Eurographics Symposium on Geometry Processing 2011
Mario Botsch and Scott Schaefer
(Guest Editors)

Volume 30 (2011), Number 5

Surface Patches from Unorganized Space Curves

Fatemeh Abbasinejad[1], Pushkar Joshi[2], Nina Amenta[1]

[1]University of California, Davis
[2]Adobe Systems Inc.

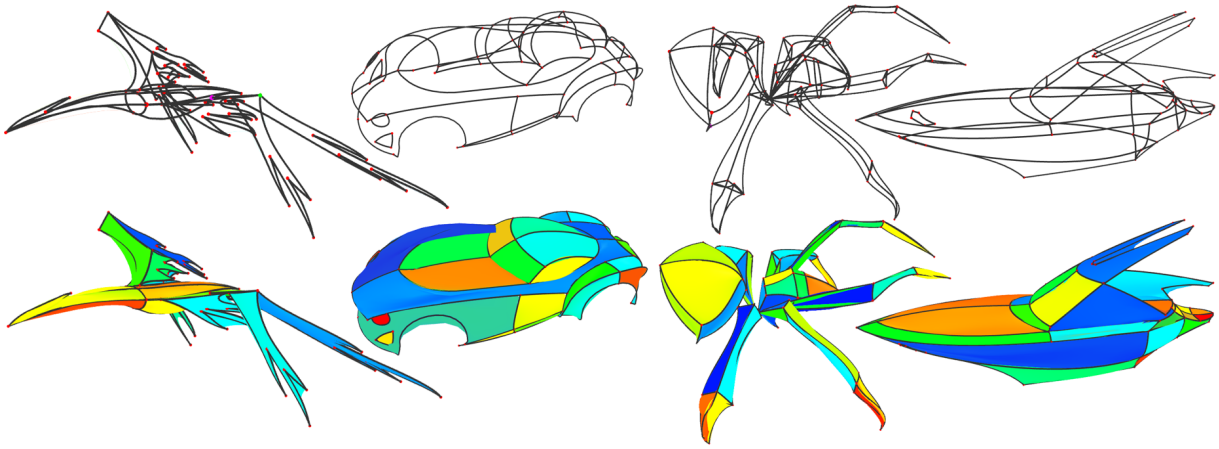


Figure 1: *Our system takes as input a wireframe model produced from a 3D curve sketch, and outputs a set of curved surface patches representing the object. The output objects need not be manifold, or connected. From left to right, phoenix, roadster, spider and boat curve networks created with ILoveSketch on top, and the output sets of surface patches on the bottom.*

Abstract

Recent 3D sketch tools produce networks of three-space curves that suggest the contours of shapes. The shapes may be non-manifold, closed three-dimensional, open two-dimensional, or mixed. We describe a system that automatically generates intuitively appealing piecewise-smooth surfaces from such a curve network, and an intelligent user interface for modifying the automatically chosen surface patches. Both the automatic and the semi-automatic parts of the system use a linear algebra representation of the set of surface patches to track the topology. On complicated inputs from ILoveSketch [BBS08], our system allows the user to build the desired surface with just a few mouse-clicks.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation, I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations

1. Introduction

Recent tools for 3D curve drawing [WS01,BBS08,SKSK09] have simplified the process of sketching 3D shapes, as a method of high-level shape design. These user interfaces are intended to allow ideas to be expressed visually, without impeding the user's creativity. For instance, many of them mimic drawing on paper. 3D sketching interfaces are particularly effective for shape ideation (i.e. the generation of concept shapes), that can be used in the early phases of collaborative design.

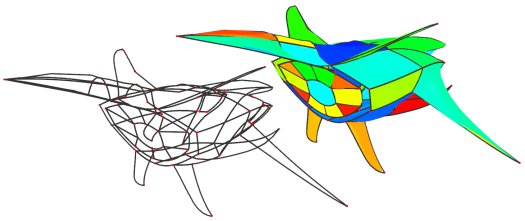


Figure 2: *An example of an ILoveSketch input (spacecraft 30) for which the automatic phase of our system finds an acceptable surface and no user interaction is required.*

The outputs of these design tools are networks of curves in $\mathbb{R}^3$. Such a curve network can usually be interpreted as a wireframe model of a piecewise-smooth surface; by "wireframe model" we mean a representation of a surface by its edges alone. Constructing the 3D surface from the wireframe model is commonly referred to as "surfacing", "lofting" or sometimes "skinning." Surfacing is an important part of the design process; it produces more realistic and less confusing visualizations of the concept shape. While surfaced sketches are generally not suitable as production specifications or final designs, they can serve as a starting point or inspiration for a modeler to construct a precise CAD model.

Surfacing also clarifies the designer's intentions. Many curve sketches are ambiguous (the boat in Figure 1 is a good example), meaning that the same curve network could be the wireframe of several different surface models. The surfaces represented by artist's 3D sketches are often non-manifold; artists can, and often do, allow three or more surface patches to meet along a curve. The desired outputs may include both two- and three-dimensional regions. This lack of topological structure greatly increases the number of mathematically possible interpretations of an input curve network.

Contribution

This paper presents a method that automates a significant portion of the surfacing process, greatly reducing the required user interaction. We focus on finding patches to form reasonable non-manifold surfaces, which has not been studied much in the past, but is necessary for 3D sketch input.

