A Sketching Interface for Feature Curve Recovery of Free-Form Surfaces

Ellen Dekkers[a,*], Leif Kobbelt[a], Richard Pawlicki[b], Randall C. Smith[c]

[a] *RWTH Aachen University, Ahornstrasse 55, 52074 Aachen, Germany*
[b] *RRP & Associates*
[c] *Oakland University*

Abstract

In this paper, we present a semi-automatic approach to efficiently and robustly recover the characteristic feature curves of a given free-form surface where we do not have to assume that the input is a proper manifold. The technique supports a sketch-based interface where the user just has to roughly sketch the location of a feature by drawing a stroke directly on the input mesh. The system then snaps this initial curve to the correct position based on a graph-cut optimization scheme that takes various surface properties into account. Additional position constraints can be placed and modified manually which allows for an interactive feature curve editing functionality. We demonstrate the usefulness of our technique by applying it to two practical scenarios. At first, feature curves can be used as handles for surface deformation, since they describe the main characteristics of an object. Our system allows the user to manipulate a curve while the underlying non-manifold surface adopts itself to the deformed feature. Secondly, we apply our technique to a practical problem scenario in reverse engineering. Here, we consider the problem of generating a statistical (PCA) shape model for car bodies. The crucial step is to establish proper feature correspondences between a large number of input models. Due to the significant shape variation, fully automatic techniques are doomed to failure. With our simple and

*Corresponding author. Phone: +49 241 8021815, Fax: +49 241 8022899

Email addresses: dekkers@cs.rwth-aachen.de (Ellen Dekkers), kobbelt@cs.rwth-aachen.de (Leif Kobbelt), richard.pawlicki@gmail.com (Richard Pawlicki), 1randall.c.smith@gmail.com (Randall C. Smith)

URL: www.graphics.rwth-aachen.de (Ellen Dekkers)

effective feature curve recovery tool, we can quickly sketch a set of characteristic features on each input model which establishes the correspondence to a pre-defined template mesh and thus allows us to generate the shape model. Finally, we can use the feature curves and the shape model to implement an intuitive modeling metaphor to explore the shape space spanned by the input models.

Keywords:
Feature extraction, sketch-based interfaces, curve-based modeling, surface registration, statistical shape model

---

1. Introduction

In this work, we address the problem of recovering smooth feature curves from free-form surfaces. We do not impose any simplifying assumptions on the characteristics of the input surfaces. Unrestricted input exhibits a variety of challenging characteristics: An input model is a triangle or general poly mesh, which is not required to be watertight or manifold, let alone genus zero. It supposedly consists of a large number of surface patches which intersect arbitrarily or contain gaps in between each other. In general, a model can include much geometric detail which imposes difficulties on the recovery of smooth curves as well as arbitrary geometric structure in its interior. Furthermore, we do not require a consistent normal orientation.

The output of our system are B-spline curves that smoothly approximate the features of the input model. Due to the complexity of the data, fully automatic approaches for feature recovery are doomed to failure. Furthermore, the aesthetic question of what a feature actually is and what a method should be able to recover cannot be modeled mathematically. However, our system provides maximal user-support in recovering what he considers a feature.

Recovering features from free-form surfaces is an intensively investigated field in Computer Graphics and CAGD. A closed network of feature curves allows for the segmentation of an input model into meaningful sub-parts which is a key issue for many further applications on meshes such as parametrization, morphing, matching, registration, or compression. There exist a large number of mesh segmentation methods, automatic as well as semi-automatic approaches, which incorporate some user-interactivity. Providing a detailed survey on mesh segmentation is beyond the scope of this work. However, one can roughly classify existing methods into two classes: The first class

tries to identify meaningful regions and then refines the borders separating them. There exists a variety of approaches in the literature which make use of watershed segmentation [1], clustering [2], [3], [4], region growing [5], or random walks [6] to group similar mesh elements into meaningful regions. Fitting of geometric primitives (e.g. [7], [8]) is suitable for decomposing models of mechanical parts in reverse engineering applications. The second class of segmentation methods aims at the identification of segment borders which implicitly define the segment regions. To extract meaningful patches, one usually wants to align the borders to mesh features. However, depending on the application, the definition of a feature varies. Graphical models on the one hand impose different requirements than engineering objects or CAD models on the other hand. For the first type of objects, the goal is to split it into meaningful parts. From cognitive theory [9], we know that human perception divides an object along significant concave features, widely referred to as the minima rule. For objects of the latter type, one mostly aims at segmenting the model into parts which can be fitted with some analytical surface and hence in many cases the minima rule is not suitable for identifying the borders. Therefore, several methods make use of snakes [10], [11], [12] to identify features in a mesh, since an energy functional can be adapted to the application-dependent definition of a feature. However, a drawback of snake-based feature extraction is their restriction to detect local minima, only. Once a snake settled to its final (locally optimal) position, it needs to be detached manually to recover the global optimum. In [2] graph cuts (e.g. [13]) were used to automatically refine an initial segment boundary within a transition region derived from fuzzy clustering.

Approaches that automatically recover global features from surface meshes [14], [15], [16] require the computation of higher-order surface derivatives thus imposing limitations (2-manifoldness) on the meshes to be processed. Furthermore, careful parameter tuning may be necessary since automatic feature extraction is based on heuristics which might recover insignificant parts. [17] require the user to provide a few control parameters and operators to be applied to the input surface and hence are calling for a skillful user.

In recent years a lot of research has been spent on sketch-based interfaces. The user draws some strokes on a 2D canvas from which the system e.g. creates smooth three dimensional surfaces (e.g. [18], [19], [20], [21]). [20] consider the sketched curves as features of the resulting model and further use them for surface editing.

The incorporation of user assistance into the task of mesh segmentation

greatly supports the detection of salient features. [5] let the user quickly draw some freehand strokes on an input model marking subparts of interest. A region growing algorithm then extracts the segment boundaries which are finally smoothed using snakes. Furthermore, the user can manipulate the resulting boundaries by, e.g., dragging them over the surface.

2. Overview & User Interface

Our system provides an intuitive sketch-based user interface that supports the user in recovering smooth feature curves from highly complex free-form surfaces where automatic approaches cannot be applied. Unlike the methods described in Sec. 1, the input to our method can be arbitrary. Since an input model may consist of a large number of surface patches, a feature curve may have to span over several patches. Furthermore, when the position of a curve is optimized it is also likely that it leaves one patch and moves onto a neighboring patch. For non-manifold input, the definition of a patch neighborhood is ambiguous and we cannot compute geodesic paths on the model. Most existing methods align feature lines to edges of the input model. Since mesh faces in CAD models often vary largely in scale, the resulting boundary may suffer from artifacts caused by the poor mesh tessellation. In contrast, our system generates smooth curves which follow feature regions of the input model regardless of the underlying tessellation.

The fundamental idea of our work is the transformation of arbitrary input into a regular intermediate structure which then allows for an evaluation of the surface characteristics. To avoid general expensive 3D mesh repair, we reconstruct the missing structural information locally. By using a "fishbone structure" (cf. Sec. 3.3) we are able to reduce the reconstruction problem from 3D surfaces to two 2D curves. The local structural information provided by the fishbone enables a graph-cut based algorithm that optimizes a curve subject to various surface properties, thereby satisfying smoothness requirements and guaranteeing to find the global optimum.

The user starts an interactive modeling session by loading an input model into our system. The interface allows for intuitive navigation in 3D space such that the model can be inspected from arbitrary viewpoints. To create a new curve, the user roughly places some points in a region of interest on the input mesh from which an an initial curve is automatically computed. Our system supports the user in the task of recovering features by automatically "snapping" the sketch curve to regions of the most likely feature location.

The user can adjust the relative weighting of the optimization objectives using a slider and scale the stiffness of the feature curve. In case he is not yet satisfied with the result, positional constraints can be imposed by pulling the curve over the input surface.

