Developable Surfaces from Arbitrary Sketched Boundaries

Kenneth Rose[†], Alla Sheffer[†], Jamie Wither[‡], Marie-Paule Cani[‡], Boris Thibert[§]

Abstract

Developable surfaces are surfaces that can be unfolded into the plane with no distortion. Although ubiquitous in our everyday surroundings, modeling them using existing tools requires significant geometric expertise and time. Our paper simplifies the modeling process by introducing an intuitive sketch-based approach for modeling developables. We develop an algorithm that given an arbitrary, user specified 3D polyline boundary, constructed using a sketching interface, generates a smooth discrete developable surface that interpolates this boundary. Our method utilizes the connection between developable surfaces and the convex hulls of their boundaries. The method explores the space of possible interpolating surfaces searching for a developable surface with desirable shape characteristics such as fairness and predictability. The algorithm is not restricted to any particular subset of developable surfaces. We demonstrate the effectiveness of our method through a series of examples, from architectural design to garments.

1. Introduction

Developable surfaces, namely those that can be unfolded into the plane with no distortion, are present in every object made from fabric, paper, leather, or metal and wood sheets. Due to their aesthetic appeal, they are frequently used in architectural design [She02], home artefacts, and modern art [Hil66, AKA06].

Despite their ubiquity, developable surfaces remain difficult to model, particularly for non-expert users. Traditional approaches [Rhi, Cat, PW01, CS02, Aum04, WT05] focus on construction of four-sided developable patches and typically require the user to explicitly specify the directrices or ruling directions of the surface. Frey [Fre02] introduced a simpler modeling approach, presenting a method for generating a discrete developable surface, or boundary triangulation, interpolating a given closed polyline. However, the method is restricted to height-field surfaces, i.e. surfaces that can be projected onto the XY plane with no self-intersection. The resulting surface depends on the choice of projection direction.

We introduce a robust and easy to use sketch-based system for modeling general developable surfaces which can be used even by non-experts to generate sophisticated sur-
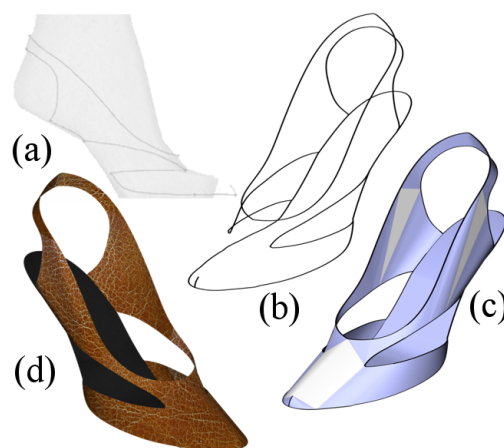

Figure 1: *Modeling developable surfaces (shoe upper and sole) from sketched boundaries: (a) upper boundary sketched over a foot model; (b) extracted contours; (c) structure of the obtained developable surface with torsal surfaces shown in blue and planar transition regions in white; (d) textured model.*

faces such as the shoe in Figure 1. Using our interface the users simply sketch the boundaries of each surface patch as a 3D polyline. If desired, they can provide additional hints to guide the construction towards a specific shape. To enable this modeling paradigm, we introduce a novel method for creating general discrete developable surfaces which interpolate *arbitrary* boundaries. We observe the correlation

---

[†] University of British Columbia
[‡] Inria Rhone-Alpes
[§] Université Joseph Fourier

between the discrete developable surfaces that interpolate a boundary polyline and the convex hull of that polyline. We use this linkage as the basis for a novel algorithm that generates interpolating developable surfaces for any given input polyline boundary. Our 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.