Our system begins with an automatic phase in which the system produces its best guess for the desired set of surface patches. This is followed by a user-interaction phase, in which the user can add and delete surface patches. Both phases make use of the topological structure of the current set of patches. We demonstrate the system on networks of curves produced by ILoveSketch [BBS08], although it could take input from other 3D sketching programs as well.

Overview

We take as input a wireframe model represented by a graph with edge curves, and output set of patch boundaries as a set of cycles in that graph. Topologically, a set of patches may or may not close off some number of three-dimensional solids, forming a model which may have both two- and three-dimensional parts. Our approach is to select an initial maximal set of patches which forms a two-dimensional surface, including every curve, but closing off no solids. We then present the user with an interface that allows her to delete and replace surface patches, and/or add additional patches which close off solids.

The automatically-created two-dimensional, possibly non-manifold but solid-free surface forms a *cycle basis* for the graph. A graph has many possible cycle bases, so we use a heuristic weighting function on cycles to define an optimal one. Our system is based on the insight that it is easy to compute an optimal cycle basis with respect to an arbitrary weighting function on cycles. This rather simple observation does not appear to have been applied before in this domain. We use a weighting function which prefers cycles with a small number of edges, which are close to being flat, and which do not separate the graph. This heuristic seems to work well on the ILoveSketch [BBS08] input, but *any* heuristic could be plugged into our framework.

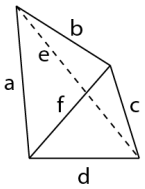
In the user-interaction phase, we help the user refine the cycle basis: when she deletes a patch, we suggest other low-weighted patches that could replace it in the basis. We also suggest patches that might close off solids, using a different heuristic that scores not only cycles but also the solids that they create. Suggesting topologically acceptable and heuristically optimal new patches gives a simple yes/no user interface, which is faster and more pleasant than having the user choose the edges of a cycle with several point-and-clicks.

We begin in Section 2 by describing the relationship of our method to previous work on patch finding. In Section 3 we describe cycle bases in more detail, and we explain the algorithm and the weighting function in Section 4. User interaction is described in Section 5. We end with some results (Section 6) and discussion (Section 8).

2. Related Work

Most prior work in patch finding has focused on two-dimensional drawings of wireframe models of manifold surfaces, sometimes with holes or non-planar faces. This problem is significantly different from ours; the input is strictly two-dimensional, rather than three-dimensional, which introduces more ambiguities, and the stronger topological constraint on the output makes the space of solutions smaller and more structured. This problem is quite difficult, and mature systems like Liu, Lee and Cham [LLC02] and Varley [VC10] include elaborate sets of heuristics and sophisticated search strategies. A few of the manifold patch finding algorithms [MW80, AW92] take three-dimensional wireframe models as input.

For non-manifold surfaces from two-dimensional draw-



f1 = abf
f2 = aed
f3 = cdf
f4 = bec

	f1	f2	f3	f4
a	1	1	0	0
b	1	0	0	1
c	0	0	1	1
d	0	1	1	0
e	0	1	0	1
f	1	0	1	0

Figure 3: *The matrix representing the four faces of a tetrahedron. We can verify that the fourth column of the matrix is the sum of the first three, using arithmetic modulo two. We also observe that geometrically the cycle bounding the fourth face is the sum of the first three, and that it closes off a solid (the interior of the tetrahedron). Any choice of three of these four columns forms a cycle basis (one of many possible cycle bases for this graph). The entire vector space of cycles defined by the wireframe model would consist of all possible sums of these four cycles; for example, the cycle a,b,c,d is the sum of a,b,f and f,c,d.*

ings, Shpitalni and Lipson [SL96] proposed a heuristic search algorithm, using the constraint that the common boundary between two faces should always be smooth. Liu and Lee [LL01] sped up their algorithm by finding minimal cycles more efficiently and by formulating a better search algorithm, reporting performance similar to ours. Sun and Lee [SL04] give some interesting analysis of when a vertex, edge or face *must* be non-manifold. Sketch inputs are less constrained than the simple CAD models considered in these papers; for some (eg. the phoenix in Figure 1), the desired surface has patches adjacent along sharp edges, and in general the inputs are more complex, curved, and more mixed-dimensional.

If the input is a three-connected planar graph, it is possible to find the correct face structure by embedding on the sphere [Han82, DB83, SL96]. Inoue et al. [ISC03] consider two-connected planar graphs, which can be embedded onto the sphere in multiple ways; they produce many solutions and then score them. Many non-manifold models, such as the roadster in Figure 1, are 2-connected planar graphs. Inoeue's algorithm would reconstruct them as solids, but possibly the algorithm could be adapted to produce non-manifold surfaces.

