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

Ellen Dekkers* Computer Graphics Group RWTH Aachen University

Leif Kobbelt† Computer Graphics Group RWTH Aachen University

Richard Pawlicki‡ RRP & Associates

Randall C. Smith§

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. 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 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 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.

CR Categories: I.3.5 [Computational Geometry and Object Modeling]: Geometric algorithms, languages, and systems

Keywords: Feature extraction, sketch-based interfaces, 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. This would either restrict the applicability of our method to surfaces that fulfill these assumptions or require a faithful and expensive mesh repair prior to processing them with our method. 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 which smoothly approximate the features of the input model. Due to the complexity of the input data, fully automatic approaches for feature recovery are doomed to failure. Furthermore, the aesthetic question of what a feature actually is and hence what a method should be able to recover cannot be modeled mathematically. However, our system provides maximal user-support in recovering what he consideres a feature. Finally, the extraction of the same set of feature curves from several input models allows for the computation of a statistical shape model. The shape space spanned by the input models can then be explored intuitively using the feature curves.

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 subparts which is a key issue for many further applications on meshes such as parameterization, 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 refine the borders separating them. There exists a variety of approaches in the literature which make use of watershed segmentation [Mangan and Whitaker 1999], clustering [Katz and Tal 2003], [Lai et al. 2006], [Shlafman et al. 2002], region growing [Ji et al. 2006], or random walks [Lai et al. 2008] to group similar mesh elements into meaningful regions. Fitting of geometric primitives (e.g. [Attene et al. 2006], [Wu and Kobbelt 2005]) 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 [Hoffman and Richards 1985], 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 [Lee and Lee 2002], [Lee et al. 2004], [Lee et al. 2005] 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 [Katz and Tal 2003] graph cuts (e.g. [Cormen et al. 2001]) were used to automatically define segment boundaries. However, they apply the graph cut to refine an initial segment boundary within a transition region derived from fuzzy clustering.

*e-mail: dekkers@cs.rwth-aachen.de
†e-mail: kobbelt@cs.rwth-aachen.de
‡e-mail: richard.pawlicki@gmail.com
§e-mail: 1randall.c.smith@gmail.com

Approaches which automatically recover global features from surface meshes [Hildebrandt et al. 2005], [Ohtake et al. 2004], [Yoshizawa et al. 2005] require the computation of higher-order surface derivatives thus imposing limitations (2-manifoldness) on the type of meshes which can be processed. Furthermore, careful parameter tuning may be necessary since automatic feature extraction is based on heuristics which might recover insignificant parts. [Hubeli and Gross 2001] 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. [Igarashi et al. 1999], [Karpenko and Hughes 2006], [Nealen et al. 2007], [Zimmermann et al. 2007]). Since human perception recognizes the shape of an object by its main characteristics, [Nealen et al. 2007] consider the sketched curves as features of the resulting model and further exploit them for surface modeling.

The incorporation of user assistance into the task of mesh segmentation greatly supports the detection of salient features which are consistent with human shape perception. [Ji et al. 2006] 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 dragging them over the surface, inserting and deleting vertices and providing freehand strokes for the replacement of boundary segments. Especially the last metaphor greatly captures human intuition.

2 Overview

Unlike the methods described in the last section, the input to our method can be arbitrary without imposing any consistency requirements. 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. Since general mesh repair is too expensive, we reconstruct the missing structural information locally. By the use of a "fishbone structure" (cf. Sec. 4.3.1) we are able to reduce the reconstruction problem from 3D surfaces to two 2D curves.

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. Since the input model is possibly non-manifold, the definition of a patch neighboorhood is ambiguous and hence we cannot compute geodesic paths on the input model. Instead, the local structural information provided by the fishbone is used for curve optimization on the surface.

Most existing methods align the 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.

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. A graph-cut based algorithm optimizes a curve subject to various surface properties thereby guaranteeing to find the global optimum. We demonstrate the usefulness of our system by applying it to the problem of generating a statistical (PCA) shape model for car bodies, a class of models which exhibit large variance in shape. By recovering the same set of feature curves from each model, we are able to establish proper feature correspondences. The registration of a template mesh to each input model is guided by the feature network and finally allows us to generate the morphable shape model. We further exploit the feature curves as an intuitive modeling metaphor for exploring the shape space spanned by the input models.

3 User Interaction

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 the system automatically computes an initial curve. Our system supports the user in the task of recovering features from the input model by automatically "snapping" the sketch curve to regions of the most likely feature location. A graph cut based optimization process shifts the curve to regions of maximum curvature or certain surface normal orientations, thereby satisfying smoothness requirements. The user can adjust the relative weighting of the optimization objectives using a slider and scale the stiffness of the feature curve.

In case the user is not yet satisfied with the optimized curve, he can impose positional constraints by pulling the curve over the input surface. When he clicks on a curve and moves the mouse, the displacement is computed in screen space. The curve's new 3D position is obtained by reprojecting the screen positions back onto the surface. 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, the computation of geodesic paths on meshes exhibiting the characteristics described above is not straightforward. In contrast, reprojecting screen positions back onto the mesh is always possible regardless of any mesh inconsistencies.