2. Background

2.1. Developable Surfaces

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

Developable and Ruled Surfaces: Developable surfaces are considered a special case of ruled surfaces. A ruled surface is a surface that can be swept out by moving a line in space along a prescribed *directrix curve* [dC76]. It is well-known that a G^2 surface is developable if and only if it is a ruled surface whose normals are constant along each ruling [PW01]. Therefore, ruled surfaces can be classified as developable or *warped* according to the behaviour of the tangent plane to the surface along any given ruling. Additionally, the tangent plane along a given ruling of a developable surface is almost always a supporting plane of a region on the surface containing the ruling, *i.e.* the tangent plane does not intersect the surface locally [Lay72]. A typical ruling on a developable surface therefore lies *on* the local convex hull of the surface [Fre02]. In contrast, on a warped surface the majority of rulings lie inside the local convex hull.

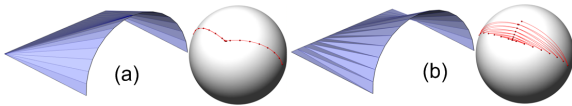**Figure 2:** *Locally convex (left) and non-convex (right) interior triangulation edges P_iP_j.*

Figure 3: *Developable and warped ruled triangulations interpolating the same polyline and their normal maps.*

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
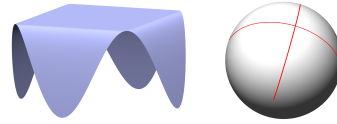
Figure 4: *A general developable surface (left) and its normal map (right).*

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, a triangulation approximates a developable surface at the limit if and only if the majority of its interior edges are *locally convex* [Fre02]. 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 [Fre02] (see Figure 2). An interior edge is *non-convex* if it lies inside this convex hull. For a triangulation to approximate a smooth developable surface, the number of non-convex edges should remain nearly constant as the sampling density of the polyline increases. Figure 3 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 3(a)). In the second case, the majority of edges are non-convex (Figure 3(b)).

Projective Geometry of Developable Surfaces: An important characteristic of developable surfaces is that their normal map is one-dimensional [PW01]. In the general case (Figure 4), the normal map is a network of curves. If the normal map is a single curve, then the directrix of the surface is a single continuous curve. Pottman and Walner [PW01] refer to these surfaces as *developable ruled surfaces* or *torsal ruled developable surfaces*. To avoid confusion with ruled surfaces we will refer to these surfaces as *torsal* developable surfaces, in contrast to *general* developable surfaces whose map is a network of curves. A general developable surface is thus made of a union of torsal developable surfaces joined together by transition planar regions [dC76], where the latter correspond to the branching points on the normal map.

2.2. Previous Work

In computer graphics and modeling, developable surfaces have raised interest in several different contexts including reconstruction from point clouds [CLL*99, Pet04] and mesh segmentation into nearly developable charts for parameterization and pattern design [JKS05, YGZS05, STL06]. The following review only covers methods for modeling developables either via developable approximation or directly.

Developable Approximation: Given an existing non-developable surface, a large number of methods aim at approximating it with one or more developable surfaces, including [MS04, WT04, LPW*06, DJW*06]. Mitani and Suzuki [MS04] approximate arbitrary meshes by triangular strips; unfolding the latter creates 2D patterns for pa-
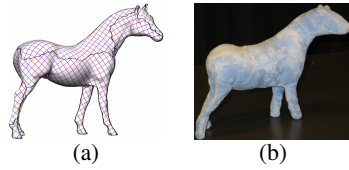
Figure 5: *Artefacts in using approximate developables [JKS05] for manufacturing: (a) approximate developable segmentation (L^2 stretch 1.01); (b) reassembled model.*

per craft toys that roughly approximate the initial geometry. Wang and Tang [WT04] increase the developability of a mesh surface by minimizing its Gaussian curvature. A similar approach is used by Frey [Fre04] to introduce singular vertices into a developable triangulation. Decaudin et al. [DJW*06] use overlapping mesh patches, computing for each mesh patch the locally best approximating developable surface and deforming the mesh towards this surface. A related problem is addressed by Liu et al. [LPW*06] who present a planarization, or development, algorithm applicable to the special case of planar quad strips. Combined with subdivision it can be used to model nearly developable surfaces.