The optimization criteria as well as positional constraints for selected points on the curve can be modified interactively. The feature curve then automatically adopts itself to the new constraints in real time providing direct feedback. We hide any technical details of the underlying algorithms such as specific parameters from the user. This makes our system easily usable for experts as well as inexperienced users.

We demonstrate the usefulness of our sketch-based interface by applying it to two practical problem scenarios. In Sec. 4.1 we use the recovered feature curves as handles for surface editing. In Sec. 4.2 we construct a statistical shape model for car bodies, where the feature curves are used to establish proper feature correspondences between the input models and furthermore allow for intuitive navigation in the resulting shape space.

Please refer to the accompanying video for a demonstration of our interface and of both application scenarios.

3. Implementation

In order to ensure high geometric quality, all curves created in an interactive modeling session are B-spline curves

$$c(t) = \sum_{i=0}^{n} d_i \cdot N_i^p(t).$$

The number of control points d_i is chosen proportionally to the length of the curve and $N_i^p(t)$ are B-spline basis functions. By default, we set the curve's degree p to three, although the system is not restricted to cubic curves.

3.1. Initial Curve Generation

Given a set of surface points sketched by the user, our system generates an initial curve by interpolation. To increase resolution, the resulting curve is sampled at equidistant locations and the samples are projected onto the closest point on the input mesh. In the following, we refer to these projected curve points as surface samples $S = \{s_0, \ldots, s_{m-1}\}$. A curve $c(t)$ embedded in the surface is then obtained by computing a chordal parametrization t_i of

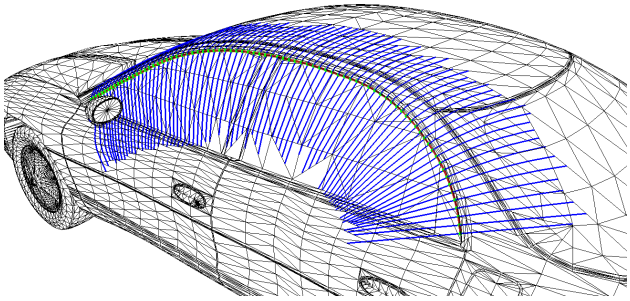


Figure 1: A fishbone structure on a car model: For each surface sample (green) a manifold rib polygon (blue) is constructed which is orthogonal to the approximating curve (red).

the samples s_i as well as a knotvector $U = \{u_0, \ldots, u_{n+p+1}\}$ that respects the distribution of the parameter values t_i [22]. We approximate the surface samples s_i in the least squares sense such that

$$E(c) = \sum_{i=0}^{m-1} \|s_i - c(t_i)\|^2$$

is minimized. Since later some samples s_j describe positional constraints imposed by the user, they are required to be interpolated exactly. Hence, a constrained least squares approximation is formulated by augmenting the approximation component $N \cdot C = A$ with interpolation constraints $M \cdot C = B$. Here, N and M are matrices of B-spline basis functions, C are the unknown control points, A are samples to approximate and B are the samples to interpolate. We minimize the sum of the errors in the approximation component subject to the interpolation constraints using Lagrange multipliers [22].

3.2. Curve Dragging

To pull a curve over the input surface, the user clicks on the curve and the system computes the closest surface sample s_c. Then, s_c is projected into the image plane and translated by a displacement defined by the movement of the mouse on the screen. The new position s'_c is obtained by reprojecting the screen position back onto the mesh, which then induces an additional interpolation constraint to the subsequent approximation.

We prefer the computation of the displacements in screen space over their computation on the 3D surface itself using geodesic distances because it allows for a more intuitive control of curve behavior. Furthermore, in

contrast to the computation of geodesic paths, reprojecting screen positions back onto the mesh is always possible regardless of any mesh inconsistencies.

3.3. Feature Curve Optimization

In order to treat all different types of surface quality constraints in a uniform manner, we do not optimize the curve $c(t)$ itself but the positions of the surface samples s_i that $c(t)$ approximates. Allowing the s_i to move freely on the surface during optimization would lead to clustering in areas which best meet the optimization criteria. We therefore restrict their trajectories to paths perpendicular to the approximating curve. On the one hand, this avoids clusters since surface samples maintain a certain distance to each other. On the other hand, it allows for easy local curve resampling in the case that the sampling density falls below or exceeds certain density thresholds.

We use a fishbone structure as proposed by [23], to define the trajectories along which the surface samples s_i are allowed to move during optimization. The feature curve $c(t)$ thereby forms the backbone while rib polygons are created as follows. Since $c(t)$ approximates a set of m surface samples $S = \{s_0, ..., s_{m-1}\}$, there exists a curve parameter value t_i for each sample s_i. At each $c(t_i)$ we define a rib by a plane orthogonal to the curve, i.e. its normal equals the curve's tangent at $c(t_i)$. Intersecting the input mesh with a rib plane traces out a set of edges which need to be joined to form a rib polygon that represents the outer contour. A complete fishbone structure is obtained by reconstruction a manifold polygon for each rib. Fig. 1 shows a fishbone for a curve on a car's roof.

Note, that the planarity property of the ribs allows us to perform the manifold reconstruction in a 2D plane. The watertight repair of 3D meshes (see e.g. [24]) is considerably more difficult than 2D polygon repair due to the much larger variety of artifacts and inconsistencies that can occur (e.g., non-manifold vertices and edges, gaps, partial overlaps, holes and self-intersections; see [25] for a survey). In the 2D case of contour repair, all we need to do is to recover the proper vertex sequence for which we suggest a local marching procedure in the following section. Notice that surface repair based on a local marching technique is not guaranteed to work correctly since objects like Moebius strips and Klein bottles might be generated accidentally.

3.3.1. Rasterization-based Manifold Reconstruction

Intersecting the input model with a rib plane creates an unordered set of edges. Assuming that the initial curve is sketched in a region of interest and

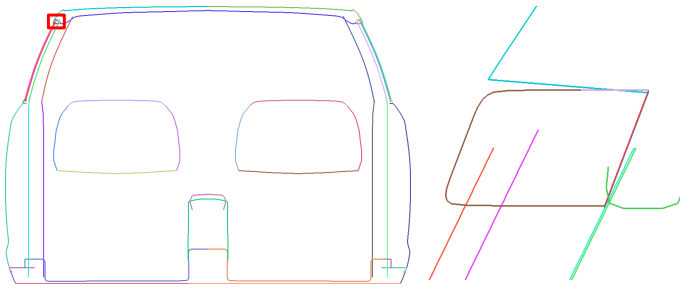


Figure 2: A slice through an input model. Surface patches intersect arbitrarily, hence a purely geometric repair algorithm is prone to failure. The image on the right shows a close-up of the area enclosed by the red box in the left image.

hence is not supposed to change its location by more than a certain distance, we can accelerate the construction of the rib polygons by reducing the search space. We therefore discard all intersections of a rib plane with the input model that do not lie within a circle with a predefined radius around the backbone. The radius defines the region around the user-sketched feature curve where the actual feature curve should be searched for. Hence the choice of this parameter depends on our expectations of how close to the real feature the user sketch lies. In our application scenarios (Sec. 4), this radius is automatically chosen to be 15 cm (car models are given in full scale).

Since the input model can be arbitrary, a manifold polygon representing the outer contour cannot be reconstructed in a simple mesh traversal. However, we can traverse each surface patch separately and connect its intersection edges to a manifold polygonal segment. What remains is the connection of these segments to a single manifold contour polygon that interpolates the outer silhouette. A simple and purely geometric approach merges the segments by concatenating them at coinciding endpoints. However, since surface patches may intersect arbitrarily, this also holds for the segments (cf. Fig. 2). Unfortunately, no robust criteria can be found to distinguish between segments that should be (partially) contained in the contour and those that do not. However, prior to performing the following rasterization-based reconstruction, we check the polygonal segments for intersection and fall back to this purely geometric approach in the case that none are detected.

