Surface Patches for 3D Sketching

Fatemeh Abbasinejad[*] University of California at Davis
Pushkar Joshi Motorola Mobility[†]
Cindy Grimm Oregon State University
Nina Amenta Lance Simons University of California at Davis

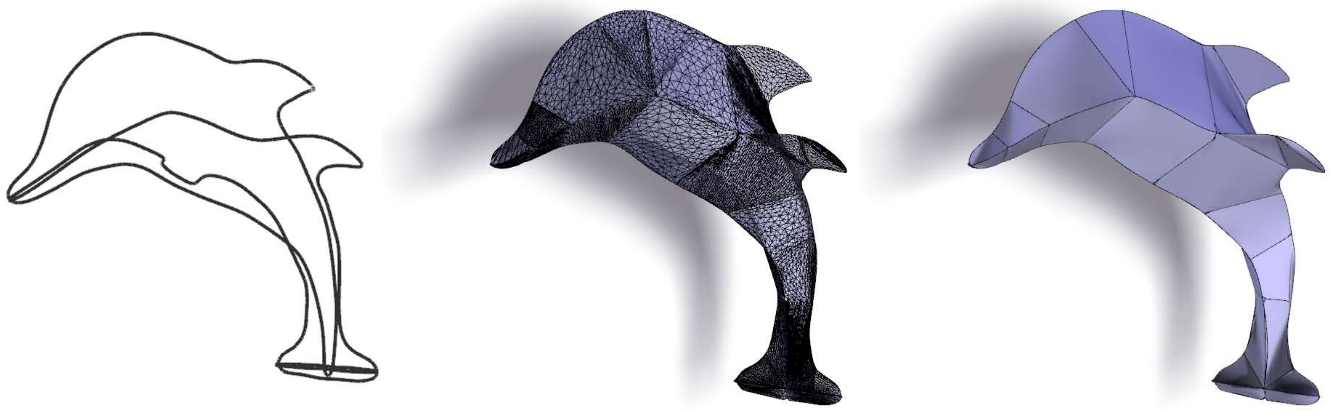
Figure 1: *The 3D sketch on the left implies a surface consisting of eight patches, four of which are complex and highly non-planar. Our system automatically divides these complex patches into smaller ones that are simpler and therefore easier to tessellate (middle). Displaying the simplified patches (right) during interactive sketching can both improve visualization and provide new opportunities for natural interaction, such as sketching on the resulting patches.*

Abstract

3D sketching is an appealing approach for creating concept shapes in the early stages of design. While curve networks alone can convey shape, surfacing the network can dramatically help with visualization and interaction. Unfortunately, surfacing a curve network is an inherently ambiguous problem, and even if the correct surface patches are identified, they can have an arbitrarily complex 3D geometry, making it challenging to produce a reasonable tessellation. In this paper we address the problem of creating light-weight surface tessellations on the fly. Our approach is to identify potential patches in the curve network, and then break complicated patches into simpler ones which can be tessellated using any simple algorithm. Our surfacing approach relies on the observation that breaking a complicated patch into a set of nearly planar ones with small total area seems to create a simple, natural-looking surfaces. We demonstrate our approach on curve networks generated by two different 3D sketching systems.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Modeling packages;

Keywords: Curve-based modeling, Sketching 3D curves, Surface patches

[*]e-mail:fabbasinejad@ucdavis.edu
[†]now at Google Inc.

1 Introduction

Three-dimensional modeling remains very difficult, despite years of research and development. Recently, tools for three-dimensional curve drawing [Wesche and Seidel 2001; Bae et al. 2008; Schmidt et al. 2009; Grimm and Joshi 2012] have introduced a new paradigm that leverages the artist's two-dimensional drawing skills to allow them to sketch networks of space curves in three dimensions. These interfaces seem to be particularly effective for ideation of concept shapes in the early phases of design. Enhancing the artist's creativity and flow is essential to this process, meaning that the drawing interface should be as un-cluttered as possible.

A natural goal is to automatically surface these three-dimensional sketched curve networks to create models. Creating these surfaces on the fly during the curve drawing process itself would help the designer in several ways as they build up a sketch. Displaying the shape and topology of the partially inferred surface would show how the system interprets the drawing so far, providing guidance on what needs to be filled in. By introducing occlusion and shading, the surface visually clarifies the three-dimensional structure of the curve network. And finally, providing surfaces on which to draw enables an interface with which the artist can sketch initial curves