Cycle basis approaches, like ours, have in the past been applied to genus-zero manifolds. The algorithm of Ganter and Uicker [GU83] starts with an arbitrary cycle basis, and pivots from one cycle basis to better one by adding cycles together as described in Section 3. Brewer and Courter [BC86] also followed this approach, optimizing the basis so that the number of edges shared by cycles in the basis is minimized. We tried a pivoting approach based on their ideas, but found that it often failed to produce a desirable cycle basis. The most relevant prior work is probably by Bagali et al. [BW95], who construct a cycle basis using a greedy algorithm. Cycles are added in order of smallest total edge length, and the topological assumption that the input is a solid of genus zero is used to reject inappropriate choices. There is also theoretical work [GH02, LR05, dP95, KMMP04, BGdV04] on greedily computing cycle bases with minimal total edge length. These papers provide polynomial-time algorithms by guaranteeing that not too many cycles are examined. Along with Varley [VC10], we observe that weighting cycles by total edge length does not seem to lead to good results in patch finding. Our main innovation is the observation that weighting *cycles*, rather than edges, allows us to encode a much more flexible variety of heuristics into the set of weights, and that any choice of cycle weights can be easily optimized via the greedy algorithm.

3. Linear Algebra Framework

Any set of two-dimensional patches forms what is called a two-dimensional cell complex. If it separates three-space into two or more connected components, we treat all but the exterior as three-dimensional solids, producing a three-dimensional cell complex (possibly containing two-dimensional as well as three-dimensional parts). Our approach to selecting an initial set of patches is to construct the largest two-dimensional complex we can before closing off any solid three-dimensional regions.

Such a set of patches is called a *cycle basis* for the set of cycles in the input graph. It is called a basis because the set of cycles - that is, the set of possible patches - forms a vector space. This vector space representation (described below) is the same as that used in topology when computing homology using the Smith Normal Form and in persistent homology computation [EH08]. It is helpful to think of this structure as an extension to two dimensions of the combinatorics of spanning trees: a spanning tree in a graph is a maximal set of edges forming no cycles, whereas here we are looking for a maximal set of cycles forming no solids.

A cycle in a graph is a set of edges such that an even number of edges meet at each vertex. Notice that under this definition, cycles may have multiple connected components, or vertices of degree greater than two. "Adding" two cycles consists of taking their symmetric difference: shared edges are deleted, and the remaining edges in either cycle are combined into the new cycle. A *cycle basis* C is a set of cycles such that every other cycle in the graph can be constructed as a sum of some subset of the cycles in C. See the example in Figure 3.

Computationally, this is implemented using matrix arithmetic over the integers mod two, also known as the Galois Field of order two, $GF(2)$. The entire vector space could be represented by a single large matrix M with a row for each edge of the input graph, and a column for each possible cycle. There is a 1 in matrix location $m_{i,j}$ if edge i belongs to cycle j, and 0 otherwise. Adding two cycles c_j, c_k cor-

responds to adding the corresponding columns in $GF(2)$, so that $1+1=0$; notice that this produces the symmetric difference, which is another cycle c_l (i.e. $c_l = c_j + c_k$). The three cycles c_j, c_k and c_l are dependent, since any one is the sum of the other two. The rank r of M, over $GF(2)$, is the size of the largest *independent* set of columns. As usual, we can select many possible bases for the vector space M, each of which is a set of columns of size r. For a connected graph, $r = m - n + 1$, where m is the number of edges in the graph and n is the number of vertices, but there is no requirement that our inputs be connected, and many are not.

4. Automatic Patch Finding

In this section we describe the automatic phase of our patch identification system. The input to this phase is an undirected graph, whose edges are smooth curves. The curves meet at a finite number of vertices, so that two curves may not overlap over a non-negligible length. There may be, and often are, multiple curves connecting the same pair of vertices. There is no requirement that the output should be a manifold.

4.1. Algorithm

Just as there are many spanning trees in a graph, there are many cycle bases; in fact, there are possibly an exponential number of both cycles and cycle bases. Our objective is to select an optimum cycle basis, where the optimum is defined as the minimum using a heuristic weighting criterion (see Section 4.2).

To produce the optimal basis, we use the standard *greedy matroid algorithm*, which can be applied to any vector space, with any weighting function on the elements. We write $W(c)$ to denote the weight of cycle c (or just W instead of $W(c)$ when clear). The weight of a cycle basis $\{c_1, \ldots, c_r\}$ is the sum of its cycle weights.

Theorem 1 [Law76] For any weighting function W, the greedy matroid algorithm produces a minimum-weight basis.

Applied to minimum spanning tree computation, the greedy matroid algorithm is Kruskal's algorithm. Applied to cycle basis computation, the greedy matroid algorithm proceeds as follows. We begin with an empty set of cycles C, and at each step we greedily attempt to add the cycle c of minimum weight $W(c)$. If c is dependent on some subset of the cycles in C, we discard it. If, on the other hand, c is independent, we add it to C and continue.

The current set of cycles C is represented as a set of column vectors as described in the previous section, and to see if a new column vector is independent of the current set, we append it to the current matrix, and compute the rank. If the rank is increased by adding the new column, the new cycle is independent of the previous set of cycles. We use the FFLAS-FFPACK package [JGDP04] for linear algebra over finite fields to compute the rank of the $0-1$ matrix.

4.2. Weighting function

Choosing the heuristic weighting function W requires tradeoffs between simplicity, efficiency and the quality of the results. Our choice combines three heuristics into a single lexicographic function W.

Our first heuristic is that we prefer short cycles over long ones. The second heuristic is borrowed from Bagali et al. [BW95]: non-separating cycles are preferred over separating cycles. A separating cycle is one for which removal of the cycle vertices and all of their adjacent edges (including the cycle edges) increases the number of connected components in the graph.