Generally speaking, the approximation approach is highly restricted, as 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 in manufacturing setups, where the surfaces need to be realised from actual planar patterns, such as sewing. In these setups the distortion caused by using unfolded patterns from approximate developables can be quite significant as demonstrated in Figure 5. In this example coming from [JKS05] the horse model was segmented into nearly developable charts unfolded into the plane with L^2 stretch of less than 1.01 [JKS05]. However as demonstrated, when the patterns created from the unfolding were sewn back together, the resulting toy horse had significantly different proportions from the initial model.

Direct Modeling: Most existing methods for modeling developable surfaces consider only torsal developable surfaces, i.e. surfaces whose normal map is a 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 [Aum04, CS02]. Users are required to clearly specify the ruling direction for the surface. Wang and Tang [WT05] 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. Pottman and Walner [PW01] use a dual space approach to define a plane-based control interface for modeling developable patches.

Controlling such an interface requires significant geometric expertise.

A highly time consuming alternative presented by some of the commercial modeling tools is to first design a planar pattern for the surface and then deform it into the desired shape using bending and physical simulation [May, Cat].

Frey [Fre02] describes a method for computing discrete height-field developable surfaces that interpolate a given polyline. This approach is more consistent with the recent trend towards sketch [KH06] or curve based [SF98] interfaces. Given a user-provided projection plane, the method first computes all 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 blank holder. This setup operates under the assumption that the projection to the plane of the desired triangulation contains no self intersections, restricting the method to height-field surfaces. The output of the method depends on the projection direction.

Descriptive geometers use local convexity to manually locate regions on a boundary curve that can be interpolated by torsal surfaces and planes [PW01] and then connect those into a single surface. Inspired by those as well as Frey [Fre02] and Wang and Tang [WT05] we use the local convexity property of developable surfaces to guide our algorithm. However, our method is not restricted to special limited cases of height-fields or strips and thus to the best of our knowledge is the first algorithmic approach for robustly modeling any type of developable surfaces. Our method requires far less user input than most existing techniques allowing non-expert users to create complex models. Finally, by exploring the space of possible interpolating surfaces, it allows greater user control of the resulting surfaces.

Sketch-based Modeling: Modeling of surfaces using networks of boundary curves described via sketching [KH06, IMT99] or 3D manipulation [SF98] is becoming increasingly popular. Recently, Decaudin et al [DJW*06] proposed a sketch-based system for modeling garments, which are a special case of developable surfaces. They inferred a non-developable surface from the sketch requiring subsequent developable approximation. We adopt the sketching framework for modeling of developable surfaces and use it to obtain the 3D boundaries of the modeled surfaces (Section 6).

3. Toward Locally Convex Triangulations

Section 2.1 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. We now
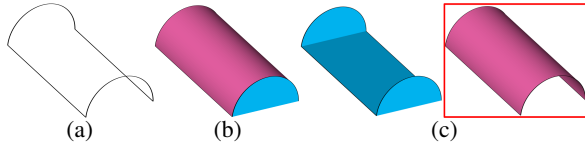
Figure 6: *Envelope triangulations for a polyline that lies on its convex hull: (a) polyline; (b) convex hull with envelopes; (c) the two envelope triangulations, the framed (right) one is the one selected by the algorithm in Section 5.*
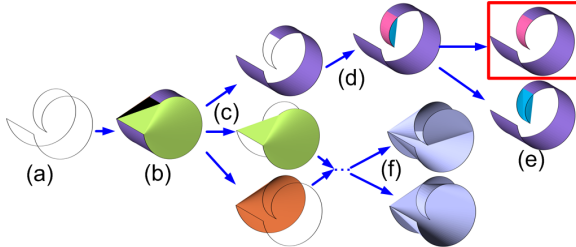

Figure 7: *Extracting a locally convex triangulation: (a) boundary; (b) convex hull with extracted charts (interior triangle shown in black) (c) individual charts and remaining subloops after subtraction; (d) recursing on the subloop formed by removing the purple chart; (e) resulting triangulations (the framed triangulation is the one returned by the algorithm in Section 5); (f) two of the triangulations created with different chart choices.*

describe a general method for obtaining such triangulations. Section 5 extends this method to search for triangulations which satisfy additional requirements.