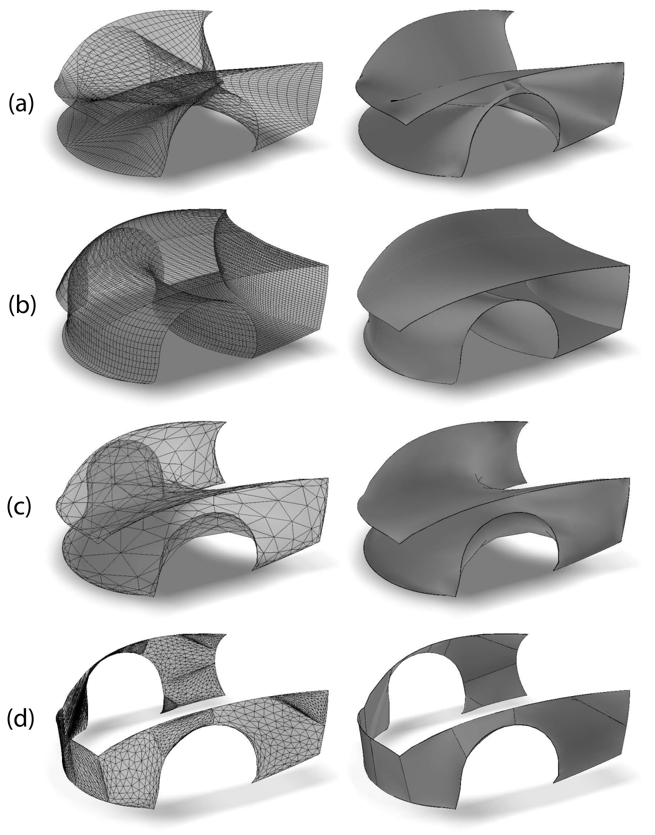

Figure 2: *Four methods to create a surface approximation for a single, complex patch boundary: (a) the lofting curve networks of [Schaefer et al. 2004] (b) the recent design-driven quadrangulation of [Bessmeltsev et al. 2012], which produces an approximation to a ruled surface but not the one of minimal area (c) linearized Laplacian smoothing [Abbasinejad et al. 2011], which approximates the minimal (general) surface, and (d) our method, which approximates the minimal ruled surface.*

with unambiguous depth, similar to the temporary planes and surfaces already used by sketching programs [Bae et al. 2008; Schmidt et al. 2009; Grimm and Joshi 2012]. Surfaces could even be used to create physical prototypes through 3D printing.

Unfortunately, inferring surfaces from sketches, especially partial ones, is difficult, since the shape implied by a network of unconstrained space-curves is often ambiguous, both topologically and geometrically. For example, it is not clear which of the cycles in a curve network are intended to bound surface patches. This problem was recently addressed by Abbasinejad et al. [2011]; we build on their approach, using their system to identify cycles bounding potential surface patches. Once a cycle has been identified there is still the problem of constructing a patch surface that has that boundary. Complicated patch boundaries in $\mathbb{R}^3$ can, unfortunately, have many possible surface interpretations, with very different geometry (see Figure 2). Given only a closed cycle and knowing nothing else about the designer's intent, our focus in this paper is to quickly produce as simple a surface as we can.

A curve cycle that is nearly planar and has a convex boundary is easy to tesselate (that is, create a surface mesh for), and there are many techniques for doing so. We focus here on the highly non-planar, non-convex, and geometrically complicated patch boundaries that are difficult to tessellate directly with any current technique (see Section 2).

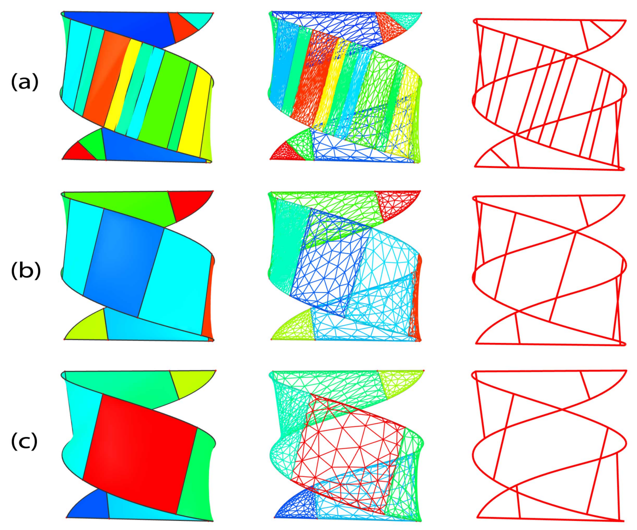

Figure 3: *A twisted ribbon segmented using different values of the parameter α, which balances the cost of adding a new chord with the improvement in the flatness of the resulting sub-cycles; as α increases, the number of chords selected decreases. From top to bottom, $\alpha = 0$, $\alpha = 0.2$ and $\alpha = 0.4$.*

Our contribution is that we provide a light-weight approximation of an appropriate surface patch which can be computed quickly and used during interactive sketching. This output is not necessarily a final, high-quality surface, although it could certainly serve as an input or a hint for a higher-quality surface generation method. Using curve networks produced using two different sketching systems, ILoveSketch [Bae et al. 2008] and JustDrawIt [Grimm and Joshi 2012], we demonstrate that we produce simple, intuitively reasonable surfaces, even given quite complex boundary cycles.

Rather than numerically optimize a mesh, which might be prohibitively slow, our approach is to divide the complex patches into simpler ones, each of which can be easily tessellated. Of course, there are many ways to do this. We show that optimizing for a small number of nearly planar sub-patches, with small total area, produces a reasonable surface approximation even for very complicated patches. This method approximates computing a minimal ruled surface. Our method has one main parameter that can be tuned.

2 Related Work

Many systems that provide sketching interfaces for designing surfaces in $\mathbb{R}^3$ have been developed, including [Zeleznik et al. 1996; Igarashi et al. 1999; Nealen et al. 2007; Rivers et al. 2010; Schmidt et al. 2006; SketchUp 2012]. In these systems the user explicitly creates and interacts with surfaces using sketching and other controls. We are interested in supporting systems, eg. [Wesche and Seidel 2001; Bae et al. 2008; Schmidt et al. 2009; Grimm and Joshi 2012], that allow the user to sketch curves in three dimensions, without explicitly creating surfaces. These interactive systems produce networks of space curves, rather than surface or solid models.

