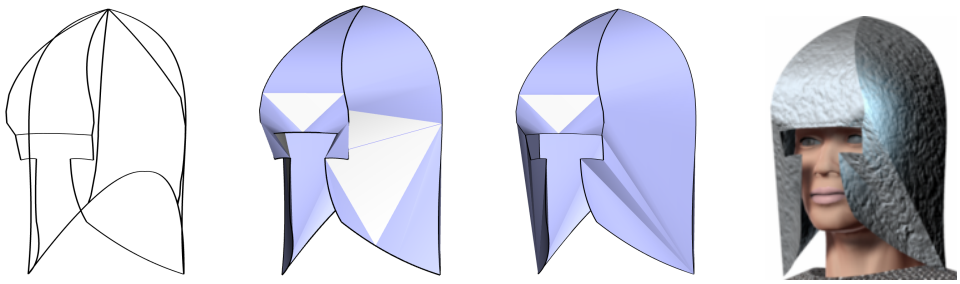
5.3 Paper Design

Paper and foil are both commonly used materials used in the design of home artifacts. The tulip lamp modeled from developable petals and leaves (Figure 5.8) is mimicking Art-Nouveau paper lamps. The flower is created by duplicating and scaling a developable petal surface. While the gold-foil leaves are composite developable surfaces the paper petals are torsal ruled surfaces and thus could be modeled by previous techniques, e.g. [40]. However, in contrast to these approaches, with our approach the user is not required to specify the ruling directions or even know what ruling directions are, allowing non-experts to use the system. Furthermore, the method, not the user, is able to determine that a single torsal developable surface interpolating the boundary exists, a non-trivial observation, making the method more attractive for non-experts.



(a) Boundaries (b) Surface Structures (c) Rendering

Figure 5.8: Tulip paper lamp with developable paper petals and gold-foil leaves

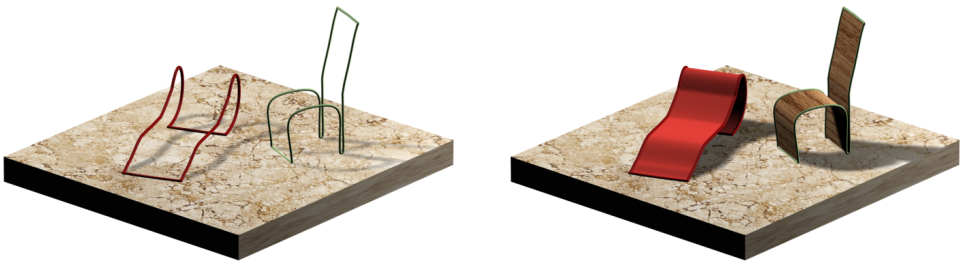


(a) Boundary Network (b) Smooth Solution (c) Sharp Angles Solution (d) Rendering of (c)

Figure 5.9: Helmet. (b) Solution obtained with a smoothness threshold of 60°; (c) Solution obtained with a smoothness threshold of 100°. By relaxing the smoothness threshold, we create the appearance of metal ridges.

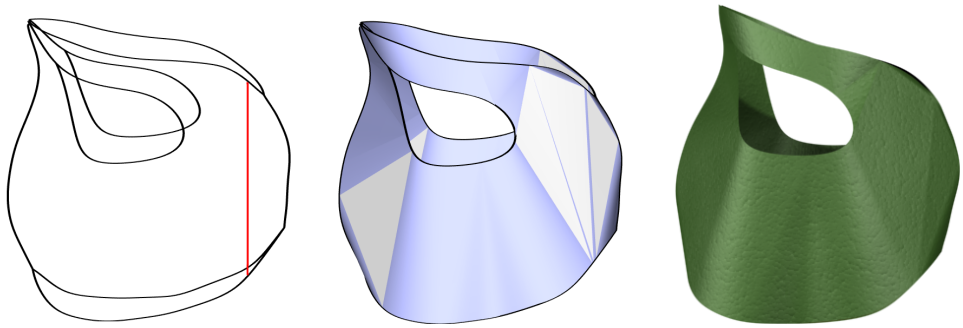
5.4 Leather, Wood Veneer, and Metal Goods