We propose a rasterization-based approach that is able to robustly reconstruct a planar manifold polygon given an arbitrary set of polygonal segments. We start by rendering all segments into an offline buffer to create an image I showing a slice through the input mesh. Since small gaps may occur

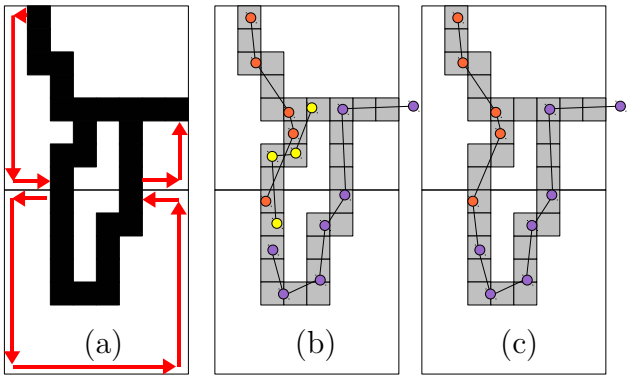


Figure 3: Manifold extraction: Our algorithm gains topology information for the reconstruction of a contour in an image traversal (a). Identifying the correct intersections of grid edges with polygon edges (b) allows for the construction of a manifold contour polygon (c) from a set of arbitrarily intersecting polygonal segments (orange, yellow, purple).

in-between the segments, we execute k dilation steps on I, where k depends on the size of the gaps to be closed. During the subsequent k erosion steps, we follow the approach of [24] and erode a foreground pixel $p_{x,y}$ if and only if it satisfies both of the following two conditions: First, $p_{x,y}$ was not set as foreground pixel during the initial rasterization. Secondly, when cycling the 8-pixel neighborhood of $p_{x,y}$ we encounter at most one switch from background to foreground and vice versa. This ensures that in the resulting image small gaps are closed while the original topology is preserved otherwise.

The idea of our algorithm is to reconstruct a manifold polygon by walking around the outside of the contour. To accelerate the computation we create a uniform axes-aligned grid that partitions the image into quadrangular cells. Our algorithm then identifies all cells containing geometry information and forming the contour by walking along the grid edges. Simultaneously, it identifies the edges within the visited cells that constitute the contour and adds them to the manifold polygon in the correct order.

Consider Fig. 3 for an illustration of our method: (a) shows two exemplary cells of size 8×8 pixels. Suppose we detect a first foreground pixel (black) on the top grid edge of the upper cell as starting point for the contour extraction. Starting from there, we walk along the cell's grid edges in counterclockwise direction and compare the colors of neighboring pixels. The next occurrence of a black pixel indicates at which edge the contour leaves the current cell and enters one of the four neighboring cells.

The image traversal serves as topological guide for the construction of a

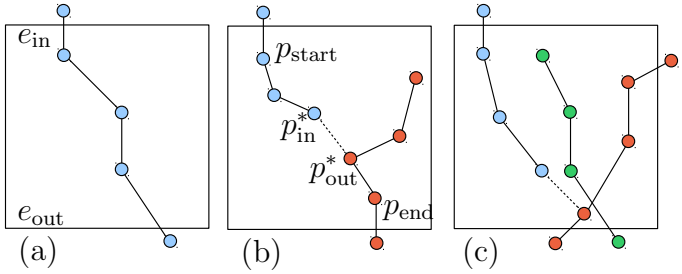


Figure 4: Different cases occurring in a single cell during the contour reconstruction: (a) One segment intersects the cell. (b) Two segments intersecting the cell are connected to a single manifold segment. (c) Ambiguous case: Grid edges are intersected multiple times.

manifold contour polygon. We add the points within each visited cell to the polygon depending on the cell characteristics depicted in Fig. 4. A cell can be intersected by a single polygonal segment (a), or by multiple segments. In the latter case, we need to distinguish between a situation where both the entering and the leaving grid edge are intersected by exactly one segment (b), and the situation in which the entering and/or the leaving grid edge is intersected multiple times by possibly different segments (c).

Suppose we enter a cell on the top edge e_{in}, which is intersected by a segment S_{in}, and leave it on the bottom edge e_{out}, which is intersected by a segment S_{out}. If $S_{\text{in}} = S_{\text{out}}$ (a) we simply add all points $p_i \in S_{\text{in}}$ within the cell to the final manifold polygon. If $S_{\text{in}} \neq S_{\text{out}}$ (b) we compute two point sets $P_{\text{in}} \subset S_{\text{in}}$ containing all points within the cell that belong to segment S_{in} and $P_{\text{out}} \subset S_{\text{out}}$ containing all points that belong to S_{out}. We then compute the pair of points $p^*_{\text{in}} \in P_{\text{in}}$ and $p^*_{\text{out}} \in P_{\text{out}}$ having the minimum distance

$$\arg\min_{p_{\text{in}} \in S_{\text{in}}} \arg\min_{p_{\text{out}} \in S_{\text{out}}} \{\text{dist}(p_{\text{in}}, \overline{p_{\text{out}-1} p_{\text{out}}})\}.$$

We add the point set $\{p \in P_{\text{in}} | p_{\text{start}} \leq p \leq p^*_{\text{in}}\}$, i.e., all points between p_{start} and p^*_{in} on segment S_{in}, and then add the point set $\{p \in P_{\text{out}} | p^*_{\text{out}} \leq p \leq p_{\text{end}}\}$, i.e., all points between p^*_{out} and p_{end} on segment S_{out}, in the correct order to the final polygon. The relation $<$ thereby refers to the index-based explicit ordering of points in a manifold polygon. By p_{start} and p_{end} we denote the endpoint of the edge in S_{in} or S_{out}, respectively, which intersects the respective grid edge and lies within the current cell. In the case that we encounter a situation with more than one intersection on the leaving grid edge (cf. Fig. 4 (c)), the choice for the leaving segment S_{out} becomes ambiguous. Since our

goal is to reconstruct the outer contour, we simply chose the first intersection we encounter while traversing the grid edge, i.e., the intersection closest to the edge's starting point (red segment in Fig. 4(c)). Although the contour extraction is topology-driven, the geometrical decision is inevitable, since several intersections may be contained within one foreground pixel and the image traversal does not provide enough topology information. The selected outgoing intersection is kept as incoming intersection for the neighboring cell and hence does not need to be recomputed.

Since our method extracts a closed manifold contour polygon it happens that if the input is not closed, we extract the inside of the input as well. However, since the reconstruction is performed in a 2D plane, we are able to compute consistent normals at each vertex of the resulting manifold polygon. This on the one hand allows for distinguishing between the outer and the inner part of the contour and furthermore overcomes the possible lack of a consistent normal orientation in the input model.

In our experiments, we found an image resolution of 800×600 pixels and $k = 1$ dilation and erosion steps together with a grid resolution of 8×8 pixels sufficient for the construction of precise contour polygons.

3.3.2. Graph-Cut Optimization

A fishbone with one manifold rib polygon per surface sample establishes a locally regular structure in highly unstructured surface areas and allows us to optimize a feature curve $c(t)$. Instead of optimizing $c(t)$ directly, we compute optimal positions of the samples s_i on the input surface and re-approximate them by $c(t)$. We optimize the positions of the surface samples w.r.t. a set of criteria: They include external forces modeling surface properties as well as an internal force imposing smoothness requirements to the optimization problem. The latter is crucial since we would like the resulting polygon of surface samples to exhibit low geodesic curvature.

We formulate the task of extracting optimal sample positions as a graph cut problem as illustrated in Fig. 5. Each rib polygon b_i is sampled at dense but discrete locations $b_{i,j}$, constituting a set of possible positions of the associated surface sample s_i (a). By sampling each rib, we construct a planar regular quad graph structure $G = (V, E)$ with one column per rib (b). Notice that the edges E, not the vertices of column i in G represent the samples $b_{i,j}$ on rib i. Edges connecting samples of the same rib (vertical edges) are in the following referred to as *rib edges* whereas edges connecting vertices of neighboring ribs are referred to as *cross edges*.