These systems raise the question of how to tessellate a non-planar patch boundary. Advancing front propagation methods construct the patch surface by inward propagation of rays or flow lines from

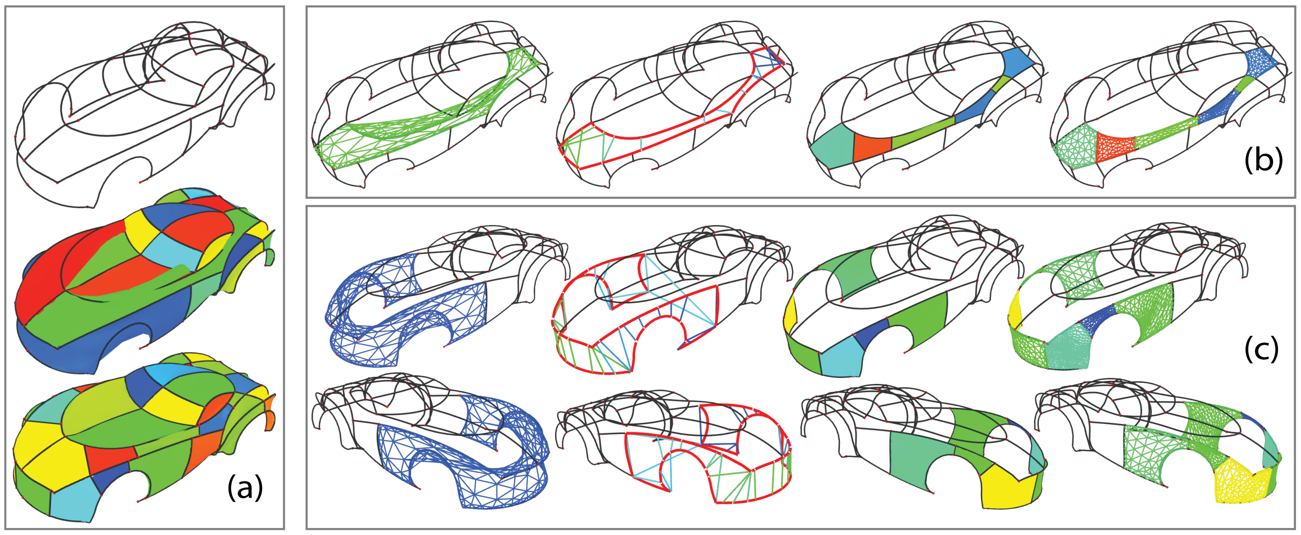

Figure 4: *Simplifying complex surface patches: The input curve network of the "roadster" model from the collection of ILoveSketch [Bae et al. 2008] models is shown in (a), and the steps of our algorithm on two of the complex patches in (b) and (c). Directly tessellating surface patches bounded by complex cycles results in surfaces that are clearly inconsistent with the designer's intentions (b and c, left). The first step in the decomposition of the complex cycles is to select a set of candidate chords (second column, b and c). From this set we find an optimal subset of chords with respect to a heuristically chosen function, producing nearly-planar parts (third column). These are easily tessellated (fourth column).*

the boundary. The design-driven quadrangulation of [Bessmeltsev et al. 2012] is a state-of-the-art method, inspired by the same problem we address. It uses flow lines between matched pairs of patch boundary segments. The combinatorial problem of finding an appropriate matching is difficult for complex and especially irregular cycle boundaries, but in situations where the designer's intention is clearly represented by paired boundary curves, the resulting surfaces are excellent. Another successful approach is the lofting curves method [Schaefer et al. 2004]. It uses subdivision to construct smooth surfaces by tessellating n-sided patches. In Figure 2, we show the results of both [Bessmeltsev et al. 2012] and [Schaefer et al. 2004], as well as the simple linearized Laplacian smoothing included with [Abbasinejad et al. 2011], on a difficult example patch from one of the ILoveSketch [Bae et al. 2008] models. Given the patch boundary, all the previous methods produce unnecessarily complex and unsatisfactory surfaces.

Our approach is similar to that of Rose et al. [2007], who used developable surfaces. They use optimization to select a developable surface that includes many triangles of the convex hull of the input cycle. We use the somewhat broader class of ruled surfaces (a bilinear surface, for example, is ruled but not developable), within which we optimize for minimal area. Minimizing area over the class of ruled surfaces gives a different result than minimizing area over all surfaces (eg. via Laplacian smoothing). A minimal-area triangulation is the solution to the discrete version of this problem, and it can be computed by dynamic programming, as shown by Barequet and Sharir [1995], but this $O(n^3)$ time algorithm is not usable in interactive situations. Instead, we build a much rougher heuristic approximation to the minimal-area ruled surface by computing a decomposition of a complex cycle boundary into a small number of nearly-planar patches. Although in practice we examine many fewer sub-problems than would be required for a complete triangulation, we can no longer guarantee that the number of sub-problems is polynomial, so we do not get a polynomial bound on the running time. Of course, the fact that our algorithm is potentially exponential does not imply that an efficient polynomial-time approximation to the minimal-area ruled surface does not exist.

