EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling (2007)
M. van de Panne, E. Saund (Editors)

SilSketch: Automated Sketch-Based Editing of Surface Meshes

Johannes Zimmermann Andrew Nealen Marc Alexa

TU Berlin

Abstract

We introduce an over-sketching interface for feature-preserving surface mesh editing. The user sketches a stroke that is the suggested position of part of a silhouette of the displayed surface. The system then segments all image-space silhouettes of the projected surface, identifies among all silhouette segments the best matching part, derives vertices in the surface mesh corresponding to the silhouette part, selects a sub-region of the mesh to be modified, and feeds appropriately modified vertex positions together with the sub-mesh into a mesh deformation tool. The overall algorithm has been designed to enable interactive modification of the surface – yielding a surface editing system that comes close to the experience of sketching 3D models on paper.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling - Modeling packages; I.3.6 [Methodology and Techniques]: Interaction techniques; General Terms: Sketch Based Modeling, Deformations, Laplacian Surface Editing, Differential Geometry, Sketching

1. Introduction

The process of generating 3D shapes in engineering or content creation typically goes through several *design reviews*: renderings of the shapes are viewed on paper or a screen, and designers indicate necessary changes. Oftentimes designers sketch replacements of feature lines onto the rendering. This information is then taken as the basis of the next cycle of modifications to the shape.

We present a surface mesh editing system motivated by design reviews: given nothing but the over-sketch of a feature line, it automatically deforms the mesh geometry to accommodate the indicated modification. Building on existing mesh deformation tools [SLCO*04, NSACO05], the main feature of our work is the *automatic* derivation of all necessary parameters that these systems require as input in *real-time*.

In particular, Laplacian Surface Editing [SLCO*04], but also most other recent mesh deformation techniques (e.g., [YZX*04, BPG06]) require the selection of: handle vertices, the displacement for these handle vertices and a region of interest (ROI), representing the part of the mesh to be modified to accommodate the displaced handle vertices. For our system, we need to compute this information from the over-sketched feature line alone; and we do this in fractions of a second. The steps described below comprise our system (see also Fig. 1) – breaking down the problem into these steps and performing each step in few milliseconds are the main contributions of our work:

1. Based on the screen projection of the shape, a subset of pixels lying on potential feature lines is identified. These pixels are then segmented and converted to image-space polylines as the set of candidate feature lines.
2. The user-sketch is matched against all polylines to find the corresponding part on a feature line.
3. Based on the correspondence in image-space, a set of handle vertices in the surface mesh is selected. The image-space projection of these vertices covers the detected part of the feature line.
4. New positions for the handle vertices are derived from the displacements in image-space between the projection of the handle vertices and the user's sketch; these are the necessary displacements.
5. A part of the surface mesh around the handle vertices, computed by region growing, is defined as the ROI.

Note that in steps 3,4, and 5 we compute the necessary input for shape deformation, while steps 1 and 2 are required to identify the input, based only on the user-sketch.