Figures 5.9, 5.10, 5.11, 5.12, and 5.13 show a variety of objects designed from flat sheet materials: a metal helmet, chairs, a leather purse, a shoe, and a glove. Despite the complexity of the modeled surfaces, no modeling expertise was required when sketching them using free-form drawing. The examples also show the control mechanisms available to the user, such as the use of rulings to guide the construction of the purse as explained in Section 4.6.1 and the impact of smoothness threshold in the helmet example, where we relaxed the threshold to create the appearance of metal ridges. The shoe example (Figure 5.12) was inspired by a pair of actual woman's shoes (Figure 5.12(a)). The actual shoes were fabricated with a single piece of leather and were developable almost everywhere, the only exception being the tip where the leather was rounded using a heating process. To model this effect using only developable surfaces, we inserted a rounded dart at the tip of our boundary (5.12(b)). Like the garment examples (Section 5.1), all of these results are analytically developable, permitting real life replicas to be manufactured.



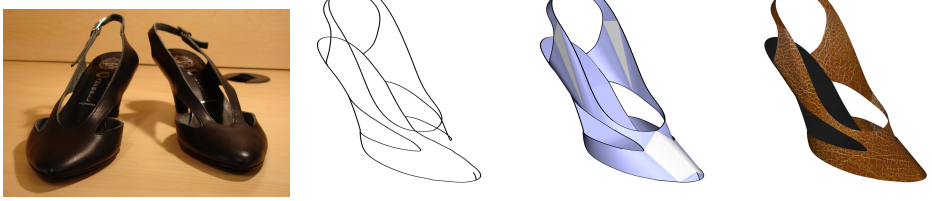
(a) Boundaries (b) Rendering

Figure 5.10: Chairs



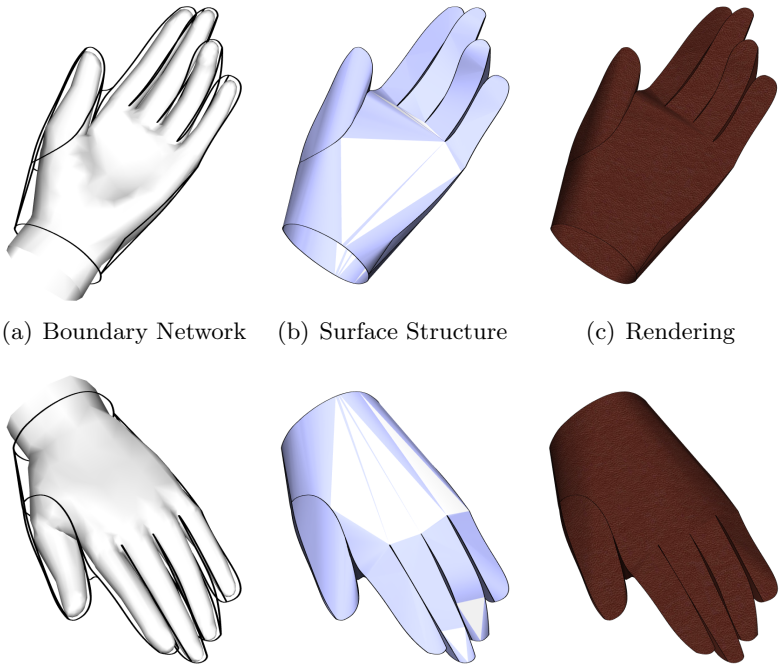
(a) Boundary Network and Specified Ruling (b) Surface Structure of (a) (c) Rendering of (b)

Figure 5.11: Purse



(a) Reference Photo (b) Boundary (c) Surface Structure (d) Rendering

Figure 5.12: Shoe



(a) Boundary Network (b) Surface Structure (c) Rendering

(d) Boundary Network (e) Surface Structure (f) Rendering

Figure 5.13: Glove

Chapter 6

Summary

This thesis presents an algorithm for computing developable surfaces interpolating arbitrary 3D polygonal boundaries. Combining this algorithm with an interface for editing 3D curves results in an easy to use system for modeling composite developable surfaces in which users simply specify the boundaries of each surface patch. Unlike many previous approaches for modeling developable surfaces, significant geometric expertise is not prerequisite, which is an attractive option for non-expert users. However, the method permits user interaction and optimization of different surface properties, granting more advanced users the ability to finely control the output surface.

The algorithm is based on the linkage between between a developable surface interpolating a boundary curve and the convex hull of that curve. The method explores the space of possible interpolating developable surfaces searching for solutions which have a desired set of shape properties. The presented approach is fairly generic and can be easily extended to handle additional shape metrics. The runtime of the algorithm is strongly affected by the complexity of the input boundary, and to a lesser degree, the number of vertices on the boundary. In practice, the overall runtime is low, ranging from a few seconds for simple boundaries to a few minutes for more complex boundaries.