There are a variety of other non-optimization approaches as well. The linearized Laplacian method used in [Abbasinejad et al. 2011] projects the cycle to the plane, meshes the interior, and then relaxes the vertex positions in $\mathbb{R}^3$. Jun [2005] projects the cycle to the plane, and uses any crossings to decompose the cycle into simple pieces. Brunton et al. [2009] use an energy minimization to unfold the cycle into a simple closed curve in the plane.

Filling a hole in a mesh, which might have a complex boundary, is a similar problem. There the mesh provides additional information, including normals. Scattered data interpolation approaches [Tekumalla and Cohen 2004; Brazil et al. 2010; Branch et al. 2006; Podolak and Rusinkiewicz 2005] require normals or existing surfaces. The volumetric approaches [Brazil et al. 2010; Podolak and Rusinkiewicz 2005] can fill complex holes, but assume that the desired result is a watertight mesh, which is not always true in our case.

3 Algorithm

Our goal is to decompose a complex cycle into a small number of nearly-planar sub-cycles, which are easy to tessellate with one of the simple algorithms. Since the cycle is presented as a polygonal approximation, we could always decompose it into perfectly planar parts, approximating a ruled surface, by triangulating all of its vertices. But this would result in an excessive number of long, thin triangles; instead we search for a small number of nearly-planar sub-cycles (each of which could be triangulated in many ways, all producing the nearly the same area). There are an exponential number of ways to triangulate a piecewise-linear cycle, and even more ways to divide a cycle into sub-cycles. We make a heuristic choice among the possible decompositions based on the principle that the resulting surface should have small total area.

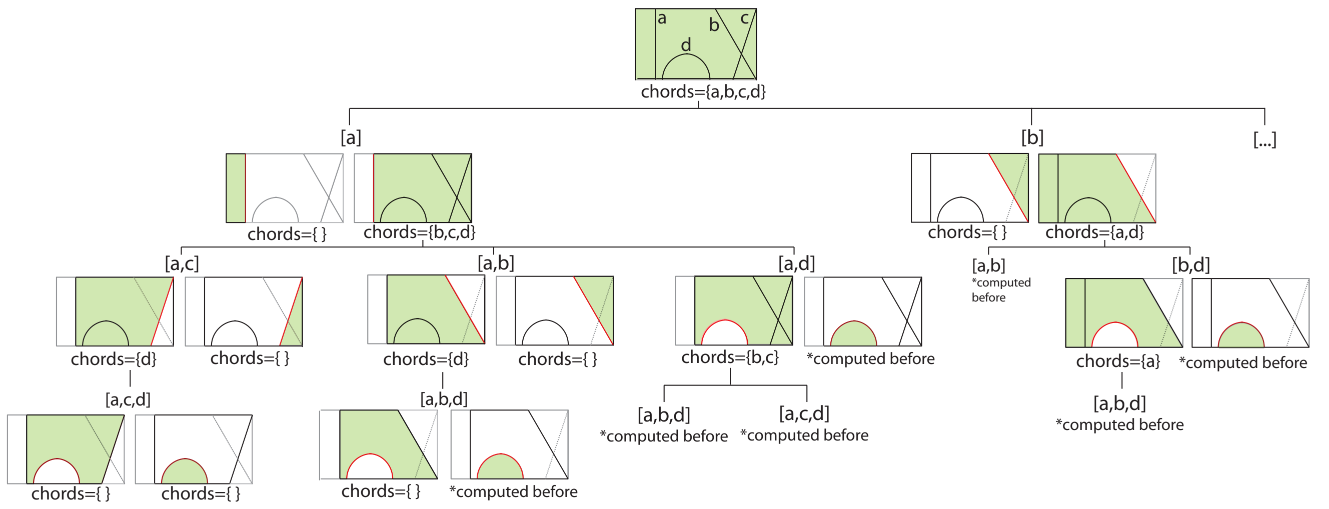

Figure 5: *Illustration of a hypothetical example: For each input sub-cycle K (green box), we sub-divide into a left and right sub-problem by selecting one of its chords c_i (in red). A chord is passed into a sub-problem if and only if both of its endpoints lie in the sub-cycle; otherwise we say the chord is a crossing chord and is not included in either subproblem. For example, the diagonal chord c (upper right, sub-dividing on b) is not included in either sub-problem. Many sub-cycles show up as part of other sub-cycles; these are computed only once and the score stored. Note that the recursive algorithm does a depth first search traversal on the sub-problems.*

We decompose complex cycles by introducing *chords*, that is, line segments connecting two vertices on the cycle. There are two main steps to choosing the chords. First, we use heuristics to select a small set of candidate chords, cutting the problem down to a reasonable size. Second, we use an exhaustive search, with memoization, to select a subset of the candidate chords which gives a decomposition minimizing our heuristic function (see below). The algorithm is illustrated in Figure 4. We now describe these two steps in detail.

3.1 Selecting candidate chords

Straight segments on the boundary of the input cycle can always be assigned to flat sub-cycles, but segments with significant curvature may need to be broken up. Our heuristic selects chords that connect sharp corners or sample points from regions of significant curvature, where the chords are roughly perpendicular to the patch boundaries.

We begin the process of selecting candidate chords by choosing potential chord endpoints. We estimate curvature in the polygonal approximation of the cycle by computing the angle between adjacent segments. Any angle greater than $20°$is considered a sharp corner and is always included as a potential segment endpoint. In smoother regions which still have significant overall curvature, sample points are identified by walking along the edges of the cycle. Local maxima of curvature are selected if possible, and otherwise samples are placed so that the total curvature between any two samples is bounded.