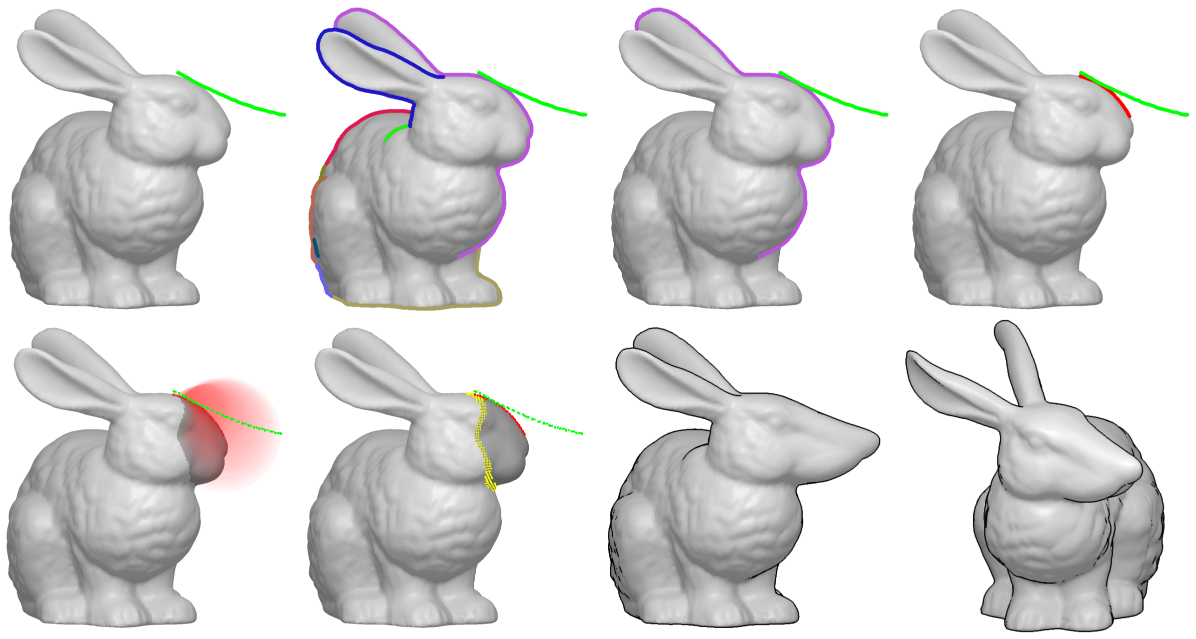

Figure 1: *Algorithm pipeline. Top row, from left to right: a) user-sketch, b) image-space silhouettes, c) retained silhouettes after proximity culling, d) handle estimation; Bottom row, left to right: e) correspondences and ROI estimation by bounding volumes, f) setup for Laplacian Surface Editing, g) and h) deformation result. Note that the user only sees a), g) and h).*

2. Related Work and System Design

Sketch-based interfaces are a very popular method for creation and deformation of 3D surface meshes [IMT99, KSvdP07, KS07]. Deriving the parameters for mesh deformation from sketches only is not new: Kho and Garland [KG05] derive ROI and handle vertices from sketching onto the projected shape, essentially implying a skeleton for a cylindrical part. A second stroke then suggests a modification of the skeleton, and the shape is deformed according to the deformed skeleton. However, according to Hoffman and Singh [HS97], we recognize objects mainly by a few feature lines, namely silhouettes and concave creases. Since the process of paper-based sketching relies exactly on these features, we feel it is more natural to use them as the basis for our over-sketching mesh deformation tool. This line of thought is similar to Nealen et al. [NSACO05]. They have enhanced Laplacian Surface Editing techniques to work in the setting of prescribing new silhouettes. In particular, this requires positional constraints defined on mesh edges and finding the correspondence between a pre-selected silhouette of the mesh and the over-sketched silhouette. In their system the user manually selects the ROI and a part of one of the silhouettes as a pre-process. In our system, all these manual selections are now automated; the user only provides a single stroke, from which handle and ROI are estimated (Figs. 1 and 2).

We have also observed that computing silhouettes from the mesh representation (i.e. in object-space) has problems:

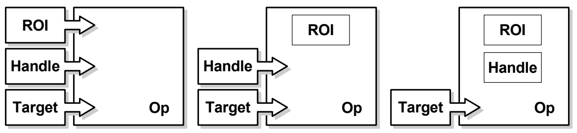

Figure 2: *Required user interaction (from left to right): Nealen et al. [NSACO05], Kho and Garland [KG05], and our approach .*

the silhouette path on the mesh might fold onto itself when projected to image-space, i.e. a point of the silhouette in image-space could map to several pieces of the silhouette on the mesh. As a result, the mapping from the sketch to handle vertices could be ill-defined. More generally, the complexity of the silhouette path on the surface is not necessarily reflected in its image-space projection, making a reasonable mapping from the sketch to vertices on the mesh difficult.

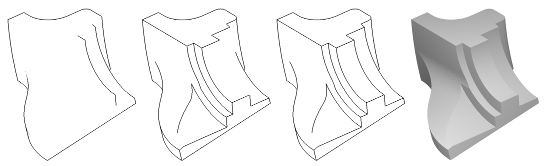

Figure 3: *Depth map discontinuities, Normal map discontinuities, combined discontinuities, shaded scene (left to right).*

Because of these problems we detect silhouettes in image-space, and then try to identify vertices in the mesh that would map onto the detected region in image-space. Image-space

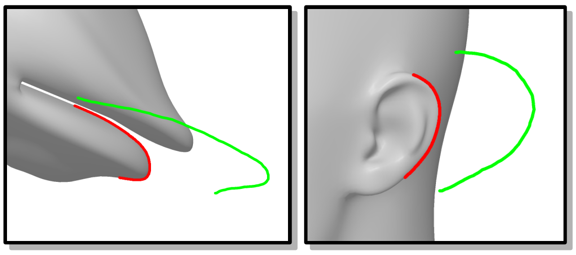

Figure 4: *Handle estimation due to the similarity of handle candidate (red) and targeted deformation (green).*

silhouettes are usually obtained using edge detection filters on the depth map and/or normal map of the shape [Her99]. Typically, the conversion from raster-based edge pixels to vector-based polylines is then achieved by applying some morphological operations (e.g. thinning) and finally tracing (e.g. chain codes). We have decided to restrict the set of feature lines to discontinuities in the depth map. This approach shows a feasible trade-off between quantity of feature lines vs. their significance (see Fig. 3).