To combine these two heuristics, we define for each cycle the quantity

$$k = \text{number of edges} + (2+\varepsilon)s$$

where s is zero if the cycle is non-separating and one if it is separating. Ordering the cycles by k means that we try first non-separating cycles of length $2, 3$ and then 4, followed by separating cycles of length 2, non-separating cycles of length 5, separating cycles of length 3, and so on. See Algorithm 1 for a bit more detail on how we generate and examine cycles in this order.

Cycles with the same value of k are ordered geometrically, by the volumes of their axis-aligned bounding boxes, from smallest to largest. Since the curves are given as ordered sets of sample points, the computation of bounding boxes is trivial. This choice favors nearly flat faces which are well-aligned with the coordinate planes. It works surprisingly well on the ILoveSketch [BBS08] data, perhaps because the wireframe models are drawn aligned with the coordinate system. More sophisticated geometric heuristics, for instance, the volume of the approximating ellipsoid found by PCA, would of course be possible. But the limitations that we see in the current system (see Section 8) do not seem to be related to this design choice.

Combined with the fact that we consider the cycles in order of edge length, this means that the cycle basis we find optimizes the following lexicographic objective function W: it finds a basis with minimum total summed value of k, and of the many bases with that total sum of k, it finds one that minimizes the sum of the volumes of the bounding boxes.

5. Topology-aware User Interface

Our user interface allows the user to both add patches to close solids, and to delete patches, replacing them with alternative choices. The naive user interface for deletion is fine: point at the patch, and click to delete it. But to add a patch, either to close a solid or to replace a deleted patch, the naive user interface is to indicate a sequence of edges, requiring several delicate point-and-clicks, each taking a second or so. This rapidly becomes tedious, which is the reason that automatic patch finding is desirable at all. Since some user

Algorithm 1 Finding an optimal cycle basis

{INPUT: Graph $[V,E]$}
{OUTPUT: Cycle basis represented by matrix C }
$C = []$ {empty matrix}
for all $i \in \{2,\ldots,n\}$ **do**
  for all $v \in V$ **do**
    Find all cycles of length i containing v, by breadth-first search.
    Store separating cycles in set S_i, and non-separating cycles in set N_i, removing duplicates.
  end for
  Rank the cycles in N_i by bounding box volume.
  for all cycles $c \in N_i$, in rank order **do**
    Add vector representation of c to C.
    Compute rank of new matrix C.
    If rank did not increase, remove c and discard.
  end for
  if $i \geq 4$ **then**
    Rank the cycles in S_{i-2} by bounding box volume.
    for all cycles $c \in S_{i-2}$, in rank order **do**
      Add vector representation of c to C.
      Compute rank of new matrix C.
      If rank did not increase, remove c and discard.
    end for
  end if
end for

interaction cannot be completely avoided, we use geometric/topological analysis to provide a better, *semi-automated* user interface. In our interface, we show the user patches that she might want to add, each of which can be accepted or rejected with a single click, in a small fraction of a second. The user can always fall back to the naive interface, but we did this for only two of the 34 models we have built.

Deleting a patch leaves an incomplete cycle basis. As a replacement patch, we suggest the next-best patch again using the heuristic weighting function of Section 4.2.

Adding a patch to a set of cycles of full rank closes off a new solid. Our system suggest closing patches using a different set of heuristics, which consider the solids as well as the cycles. The first heuristic is again to prefer adding cycles with few edges. Second, we prefer a patch c which includes many edges of degree one, and makes them degree two in the closed model; a simple example would be the final square patch closing off a cube. Third, we use the heuristic that the area of the patch c closing off the solid should be small relative to the area of the solid itself. This favors larger solids, while not allowing large solids to be created by adding large patches. We use the area of the largest side of the axis-aligned bounding box of a patch as an estimate of its area, since it is not easy to compute the area of a non-planar patch. Again, we found that using the bounding box works surprisingly well.

To detect the new solid potentially formed by the addition of cycle c, we again use the linear algebra mechanics of Section 3. Let C be the matrix of rank r whose column represent the current cycle basis (irrespective of whether the model currently includes some solids already). We remove each cycle c_i in turn from C, and then consider the rank of $C - c_i + c$. If the rank is r, we know that c_i would complete a cycle containing c, and if the rank is $r-1$, we know that the cycle containing c is included in the set $C - c_i$.

We combine the three heuristics in the following formula, where cycles producing a large value of $W_s(c)$ are preferred:

$$W_s(c) = 10 * (a + b) + k$$

where a is the ratio *total area estimate of solid s / area estimate of cycle c* (i.e. the third heuristic), b is the number of edges in c that have degree one in the current model (i.e. the second heuristic), and k is the number of edges in cycle c.

6. Results

As mentioned before, we demonstrate our method by finding patches in sample 3D wireframe models from ILoveSketch [BBS08].

6.1. Pre-processing

The ILoveSketch [BBS08] output consists of unoriented piecewise-linear curves (although the user enters curves with a specific orientation, we do not assume that this is meaningful), represented as sequences of points. Intersections are not indicated, and in fact for most of the sketches the curves fail to intersect each other at all. To produce a graph, we assume that two curves intersect when they pass within a user-defined threshold, and place the intersection at the closest point. The threshold varies because different models are more or less carefully drawn. When an intersection is found the curves are automatically snapped together, with the displacement interpolated across the edge to avoid introducing sharp bends. We introduced some intersections which were not found automatically, using Maya, so that we could get all of the visually more interesting models. Dangling edges (with a vertex of degree one) were removed. We assume that the input curves are intended to be smooth, and we interpret sharp corners within a curve as vertices of degree two. Without too much trouble, we turned 49 ILoveSketch models into reasonable input graphs.