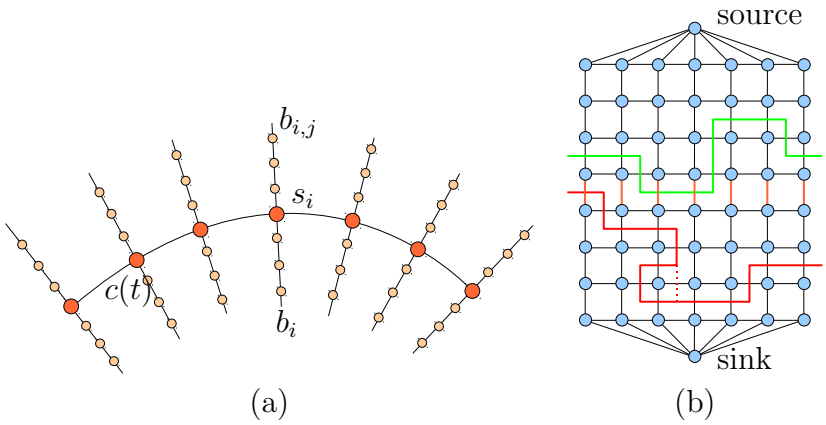


Figure 5: Fishbone structure: (a) The feature curve $c(t)$ forms the backbone of a fishbone structure with one orthogonal rib b_i per surface sample s_i. The optimal positions of the surface samples are obtained by embedding the fishbone into a planar graph structure (b) and computing the minimum cut. The orange edges represent the backbone samples s_i. Our algorithm guarantees the cut to be monotonic (green line) by assigning appropriate edge capacities. A cut as depicted by the solid red line which results in ambiguous sample positions is impossible since there always exists a cheaper cut (dashed red line).

Capacities are assigned to rib edges by evaluating the following two external optimization criteria at the corresponding rib samples $b_{i,j}$. The first criterion pulls the curve to regions of maximum curvature. We therefore evaluate the curvature at each sample $b_{i,j}$ as the average of the discrete normal curvature in the directions of the adjacent rib edges $e_0 = (b_{i,j-1}, b_{i,j})$ and $e_1 = (b_{i,j}, b_{i,j+1})$:

$$\omega_{\text{curv}}(b_{i,j}) = \frac{1}{2}(|\kappa_k| + |\kappa_{k+1}|), \text{with} \quad \kappa_k = \frac{2 \cdot (b_{i,k-1} - b_{i,k}) \cdot n}{\|b_{i,k-1} - b_{i,k}\|^2}$$

and n being the edge normal. The second criterion promotes feature curves to approximate isophotes and hence encourages them to align to surface regions with a specific normal angle. We evaluate the surface normal angle with one of the axes of the coordinate frame at each rib sample $b_{i,j}$ as

$$\omega_{\text{normal}}(b_{i,j}) = |\arccos(n_{i,j} \cdot a) - \beta|$$

where β is the desired surface normal angle and a is the x-, y-, or z-unit vector of the coordinate frame. The surface normal $n_{i,j}$ at sample $b_{i,j}$ is obtained by linear interpolation of the normals at the endpoints of the current polygon

edge. Since the criteria measure different quantities we normalize ω_{curv} and ω_{normal} to the interval $[0, 1]$. To obtain the capacity τ of a rib edge in G, we simply compute the weighted sum over the normalized external forces:

$$\tau(b_{i,j}) = \alpha \cdot \omega_{\text{curv}}(b_{i,j}) + (1 - \alpha) * \omega_{\text{normal}}(b_{i,j}). \quad (1)$$

The parameter α allows for weighting of the criteria. Note, that optimization objectives are often application dependent and the external criteria presented here are examples. It is straightforward to integrate any other forces that can be evaluated on fishbone ribs.

The capacities assigned to rib edges model external forces that shift the samples to those positions on the surface that best meet the optimization criteria. However, optimizing the sample positions subject to these forces only can lead to an arbitrary zig-zag shape of the resulting surface sample polygon. We therefore introduce an internal smoothness criterion by assigning an appropriate capacity to cross edges. This capacity θ is set to be always greater than all capacities associated to rib edges

$$\theta \geq \max_{i,j} \tau(b_{i,j}).$$

which ensures that in each column of G exactly one rib edge is cut. Hence, the resulting minimal cut through G is always monotonic (green line in Fig. 5(b)). A cut as depicted by the red solid line, which would lead to more than one optimal position for a surface sample, cannot occur since its costs are guaranteed to be larger than the costs of the cut depicted by the dashed line.

The user can modify the stiffness of the resulting curve by adjusting a stiffness factor $f \geq 1$ which scales the capacity of cross edges. A positional constraint for a surface sample s_c is integrated into the graph embedding by assigning a capacity equal to zero to the graph edge associated to the rib sample $b_{c,j}$ that has minimum distance to the desired surface position, while the capacities of all other edges in column c are set to infinity.

Finally, the top row of vertices of G is connected to the source, the bottom row is connected to the sink and all edges connecting source and sink are assigned an infinite capacity guaranteeing they are never cut. When computing the minimal cut through G, the quad-structure of G ensures that each column is cut at least once, since otherwise it is impossible to disconnect source and sink. Simultaneously, the choice of the capacities ensures that in each column exactly one vertical edge is cut. The rib sample b_i^* associated with the edge cut in column i is the optimal position for the surface

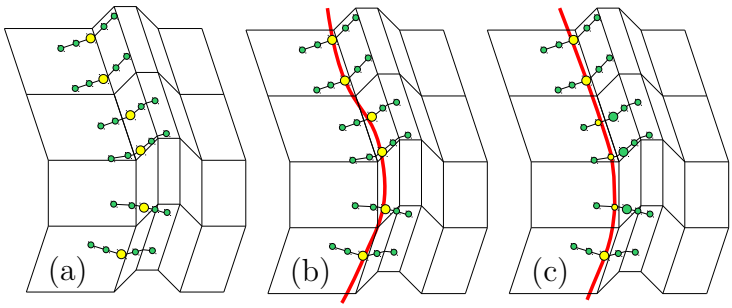


Figure 6: Geometric detail in the input model may induce the projection of surface samples (large dots) to both sides of a sharp feature and hence results in a zig-zag shape of the backbone samples (a). Constructing the fishbone and running the graph cut optimization without synchronizing the ribs results in a feature curve suffering from high curvature (b). Our rib synchronization establishes a smoothed line of backbone samples (yellow dots) and hence results in a smooth optimized feature curve (c).

sample s_i. After reprojecting the b_i^* onto the input model, yielding the final optimal surface samples $S^* = \{s_0^*, ..., s_{m-1}^*\}$, we recompute a least squares approximation of S^* to obtain the optimized feature curve $c(t)$.

A special case that needs to be taken into account in the curve optimization is posed by geometric detail contained in the input model (cf. Fig. 6). During the projection of an initial sketch curve (cf. Sec. 3.1), the surface samples s_i are projected onto different sides of a sharp feature (a). Regardless of the lack of smoothness, the graph cut embedding would encode the s_i as opposite horizontal edges in the center row of the graph structure. Hence, the zig-zag shape is assumed to be maximally smooth since no horizontal edges would need to be cut. A curve that approximates these samples suffers from high curvature as illustrated in Fig. 6 (b). To overcome this problem, we compute an index offset to "synchronize" neighboring ribs prior to the computation of the graph cut embedding. Sampling each rib as usual and starting at the backbone sample $\tilde{s}_i = s_i$ of rib i (with i being either the first or some rib in the middle), we determine the sample on the neighboring rib $i + 1$ that has minimum Euclidean distance to $\tilde{s}_i$ and define it as the new backbone sample $\tilde{s}_{i+1}$. The number of samples in between the original surface sample s_{i+1} of rib $i + 1$ and the new sample $\tilde{s}_{i+1}$ determines in which direction and how many steps the rib must be shifted to smooth the backbone. Running the graph cut optimization on the synchronized fishbone produces a smoother feature curve (cf. Fig. 6(c)).