Next, for each potential endpoint e, we search for two types of potential partner points p:

- closest other point on the cycle, in three-dimensional Euclidian distance
- and nearest sharp corner

such that the chord connecting them will be nearly perpendicular to the cycle at e. By "nearly perpendicular", we mean that we require the angle between the chord and the normal plane to the curve at e to be at most a threshold value m. If the partner point p is a sharp corner we use $m = 40°$, while if p is closest in Euclidean distance, we only allow $m = 30°$. These threshold values are fixed for all of our examples.

For the closest value in Euclidean distance, if there is already another potential endpoint which meets the criteria we use it, and if not we create a new endpoint for the new candidate chord. If there are no corner points meeting the criteria, we do not create the new candidate chord. If a corner point and the closest point are close together, we use the corner point and ignore the closest point. This ensures at least one candidate chord per endpoint.

3.2 Patch decomposition

From the set of candidate chords, we then want to pick a subset that decomposes the cycle into a small number of reasonably flat sub-cycles, each with small area. We do this by optimizing a heuristic cost function over subsets of chords. The cost function is intended to be fast to evaluate.

We describe this step of the algorithm in a top-down fashion. The initial problem is the input cycle K and its set of chords $chords[]$. We consider all possible methods of dividing the problem into two sub-problems, left and right, with a single chord c_i, and of these possibilities, we choose the optimal one according to a cost function defined below.

The division cost $S(K, c_i)$ of K by c_i is the cost of its two sub-problems, computed recursively, plus a cost associated with c_i.

$$S(K, c_i) = \text{Sub}(K_{\text{left}}) + \text{Sub}(K_{\text{right}}) + \alpha\,\text{Cost}(c_i) \qquad (1)$$

The scalar parameter α is the only one associated with this part of the algorithm and the only one we have adjusted for different inputs. When α is zero, a maximal set of chords is selected; the larger α is, the fewer sub-cycles we have in the final decomposition. This effect is illustrated in Figure 3. We used $\alpha = 0.1$ for all of the models except the Figure 4, where $\alpha = 0.6$. The choice of α has no effect on the running time, since it does not limit the number of subproblems considered by the algorithm.

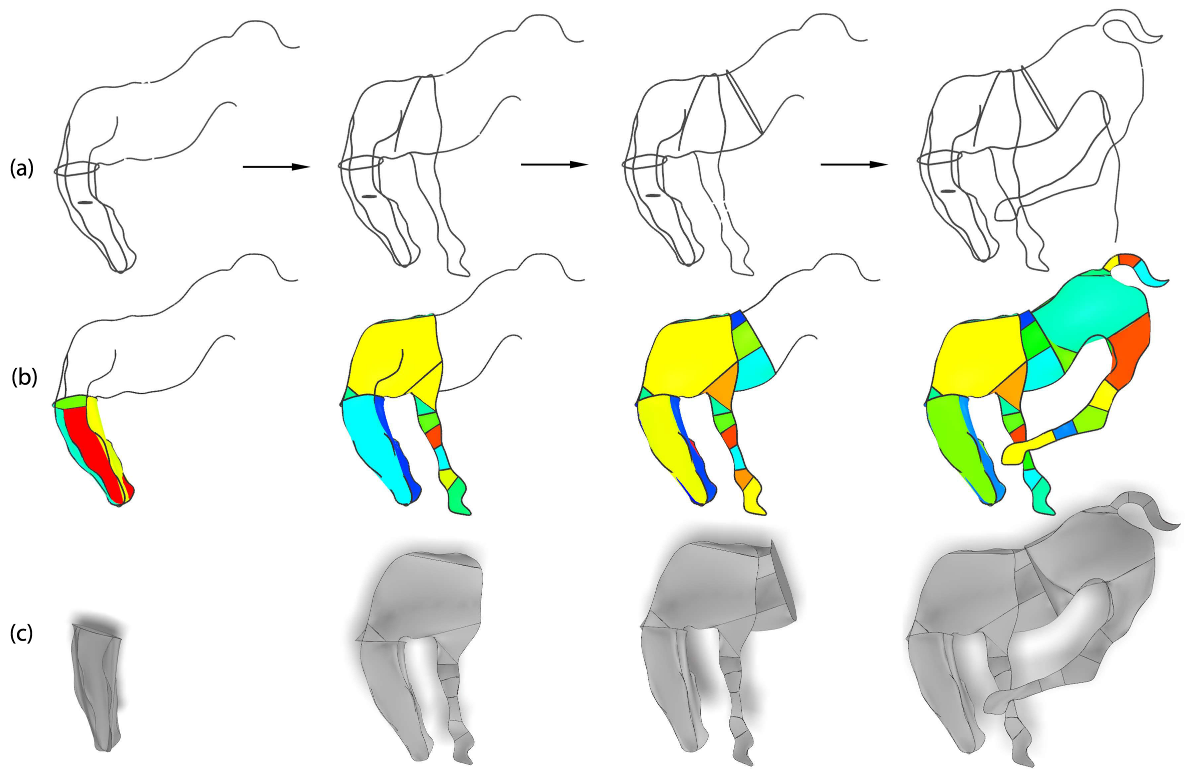

Figure 6: *An example of how our method might be used in a sketching system: (a) from left to right, the artist adds more strokes to the sketch (b) complex patches are decomposed using our method (c) the artist can choose a rendered view of the surfaces, in which the partial shape implied by the sketch is comprehensible.*

To compute $Sub(K)$, we compare the costs of the divisions induced by trying each of the possible chords c_i, and also the cost $S(K, \emptyset)$ of not cutting the problem with any chords at all, and choose the lowest-cost option:

$$\text{Sub}(K) = \min(\ \{S(K, c_i)\ |\forall c_i\} + S(K, \emptyset)\ ) \tag{2}$$

For a sub-problem with no chords the cost is given by $S(K, \emptyset)$.