The optimization criteria as well as positional constraints for selected points on the curve can be modified by the user interactively. The feature curve then automatically adopts itself to the new constraints in real time providing direct feedback for the user. 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.

Features are characterteristic curves which describe the essential properties of a surface. Once these curves are recovered, they can be manipulated and hence allow for interactive surface modeling (cf. Sec. 5).

Please refer to the accompanying video for a demonstration of our interface and an example on surface modeling using feature curves.

4 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.

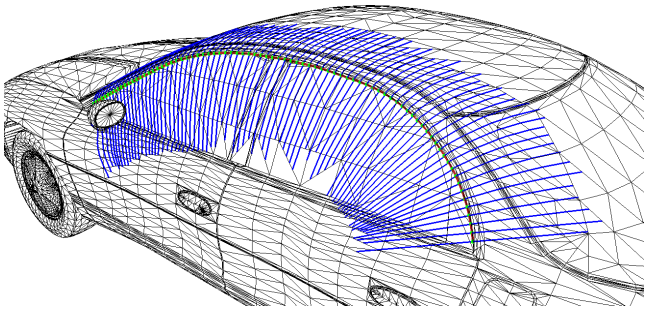


Figure 1: *A fishbone structure on a model of a car: For each surface sample (green dots) a manifold rip polygon (blue) is constructed which is orthogonal to the approximating curve (red).*

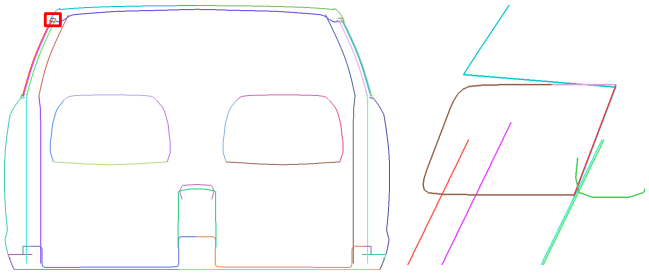


Figure 2: *A slice through an input model. Surface patches intersect arbitrarily, hence a purely geometric repair algorithm is prone to failure. The image on the right shows a close-up of the area enclosed by the red box in the left image.*

4.1 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 parameterization of the samples s_i as well as a knotvector $U = \{u_0, \ldots, u_{n+p+1}\}$ that respects the distribution of the parameters [Piegl and Tiller 1995]. 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 the sample positions one wants 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 [Piegl and Tiller 1995] and improve the fit using classical parameter correction [Hoschek 1988]. Note, that since we compute an approximation of the s_i, the resulting curve is not exactly embedded and the $c(t_i)$ do not necessarily lie on the surface, except for the interpolation constraints.

4.2 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. To allow for intuitive modeling, we compute the displacement in screen space. We therefore project s_c into the image plane and translate it 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.

4.3 Feature Curve Optimization

In order to treat all different types of surface quality constraints in a uniform manner, we do not optimize a curve $c(t)$ itself on the surface. Instead, we optimize the positions of the surface samples s_i which are then approximated by $c(t)$. Allowing the samples to move freely on the surface during optimization would lead to clustering in surface areas which best meet the optimization criteria. We therefore restrict their trajectories to paths perpendicular to the approximating curve. On the one hand, this ensures that surface samples maintain a certain distance to each other and hence the generation of clusters is avoided. 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.

4.3.1 Fishbone Structure

We use a fishbone structure as proposed by [Botsch and Kobbelt 2001], to define the trajectories along which the surface samples s_i are allowed to move during the curve optimization. The feature curve $c(t)$ thereby forms the backbone while rib curves are created as follows (cf. Fig. 1). Since $c(t)$ approximates a set of m surface samples $S = \{s_0, ..., s_{m-1}\}$, there exists a curve parameter 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 contour polygon. In the following section, we describe our rasterization-based algorithm for 2D manifold repair in detail. A complete fishbone structure is finally obtained by reconstruction a manifold contour polygon for each rib. Fig. 1 shows a fishbone for a curve on a car's roof.

Notice that the planarity property of the fishbone ribs allows us to perform all following computations in a 2D plane which renders the task of creating manifolds considerably simpler compared to 3D mesh repair (see e.g. [Bischoff et al. 2005]). To accelerate the construction of the polygons, we reduce the search space by discarding all intersections of a rib plane with the input model that do not lie within a circle with a predefined radius around the backbone. Rejecting intersections outside a certain region is feasible since we assume the curve to be sketched in a region of interest and hence it is not supposed to change its location by more than a certain distance.