The costs of the computation of the fishbone depend on the number of ribs

that need to be reconstructed using the rasterization approach in case that the simple geometric approach cannot be applied due to inconsistencies in the input model. When the geometric approach is chosen, the entire fishbone is constructed in real time. It takes additional 0.3 sec. per execution of the rasterization-based repair algorithm (on an Intel Core2 Duo 2.66GHz with 4GB RAM). The number of executions depends on the complexity of the model's area that is being repaired. In our experiments we observed that on average the image-based repair algorithm was selected (instead of) the geometric fallback approach for about ten percent of the rib polygons.

4. Applications

We demonstrate the usefulness of our system by applying it to two practical problems: feature curve-based surface modeling (cf. Sec. 4.1) and the generation of a statistical shape model for car bodies (cf. Sec. 4.2).

4.1. Feature Curve Based Surface Modeling

Feature curves describe the main characteristics of a surface and hence modifying them changes the overall shape of the surface in a meaningful way. There exists a variety of surface deformation methods in the literature on which [26] provide an excellent overview. A widely used paradigm to deform a surface is to let the user define a region of interest (ROI) on a model and to edit this region by moving or rotating a handle. This rigid handle metaphor is well suited for our setting, since we would like to manipulate feature curves and propagate the modification to the input model. For surface deformation, we use the well-known "Laplacian surface editing" technique [27].

Laplacian based modeling approaches require the input model to be manifold and hence they cannot be applied to our models directly. However, during curve optimization a fishbone structure has been established that reconstructs the missing manifold information locally. Now, we further exploit this structure to enable Laplacian surface editing on unrestricted input models in the following way: From the fishbone, a manifold mesh patch is constructed, which uses a feature curve as deformation handle. After manipulating the curve, this patch is deformed accordingly using Laplacian surface editing and finally its deformation is transferred to the non-manifold input model using normal displacements.

In more detail, our methods works as follows (consider Fig. 7(a) for an illustration): A feature curve $c(t)$ constitutes the backbone of its associated

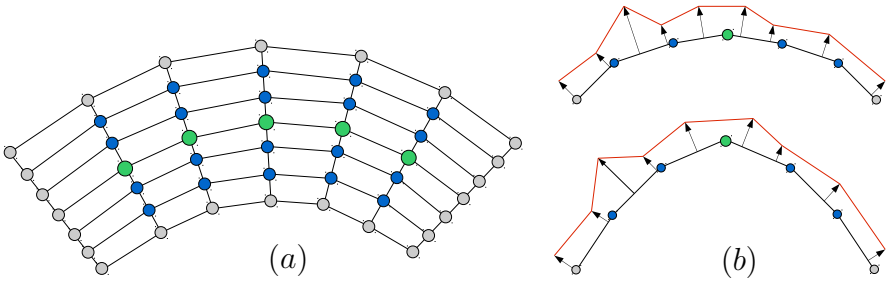


Figure 7: Curve-based modeling: (a) We exploit the regular structure of a fishbone (cf. Fig. 5(a)) to generate a manifold mesh patch. The handle vertices (green) are directly related to the feature curve, gray vertices represent fixed boundary constraints while the remaining inner vertices (blue) represent the deformable region of the mesh. (b) 2D illustration. Top: Initial configuration, bottom: configuration after deformation. When the user modifies a feature curve, the positions of the associated handle vertices (green) in the fishbone patch are updated accordingly, while the blue vertices are deformed using a Laplacian-based modeling technique. The deformation of the patch (black) is transferred to the input model (red) using normal displacements.

fishbone (cf. Fig. 5(a)) and together with its ribs naturally induces a regular quad structure, from which we construct a local manifold mesh patch P by triangulating neighboring rib samples. There exist three types of vertices in this mesh: handle vertices (green) constituting the handle region, modeling vertices (blue) representing the deformable ROI, and boundary vertices (gray) which impose boundary constraints into the modeling and which represent the fixed part during the deformation. We select the handle vertices to be directly related to the feature curve and hence they are given by the center row in the fishbone patch P. All inner vertices of P constitute the ROI, which later undergoes the deformation, while all boundary vertices describe the non-deformable boundary constraints.

Since in the end we do not want to deform a fishbone patch which locally resamples the input geometry but the input model itself, we must transfer the deformation of the patch to the model. Therefore all model vertices v_i that are covered by the patch P are identified and project onto their closest point on P. This yields projected positions v_i^{proj} which can be described as a barycentric combination of the vertices of the patch triangle $\Delta_P(w_i, w_j, w_k)$ onto which v_i was projected. Furthermore, we store the normal displacement $n_i = v_i^{\text{proj}} - v_i$ in the local coordinate system of the triangle Δ_P.

Our feature curve based surface modeling then works as follows (cf. Fig. 7(b)): The user manipulates a feature curve in 3D space which directly

defines new positions for the handle vertices in the associated fishbone patch. The thereby induced Laplacian modeling step yields a deformed patch P^* that fulfills these positional handle constraints. Finally, the new position v_i^* of a model vertex v_i that was covered by the original fishbone patch is obtained by applying its barycentric coordinates to the deformed patch triangle $\Delta_{P^*}(w_i^*, w_j^*, w_k^*)$ and adding the normal displacement n_i^* which is transformed according to the deformed triangle Δ_{P^*}:

$$v_i^* = (\alpha_i \cdot w_i^* + \beta_i \cdot w_j^* + \gamma_i \cdot w_k^*) + n_i^*$$

Here, $\alpha_i, \beta_i, \gamma_i$ are the barycentric coordinates of the projected model vertex v_i^{proj} w.r.t. the undeformed patch triangle $\Delta_P(w_i, w_j, w_k)$.

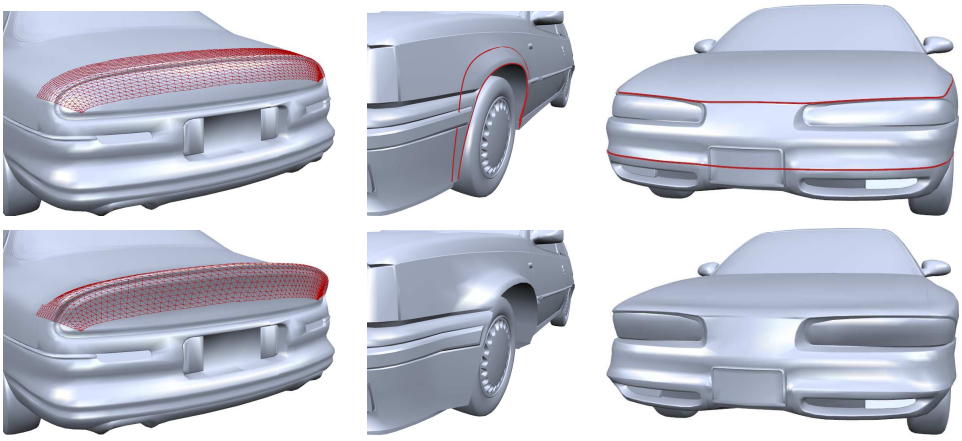
To control the range of a feature curve-based deformation on the input model, the user can adjust the width of the fishbone patch, i.e., the length of the ribs using the mouse wheel. Short ribs result in a local deformation while long ribs allow for editing larger regions of the input model.

Since the user often modifies more than one feature curve while editing the input model, a model vertex may be influenced by multiple fishbone patches. In these cases, we compute the final vertex position as the average of its deformed positions in all fishbone patches.