The following observation 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.* We rely on this observation to narrow the search space when looking for desirable triangulations.

The convex hull of a closed polyline is a triangular mesh, containing a subset of the polyline's vertices. If the polyline lies entirely on its convex hull, it separates the hull into two triangulations, the left and right hull *envelopes*, defined with respect to the boundary orientation (Figure 6). 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 does not lie entirely on the convex hull, it will not separate the hull into two envelopes and a more sophisticated modeling strategy is required. The convex hull of almost every closed sufficiently smooth space curve consists of planar triangles and torsal developable surfaces [Sed86], 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 con-

secutive triangles on the polyline's hull having edges on the polyline (Figure 7(b,c)). Formally we define such charts 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 the triangles are oriented consistently with respect to the polyline.

The second requirement implies that charts are separated from each other by *interior* triangles, i.e. triangles of the convex hull with no edges on the polyline (shown in black in Figure 7(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 7(c)). If the subloops lie on *their* convex hulls, their left and right envelopes will provide triangulations, which together with the removed chart will interpolate the original polyline (Figure 7(d)). 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 7(d) and (e), 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. 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 3, is *smoothness*: requiring the dihedral angles between adjacent triangles to be low. Even
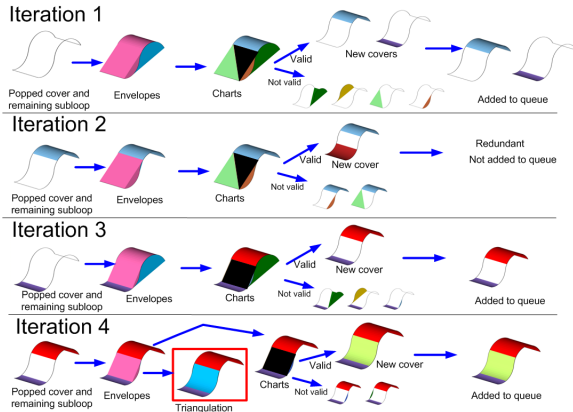
Figure 8: *Algorithm stages on a simple example (interior triangles shown in black). The framed triangulation is the output. The cover pushed into the queue in iteration four will be discarded at iteration five in stage 1 (it is not better than the best triangulation).*

with these two restrictions, there may exist multiple boundary triangulations providing a valid solution (see Figure 7 (e),(f)), raising the question which of these the user expects to obtain when specifying a particular boundary. Clearly, when designing a modeling tool, *predictability* is a desirable property. Human perception studies indicate that "simplicity is a principle that guides our perception..." [Ben96]. This principle is well known in Gestalt theory and is commonly used in sketch interpretation [KH06]. In our setup, 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 [Fre02] and Wang and Tang [WT05] 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 5.1.1).

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.

5. Branch-and-Bound Search Algorithm

We now extend the basic methodology described in Section 3 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 polyline boundaries that lie on their convex hull, the two envelopes mentioned above are not necessarily the best solutions with respect to smoothness (see the pink and blue envelopes at iteration one in Figure 8). 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 slightly 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 desired.

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 8, 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 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 8, 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 this definition of charts, our algorithm computes boundary triangulations that are unions of charts and envelopes. The algorithm uses a variation of the branch-and-bound approach [CLR90], which helps drive the search towards a good solution while avoiding the exploration of non-promising ones. The method uses a priority queue of sets of charts, or *covers*, that interpolate segments of the polyline (Figure 8). The queue is initialized with the empty cover. The priority function of the queue is based on a potential metric (Section 5.1.2) 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 5.1.1). 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 8 and the attached video.