4.3.2 Rasterization-based manifold reconstruction

Intersecting the unstructured input model with a plane first creates an unordered set of edges. Since we do not require the input mesh to be watertight, it may consist of several unconnected surface patches that intersect arbitrarily and hence a manifold polygon representing the contour cannot be reconstructed in a simple mesh traversal. However, we can combine the unordered edge set to a set of unconnected polygonal segments by traversing each intersected surface patch separately and connecting its intersection edges to a manifold polygonal segment. What remains is the connection of the resulting

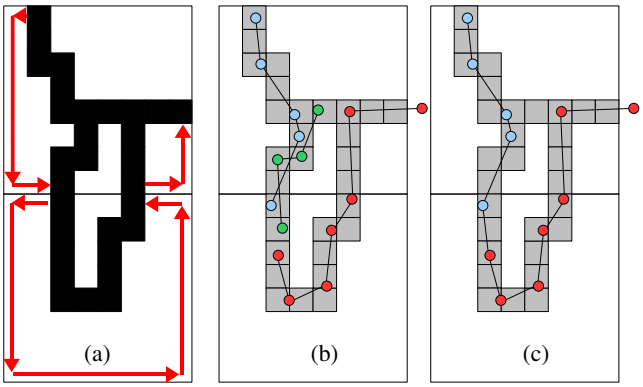


Figure 3: *Manifold extraction: Our algorithm gains topology information for the reconstruction of a contour in an image traversal (a). Identifying the correct intersections of grid edges with polygon edges (b) allows for the construction of a manifold contour polygon (c) from a set of arbitrarily intersecting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

segments to a single manifold contour polygon that interpolates the outer silhouette. In a simple and purely geometric approach, polygonal segments are merged to a manifold contour by concatenating them at coinciding endpoints. However, since surface patches may intersect arbitrarily, this also holds for the segments (cf. Fig. 2). Hence, the segments need to be pairwise checked for intersection and those parts constituting the outer contour need to be identified and connected. Unfortunately, this approach may fail since 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 the purely geometric approach in the case that no intersections are detected.

We propose a rasterization-based approach which 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 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 [Bischoff and Kobbelt 2005] 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 walk around the outside of the contour to be reconstructed. To accelerate the computation we create a uniform axes-aligned grid that partitions the image into quadrangular cells. For each cell $c_{i,j}$ we store the end points of all polygon edges rendered into $c_{i,j}$. Furthermore, we store those polygon edges that intersect one of the cell's four grid edges, together with the point of intersection. Our extraction algorithm then identifies all cells containing geometry information and forming the contour, by walking along the grid edges in counterclockwise direction. 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. This pixel indicates where we enter the upper cell. 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 cell (bottom edge). Simultaneously, it indicates at which grid edge we enter the neighboring cell (top edge). Hence, after examining cell $c_{i,j}$ we continue with one of the four neighboring cells $c_{i\pm1,j}$ or $c_{i,j\pm1}$ depending on the grid edge where $c_{i,j}$ is left.

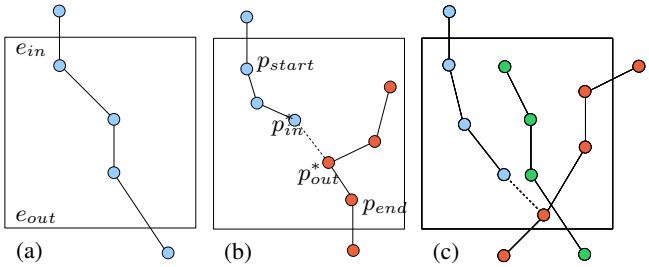


Figure 4: *Different cases occurring in a single cell during the contour reconstruction: (a) One segment intersects the cell. (b) Two segments intersecting the cell are connected to a single manifold segment. (c) Ambiguous case: Grid edges are intersected multiple times.*

The image traversal serves as topological guide for the construction of a manifold contour polygon. We add the points within each visited cell to the manifold polygon depending on the cell characteristics depicted in Fig. 4. A cell can be intersected by one 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} and leave it on the bottom edge e_{out} as illustrated in Fig. 4. The top edge e_{in} is intersected by a segment S_{in} while e_{out} is intersected by a segment S_{out}. In the case that $S_{in} = S_{out}$ (a) we simply add all points $p_i \in S_{in}$ within the cell to the final manifold polygon. In the case that $S_{in} \neq S_{out}$ (b) we compute two point sets $P_{in} \subset S_{in}$ containing all points within the cell that belong to segment S_{in} and $P_{out} \subset S_{out}$ containing all points that belong to segment S_{out}. We then compute the pair of points $p_{in}^* \in P_{in}$ and $p_{out}^* \in P_{out}$ having the minimum distance

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

We add the point set $\{p \in P_{in} | p_{start} \leq p \leq p_{in}^*\}$, i.e., all points between p_{start} and p_{in}^* on segment S_{in}, and then add the point set $\{p \in P_{out} | p_{out}^* \leq p \leq p_{end}\}$, 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. There are multiple intersections of the bottom edge with polygonal segments but our goal is the reconstruction of the outer contour. Hence, 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 then kept as incoming intersection for the neighboring cell and hence does not need to be recomputed.

Our method allows for the extraction of a globally manifold contour polygon from an arbitrary complex set of polygonal segments. Since we always generate a closed contour, it happens that if the input is not closed, we generate the inside of the input as well. However, since the manifold 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.