When dividing K by c_i, another chord c_j is considered to be part of a sub-problem if and only if both of its endpoints lie in the same sub-cycle; otherwise we say c_j crosses c_i, and c_j is not included in either subproblem.

It remains to define the cost functions S and Cost. $S(K, \emptyset)$ is intended to estimate of the flatness of cycle K. We use the volume of the rectangular box implied by the Principle Components of the set of vertices of the polygonal representation of K (the product of the three eigenvalues of the covariance matrix). While this is clearly not a perfect measure of flatness, it is easy and fast to compute, and it gives good results on the examples we have.

Assigning a cost for each chord is necessary to keep down the number of sub-cycles in the output, and to bias the results towards a set of cycles with small area. We associate a cost with each chord which is linear in its length. This cost is inspired by our goal of approximating a minimal-area ruled surface. A ruled surface is the limit of a triangulation of the cycle boundary as the density of sampling increases, and its area is linear in the lengths of the chords. Choosing shorter chords to dissect the cycle is intended to encourages the implied chords filling in the almost-planar patches to be shorter as well, for example in Figure 2.

We normalize Cost(c_i) by the total length len(K) of the sub-cycle being decomposed.

$$\text{Cost}(c_i) = \text{len}(c_i)/\text{len}(K) \tag{3}$$

The algorithm as described so far is inefficient, even for an exponential search, since many of the top-down problems will lead to identical sub-problems. This is illustrated by the example in Figure 5. Inspired by the dynamic programming algorithm for optimal polygon triangulation [Barequet and Sharir 1995], we improve efficiency by *memoization*: storing the score $S(K)$ for each sub-problem in a hash table, so that no score is ever computed twice. The hash key is the sorted list of the unique identifiers of all of the chords on the boundary of cycle K. Thus, the total number of possible sub-cycles determines the running time. Unfortunately this is still potentially exponential in n, the number of candidate chords. To see this, note that the boundary of an output sub-cycle might include an arbitrary number of chords, where each pair of chords is separated by arcs of the original cycle. Any subset of chords could be dropped from the boundary to create a new sub-cycle, leading to an exponential problem size. However, n tends to be small since we choose a small number of candidate chords, and the structures of the problems are usually well-behaved (most regions are bounded by a small number of chords). Thus, nearly all of the patches can be subdivided in under a second (see Table 1).

4 Results

Our patch decomposition scheme produces intuitively reasonable results on complicated free-form patches, greatly improving the perception of the sketched model. In addition to the results presented here, we refer the user to the short accompanying video.

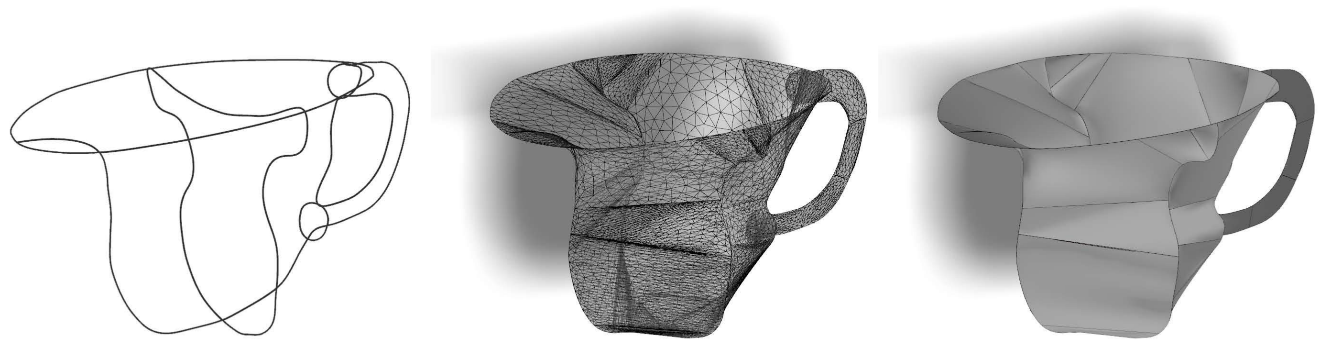

Figure 7: *Creating a teacup from two largely planar silhouette curves and three cross-sections. Note that the handle surface is flat because there is no cross-section defined for it yet.*