Matching a segment of a silhouette in image-space to the user-sketch requires a metric, defining the distance between polylines. This metric should resemble human perception of similarity. We have found that the important features are proximity to the candidate feature lines and intrinsic shape (see Fig. 4). By intrinsic shape we mean similarity regardless of position and orientation in space. To maximize this intrinsic shape similarity we use a method by Cohen and Guibas [CG97].

We determine the handle mesh vertices corresponding to the silhouette segment by selecting vertices which are close to the handle in image-space. The displacements for these vertices are derived from displacements in image-space.

We consider defining the ROI as a form of mesh segmentation, for which various geometry-based methods are described (see [KT03, JLCW06]), and even image-based approaches are conceivable (see [HJBJ*96, PP93]). Whereas image-based approaches obviously suffer from occlusion, geometry-based methods are only restricted by the requirement for interactive response times. Generally, topologically growing the ROI from the handle vertices is a feasible method.

Once we have defined handle vertices, their transformed target positions and the region of interest, the application of Laplacian Surface Editing is straightforward. Note that the user only provides 2D input and we have found that preserving the scale in depth leads to more intuitive results than scaling isotropically in 3D. Interestingly, several of the refinements of Laplacian Surface Editing (such as [SLCO*04]) favor isotropic scaling. For this reason we are currently using an approach in the spirit of [LSCOL04], where local transformations of each frame are estimated a priori. We like to stress that other mesh deformation tools could be used as well.

3. Interface

Our user interface consists of a single rendering window with an orthogonal projection, embedded controls for navigation, and the capability of drawing viewport-aligned strokes (enabled by default). Holding some meta key activates the embedded navigation controls, with which the user can drag the mesh along the horizontal and vertical axis, rotate it by tapping beside it and dragging the mouse, and scale the current projection by clicking and dragging two invisible sliders on the left and right screen boundaries.

If the user has determined an appropriate view, placing a sketch near the silhouette implies a deformation. The system identifies the appropriate parameters (see following sections) and then displays the result. The user has the option to approve this deformation or to apply refinements by oversketching the new silhouette path.

4. Algorithm

The user sketches the desired deformation result as a view-dependent polyline. This polyline simply consists of tracked mouse events, and we apply the Douglas-Peucker algorithm [DP73] to obtain a simplified version. In the following sections we detail the steps of our algorithm.

4.1. Image-Space Silhouettes

In this section, we describe how to retrieve image-space 2D polylines that describe discontinuities in the depth map (and therefore silhouettes) of the scene using two steps of detection and extraction. We developed a method that exploits the properties of a synthetic scene (= absence of noise) to speed up our algorithm, rather than relying on well established methods like the Canny edge detector [Can86] or morphological operations.

4.1.1. Silhouette Detection

We determine discontinuities in the depth map by applying a 4-neighborhood Laplacian edge detection filter on each pixel p, along with some threshold θ_p:

$$sil(p) := D^2_{xy}[depth(p)] > \theta_p \tag{1}$$

We retrieve only edge pixels that describe the foreground of a discontinuity, since we define the depth range of the scene to (near, far) [0, 1] and use θ_p as a threshold for the *signed* filter response. Depending on the choice of θ_p (we recommend 0.005), the binary images retrieved consist of continuous silhouette paths (Fig. 5, left). Note though, that these paths can be more than a single pixel wide, especially in areas of high curvature.

4.1.2. Silhouette Extraction

For the subsequent handle estimation (Sec. 4.2), we need to convert the silhouette pixel paths into a set of image-space polylines. Aiming for simplicity and speed, we developed a greedy segmentation algorithm, which relies only on local criteria for silhouette tracing.

The basic idea of tracing connected components of the silhouettes is that silhouette pixels in the image are neighbors on a silhouette segment if they have similar depth. In other words, two neighboring silhouette pixels a and b are depth continuous if

$$cont(a,b) := \|depth(a) - depth(b)\| < \theta_n. \quad (2)$$

Remember that the silhouette pixels form a path that could be wider than a single pixel, making the conversion to a polyline ambiguous. Some approaches use the morphological operation of thinning to correct this problem. However, applying morphological operations on the binary silhouette image may result in silhouette paths that are continuous in 2D, but discontinuous in depth. This is illustrated in Fig. 6b: the silhouette terminates on pixel f_c if n_7 is removed by erosion, and $\|depth(f_c) - depth(n_0)\|$ exceeds θ_n. In this case, n_7 is exactly the pixel that stitches the silhouette together. Instead of developing depth sensitive morphological operations , we solve this issue by using a local tracing criterion.