Results. Fig. 8 depicts some results we obtained with our system. In (a) a feature curve on a car's trunk was edited to create a spoiler. In (b) two feature curves were recovered from a wheel opening and were modified to expand the car's wheelhouse. In the third example (c) we show the front of a car model, where one feature curves was recovered from the lower part of the bumper and another one from the top of the headlights. Both curves were deformed to make the car's front area appear more bulky. In figures (b) and (c) the fishbone patches are omitted to avoid overloading the images.

4.2. A Morphable Shape Model for Cars

In our second application we apply our system to the practical problem of generating a morphable shape model for car bodies. After recovering a set of feature curves from an input model, we automatically construct a feature network which establishes correspondences to a pre-defined template mesh. The input model is then registered to the template guided by the feature network in a fully automatical way. Having registered a set of car bodies to the same template finally allows for the construction of a statistical



(a) (b) (c)

Figure 8: Curve-based editing of car models. (a) A manifold mesh patch (red) is constructed from the feature curve's fishbone (top). The deformation of the patch is transferred to the non-manifold input model (bottom). (b),(c): The input model with the recovered feature curves (top) and a result obtained with our modeling system (bottom).

(PCA) shape model. In addition, the feature network implements an intuitive metaphor for exploring the shape space spanned by the input models.

There exists a variety of related work in the context of conceptual design for automotive shapes. We will briefly review three approaches that are closely related to our work. [28] align a 3D template model to 2D sketches. The aligned model is then refined by tracing a car's characteristic lines on the sketch. [29] represent a car by a network of polylines capturing the main features. The user can manipulate the network by pulling single vertices and sketching over lines to generate new shapes. However, the feature network was extracted from input models in a solely manual process. In [30] a framework for navigation in a shape space of registered models of automotive shapes is presented. Again, the registration problem is solved in a tedious manual preprocessing step. The user can explore the shape space by dragging single feature points or manipulating shape space parameters.

4.2.1. Registration

To reduce distortion in the later PCA model, we segment a car into four components: roof, body and front and rear wheelhouse. We thereby employ the symmetry w.r.t. the plane that separates the driver's side from the passenger's side and register the driver's half only, while the remaining

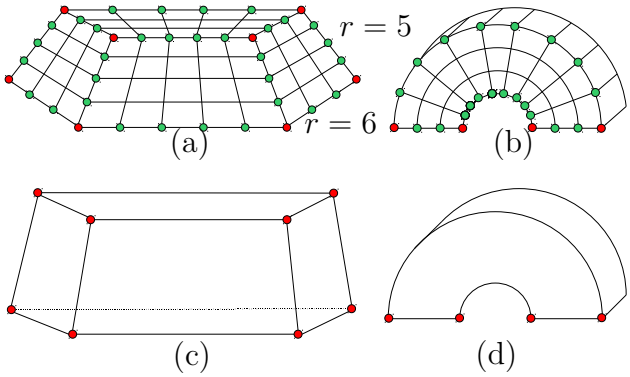


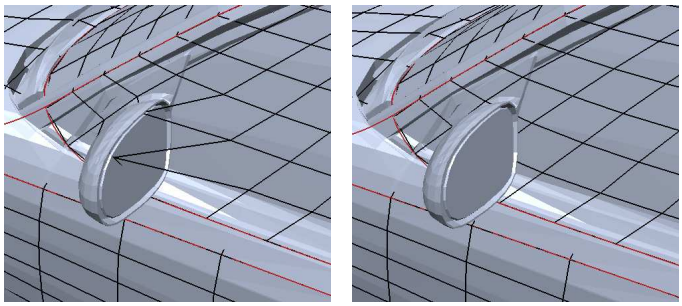
Figure 9: Template mesh and associated curve network: (a) Roof/body mesh, (b) wheelhouse mesh, (c) roof/body network, (d) wheelhouse network.

half is obtained by mirroring the result. For each component, we define a network of feature curves as well as a template mesh as depicted in Fig. 9. An input model is registered in a two-step procedure.

Network construction. Since usually the endpoints of a recovered feature curve are not placed precisely, we compute the feature endpoints by pairwise intersecting the input curves yielding the red nodes in the curve network in Fig. 9 (c) and (d). If three feature curves do not intersect in a common point, we compute the average of the three pairwise intersection points.

Mapping. In the second step, a pre-defined template quad mesh is mapped to its associated curve network thereby roughly approximating the underlying geometry of the input model.

Our system constructs a template mesh on-the-fly letting the user define its resolution. Note that the computation of a PCA shape model requires a common resolution of all registered models. Fig. 9 (a) shows an exemplary template mesh where the roof and the body are a topological cube with four or five faces, respectively. The wheelhouse component is depicted in (b). Each template component contains feature vertices (red and green) and non-feature vertices (not highlighted). The red feature vertices directly correspond to the red intersections of feature curves in (c) and (d). The green feature vertices correspond to samples on the network, which are obtained by uniformly sampling the relevant interval of the respective feature curve at $r - 2$ locations, depending on the template resolution r.



(a) (b)

Figure 10: Mapping problem: (a) The outmost intersection of a vertex normal with the input model may lie on geometric detail which should not be captured with the template mesh. (b) We eliminate these outliers in the mapping step by integrating a vertex regularization into the sampling of the input surface.

While the final positions of the feature vertices are defined by the curve network, the final positions of non-feature vertices are computed as a combination of a sampling of the input surface and a vertex regularization. First, we construct a uniform Laplace system with all feature vertices as positional boundary constraints and solve for the coordinates of the non-feature vertices p_i^{mem}. This distributes the non-feature vertices on the membrane surface spanned by the feature vertices (for more details, please see [31]) and allows us to compute vertex normals n_i. In the following projection step, we compute the outmost intersection of the vertex normal with the model, i.e.,

$$p_i^{\text{inter}} = p_i^{\text{mem}} + \gamma_{\max} \cdot n_i$$

is the point of intersection with the maximal parameter $\gamma_{\max}$. Notice that if p_i^{mem} lies outside the model, the outmost intersection lies inside the template and hence the maximal parameter $\gamma_{\max}$ is negative.

In situations where the geometric detail is finer than what can be captured with the template mesh resolution, alias effects occur. E.g. a vertex that is supposed to lie on the driver's side window may be projected onto the exterior mirror since this detail is further outside (cf. Fig. 10). To overcome this problem, we include the positions of neighboring vertices to detect if an intersection is reliable. The optimal position p_i^* of a non-feature vertex v_i is supposed to be some weighted average of the outmost mesh intersection p_i^{inter} and the membrane position p_i^{mem}. The larger the distance between the two positions, the less reliable is considered the intersection p_i^{inter}. To define

an appropriate weight, we compute

$$d_i = \min\left(\frac{\|p_i^{\text{inter}} - p_i^{\text{mem}}\|}{\delta \cdot e_{\min}}, 1\right),$$

which measures the ratio of the distance between p_i^{inter} and p_i^{mem} and the minimum length $e_{\min}$ of edges adjacent to v_i. Taking the minimum of this ratio and 1 establishes a smooth transition between p^{inter} and p^{mem} and at the same time discards outliers by clamping them to 1. In our experiments, we found $\delta \in [1, 3]$ to lead to the best results. The following transfer function finally maps vertex distances to weights $\omega \in [0, 1]$:

$$\begin{aligned} \omega_{\text{inter}}(v_i) &= 1 - d_i \\ \omega_{\text{mem}}(v_i) &= d_i \end{aligned} \tag{2}$$

Considering the membrane position as well as the mesh intersection additionally enables us to correctly handle vertices for which an intersection of their normal with the mesh cannot be computed due to gaps between neighboring surface patches. If there is no intersection, we simply use the membrane position only.