As demonstrated by the numerous examples in Chapter 5, the method robustly computes developable surfaces interpolating a vast array of input boundaries and can be used to model an assortment of developable shapes. Darts are directly supported by the method, making it especially practical for garment design setups. In all of the presented examples, since the results are analytically developable, the flattened 2D patterns are available. This

permits real replicas of the shapes to be fabricated.

Chapter 7

Future Work

The algorithm presented in Chapter 4 presented smoothness, predictability, and fairness as desirable surface characteristics. These characteristics were used to guide the algorithm towards an interpolating surface that optimized these traits. However, one could consider alternative properties to optimize for. For example, Wang and Tang [40] identify minimal surface area, minimal bending energy, minimal mean curvature variation, and minimal normal variation as optimization objectives. Exploring these and other surface metrics would permit additional user control and further improve the algorithm.

Given a network of boundary curves, the algorithm runs separately on each surface patch, returning a developable surface interpolating each. The resulting composite developable surface thus has C^0 continuity across its seam lines and the surface is not developable across seams. However, in certain designs, higher orders of continuity are required. For example, C^2 continuity is necessary to ensure continuous highlights and reflection lines. One approach to support higher geometric continuity would be to modify the algorithm so that it is globally aware of all of the patches, not just locally aware of its current patch. When initially selecting charts on the convex hull of a patch, it could favour charts having a corresponding "continuation chart" on the convex hull of an adjacent patch.

Additional support for singularities would be an interesting endeavour. Though the algorithm currently supports darts as a mechanism for introducing singular lines, less restrictive results may be achieved by exploring other types of singularities (e.g., crescent singularities and stretching ridges [26]).

Finally, formalized user studies in which individuals are tasked with de-

signing developable objects would be beneficial. In addition to helping identify strong and weak points of the user interface, user studies could possibly provide a notion of the amount of design experience required to use the system.

Bibliography

[1] Autodesk 3ds Max. http://www.autodesk.com/3dsmax.

[2] Sung Joon Ahn. *Least Squares Orthogonal Distance Fitting of Curves and Surfaces in Space (Lecture Notes in Computer Science)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2004.

[3] Autodesk AliasStudio. http://www.autodesk.com/aliasstudio.

[4] Günter Aumann. Interpolation with developable Bézier patches. *Computer Aided Geometric Design*, 8(5):409–420, November 1991.

[5] Günter Aumann. Degree elevation and developable Bézier surfaces. *Computer Aided Geometric Design*, 21(7):661–670, 2004.

[6] John. G. Benjafield. *The developmental point of view. A history of psychology*. Needham Heights, MA: Simon and Schuster Company, 1996.

[7] bluffton.edu. http://www.bluffton.edu/~sullivanm/ohio/cleveland/gehry/0067minuslights.jpg.

[8] Pengbo Bo and Wenping Wang. Geodesic-controlled developable surfaces for modeling paper bending. *Eurographics (Computer Graphics Forum)*, 26(3), 2007.

[9] burdamode.com. http://www.burdamode.com/Free_Downloads,1333669-1413206-1333668,enEN.html.

[10] Catia. http://www.3ds.com/products-solutions/plm-solutions/catia/all-products/domain/Mechanical_Design/product/SMD.

[11] Julie Steele Chalfant. Analysis and design of developable surfaces for shipbuilding. Master's thesis, Massachusetts Institute of Technology, June 1997.

[12] H.-Y. Chen, I.-K. Lee, Stefan Leopoldseder, Helmut Pottmann, Thomas Randrup, and Johannes Wallner. On surface approximation using developable surfaces. *Graphical Models and Image Processing*, 61(2):110–124, 1999.

[13] Chih-Hsing Chu and Carlo H. Séquin. Developable Bézier patches: properties and design. *Computer-Aided Design*, 34(7):511–527, 2002.

[14] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[15] Philippe Decaudin, Dan Julius, Jamie Wither, Laurence Boissieux, Alla Sheffer, and Marie-Paule Cani. Virtual garments: A fully geometric approach for clothing design. *Computer Graphics Forum (Proc. Eurographics)*, 25(3):625–634, Sep 2006.