4.3.3 Graph-Cut Optimization

A complete fishbone with one manifold rib polygon per surface sample establishes a locally regular structure in highly unstructured surface areas which 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. Otherwise, the approximation of a noisy set of samples would result in a curve which suffers from high curvature or approximation errors.

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 (cf. Fig. 5(a)). By sampling each rib, we construct a planar quad-regular graph structure $G = (V, E)$ with one column per rib (cf. Fig. 5(b)). Notice that the edges E, not the vertices of column i in G represent the samples $b_{i,j}$ on rib i. Hence, having k samples on rib i, there are k edges and $k + 1$ vertices in column i of G. 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*. Capacities are assigned to rib edges by evaluating the following external optimization criteria at the corresponding rib samples $b_{i,j}$.

We introduce two external criteria which model the optimal surface requirements. 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 rip edges $e_0 = (b_{i,j-1}, b_{i,j})$ and $e_1 = (b_{i,j}, b_{i,j+1})$:

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

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

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

with β being the desired surface normal angle and a being 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 dependend and the external criteria presented here are examples. It is straightforward to integrate any other forces which can be evaluated on fishbone rips.

The capacities assigned to edges along the ribs model external forces that shift the samples to those positions on the surface which best meet the optimization criteria. However, optimizing the sample positions subject to these forces only, without imposing any internal smoothness requirements, may 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 (horizontal edges in Fig. 5(b)). This capacity θ is set to be always greater than all capacities associated to rib edges:

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

This ensures that in each column of G (along a rib) exactly one vertical 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

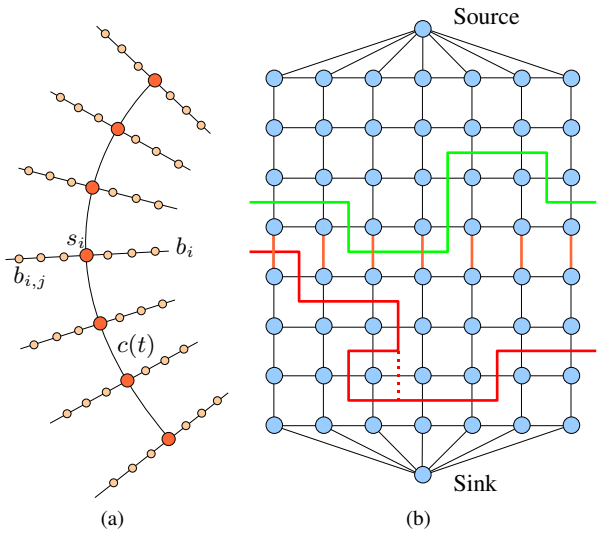


Figure 5: *Fishbone structure: (a) The feature curve $c(t)$ forms the backbone of a fishbone structure with one orthogonal rib b_i per surface sample s_i. The optimal positions of the surface samples are obtained by embedding the fishbone into a planar graph structure (b) and computing the minimum cut. 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).*

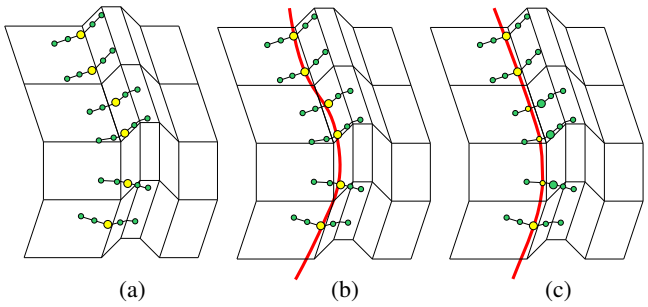


Figure 6: *Geometric detail in the input model may induce the projection of surface samples (large dots) to both sides of a sharp feature and hence results in a zig-zag shape of the backbone samples (a). Constructing the fishbone and running the graph cut optimization without synchronizing the ribs results 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).*

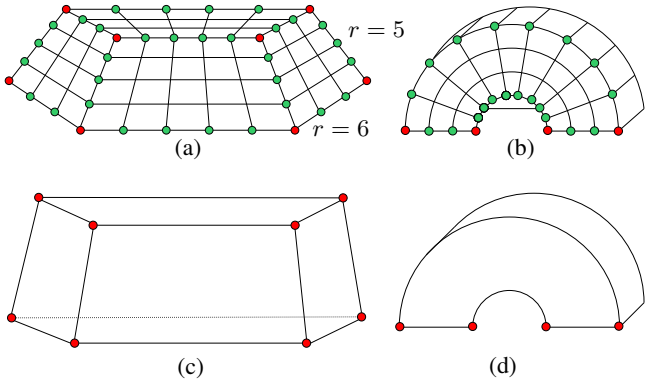


Figure 7: *Template mesh and associated curve network: (a) Roof/body mesh, (b) wheelhouse mesh, (c) roof/body network, (d) wheelhouse network.*

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 red line. The user can modify the stiffness of the resulting curve by adjusting a stiffness factor $f \geq 1$ which is used for scaling 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 $b_{c,j}$ which 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. Computing the minimal cut through G results in a set of graph edges that are cut. The quad-structure of G ensures that each column of G 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 sample s_i. Since the rib samples were created by sampling rib polygons which may span over gaps in the input model, they do not necessarily lie on the surface. We therefore reproject the b_i^* onto the input model resulting in the final optimal surface samples $S^* = \{s_0^*, ..., s_{m-1}^*\}$. Re-computing a least squares approximation finally yields the optimized feature curve $c(t)$ in demand.