Algorithm 1 $Sub(K)$: Finding the cost of sub-cycle K

```
INPUT: set of chords $C[]$
OUTPUT: cost of sub-cycle $Sub(K)$ and best cutting chord $c_{\text{best}}$ (if any)
$K \leftarrow$ input set of chords $C[]$ creates a sub-cycle
do a look up in the hash table to see if you have computed the cost of $K$ already
if (in the hash table) then
    extract information from the hash
    $c_{\text{best}} \leftarrow$ best cutting chord if any
    $sub \leftarrow$ cost of sub-cycle $K$
    return $[sub, c_{\text{best}}]$
else
    $sub \leftarrow$ compute the cost of the sub-problem by not cutting with any chord
    $chords[] \leftarrow$ extract all chords within this sub-cycle $K$
    if ($chords[].empty()$) then
        insert computed cost $sub$ into the hash table
        return $[sub, \emptyset]$
    end if
    for all ($c_i \in chords[]$) do
        $[K_{\text{left}}, K_{\text{right}}] \leftarrow$ divide $K$ by $c_i$
        $S(K, c_i) = Sub(K_{\text{left}}) + Sub(K_{\text{right}}) + \alpha\ Cost(c_i)$
        $[S_{\text{best}}, c_{\text{best}}] \leftarrow$ maintain $\min(S(K, c_i), sub)$
    end for
    insert $S_{\text{best}}$ and $c_{\text{best}}$ as the computed cost of Sub(K) into the hash table
    return $[S_{\text{best}}, c_{\text{best}}]$
end if
```

We use the output of two different 3D sketching systems, ILoveSketch [Bae et al. 2008] and JustDrawIt [Grimm and Joshi 2012], as test data. ILoveSketch is oriented towards conceptual product design and allows the artist to produce symmetrical sketches with mostly smooth curves. Nonetheless, artists frequently produce sketches which contain highly non-planar and complex cycles. There is a large published database of complete curve networks produced by ILoveSketch. JustDrawIt allows more free-form sketching, and produces less regular curves. For some of the JustDrawIt examples the curves were provided in the order that they were drawn by the user, allowing us to illustrate the results of our method on partial 3D sketches.

In all of our examples, we began by running the patch-finding algorithm of Abbasinejad et al. [2011] to select cycles bounding potential surface patches from the curve network. In some cases we used the interaction provided by their system to improve their automatic choices of which cycles to surface. For example, in the dolphin model, we requested the fourth and final patch.

Figure 1 shows our results at an intermediate phase of drawing the dolphin model with JustDrawIt. With only the four main curves (top and side silhouettes, and the tail cross-section), we can produce a surface which clarifies the essential shape of the model. In the JustDrawIt paper, several more curves (and surface normals along those curves), were needed to successfully construct a surface using the approach in [Brazil et al. 2010].

We illustrate the two steps of the algorithm in Figure 4 using two of the more difficult patches from the ILoveSketch "roadster" model, and in Figure 2 compare our decomposition method on one of those patches to three direct tessellation algorithms: the lofting curve networks of Schaefer et al. [Schaefer et al. 2004], the linearized Laplacian minimization used in the Abbasinejad et al. [Abbasinejad et al. 2011] system, and the recent design-driven quadrangulation method of Bessmeltsev et al. [Bessmeltsev et al. 2012]. In this case, our approach clearly does the best at reflecting the user's intention of outlining the bumper of a car. Both the lofting curves and the linearized Laplacian methods found an approximation to a minimal surface, which is not an appropriate solution here. The design-driven quadrangulation produced a nearly-flat solution with a larger surface area than the one selected by our method.

In Figure 6 we use the horse sketch from the JustDrawIt paper to illustrate how our method works on partial sketches. This sequence is intended to give a sense of how the light-weight surfaces produced by our decomposition algorithm would work within a sketching interface, giving the user a clear sense of what the implied model is at any stage of the process. When the user sketches in large parts of the model with a single curve (eg. the shoulder and leg patch, and then the back of the horse including a leg and the tail), complex non-planar cycles are created. These large patches are not correctly tessellated by any of the direct methods. At each of the four different stages of the sketch shown, the only input from the user is the network of 3D curves (Figure 6, top). Patches are automatically found independently at each stage, and then all of the patches are then decomposed using our algorithm; some of them, such as the head patches, require no additional chords. The final output patches, all nearly planar, are tessellated using the linearized Laplacian method. Note that, although the desired final geometry of the horse may be a closed, inflated surface, many of the intermediate stages are not.

The teacup in Figure 7 is also a partially-completed sketch drawn using JustDrawIt. This partial sketch illustrates how several planar curves can be combined to produce highly non-linear patches. The

Patch on Model	Chords		Number of	Time
	Candidates	Chosen	sub-problems	(secs)
Dolphin(bottom right)	17	10	467	0.19
Dolphin(bottom left)	23	10	1343	0.5
Dolphin(top left)	17	12	930	0.83
Dolphin(top right)	28	12	1484	0.47
Roadster(front)	32	7	1108	0.14
Roadster(side)	16	5	251	0.02
Horse(shoulder)	18	9	699	0.14
Horse(middle)	8	4	74	0.003
Horse(back)	30	15	12319	6.5
Teacup(front left)	10	6	158	0.01
Teacup(front right)	8	6	119	0.01
Teacup(back right)	13	7	440	0.07
Teacup(back left)	15	6	679	0.12
Teacup(handle)	8	5	83	0.005

Table 1: *Timings and statistics for the patch decompositions shown in this paper.* $\alpha = 0.1$ *except for the roadster front, where* $\alpha = 0.6$.

topology of the body of the cup is accurately captured (although presumably the user would want smooth geometry for the final shape). Note that the handle of the cup, which does not yet have a sufficient number of cross-section curves to define it as a tube, nevertheless results in a plausible surface.