The idea for the local tracing is to favor silhouette paths with lower curvature in image-space, i.e. straight silhouettes are favored over ones with sharp corners. The criterion is implemented as a priority map relative to the direction from which we entered the current silhouette pixel (see Figs. 6 and 7: a smaller number in the mask around f_c indicates higher priority). Based on the priority mask, silhouette edge paths are formed by selecting from depth continuous silhouette pixels.

However, correctly identifying endpoints of silhouette paths requires extra attention. A silhouette path ends in surface creases; and it might appear to end in sharp creases of the silhouette (see Fig. 5). It also ends in image-space when the silhouette is obstructed by another part of the surface, in which case it connects to another silhouette (see Fig. 7). Our basic tracing algorithm would correctly identify endpoints in surface creases, however, it might also classify sharp corners as endpoints and could connect unconnected parts of the silhouettes if they happen to have almost similar depth. To avoid terminating in sharp corners, we remove the tips of silhouettes. Note that surface creases are surrounded by pixels with almost similar depth in the depth image, while tips of the silhouette are not (see Fig. 5). So we remove tips by repeatedly removing silhouette pixels if they have less than two depth continuous 8-neighbors in the depth image (see Fig. 6, second image). As an additional criterion for identifying connected silhouette pixel we use consistency of the surface normals along the silhouette (see Fig. 7). As we are only interested in the orientation of the normals, it is sufficient to consider the gradients of the depth map.

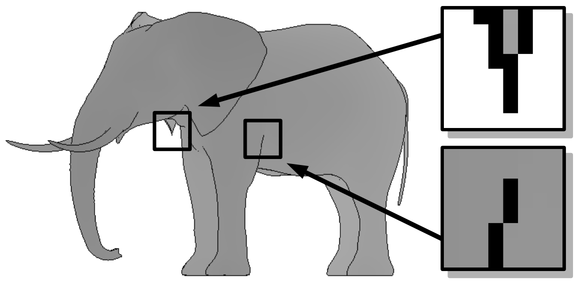

Figure 5: *Depth map with binary overlay from Eqn. 1 (left), degenerated silhouette feature (top, right), silhouette caused by a surface crease (bottom, right).*

In detail, our silhouette extraction algorithm creates silhouette polylines $S : \{(v_1,d_1),...,(v_n,d_n)\}$ described by vertices $v_i \in \mathbb{R}^2$ and depth values $d_i \in \mathbb{R}$, by scanning the binary silhouette image row by row, and extracting feature paths for any encountered silhouette pixel $f_c : (v_c,d_c)$ according to the following algorithm:

1. Create $S = \emptyset$.
2. Append f_c to S.
3. Determine next silhouette pixel f_n, where
   a) f_n is adjacent to f_c,
   b) f_n is depth continuous to f_c according to Eqn.2,
   c) f_n maintains the orientation of depth map gradients w.r.t. the current tracing direction (see Fig. 7), and
   d) the tracing direction turn caused by f_n is minimal.
4. Mark f_c as a non-silhouette pixel.
5. Assign f_n to f_c.
6. Repeat on 2. until $f_c = NIL$.

Note that a) and b) are determined by Eqns. 1, and 2 respectively, whereas c) ensures continuity of the normals along the silhouette paths (Fig. 7). Furthermore, d) is the tracing criterion, navigating the tracing algorithm through silhouette paths wider than a single pixel.

Since scanning the silhouette image row by row typically encounters a silhouette somewhere inside its path, the tracing algorithm is applied twice for any initial pixel, in opposite directions.

4.2. Handle Estimation

To derive the actual handle polyline (a subset of all silhouette polylines), we introduce an estimation metric which reflects the likelihood that an arbitrary silhouette segment is a good handle w.r.t. the user-sketch (target polyline). As pointed out before, this scoring function relies on both proximity and similarity.

First, we substitute the silhouette polylines by simplified delegates (polylines as well, see [DP73]), and reduce the silhouettes by culling according to a proximity criterion (see Figs. 1b and 1c).