[16] Manfredo Perdigao do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, 1976.

[17] William H. Frey. Boundary triangulations approximating developable surfaces that interpolate a closed space curve. *International Journal of Foundations of Computer Science*, 13:285–302, 2002.

[18] William H. Frey. Modeling buckled developable surfaces by triangulation. *Computer-Aided Design*, 36(4):299–313, 2004.

[19] Carl Friedrich Gauss. *Disquisitiones generales circa superficies curvas*. Comm. Soc. Reg. Sc. Gott. Rec., 1828.

[20] Dan Julius, Vladislav Kraevoy, and Alla Sheffer. D-charts: Quasi-developable mesh segmentation. *Computer Graphics Forum (Proc. Eurographics)*, 24(3):581–590, 2005.

[21] Olga A. Karpenko and John F. Hughes. Smoothsketch: 3D free-form shapes from complex sketches. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 589–598, New York, NY, USA, 2006. ACM Press.

[22] Erwin Kreyszig. *Differential Geometry*. Dover Publications, 1991.

[23] Steven R. Lay. *Convex Sets and their Applications*. Wiley, New York, 1972.

[24] Stefan Leopoldseder and Helmut Pottmann. Approximation of developable surfaces with cone spline surfaces. *Computer-aided Design*, 30(7):571–582, 1998.

[25] Yang Liu, Helmut Pottmann, Johannes Wallner, Yong-Liang Yang, and Wenping Wang. Geometric modeling with conical meshes and developable surfaces. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 681–689, New York, NY, USA, 2006. ACM Press.

[26] Alexander Lobkovsky, Sharon Gentges, Hao Li, David Morse, and T.T. Witten. Scaling properties of stretching ridges in a crumpled elastic sheet. *Science*, 270(5241):1482–1485, December 1995.

[27] Autodesk MayaCloth. `http://caad.arch.ethz.ch/info/maya/manual/MayaCloth`.

[28] Jun Mitani and Hiromasa Suzuki. Making papercraft toys from meshes using strip-based approximate unfolding. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 259–263, New York, NY, USA, 2004. ACM Press.

[29] Martin Peternell. Developable surface fitting to point clouds. In *Computer Aided Geometric Design*, pages 785–803, 2004.

[30] Helmut Pottmann and Johannes Wallner. Approximation algorithms for developable surfaces. *Computer Aided Geometric Design*, 16(6):539–556, July 1999.

[31] Helmut Pottmann and Johannes Wallner. *Computational Line Geometry*. Springer Verlag, 2001.

[32] Kenneth Rose, Alla Sheffer, Jamie Wither, Marie-Paule Cani, and Boris Thibert. Developable surfaces from arbitrary sketched boundaries. *Eurographics Symposium on Geometry Processing*, pages 163 – 172, 2007.

[33] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition, 2003.

[34] Vyacheslav D. Sedykh. Structure of the convex hull of a space curve. *Journal of Mathematical Sciences*, 33(4):1140–1153, 1986.

[35] Idan Shatz, Ayellet Tal, and George Leifman. Paper craft models from meshes. *The Visual Computer*, 22(9-11):825–834, September 2006.

[36] Dennis R. Shelden. *Digital surface representation and the constructibility of Gehry's architecture*. MIT, 2002.

[37] Meng Sun. A technique for constructing developable surfaces. Master's thesis, University of Toronto, June 1995.

[38] synfo.com. `http://www.synfo.com/theyachtreport/articles/Explorer1.jpg`.

[39] Charlie C.L. Wang and Kai Tang. Achieving developability of a polygonal surface by minimum deformation: a study of global and local optimization approaches. *The Visual Computer*, 20(8-9):521–539, 2004.

[40] Charlie C.L. Wang and Kai Tang. Optimal boundary triangulations of an interpolating ruled surface. *Journal of Computing and Information Science in Engineering, ASME Transactions*, 5(4):291–301, 2005.

[41] Charlie C.L. Wang, Yu Wang, and Matthew Yuen. On increasing the developability of a trimmed nurbs surface. *Engineering with Computers*, 20(1):54–64, March 2004.

[42] Wenping Wang. Personal Communication, 2007.

[43] Hitoshi Yamauchi, Stefan Gumhold, Rhaleb Zayer, and Hans-Peter Seidel. Mesh segmentation driven by gaussian curvature. *The Visual Computer*, 21(8-10):649–658, September 2005.