In a complete sketch-based modeling system where a surfacing algorithm such as ours would be used, the curve drawing interface should probably include snapping of intersection points, which would obviate much of this preprocessing.

6.2. Results

Of the 49 input graphs we could use, we successfully produced sets of surface patches from 34 of them. We defined a

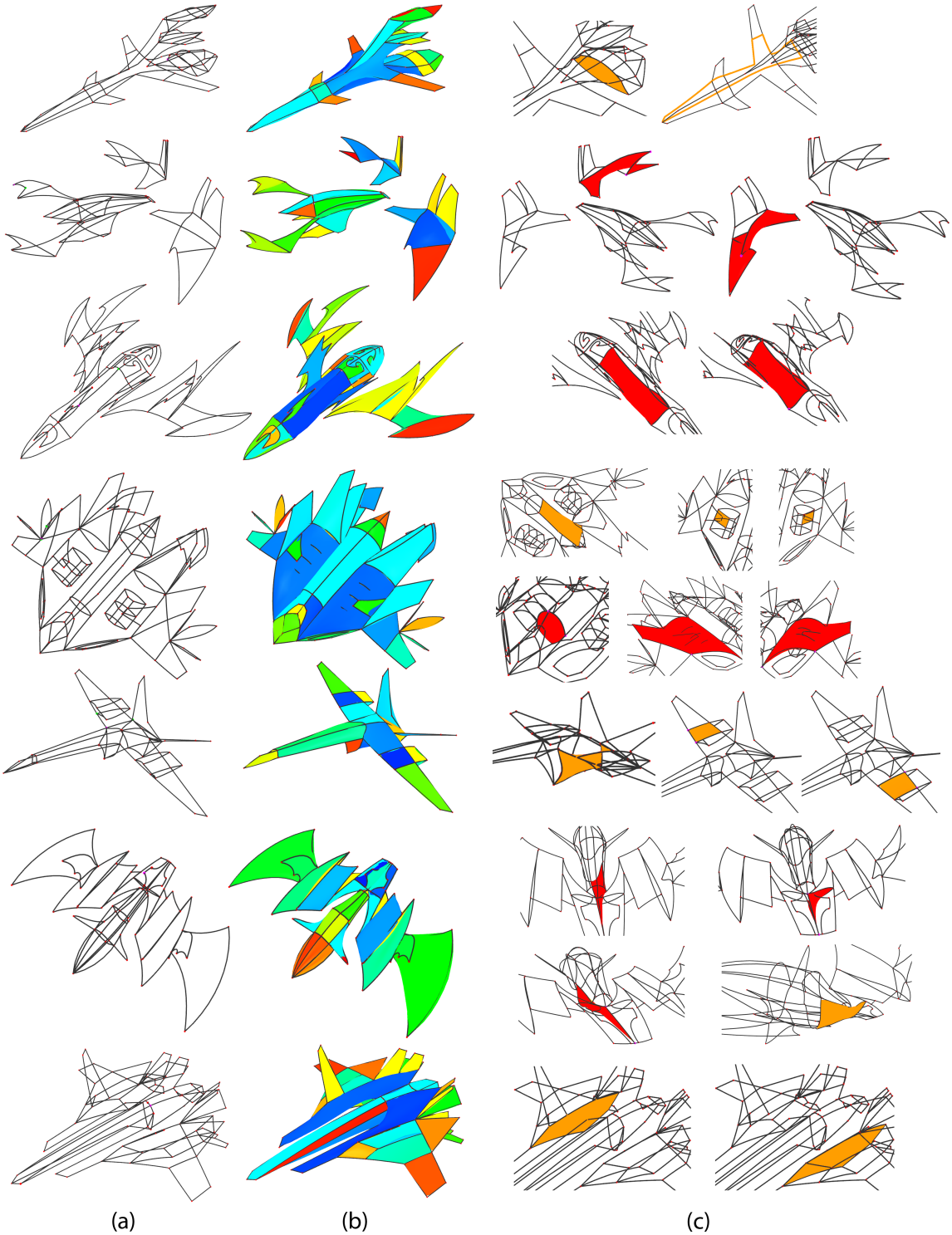


Figure 4: *Some examples of patches added and deleted in user interaction. (a) Input wireframe models provided from ILoveSketch [BBS08]. Models in order from top are spacecrafts 84, 9, 63, 12, 20, 81 and 31 (b) Our algorithm successfully finds almost all the expected patches on the models. (c) Only a few patches are needed to close solids (orange) or were not an intended patch (red). We have shown the patches from different angles of the model for better visualization (only three of the (5/4) deleted patches are show for spacecraft (12/81); others are mirror images). Examples of user interaction are included in the accompanying video. Statistics on more models can be seen in Table 1.*