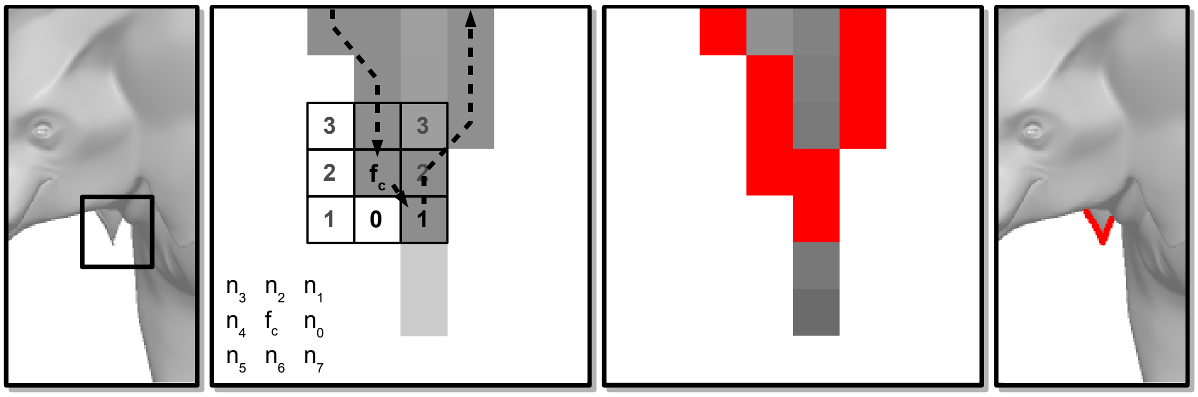

Figure 6: *Tracing the silhouette path near a degenerate feature (from left to right): a) Elephant's ear, b) tracing step ($f_c \longrightarrow n_7$) with priority map, neighborhood index (bottom left) and a degenerate feature in light grey (which is removed in a pre-processing step), c) final silhouette path, d) extracted silhouette.*

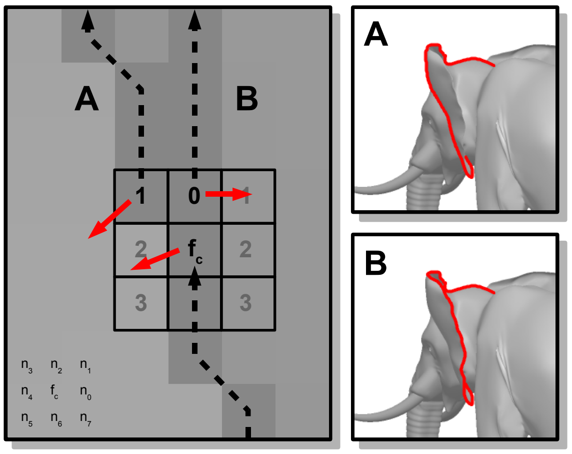

Figure 7: *Maintaining depth map gradient orientation. Path A shows how our tracing algorithm maintains depth map gradient orientation with respect to the tracing direction (gradients shown as arrows per pixel). If we disregard these gradients, the tracing algorithm will track a bogus silhouette, in this case path B, due to the preferred tracing direction. Note though, that the silhouette part from path B, which is missing in path A, will be a separate silhouette segment after all silhouettes have been traced.*

The criterion on similarity is derived from the *Polyline Shape Search Problem (PSSP)* described by Cohen and Guibas [CG97]. First, we compute Turning Angle Summaries (TASs) $\{(s_0,t_0),\dots,(s_n,t_n)\}$ from the edges $\{e_0,\dots e_n\}$ of the target and silhouette polylines by concatenating tuples of edge lengths s_i and cumulative turning angles t_i, where

$$s_i = \| e_i \|, \quad t_i = \begin{cases} \measuredangle(e_0,0) & \text{if } i=0 \\ \measuredangle(e_{i-1},e_i)+t_{i-1} & \text{if } i>0 \end{cases} \tag{3}$$

Please note that these summaries lack the representation of absolute coordinates, but they do retain the polyline arclength. Furthermore, rotating a polyline relative to its head results in a shift of its TAS along the turning angle axis, whereas isotropic scaling results in stretching its TAS along the arclength axis (see Fig. 8).

We match the target polyline onto a single silhouette polyline, described by its (isotropic) scale α and position (shift) β, by matching their Turning Angle Summaries (Fig. 8). The match result M_{PSSP} : $(\alpha,\ \beta,\ \gamma,\ R_{*mod})$ is described by a prescribed α and β, an optimal rotation γ, and the matching score R_{*mod}. Optimal rotation and matching score are computed by a modified version of the scoring function from [CG97]. Using finite sums of differences, I_1 and I_2 describe the linear and squared differences between the piecewise constant TASs $\Psi(s)$ of the target and $\Theta(s)$ of the silhouette polylines (Fig. 8):

$$\begin{aligned} I_1(\alpha,\beta) &= \int_{s=\beta}^{\beta+\alpha} \left(\Theta(s) - \Psi\left(\frac{s-\beta}{\alpha}\right)\right) ds, \\ I_2(\alpha,\beta) &= \int_{s=\beta}^{\beta+\alpha} \left(\Theta(s) - \Psi\left(\frac{s-\beta}{\alpha}\right)\right)^2 ds. \end{aligned} \tag{4}$$