We formulate the computation of the final vertex positions as a weighted constrained least squares problem: The approximation component

$$W \cdot \left[\frac{L}{K}\right] \cdot X = \left[\frac{0}{Q}\right] \tag{3}$$

consist of two parts: Matrix L is a uniform Laplace system modeling the membrane positions while matrix K is generated by starting with an identity matrix and removing all rows corresponding to vertices for which an intersection cannot be computed. Finally, we multiply Eq. (3) with a diagonal matrix W containing the weights computed with Eq. (2) to account for the reliability of the surface intersections. The points of intersection are stored in the vector Q. We augment Eq. (3) with interpolation constraints

$$M \cdot X = P \tag{4}$$

given by the positions of the feature vertices P. Minimizing the sum of the errors in Eq. (3) subject to the interpolation constraints (4) using Lagrange multipliers (cf. Sec. 3.1) yields the final positions for all template vertices.

Notice that computing the outmost intersection of the vertex normal with the mesh is not always feasible. For template vertices of the body that are actually occluded by the roof or a wheelhouse, we cannot compute a meaningful new 3D position on the input model. To determine these vertices, we intersect the normals of all body vertices with the roof and both wheelhouses. If an intersection is detected they are excluded from the projection onto the input model and only their membrane position is used in the optimization.

4.2.2. Statistical Shape Model

The registration of a number of input models to the same template mesh establishes full vertex correspondence and hence enables us to construct a statistical shape model. We represent the geometry of each registered model by a shape vector $X_i = (x_0, y_0, z_0, \ldots, x_{n-1}, y_{n-1}, z_{n-1})^T \in \mathbb{R}^{3n}$ which contains the x-, y- and z-coordinates of the n vertices of the template mesh. Writing all k shapes as columns in a data matrix $X = [X_0, X_1, \ldots, X_{k-1}]$ and subtracting the mean shape $\overline{X}$ allows for the computation of a statistical shape model by applying Principal Component Analysis (PCA) to X. A new shape S can then be computed as

$$S = \overline{X} + \Phi\alpha \tag{5}$$

with $\Phi \in \mathbb{R}^{3n \times k-1}$ being the matrix of eigenvectors of the covariance matrix and $\alpha = (\alpha_0, \ldots \alpha_{k-2})^T$ being a vector of coefficients. Note that there are only $k-1$ meaningful eigenvectors since the mean was subtracted from the input shapes.

The benefit of a PCA shape model compared to simple affine combinations of the input shapes is that the PCA model allows for dimensionality reduction. Selecting the eigenvectors corresponding to the largest $l < k-1$ eigenvectors in Eq. (6) enables the user to explore the shape space containing the most important variances. This is especially useful when the number of input models becomes large.

Since we would like to enable the user to explore the shape space spanned by the input models in an intuitive way, we need to implement a suitable modeling metaphor. On the one hand, defining target positions by repositioning individual vertices of the shape S is too laborious. On the other hand, manipulating the coefficients α_i using, e.g., one slider per coefficient does not provide intuitive control. However, the feature vertices in a shape are related to feature curves. Since these curves express the main characteristics of a surface, we employ them as an intuitive modeling metaphor.

Every manipulation of a feature curve on the PCA model defines a set of target vertex positions $T \in \mathbb{R}^{3m}$ where $m < n$ is the number of feature vertices in the template mesh. To compute the shape S^* that best approximates the target positions, we compute the PCA coefficients α_i such that

$$\|T - B(\overline{X} + \Phi\alpha)\|^2 \rightarrow \min \tag{6}$$

is minimized. Matrix B eliminates all rows in $\overline{X}$ and Φ that do not represent a feature vertex. Since the number m of feature vertices is supposed to be larger than the number k of input shapes, the coefficients α are obtained by solving the overdetermined system (6) in the least squares sense. The final shape S^* is then computed using Eq. (5). To avoid degenerated configurations in which S^* does not lie within the shape space spanned by the input models, we clamp the α_i to the bounding box of the coefficients expressing the original input models in the PCA space. Hence, the α_i satisfy $\lambda_i^{\min} \leq \alpha_i \leq \lambda_i^{\max}$.

4.2.3. Results

We applied the registration to a set of complex 3D models of car bodies, which are triangle as well as general poly meshes that are non-manifold and lack a consistent normal orientation. Each model consist of 100 to 300 surface patches that may intersect and contain gaps between each other. The complexity ranges from 10k to 40k vertices. From each model we recovered a complete curve network, which took between 10 and 20 minutes per model. Fig. 11 shows some example models together with the recovered feature curves in the top rows as well as the result of the fully automatic registration in the bottom rows. The resolution of the template mesh was chosen as $n = 1614$ vertices in total.

In general, prior to defining a feature network the question about what a feature actually is, needs to be answered. Once the topology of the feature network is defined, every other geometric structure is no feature by definition. The pre-defined set of feature curves used in the registration is considered the lowest common denominator since we are always able to identify all curves on each model. However, some cars exhibit additional characteristic lines as, e.g., sharp edges on the engine hood or the trunk area. We are aware that these additional characteristics can by missed by our template mapping since we can only guarantee the proper alignment of those edges that are defined as features in the template. Increasing the set of feature curves would on the one hand allow for a more precise approximation of the input model but on

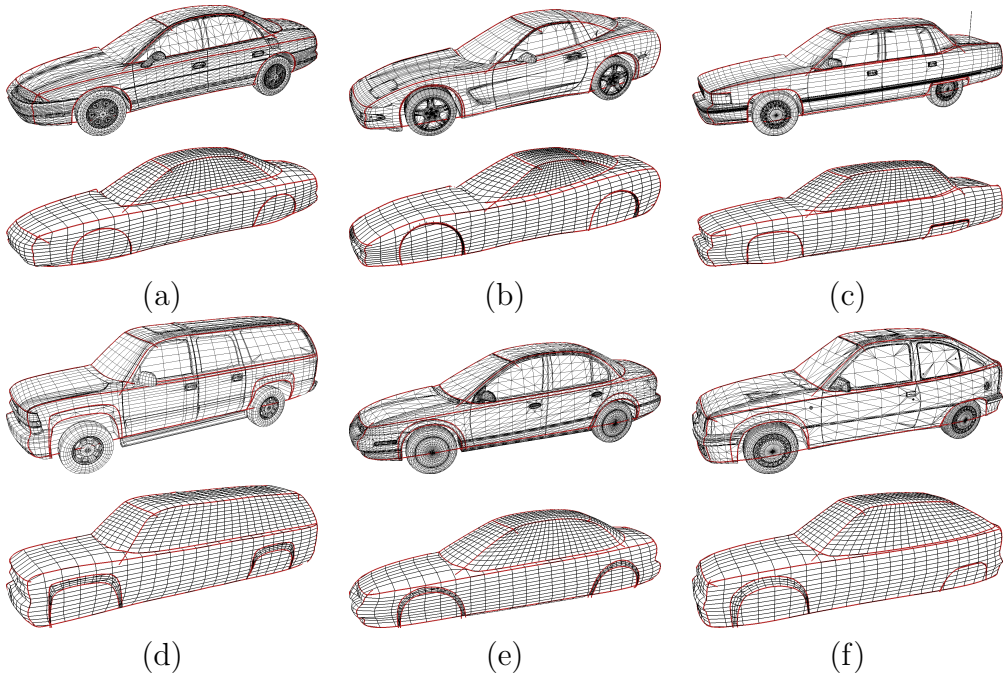


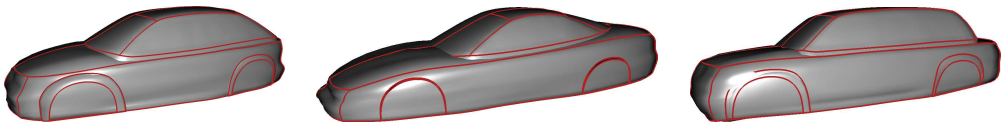
Figure 11: Top rows: Six example models of car bodies (driver's half) from which we recovered a set of feature curves. The recovery took between 10 and 20 minutes per model. Bottom rows: Results we obtained with our fully automatic registration method.

the other hand evoke the problem of how to correctly place the additional curves in cases when a car does not exhibit the additional features. To avoid ambiguity, we restricted ourselves to the minimal set of feature curves.