1. **Pop Cover:** The algorithm pops a cover C from the pri-

ority 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 3) 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 triangles defines a triangulation of the original boundary. Triangulations having interior dihedral angles above the smoothness threshold are discarded. In Figure 8, 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 5.1.1). 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 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, N is discarded. In Figure 8, 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 5.1.1. Otherwise, the algorithm goes back to Stage 1.

The pseudocode for the algorithm is summarized in Figure 9. As shown, the entire main loop consists of approximately twenty lines of code.

Darts[†] **and Multiple Boundaries:** Our method is the first to our knowledge to seamlessly handle darts as well as multiple boundary loops. The processing of darts is straightfor-

---

[†] *Darts*, or duplicate edges, are frequently used in design setups such as garment making, to introduce points or lines of non-zero curvature onto the surface (see Figure 11).

```
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 9: *Pseudocode of main loop.*

ward and requires only minor data-structure modifications. 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.

5.1. Metrics
5.1.1. Triangulation and Cover Quality

Triangulation Quality: When evaluating triangulation quality, we consider two of the criteria discussed in Section 4: 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 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.

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 it, 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[‡]. 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 a least-squares 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. 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[§]. Thus for each such subloop we add to the fairness 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).

5.1.2. 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 in descending order based on the largest consecutive chart area.
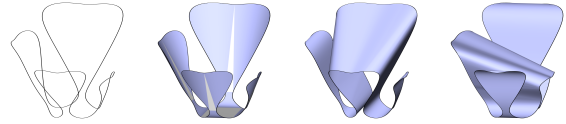
Figure 10: *Three of the alternative triangulations for the gazebo boundary in Figure 13 found by our method; the input boundary is shown on the left.*

6. Interface
6.1. Sketching

We use a fairly standard 3D curve sketching interface to create the polyline boundaries. The user can create the 3D boundary curves by first sketching them in one plane and then deforming them from a different viewpoint. Additionally, similar to [DJW*06] our system infers the depth information from a single sketch when the polyline is drawn over an existing model (Figure 1 (a),(b)). 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, 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 the same boundary again, the latter is split into two loops, enabling easy generation of a boundary network.

To further influence the result the user can also sketch a few of the rulings they expect to see on the final surface. These rulings are treated as triangulation edges which are constrained to be part of the final surface. For the purse model in Figure 12 we used this option to specify a ruling to the right of the handle, causing the purse to bulge outwards instead of curving inside. 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.