Model	#Curves	Total Time (sec.)	Patch Time (sec.)	# Patches Auto.	# Patches Manual
Jetfighter	65	5	1	23	2 add 1 del.
Spacecraft19	65	4	1	26	2 add 2 del.
Spacecraft26	66	14	1	24	2 add
Spacecraft56	74	4	2	20	1 del.
Speaker	76	23	2	27	1 add
Spacecraft20	77	13	3	34	3 add
Phoenix	80	3	~0	17	1 add
Spacecraft37	81	6	1	23	1 add 2 del.
Spacecraft9	86	3	1	37	2 del.
Roadster	86	18	1	31	-
Spacecraft75	88	9	1	22	1 del.
Spacecraft13	89	17	4	30	2 add
Spacecraft59	90	10	3	32	4 del.
Spacecraft36	95	18	5	34	1 add
Spacecraft38	95	23	6	36	2 del.
Boat	94	36	9	37	1 add 6 del.
Spacecraft57	107	14	6	37	-
Spacecraft84	108	33	9	42	2 add
Spacecraft33	109	20	8	38	1 add 2 del.
Spacecraft44	110	47	19	48	1 add 2 del.
Spacecraft77	111	12	4	31	-
Spacecraft81	111	28	12	43	1 add 4 del.
Spacecraft63	117	7	2	34	2 del.
Spacecraft85	126	167	16	54	3 add 2 del.
Spacecraft92	127	13	6	50	2 add 8 del.
Spacecraft30	132	69	30	50	-
Spacecraft49	139	9	4	39	5 add 8 del.
Spider	139	22	15	71	8 add
Spacecraft87	139	34	20	52	6 add 6 del.
Spacecraft90	140	30	15	45	2 add 3 del.
Spacecraft31	142	43	16	44	2 add
Spacecraft10	146	14	6	40	-
Spacecraft88	162	124	85	71	3 add 4 del.
Spacecraft12	196	66	51	70	3 add 5 del.

Table 1: *This table shows processing time and the amount of interaction required to produce the final models in the 34/49 cases in which our method could be used easily without fallback to the naive user interface. Total time includes generating the minimum-area surfaces for the patches. Deletes are delete and replace operations.*

case to be successful if we get almost all of the patches automatically, the resulting surface is what we expected, and we could make any necessary edits and close off any solids with a few clicks, without resorting to the naive interface. Figures 1,2 and 4 show some of the original wireframe models and the resulting patch sets. Table 1 shows the timing and the amount of manual interaction required to construct all of the 34 models. All examples were run on a 2.4GHz Intel Quad Core processor.

The 15 graphs for which we failed to produce a good model within a few mouse clicks illustrate a number of limitations of our algorithm, as well as the difficulty of the problem. These are discussed in Section 8.

6.3. User Feedback

We asked a professional artist and 3D modeler to informally validate all of our ILoveSketch examples (Table 1), in order check that our subjective decisions about which were the "right" patches were reasonable. We showed him screenshots of the wireframe models followed by those of our surface patch models, and asked him to evaluate the results based on the following two questions: "Are there any missing or unexpected patches?" and "do the surface patches help you understand the 3D shape better"? Overall, he felt strongly that the surfacing was appropriate and improved his understanding of the 3D shape. He did not see any missing or unexpected patches in any of the images. He reported that five examples did not improve his understanding of the intended shape (the curve sketches were already sufficiently simple or clear), while in eight examples the patches significantly improved his understanding of the intended shape (the wireframes were very confusing), and in most cases the patches gave a somewhat clearer presentation of the intended shape; there were no sketches for which the patches obscured or worsened his understanding of the intended shape.

We also went through the interactive process on three fairly complicated wireframe models (phoenix, and spacecraft 63 and 81). The user reported that in all cases, there were no unexpected patches produced by the automatic generation phase (although in our results we had chosen to delete and replace some patches on the two spacecraft; this might suggest that our criteria for success was a little too strict). In all three cases he was able to quickly choose the missing surface patch that would close off an appropriate volume. He also emphasized that in 3D as well as 2D the surface patches helped him perceive the 3D shape *faster* than the wireframe model, even though perceiving shape from wireframe models is easier in 3D than it is from 2D images.

The main drawback of our system according to the artist was the quality of the surfaces filling the patches (Section 7), and improving this surfacing is indeed one of our main goals in future work.

7. Creating Surfaces from Patch Boundaries

The main contribution of our paper is the identification of the cycles forming the surface patches. But these patches can have highly complex and non-planar boundaries, so that rendering them for visualization is also a non-trivial problem. For the sake of completeness, in this section we describe the process we used for generating patch surfaces.

We map the patch boundary points onto a circle, using the arc-length along the patch boundary as the parametric distance along the circle. The circle is then triangulated, adding interior points, using Triangle [She96]. The points on the boundary are mapped back to their input positions, and the positions of the new points on the interior of the