Given the arclength l of the target polyline, we compute optimal rotation

$$\gamma = \gamma_*(\alpha,\beta) = \frac{I_1}{\alpha l}, \tag{5}$$

and matching score

$$R_{*mod}(\alpha,\beta) = \frac{1}{\alpha l}\left(\frac{I_2(\alpha,\beta)}{\alpha l} - \left(\frac{I_1(\alpha,\beta)}{\alpha l}\right)^2\right). \tag{6}$$

Cohen and Guibas retrieve matches for all segments (α,β) by using a topological sweep algorithm [EG86] to match the respective Turning Angle Summaries in scale/position space. However, since this approach needs $O(m^2n^2)$ time for m silhouette edges and n target edges, we

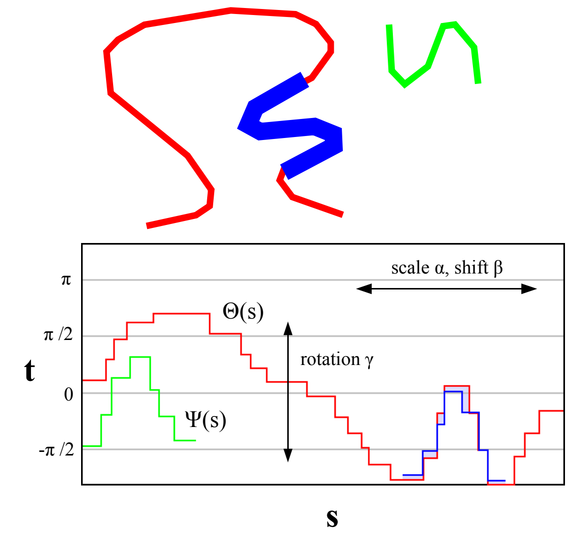

Figure 8: *Top: the short, green target polyline, red silhouette, and best-match (blue/thick) shown as a subset of the red silhouette polyline. Bottom: arclength vs. cumulative turning angle representations of target $\Psi(s)$, silhouette $\Theta(s)$, and best-match polylines (bottom).*

decided to probe only a discrete number of sample segments in Eqn. 6 in $O(m+n)$ time per segment. Specifically, we match the target polyline to sample segments of a silhouette polyline by discretely sampling α and β respectively.

For the proximity criterion we compute the distances of corresponding endpoints of the two polylines, retrieving a near and far value $Prox_{near}$, $Prox_{far}$. Then we apply a final scoring function on the obtained per-silhouette match results:

$$R := 1/(1+w_1 Prox_{near} + w_2 Prox_{far} + w_3 R_{*mod})^2 \qquad (7)$$

Iterating over all silhouettes, we select the segment with the highest score, and extract the deformation handle from the respective full-res silhouette by using (α,β) of its matching record M_{PSSP}.

4.3. Finding Handle/Target Correspondences

Given the polylines of deformation handle and target, we need to determine the corresponding mesh vertices and their transformed positions respectively.

Using both the image-space handle pixels, as well as the corresponding depth map, we construct an object-space bounding volume for each handle pixel (see Fig. 9). A mesh vertex is classified as a handle vertex if it lies in the union of these bounding volumes.

The transformed positions for these handle vertices are computed by mapping their handle-relative positions onto the target polyline. Specifically, we determine the position (s,d) for each handle vertex, where the arclength position s is given by its orthogonal projection of length d. Both handle and target polylines are parameterized uniformly in $[0,1]$ and the target position (s',d') is scaled accordingly.

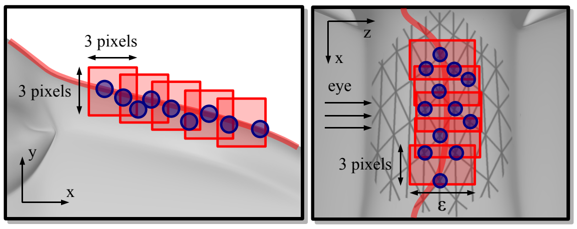

Figure 9: *Mesh vertices that are classified as handle members (blue circles) using one bounding volume (red box) for each image-space handle pixel. Left: view from the editor, right: view from top (silhouette indicated as a red line in both views).*

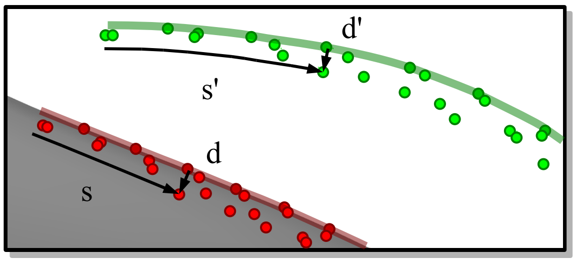

Figure 10: *Mapping of handle relative arclength position s and diplacement d (red) onto the target polyline (green).*

4.4. ROI Estimation

To complete the deformation setup, we have to select the final ROI of the mesh according to some context sensitive criterion. We grow the ROI from the handle vertices. To control the expansion, we constrain the ROI to lie within a union of bounding volumes, which consists of one volume per handle vertex.