A special case that needs to be taken into account in the curve optimization is posed by geometric detail contained in the input model. The projection of an initial sketch curve (cf. Sec. 4.1) may produce surface samples which are located on different sides of a geometric detail. Hence, we need to compute an index offset to "synchronize" neighboring ribs prior to the computation of the graph cut embedding. Consider Fig. 6 for an illustration. The surface samples are projected onto different sides of a sharp feature (a). Regardless of the lack of smoothness, the graph cut embedding would encode the surface samples 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 which approximates these samples suffers from high curvature as illustrated in Fig. 6 (b). To overcome this problem, we shift the ribs in the following way: Firstly, we sample each rib as usual. 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 $i + 1$ which has minimum Euclidean distance to $\tilde{s}_i$ and define it as the new backbone sample $\tilde{s}_{i+1}$. The number of samples inbetween 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 such that the backbone is guaranteed to be smooth. Running the graph cut optimization on the shifted fishbone structure reduces the zig-zag shape of the sample polyline to a minimum and finally produces a smoother feature curve (cf. Fig. 6 (c)). Hence, the rib synchronization establishes a local structure which allows us to execute the graph cut optimization on arbitrary geometry.

5 Application: A Morphable Shape Model for Cars

We demonstrate the usefulness of our sketch-based interface by applying it to the recovery of feature curves from a set of car body models. Having extracted 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 template finally allows for the construction of a statistical (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 which are closely related to our work. [Kara and Shimada 2008] align a 3D template model to 2D sketches. The aligned model is then refined by tracing a car's characteristic lines on the sketch. [Kókai et al. 2007] 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 [Smith et al. 2007] 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 spanned by the models by dragging single feature points or manipulating shape space parameters.

5.1 Registration

A car has a predefined coordinate frame where the origin is located in the middle of the front wheel axle, the x-axis points in direction of the trunk, the y-axis at the front-passenger's side and the z-axis upwards. To reduce distortion in the later PCA model, we segment a car into four components: the roof, the body and front and rear wheelhouse. We thereby employ the symmetry w.r.t. the x-z plane and register the driver's left half only, while the remaining 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. 7. We register an input model in a two-step procedure.

5.1.1 Network construction

In the feature curves recovered by the user, the endpoints of a feature are not placed precisely. Hence, we have to compute the feature endpoints by pairwise intersecting the input curves yielding the red nodes in the curve network depicted in Fig. 7 (c) and (d). If three feature curves do not intersect in a common point, we compute the average of the three pairwise intersection points.

5.1.2 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.

There are two types of vertices in a template mesh: feature vertices and non-feature vertices. 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.

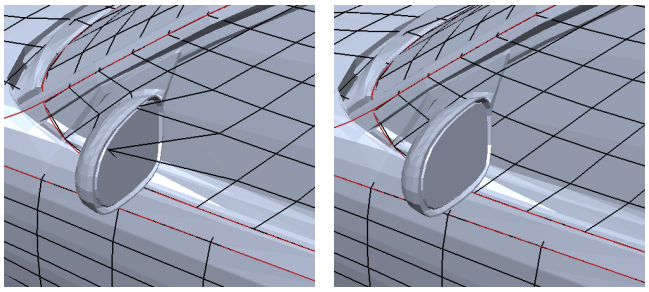
Our system constructs a template mesh on-the-fly letting the user define its resolution. Note that the computation of a PCA shape model requires a common resolution of all registered models. Fig. 7 (a) shows an exemplary template mesh where the roof is a topological cube with four faces while the body component contains an additional face on the bottom. The wheelhouse component is depicted in (b). Each template component contains feature vertices (highlighted in red) which directly correspond to the red intersections of feature curves in Fig. (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.

The curve samples represent the final coordinates of all template feature vertices. The final positions of the non-feature vertices are computed in two steps: 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 [Kobbelt et al. 1998]) 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.

However, computing the outmost intersection may lead to erroneous results as well. 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 which 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. 8). 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



(a) (b)

Figure 8: *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.*

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

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. We take the minimum of this ratio and 1 to establish a smooth transition between p^{inter} and p^{mem} and at the same time discarding 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 of the following form: The approximation component

$$W \cdot \begin{bmatrix} L \\ K \end{bmatrix} \cdot X = \begin{bmatrix} 0 \\ Q \end{bmatrix} \tag{3}$$

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. We minimize the sum of the errors in Eq. (3) subject to the interpolation constraints (4) using Lagrange multipliers (cf. Sec. 4.1) which 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 car body which are actually occluded by either the roof or one of the wheelhouses, 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 as well as with both wheelhouses. All vertices for which an intersection of their normal with one of the other components was detected are then excluded from the projection onto the input model. Instead, we only use their membrane position in the optimization.