Running times, number of candidate chords, and final number of chords appear in Table 1. The running times are adequate for interactive sketching applications, where each patch has to be decomposed exactly once at the end of a curve edit cycle, not at every frame. The longest running time was for the complex patch at the back of the horse, which involved a large set of candidate chords with a complicated structure. All examples were run on a 2.4GHz Intel Quad Core processor.

5 Conclusions and Limitations

Our algorithm is heuristic, and provides no guarantees on running time or avoidance of self-intersections. There may be other heuristics that would work within the framework of this algorithm to produce better patches or chords, but in practice these two simple heuristics (near-planar surfaces and chord-length minimization) appear to work well.

The chords and patches we currently produce use only the patch boundary positions as input; as such, they tend to be faceted and are not smooth across the boundaries. Including surface normal information across boundaries and using a minimization approach to produce curved chords would give more visually-pleasing surfaces, at increased computational cost.

Acknowledgements

We are deeply grateful to our colleagues Scott Schaefer, Alla Sheffer and Caoyu Wang for providing their code. We gratefully acknowledge the support of NSF grants IIS-0964357 and IIS-1117663, and a gift from Adobe Research.

References

ABBASINEJAD, F., JOSHI, P., AND AMENTA, N. 2011. Surface patches from unorganized space curves. *Comput. Graph. Forum 30*, 5, 1379–1387.

BAE, S.-H., BALAKRISHNAN, R., AND SINGH, K. 2008. Ilovesketch: as-natural-as-possible sketching system for creating 3d curve models. In *Proceedings of the 21st annual ACM Symposium on User Interface Software and Technology*, ACM, New York, NY, USA, UIST '08, 151–160.

BAREQUET, G., AND SHARIR, M. 1995. Filling gaps in the boundary of a polyhedron. *Computer Aided Geometric Design 12*, 2, 207–229.

BESSMELTSEV, M., WANG, C., SHEFFER, A., AND SINGH, K. 2012. Design-driven quadrangulation of closed 3d curves. *ACM Trans. Graph. 31*, 6 (Nov.), 178:1–178:11.

BRANCH, J., PRIETO, F., AND BOULANGER, P. 2006. Automatic hole-filling of triangular meshes using local radial basis function. In *3D Data Processing, Visualization, and Transmission, Third International Symposium on*, IEEE, 727–734.

BRAZIL, E., MACEDO, I., SOUSA, M., DE FIGUEIREDO, L., AND VELHO, L. 2010. Sketching variational hermite-rbf implicits. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling*, The Eurographics Association, 1–8.

BRUNTON, A., WUHRER, S., SHU, C., BOSE, P., AND DEMAINE, E. 2009. Filling holes in triangular meshes by curve unfolding. In *Proceedings. International Conference on Shape Modeling and Applications (SMI 2009),*.

GRIMM, C., AND JOSHI, P. 2012. Just drawit: a 3d sketching system. In *Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SBIM '12, 121–130.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: a sketching interface for 3d freeform design. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 409–416.

JUN, Y. 2005. A piecewise hole filling algorithm in reverse engineering. *Computer-Aided Design 37*, 2, 263–270.

NEALEN, A., IGARASHI, T., SORKINE, O., AND ALEXA, M. 2007. Fibermesh: designing freeform surfaces with 3d curves. In *ACM Transactions on Graphics (TOG)*, vol. 26, ACM, 41.

PODOLAK, J., AND RUSINKIEWICZ, S. 2005. Atomic volumes for mesh completion. In *Proceedings eurographics symposium on geometry processing*, Citeseer, 33–41.

RIVERS, A., DURAND, F., AND IGARASHI, T. 2010. 3D modeling with silhouettes. *ACM Transactions on Graphics 29*, 4.

ROSE, K., SHEFFER, A., WITHER, J., CANI, M.-P., AND THIBERT, B. 2007. Developable surfaces from arbitrary sketched boundaries. In *Proceedings of the Eurographics Symposium Geometry Processing*, 163172.

SCHAEFER, S., WARREN, J., AND ZORIN, D. 2004. Lofting curve networks using subdivision surfaces. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, ACM, New York, NY, USA, SGP '04, 103–114.

SCHMIDT, R., WYVILL, B., SOUSA, M., AND JORGE, J. 2006. Shapeshop: Sketch-based solid modeling with blobtrees. In *ACM SIGGRAPH 2006 Courses*, ACM, 14.

SCHMIDT, R., KHAN, A., SINGH, K., AND KURTENBACH, G. 2009. Analytic drawing of 3D scaffolds. In *Proceedings of ACM SIGGRAPH Asia 2009*, ACM, New York, NY, USA, SIGGRAPH Asia '09, 149:1–149:10.

SKETCHUP, 2012. Trimble SketchUp. `http://www.sketchup.com/`.

TEKUMALLA, L., AND COHEN, E. 2004. A hole-filling algorithm for triangular meshes. Tech. Rep. UUCS-04-019, University of Utah.

WESCHE, G., AND SEIDEL, H.-P. 2001. Freedrawer: a free-form sketching system on the responsive workbench. In *Proceedings of the ACM Symposium on Virtual reality software and technology*, ACM, New York, NY, USA, VRST '01, 167–174.

ZELEZNIK, R., HERNDON, K., AND HUGHES, J. 1996. Sketch: an interface for sketching 3d scenes. In *Proceedings of SIGGRAPH*, ACM, 163–170.