6.2. Overriding Optimal Selection

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 13 we had a particular orientation in mind. 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 was se-

---

[‡] 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.
[§] We arrive at $n-1$ as the number of interior edges in the triangulation $n-3$ plus the two edges adjacent to the charts.
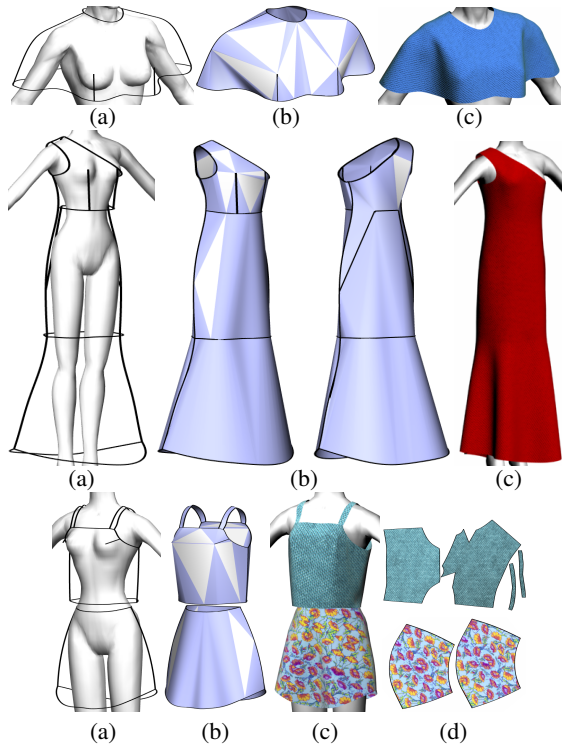
(a)   (b)   (c)

(a)   (b)   (c)

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

Figure 11: *Modeling garments: poncho (modeled from two developable panels, front and back), dress (modeled from seven panels), tanktop (four panels) and skirt (two panels). (a) Input boundary networks. (b) Modeled surfaces (coloring shows the surface structure with torsal developable surfaces shown in blue and interior triangles, corresponding to planar transition regions, in white). (c) Textured garments worn by a mannequin (basic simulation [3DM] was applied to account for collisions and gravity). Last row (d) the planar patterns for tanktop and skirt.*

lected this way. Alternatives found by the method are shown in Figure 10.

7. Results and Implementation

Throughout the paper we demonstrate the application of our method on a variety of inputs coming from different application areas where developables are used. Figure 11 shows 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 [May] 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 [DJW*06, May, 3DM]. In the examples in this paper, we fo-
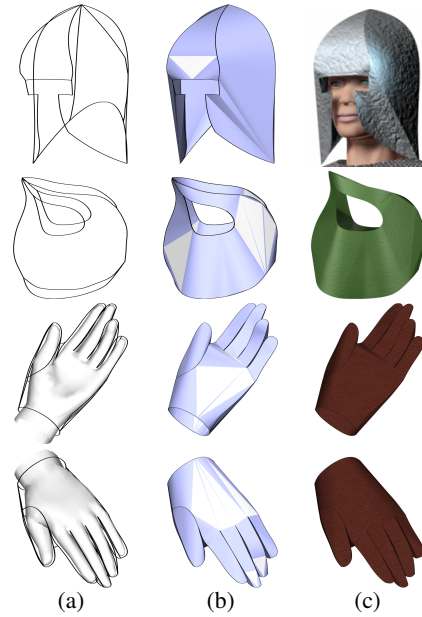

(a)   (b)   (c)

Figure 12: *Metal and leather: helmet (six panels), purse (three panels), and glove (ten panels).*

cused on obtaining the rest shapes. We then used a standard simulation tool [3DM] to visualise the garment behaviour 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 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 seams. As shown in Figure 11(b) in most cases the created surfaces are general developable surfaces, each containing several torsal developable surfaces connected by planar regions. Several examples, including the dress and tanktop, contain darts as part of the input polylines, robustly handled by our method. Since the created surfaces are analytically developable, the patterns (e.g. Figure 11(d)) can be used as-is to create reliable real-life replicas of the garments and the garment texture exhibits no distortion.

Figure 12 shows a variety of objects designed from flat sheet materials: a helmet, a leather bag and a glove. Despite the complexity of the modeled surfaces (12 (b)) no modeling expertise is required when sketching them using free-form drawing. The examples also show the control mechanism available for the user, such as the use of rulings to guide the construction of the purse as explained in Section 6 and the impact of smoothness threshold in the helmet example, where we relaxed the threshold to create the appearance of metal ridges.

Figure 13 shows examples of architectural structures generated using our method. The gazebo is an example of a complex general surface which cannot be projected to a plane without intersection and hence could not be generated by
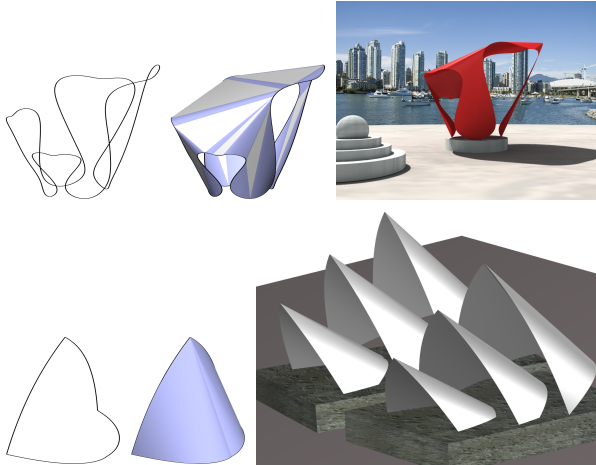
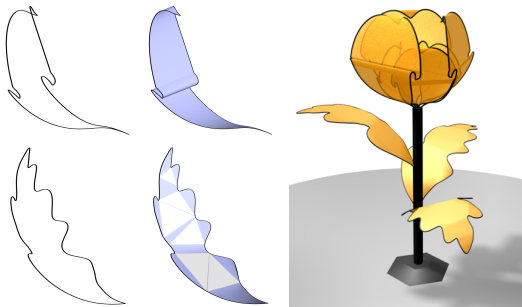**Figure 13:** *Architectural examples: gazebo and opera house.*

Figure 14: *Tulip paper lamp with developable paper petals and gold-foil leaves.*