5.2 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 PCA shape S are related to feature curves which were recovered from the input models and hence establish proper feature correspondence. Since feature curves express the essential characteristics of a surface, we employ the curves 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^* which 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 which 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 require the α_i to lie within the bounding box spanned by the coefficients α_i which express the original input models in the PCA space and hence to satisfy $\lambda_i^{\min} \leq \alpha_i \leq \lambda_i^{\max}$.

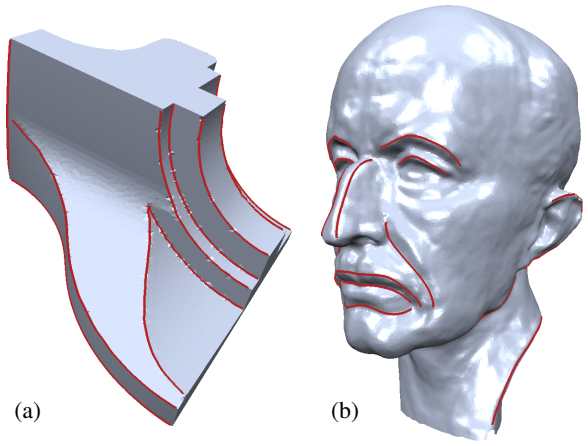


Figure 9: *Feature curves recovered from (a) a noisy fandisk model and (b) the Max Planck head model.*

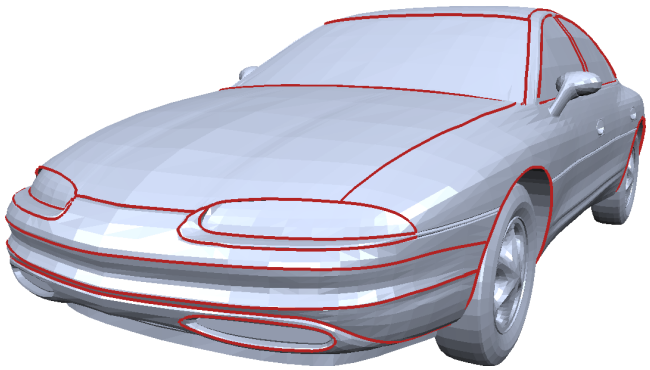


Figure 10: *Sharp features as well as features with varying intensity are recovered from a complex car body model.*

6 Results

We used our sketch-based interface to recover feature curves from a variety of input models. Depending on the class of models, the definition of a feature varies. Feature extraction from manifold models as depicted in Fig. 9 which may even exhibit sharp features as the fandisk in (a) is a simpler task. All features are easy to identify due to their high curvature. Although these meshes could be processed with automatic approaches as well, we present them for the sake of completeness. In contrast, automatic approaches fail to detect features which largely vary in their intensity or which may even vanish in some surface regions. Furthermore, the lack of any constraints on the input data renders the evaluation of surface criteria impossible without any prior mesh repair.

Our system locally reconstructs structural information which allows us to extract features from highly complex input data. The costs of the initial computation of the fishbone depend on the number of rips which need to be reconstructed using the rasterization-based approach (cf. Sec. 4.3.2) in case that the simple geometric stitching approach fails due to the inconsistency of the input model. In the

case that the geometric approach succeeds, the complete 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).

From the car model in Fig. 10, we successfully recovered sharp features as well as feature curves which vanish in some regions (e.g. where the bumper merges with the area around the front wheelhouse). The recovery of the complete set of feature curves took approx. 10 minutes.

In our application scenario, we recovered smooth feature curves from complex 3D models of car bodies. The models are triangle as well as general poly meshes which are non-manifold and lack a consistent normal orientation. Each model consist of 100 to 300 surface patches which 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 exemplary 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 of car bodies is considered the lowest common denominator since we are always able to identify all of the curves on each car model. However, some cars exhibit more characteristic lines especially on the front of the car or 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 the other hand evoke the problem of how to correctly place the additional curves in cases when a car body 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 which allows for intuitive exploration of 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 exemplary exploration of the shape space.

7 Conclusions

We have presented an intuitive sketch-based user interface which 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 which globally optimizes a feature curve w.r.t. several surface criteria, regardless of the complexity and consistency of the input model.

We demonstrated the usefulness of our method by applying it to the practical problem of the registration of car body models. Recovering the same network of feature curves from several input models allows for the construction of a PCA shape model. 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.