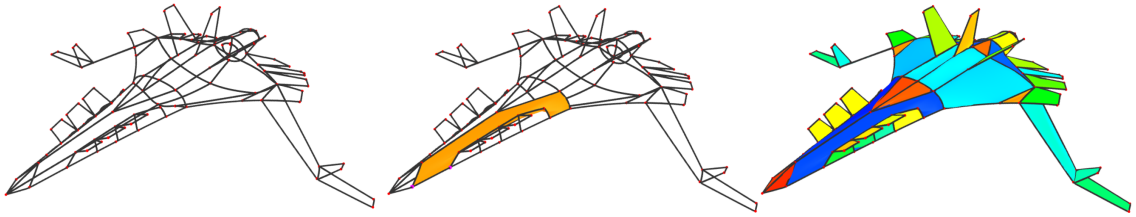


Figure 5: *An example of a model (spacecraft 79) in which we had to resort to the naive user interface (and hence not included among the success cases in Table 1). Here we wanted the patch shown in orange. The system stubbornly wanted to make a big patch with the little yellow and green squares (windows?) pasted up against it. Deleting that undesired patch led only to more complicated versions of the same thing, eg. going around one of the windows with the others pasted against it. Detecting and interpreting patches that lie on top of one another is one of the limitations of our system.*

patch are computed by solving a linearized Laplacian problem over the triangulated 2D domain. Essentially, we obtain a linear approximation of a soap film surface that interpolates the boundary. This process is commonly used by others for tessellating arbitrarily non-planar boundaries (for e.g. by Mehra et al. [MZL*09]). This method does not behave properly when used for patch boundaries with large concavities (e.g. the Phoenix wings in Figure 4(top)), but the generated surfaces seem good enough for our visualizations.

Each patch is currently rendered independently of the others, which leads to some artifacts (eg. intersecting patches). Future work will address the step of improving the overall surfaces. We expect this to require automatically extracting the correct patch normals for the boundary points (possibly again allowing for user interaction). With G^1 boundary conditions for each patch, we could generate the patch surfaces using more sophisticated methods like those in Freedrawer [WS01], non-linear Willmore flow [BS05] or variational radial basis functions [BMS*10].

8. Limitations and Discussion

We have developed a simple and robust framework for extracting patches from 3D wireframe models. To our knowledge, ours is the first system that handles the kind of topologically unconstrained 3D input data produced by modern designer-friendly 3D sketching interfaces.

Fifteen of the ILoveSketch inputs produced undesired outputs, illustrating some of the limitations of our system. A couple of these inputs consisted of curves which did not intersect in a way that either we, or the system, could interpret as a wireframe model of an object, although they looked visually suggestive. Another few included cycles consisting of a single curve, with no vertices, which we do not generate (we could and will).

But most were situations like the one illustrated in Figure 5, in which the heuristics failed because the sketches included curves which represented features or decorations on the surface of the object. This leads to problems because our heuristics unfortunately have no problem with generating patches which overlap each other.

We also never generate cycles with more than one connected component, that is, patches with holes, which causes similar problems. Patches with holes are perfectly valid cycles and can be included with no problem in the linear algebra framework. Again, finding patches which overlap each other would be a way to decide when to generate a patch with a hole. The reason we do not find them is not a limitation of that framework but that we currently never generate them as candidate patches.

Better heuristics in other senses are possible as well. We are not using the fact that the ILoveSketch inputs are symmetrical, for instance; this might be useful for automatically generating a closing patch (often its symmetric partner is already included in the cycle basis), or mirroring user interactions. It would also be interesting to derive heuristics based on studies of the perception of 3D wireframes.

There are some heuristics that are properties of sets of patches, and so cannot be expressed as weights on the cycles. One is the idea employed by Shpitalni and Lipson [SL96] and Liu and Lee [LL01] that a pair of patches should meet along a smooth curve. Also, we do not encourage edges to be manifold (exactly two adjacent patches), and most importantly on the ILoveSketch inputs we fail to discourage overlapping patches, as described above. Improving the properties of patch sets seems to be the next logical step for our automatic system. We have begun to experiment with an basis improvement phase, after the computation of the heuristically optimal cycle basis but before the interactive phase.

Memory is an issue for our current implementation. We generate an exponential number of cycles in the process of finding the basis, and save them to use later in the interactive phase. This would be better handled by regenerating cycles if needed rather than saving them. We also have some memory issues in the optimization of the surface generation step. Because of these problems, we could not handle some of the largest ILoveSketch inputs.

Acknowledgments

This project was supported by NSF Grant IS-0964357 and Adobe Collaborative Research Funding, for which we are grateful. We thank Dan Anthony Alcantara for help with the video, Dmitry Morozov for topological advice, and Daniel Presedo from the Photoshop group at Adobe for doing the informal evaluation.

References

[AW92] AGARWAL S. C., WAGGENSPACK W. N.: Decomposition method for extracting face topologies from wireframe models. *Computer-Aided Design 24*, 3 (1992), 123 – 140. 2

[BBS08] BAE S.-H., BALAKRISHNAN R., SINGH K.: 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* (New York, NY, USA, 2008), UIST '08, ACM, pp. 151–160. 1, 2, 4, 5, 6

[BC86] BREWER III J. A., COURTER S. M.: Automated conversion of curvilinear wire-frame models to surface boundary models; a topological approach. In *Proceedings of of ACM SIGGRAPH 1986* (New York, NY, USA, 1986), SIGGRAPH '86, ACM, pp. 171–178. 3

[BGdV04] BERGER F., GRITZMANN P., DE VRIES S.: Minimum cycle bases for network graphs. *Algorithmica 40* (June 2004), 51–62. 3

[BMS*10] BRAZIL E. V., MACEDO I., SOUSA M. C., DE FIGUEIREDO L. H., VELHO L.: Sketching variational hermite-rbf implicits. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium* (Aire-la-Ville, Switzerland, Switzerland, 2010), SBIM '10, Eurographics Association, pp. 1–8. 8