Specifically, we create a union of spheres, where each sphere center is located at the position of the respective handle vertex. Each sphere radius is set to the Euclidean distance $d_{h,s}$ between handle vertex and its transformed position. We have experimented with a variety of functions $r_s = f(d_{h,s})$, but have found that using $r_s = d_{h,s}$ already yields satisfying results: when the user sketch is far from the handle, using a larger sphere results in a larger ROI, yielding more deformable material (Fig. 11), which is a reasonable heuristic. To determine the ROI, we define the handle vertices to be the initial ROI vertex set, and grow this set by subsequently adding vertices of the mesh that are (a) adjacent to the current ROI border, and (b) are inside the union of spheres.

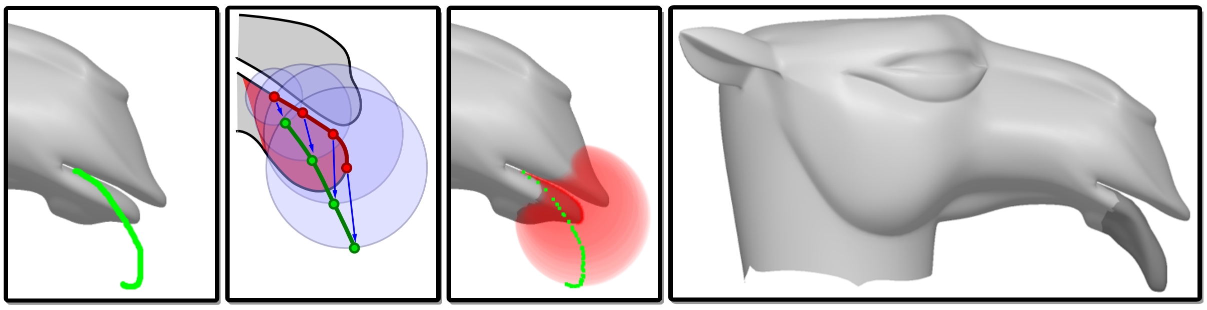

Figure 11: *Automatic ROI selection (from left to right): a) After the user places a sketch, the handle is estimated and correspondences are established. b) From these correspondences, the ROI is grown within the union of spheres, starting from the handle vertices (dark/red region, lower lip). c) Shows this for the camel lip example. d) We use the obtained vertex sets* handle, transformed handle *and* ROI *as input to the LSE algorithm. See text for more details.*

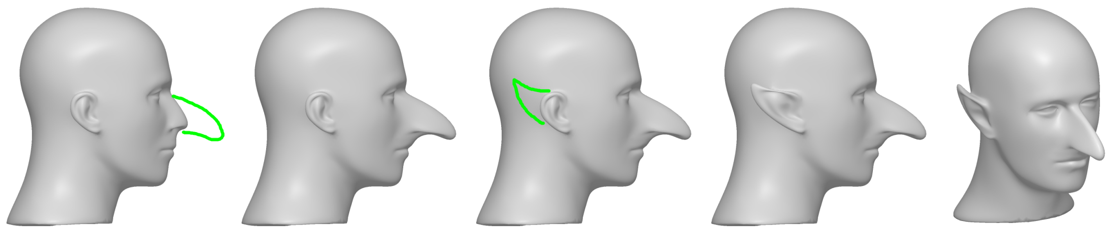

Figure 12: *The* MANNEQUIN *modeling session.*

5. Results

The modeling session shown in Fig. 12 illustrates ease of use: after the user places a stroke, the system responds interactively, presenting a deformation which generally corresponds to the users intent. All algorithmic details, which are shown in various figures in this paper, are absent from the actual user interface. For more interactive modeling sessions, please see our accompanying video.

Table 1 shows some timings obtained on a Intel Core 2 Duo 6600 processor with 2.4 GHz and 2GB memory. Extracting and segmenting the image-space silhouettes (column *Sil*) takes between 5-20% of the processing time. Handle estimation and finding handle/target correspondence (column *Handle*) depends on the density of silhouettes, as well as the number of model vertices (=5-25% overall). The column *LSE size* shows the dimensions of the sparse linear system (= number of ROI vertices), which is factored (FacLSE) and solved (SolveLSE) every time the user places a new stroke. This works interactively for ROIs up to a few thousand vertices. Of course we can also reuse the factorization as described in [NSACO05]. Note that in all cases, our algorithms (Sil + Handle + ROI) use less time than LSE setup, factorization and solve (FacLSE + SolveLSE).