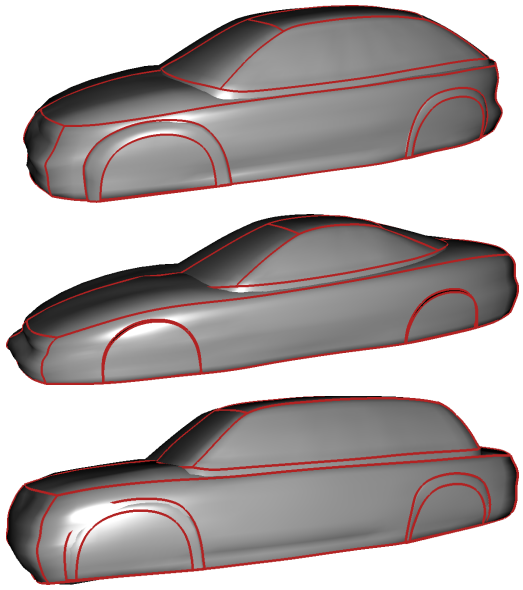


Figure 12: *New car models can be modeled intuitively by manipulating the feature network. Our system computes the best matching PCA coefficients in the least squares sense.*

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 which reconstructs fishbone ribs from points which 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 geometry.

Acknowledgements

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

References

ATTENE, M., FALCIDIENO, B., AND SPAGNUOLO, M. 2006. Hierarchical mesh segmentation based on fitting primitives. *Vis. Comput. 22*, 3, 181–193.

BISCHOFF, S., AND KOBBELT, L. 2005. Structure preserving cad model repair. *Comput. Graph. Forum 24*, 3, 527–536.

BISCHOFF, S., PAVIC, D., AND KOBBELT, L. 2005. Automatic restoration of polygon models. *ACM Trans. Graph. 24*, 4, 1332–1352.

BOTSCH, M., AND KOBBELT, L. 2001. Resampling feature and blend regions in polygonal meshes for surface anti-aliasing. *Computer Graphics Forum 20*, 3, 402–410.

CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., AND STEIN, C. 2001. *Introduction to Algorithms*, 2nd ed. The MIT Press.

HILDEBRANDT, K., POLTHIER, K., AND WARDETZKY, M. 2005. Smooth feature lines on surface meshes. In *Proc. Eurographics symposium on Geometry processing*, 85–90.

HOFFMAN, D. D., AND RICHARDS, W. A. 1985. Parts of recognition. *Cognition 18*, 65–98.

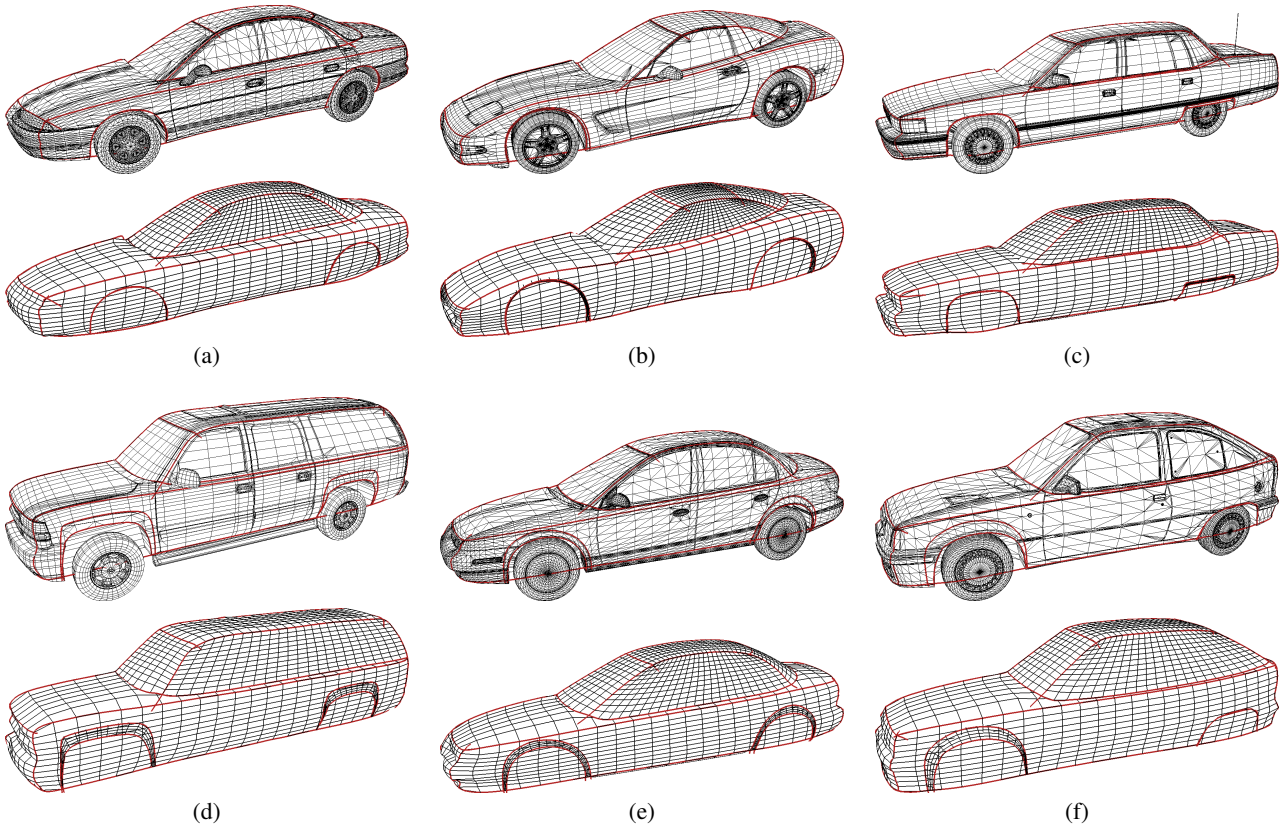


Figure 11: *Six exemplary models of car bodies (driver's half) from which we recovered a set of feature curves. The recovery took between 10 and 20 minutes per model. The lower images depict the results obtained with our fully automatic registration method.*