From a database of 13 registered car models we computed a statistical shape model and explored the shape space using the feature network. Fig. 12 depicts three exemplary results obtained with our system. Please refer to the accompanying video for an illustration of all registered input models as well as an exploration of the shape space.

5. Conclusions

We have presented an intuitive sketch-based user interface that allows for the recovery of feature curves from highly complex free-form surfaces. Using a fishbone structure, we reduce the task of reconstructing structural information on which optimization criteria can be evaluated from 3D surfaces to 2D curves. This enables a graph cut based optimization method that



(a) (b) (c)

Figure 12: New car models can be modeled intuitively by manipulating the feature network. Our system computes the best matching PCA coefficients in the least squares sense.

globally optimizes a feature curve w.r.t. several surface criteria, regardless of the complexity and consistency of the input model.

When a feature curve exhibits high geometric curvature its fishbone ribs may overlap. During optimization, the trajectories of the surface samples may hence cross each other and cause kinks and loops in the approximating curve. This problem could be solved by relaxing the orthogonality property of the ribs. However, since in our applications the goal was to recover curves that smoothly approximate the features in the input surface, this problem almost never arose.

We demonstrated the usefulness of our system by applying it to two practical problems. In the first setting, the recovered feature curves were used as modeling handles for real-time editing of highly complex input models. In our second scenario we created a statistical shape model for car bodies by recovering the same feature network from several input models. Since feature curves represent the essential characteristics of a surface the user can intuitively model new cars by manipulating the feature curves in 3D space.

In the future, we would like to extend our system such that it is able to recover feature curves from point clouds. This requires a manifold repair method that reconstructs fishbone ribs from points that lack any topology information. Furthermore, we would like to integrate additional optimization criteria into the graph-cut optimization. In the case that the input geometry exhibits some regularity in the mesh tessellation, an interesting criterion would enforce the feature curve to align to isolines contained in the input.

Acknowledgements

We would like to thank General Motors R&D for their support.

References

[1] A. P. Mangan, R. T. Whitaker, Partitioning 3d surface meshes using watershed segmentation, IEEE Transactions on Visualization and Computer Graphics 5 (4) (1999) 308–321.

[2] S. Katz, A. Tal, Hierarchical mesh decomposition using fuzzy clustering and cuts, in: Proc. ACM SIGGRAPH 2003, 2003, pp. 954–961.

[3] Y.-K. Lai, Q.-Y. Zhou, S.-M. Hu, R. R. Martin, Feature sensitive mesh segmentation, in: Proc. ACM symposium on Solid and physical modeling, 2006, pp. 17–25.

[4] S. Shlafman, A. Tal, S. Katz, Metamorphosis of polyhedral surfaces using decomposition, Comput. Graph. Forum 21 (3).

[5] Z. Ji, L. Liu, Z. Chen, G. Wang, Easy mesh cutting, Comput. Graph. Forum 25 (3) (2006) 283–291.

[6] Y.-K. Lai, S.-M. Hu, R. R. Martin, P. L. Rosin, Fast mesh segmentation using random walks, in: Proc. ACM symposium on Solid and physical modeling, 2008, pp. 183–191.

[7] M. Attene, B. Falcidieno, M. Spagnuolo, Hierarchical mesh segmentation based on fitting primitives, Vis. Comput. 22 (3) (2006) 181–193.

[8] J. Wu, L. Kobbelt, Structure recovery via hybrid variational surface approximation, Computer Graphics Forum 24 (3) (2005) 277–284.

[9] D. D. Hoffman, W. A. Richards, Parts of recognition, Cognition 18 (1985) 65–98.

[10] Y. Lee, S. Lee, Geometric snakes for triangular meshes, Comput. Graph. Forum 21 (3).

[11] Y. Lee, S. Lee, A. Shamir, D. Cohen-Or, H.-P. Seidel, Intelligent mesh scissoring using 3d snakes, in: Proc. 12th Pacific Conference on Computer Graphics and Applications, 2004, pp. 279–287.

[12] Y. Lee, S. Lee, A. Shamir, D. Cohen-Or, H.-P. Seidel, Mesh scissoring with minima rule and part salience, Comput. Aided Geom. Des. 22 (5) (2005) 444–465.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, 2nd Edition, The MIT Press, 2001.

[14] K. Hildebrandt, K. Polthier, M. Wardetzky, Smooth feature lines on surface meshes, in: Proc. Eurographics symposium on Geometry processing, 2005, pp. 85–90.

[15] Y. Ohtake, A. Belyaev, H.-P. Seidel, Ridge-valley lines on meshes via implicit surface fitting, in: Proc. ACM SIGGRAPH 2004, 2004, pp. 609–612.

[16] S. Yoshizawa, A. Belyaev, H.-P. Seidel, Fast and robust detection of crest lines on meshes, in: Proc. ACM symposium on Solid and physical modeling, 2005, pp. 227–232.

[17] A. Hubeli, M. Gross, Multiresolution feature extraction for unstructured meshes, in: Proc. Visualization 2001, 2001, pp. 287–294.

[18] T. Igarashi, S. Matsuoka, H. Tanaka, Teddy: a sketching interface for 3d freeform design, in: Proc. ACM SIGGRAPH 99, 1999, pp. 409–416.

[19] O. A. Karpenko, J. F. Hughes, Smoothsketch: 3d free-form shapes from complex sketches, in: Proc. ACM SIGGRAPH 2006, 2006, pp. 589–598.

[20] A. Nealen, T. Igarashi, O. Sorkine, M. Alexa, Fibermesh: designing freeform surfaces with 3d curves, in: Proc. ACM SIGGRAPH 2007, 2007, p. 41.

[21] J. Zimmermann, A. Nealen, M. Alexa, Silsketch: automated sketch-based editing of surface meshes, in: Proc. 4th Eurographics workshop on Sketch-based interfaces and modeling, 2007, pp. 23–30.

[22] L. Piegl, W. Tiller, The Nurbs Book, Springer, 1995.

[23] M. Botsch, L. Kobbelt, Resampling feature and blend regions in polygonal meshes for surface anti-aliasing, Computer Graphics Forum 20 (3) (2001) 402–410.

[24] S. Bischoff, L. Kobbelt, Structure preserving cad model repair, Comput. Graph. Forum 24 (3) (2005) 527–536.

[25] T. Ju, Fixing geometric errors on polygonal models: a survey, J. Comput. Sci. Technol. 24 (1) (2009) 19–29.

[26] M. Botsch, O. Sorkine, On linear variational surface deformation methods, IEEE Transactions on Visualization and Computer Graphics 14 (1) (2008) 213–230.

[27] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, H.-P. Seidel, Laplacian surface editing, in: Proc. Eurographics symposium on Geometry processing, 2004, pp. 175–184.

[28] L. B. Kara, K. Shimada, Supporting early styling design of automobiles using sketch-based 3d shape construction, Computer-Aided Design & Applications 5 (6) (2008) 867–876.

[29] I. Kókai, J. Finger, R. C. Smith, R. Pawlicki, T. Vetter, Example-based conceptual styling framework for automotive shapes, in: Proc. 4th Eurographics workshop on Sketch-based interfaces and modeling, 2007, pp. 37–44.

[30] R. Smith, R. Pawlicki, I. Kokai, J. Finger, T. Vetter, Navigating in a shape space of registered models, IEEE Transactions on Visualization and Computer Graphics 13 (6) (2007) 1552–1559.

[31] L. Kobbelt, S. Campagna, J. Vorsatz, H.-P. Seidel, Interactive multi-resolution modeling on arbitrary meshes, in: Proc. ACM SIGGRAPH 98, 1998, pp. 105–114.