any previous method for modeling developables. The Opera House model (Figure 13) was inspired by the Sydney Opera House and created by duplicating a single developable surface six times at different scales.

The tulip lamp modeled from developable petals and leaves (Figure 14) is mimicking Art-Nouveau paper lamps. The flower is created by duplicating and scaling a developable petal surface. While the gold-foil leaves are general developable surfaces the paper petals are torsal ruled surfaces and thus could be modeled by previous techniques, e.g. [WT05]. However, in contrast to these approaches in our setup 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.

These examples demonstrate the robustness of our modeling method as well as the vast array of application of developable surfaces and the varied shapes they can have.

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 estima-

tions, the method typically performs only a small number of iterations (less than a thousand for all the models shown in this paper). 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 is fairly small, varying from a few seconds for simple models such as the Opera House (Figure 13), to a few minutes for a complex model like the shoe (Figure 1). We observe that the 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.

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 sometime also influence the resulting surface. To reduce the noise and simplify the obtained hull structure, the algorithm first re-samples and smooths the polylines by fitting a piecewise B-spline curve. The algorithm then computes the center of mass C of the polyline boundary and 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. 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.

8. Summary and Limitations

To the best of our knowledge, we have presented the first algorithm for modeling developable surfaces that interpolate arbitrary polygonal boundaries. As demonstrated by the numerous examples in the paper, the method robustly computes smooth, predictable and fair developable surfaces interpolating the input boundaries and can be used to model a vast array of developable shapes. Since the method explicitly explores the space of possible solutions, users can easily select alternative solutions and explicitly specify which of the surface properties they wish to optimize. The presented approach is fairly generic and can be easily extended to handle other shape metrics.

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 developable surfaces that are entirely contained by the convex hull of their boundary. Thus, our method would not locate them. Adding

one or two extra rulings would typically break such surface into parts that partially lie on the respective boundary hulls and are thus computable by the method.

Acknowledgements

References

[3DM]   3DMAX: http://www.autodesk.com/3dsmax-support

[AKA06]   AKGUN T., KOMAN A., AKLEMAN E.: Developable sculptures of Ilhan Koman. In *Proc. Bridges* (2006).

[Aum04]   AUMANN G.: Degree elevation and developable Bézier surfaces. *Comput. Aided Geom. Des. 21*, 7 (2004), 661–670.

[Ben96]   BENJAFIELD J. G.: *The developmental point of view. A history of psychology.* Needham Heights, MA: Simon and Schuster Company, 1996.

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