HOSCHEK, J. 1988. Intrinsic parametrization for approximation. *Comput. Aided Geom. Des. 5*, 1, 27–31.

HUBELI, A., AND GROSS, M. 2001. Multiresolution feature extraction for unstructured meshes. In *Proc. Visualization 2001*, 287–294.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: a sketching interface for 3d freeform design. In *Proc. ACM SIGGRAPH 99*, 409–416.

JI, Z., LIU, L., CHEN, Z., AND WANG, G. 2006. Easy mesh cutting. *Comput. Graph. Forum 25*, 3, 283–291.

KARA, L. B., AND SHIMADA, K. 2008. Supporting early styling design of automobiles using sketch-based 3d shape construction. *Computer-Aided Design & Applications 5*, 6, 867–876.

KARPENKO, O. A., AND HUGHES, J. F. 2006. Smoothsketch: 3d free-form shapes from complex sketches. In *Proc. ACM SIGGRAPH 2006*, 589–598.

KATZ, S., AND TAL, A. 2003. Hierarchical mesh decomposition using fuzzy clustering and cuts. In *Proc. ACM SIGGRAPH 2003*, 954–961.

KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. 1998. Interactive multi-resolution modeling on arbitrary meshes. In *Proc. ACM SIGGRAPH 98*, 105–114.

KÓKAI, I., FINGER, J., SMITH, R. C., PAWLICKI, R., AND VETTER, T. 2007. Example-based conceptual styling framework for automotive shapes. In *Proc. 4th Eurographics workshop on Sketch-based interfaces and modeling*, 37–44.

LAI, Y.-K., ZHOU, Q.-Y., HU, S.-M., AND MARTIN, R. R. 2006. Feature sensitive mesh segmentation. In *Proc. ACM symposium on Solid and physical modeling*, 17–25.

LAI, Y.-K., HU, S.-M., MARTIN, R. R., AND ROSIN, P. L. 2008. Fast mesh segmentation using random walks. In *Proc. ACM symposium on Solid and physical modeling*, 183–191.

LEE, Y., AND LEE, S. 2002. Geometric snakes for triangular meshes. *Comput. Graph. Forum 21*, 3.

LEE, Y., LEE, S., SHAMIR, A., COHEN-OR, D., AND SEIDEL, H.-P. 2004. Intelligent mesh scissoring using 3d snakes. In *Proc. 12th Pacific Conference on Computer Graphics and Applications*, 279–287.

LEE, Y., LEE, S., SHAMIR, A., COHEN-OR, D., AND SEIDEL, H.-P. 2005. Mesh scissoring with minima rule and part salience. *Comput. Aided Geom. Des. 22*, 5, 444–465.

MANGAN, A. P., AND WHITAKER, R. T. 1999. Partitioning 3d surface meshes using watershed segmentation. *IEEE Transactions on Visualization and Computer Graphics 5*, 4, 308–321.

NEALEN, A., IGARASHI, T., SORKINE, O., AND ALEXA, M. 2007. Fibermesh: designing freeform surfaces with 3d curves. In *Proc. ACM SIGGRAPH 2007*, 41.

OHTAKE, Y., BELYAEV, A., AND SEIDEL, H.-P. 2004. Ridge-valley lines on meshes via implicit surface fitting. In *Proc. ACM SIGGRAPH 2004*, 609–612.

PIEGL, L., AND TILLER, W. 1995. *The Nurbs Book*. Springer.

SHLAFMAN, S., TAL, A., AND KATZ, S. 2002. Metamorphosis of polyhedral surfaces using decomposition. *Comput. Graph. Forum 21*, 3.

SMITH, R., PAWLICKI, R., KOKAI, I., FINGER, J., AND VETTER, T. 2007. Navigating in a shape space of registered models. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (Nov.-Dec.), 1552–1559.

WU, J., AND KOBBELT, L. 2005. Structure recovery via hybrid variational surface approximation. *Computer Graphics Forum 24*, 3 (Sept.), 277–284.

YOSHIZAWA, S., BELYAEV, A., AND SEIDEL, H.-P. 2005. Fast and robust detection of crest lines on meshes. In *Proc. ACM symposium on Solid and physical modeling*, 227–232.

ZIMMERMANN, J., NEALEN, A., AND ALEXA, M. 2007. Silsketch: automated sketch-based editing of surface meshes. In *Proc. 4th Eurographics workshop on Sketch-based interfaces and modeling*, 23–30.