[BS05] BOBENKO A. I., SCHRÖDER P.: Discrete willmore flow. In *Proceedings of the third Eurographics Symposium on Geometry processing* (Aire-la-Ville, Switzerland, Switzerland, 2005), SGP '05, Eurographics Association, pp. 101–110. 8

[BW95] BAGALI S., WAGGENSPACK JR. W. N.: A shortest path approach to wireframe to solid model conversion. In *Proceedings of the third ACM Symposium on Solid modeling and applications* (New York, NY, USA, 1995), SMA '95, ACM, pp. 339–350. 3, 4

[DB83] DUTTON R. D., BRIGHAM R. C.: Efficiently identifying the faces of a solid. *Computers & Graphics in Mechanical Engineering 7*, 2 (1983), 143 – 147. 3

[dP95] DE PINA J.: *Applications of Shortest Path Methods*. PhD thesis, University of Amsterdam, 1995. 3

[EH08] EDESBRUNNER H., HARER J.: Persistent homology - a survey. In *Surveys on Discrete and Computational Geometry: Twenty Years Later*, no. 453 in Contemporary Mathematics. American Mathematical Society, 2008. 3

[GH02] GOLYNSKI A., HORTON J.: A polynomial time algorithm to find the minimum cycle basis of a regular matroid. In *Proceedings of the 8th Scandinavian Workshop on Algorithm Theory* (2002), Penttonen M., Schmidt E. M., (Eds.), no. 2368 in SWAT '02, Springer Berlin / Heidelberg, pp. 200–209. 3

[GU83] GANTER M. A., UICKER J. J.: From Wire-Frame to solid geometric: Automated conversion of data representations. *Computers in Mechanical Engineering 2* (sept 1983), 40–45. 3

[Han82] HANRAHAN P. M.: Creating volume models from edge-vertex graphs. In *Proceedings of ACM SIGGRAPH 1982* (New York, NY, USA, 1982), SIGGRAPH '82, ACM, pp. 77–84. 3

[ISC03] INOUE K., SHIMADA K., CHILAKA K.: Solid model reconstruction of wireframe cad models based on topological embeddings of planar graphs. *Journal of Mechanical Design 125*, 3 (2003), 434–442. 3

[JGDP04] JEAN-GUILLAUME DUMAS P. G., PERNET C.: Ffpack: Finite field linear algebra package. In *ISSAC : International Symposium on Symbolic and Algebraic Computations* (2004). 4

[KMMP04] KAVITHA T., MEHLHORN K., MICHAIL D., PALUCH K.: A faster algorithm for minimum cycle basis of graphs. In *Automata, Languages and Programming* (2004), et al. J. D., (Ed.), vol. 3142 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 97–109. 3

[Law76] LAWLER E.: *Combinatorial Optimization: Networks and Matroids*. Holt, Reinhart and Winston, 1976. 4

[LL01] LIU J., LEE Y. T.: Graph-based method for face identification from a single 2d line drawing. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 23*, 10 (oct 2001), 1106 –1119. 3, 8

[LLC02] LIU J., LEE Y. T., CHAM W.-K.: Identifying faces in a 2d line drawing representing a manifold object. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 24*, 12 (dec 2002), 1579 – 1593. 2

[LR05] LIEBCHEN C., RIZZI R.: A greedy approach to compute a minimum cycle basis of a directed graph. *Information Processing Letters 94*, 3 (2005), 107 – 112. 3

[MW80] MARKOWSKY G., WESLEY M. A.: Fleshing out wire frames. *IBM J. Res. Dev. 24* (September 1980), 582–597. 2

[MZL*09] MEHRA R., ZHOU Q., LONG J., SHEFFER A., GOOCH A., MITRA N. J.: Abstraction of man-made shapes. In *Proceedings of ACM SIGGRAPH Asia 2009* (New York, NY, USA, 2009), SIGGRAPH Asia '09, ACM, pp. 137:1–137:10. 8

[She96] SHEWCHUK J. R.: Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, vol. 1148 of *Lecture Notes in Computer Science*. Springer-Verlag, May 1996, pp. 203–222. From the First ACM Workshop on Applied Computational Geometry. 7

[SKSK09] SCHMIDT R., KHAN A., SINGH K., KURTENBACH G.: Analytic drawing of 3D scaffolds. In *Proceedings of ACM SIGGRAPH Asia 2009* (New York, NY, USA, 2009), SIGGRAPH Asia '09, ACM, pp. 149:1–149:10. 1

[SL96] SHPITALNI M., LIPSON H.: Identification of faces in a 2D line drawing projection of a wireframe object. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 18*, 10 (oct 1996), 1000 –1012. 3, 8

[SL04] SUN Y., LEE Y. T.: Topological analysis of a single line drawing for 3d shape recovery. In *Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia* (2004), GRAPHITE '04, pp. 167–172. 3

[VC10] VARLEY P. A., COMPANY P. P.: A new algorithm for finding faces in wireframes. *Computer-Aided Design 42*, 4 (2010), 279 – 309. 2, 3

[WS01] WESCHE G., SEIDEL H.-P.: Freedrawer: a free-form sketching system on the responsive workbench. In *Proceedings of the ACM Symposium on Virtual reality software and technology* (New York, NY, USA, 2001), VRST '01, ACM, pp. 167–174. 1, 8