[CLL*99]   CHEN H.-Y., LEE I.-K., LEOPOLDSEDER S., POTTMANN H., RANDRUP T., WALLNER J.: On surface approximation using developable surfaces. *Graphical Models and Image Processing 61*, 2 (1999), 110–124.

[CLR90]   CORMEN T. H., LEISERSON C. E., RIVEST R. L.: *Introduction to Algorithms.* MIT Press, 1990.

[CS02]   CHU C. H., SEQUIN C.: Developable Bézier patches: properties and design. *Computer-Aided Design 34* (2002), 511–528.

[dC76]   DO CARMO M.: *Differential Geometry of Curves and Surfaces.* Prentice-Hall, 1976.

[DJW*06]   DECAUDIN P., JULIUS D., WITHER J., BOISSIEUX L., SHEFFER A., CANI M.-P.: Virtual garments: A fully geometric approach for clothing design. *Computer Graphics Forum (Proc. Eurographics) 25*, 3 (2006), 625–634.

[Fre02]   FREY W.: Boundary triangulations approximating developable surfaces that interpolate a closed space curve. *International Journal of Foundations of Computer science 13* (2002), 285–302.

[Fre04]   FREY W.: Modeling buckled developable surfaces by triangulation. *Computer-Aided Design 36(4)* (2004), 299–313.

[Hil66]   HILL A.: Constructivism – the European phenomenon. *Studio International 171* (1966), 140–147.

[IMT99]   IGARASHI T., MATSUOKA S., TANAKA H.: Teddy: a sketching interface for 3d freeform design. In *Proc. SIGGRAPH '99* (1999), pp. 409–416.

[JKS05]   JULIUS D., KRAEVOY V., SHEFFER A.: D-charts: Quasi-developable mesh segmentation. *Computer Graphics Forum (Proc. Eurographics) 24*, 3 (2005), 581–590.

[KH06]   KARPENKO O. A., HUGHES J.: Smoothsketch:3D free-form shapes from complex sketches. *ACM Transactions on Graphics 25*, 3 (2006), 589–598.

[Lay72]   LAY S. R.: *Convex Sets and their Applications.* Wiley, NY, 1972.

[LPW*06]   LIU Y., POTTMANN H., WALLNER J., YANG Y.-L., WANG W.: Geometric modeling with conical meshes and developable surfaces. *ACM Transactions on Graphics 25*, 3 (2006), 681–689.

[May]   MAYACLOTH: http://caad.arch.ethz.ch/info/maya/manual/MayaCloth.

[MS04]   MITANI J., SUZUKI H.: Making papercraft toys from meshes using strip-based approximate unfolding. *ACM Transactions on Graphics 23*, 3 (2004), 259–263.

[Pet04]   PETERNELL M.: Developable surface fitting to point clouds. In *Computer Aided Geometric Design* (2004), pp. 785–803.

[PW01]   POTTMANN H., WALLNER J.: *Computational Line Geometry.* Springer Verlag, 2001.

[Rhi]   RHINO3D: http://www.rhino3de.de/design/modeling/developable/.

[Sed86]   SEDYKH V.: Structure of the convex hull of a space curve. *J. of Soviet Mathematics 33*, 1 (1986), 1140–1153.

[SF98]   SINGH K., FIUME E.: Wires: A geometric deformation technique. *Computer Graphics 32*, Annual Conference Series (1998), 405–414.

[She02]   SHELDEN D. R.: *Digital surface representation and the constructibility of Gehry's architecture.* MIT, 2002.

[STL06]   SHATZ I., TAL A., LEIFMAN G.: Paper craft models from meshes. *Vis. Comput. 22*, 9 (2006), 825–834.

[WT04]   WANG C. C. L., TANG K.: Achieving developability of a polygonal surface by minimum deformation: a study of global and local optimization approaches. *The Visual Computer 20*, 8-9 (2004), 521–539.

[WT05]   WANG C. C. L., TANG K.: Optimal boundary triangulations of an interpolating ruled surface. *Journal of Computing and Information Science in Engineering, ASME Transactions 5*, 4 (2005), 291–301.

[YGZS05]   YAMAUCHI H., GUMHOLD S., ZAYER R., SEIDEL H.-P.: Mesh segmentation driven by gaussian curvature. *The Visual Computer 21*, 8-10 (September 2005), 649–658.