6. Discussion

Each of the steps in our approach presents a trade-off between fidelity and speed. And while the requirement of real-time interaction certainly restricts the algorithmic possibilities, it should also be clear that almost all over-sketches are generically ambiguous, even in the case of communication among humans – so it is unlikely that an algorithm could consistently guess correctly according to the user's expectation.

We find that the extraction and segmentation of feature lines (silhouettes) works in almost all practical cases. It might be interesting to extend the extraction to discontinuities in the normals of the shape, or even to more subtle feature lines such as suggestive contours [DFRS03]. Another set of feature lines, though invisible from the rendering but known to more experienced users, are the projections of skeleton curves used in models rigged for animation. The information deduced by our system could then be fed into modeling systems controlled by skeletons.

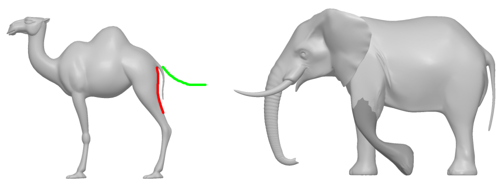

Figure 13: *Left: ambiguous handle estimation at the* CAMEL*'s tail. Right: unnatural deformation of the* ELEPHANT*'s leg due to the limitation of LSE regarding large rotations.*

Model	Feature	Sil*)	Handle*)	ROI	FacLSE	SolveLSE*)	Sum	LSE size	Figure
Bunny	Ear	109	297	15	1032	500	1953	4911 x 4911	1
CamelHead	Lip	110	250	15	250	140	765	1533 x 1533	11
Mannequin	Nose	188	219	15	485	156	1063	2013 x 2013	12
	Ear	94	62	16	609	156	937	3627 x 3627	12
all timings in msec; *) unoptimized code									

Table 1: *Some timings of our system.*

Matching the user-sketch against the feature lines works nicely, however, it might be interesting to experiment with different functions for measuring proximity and shape similarity to overcome ambiguous handle estimations (see Fig. 13 left). More fundamentally, matching is performed only against connected segments of the feature lines. The user might want to sketch something that the system has identified as different parts of the feature lines. It is unclear to us how to extend the matching process to this case.

The ROI is selected based on proximity between user-sketch and feature line in image-space. This turned out to be simple and effective, yet it disregards apparent features of the shape. We believe the results could be improved by including information such as curvature and other features in image-space into our region growing approach. Another way of improving on the selection of the ROI would be to involve the user, perhaps by defining a special stroke indicating parts that may not move.

Looking at the deformation example in Fig. 13 (right), it is clear that LSE is not a universally applicable deformation tool. However, it should be feasible to use the information gathered by the handle estimation such as rotation and scale of the best handle match in the deformation step.

Finally, as the system is almost generic with regard to the type of surface representation and the deformation tool, it would be very interesting to also try this approach in other settings.

References

[BPG06] Botsch M., Pauly M., Gross M.: PriMo: coupled prisms for intuitive surface modeling. In *Eurographics Symposium on Geometry Processing* (2006), pp. 11–20.

[Can86] Canny J.: A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell. 8*, 6 (1986), 679–698.

[CG97] Cohen S. D., Guibas L. J.: Partial matching of planar polylines under similarity transformations. In *SODA: ACM-SIAM Symposium on Discrete Algorithms* (1997).

[DFRS03] DeCarlo D., Finkelstein A., Rusinkiewicz S., Santella A.: Suggestive contours for conveying shape. *ACM Trans. Graph. 22*, 3 (2003), 848–855.

[DP73] Douglas D., Peucker T.: Algorithms for the reduction of the number of points required to represent a hne or its caricature. *The Canadian Cartographer*, 10(2) (1973), 112–122.

[EG86] Edelsbrunner H., Guibas L. J.: Topologically sweeping an arrangement. In *STOC '86: Proceedings of the eighteenth annual ACM symposium on Theory of computing* (1986), pp. 389–403.

[Her99] Hertzmann A.: Introduction to 3D non-photorealistic rendering: Silhouettes and outlines. In *Non-Photorealistic Rendering. SIGGRAPH 99 Course Notes.* (1999).

[HJBJ*96] Hoover A., Jean-Baptiste G., Jiang X., Flynn P. J., Bunke H., Goldgof D. B., Bowyer K. K., Eggert D. W., Fitzgibbon A. W., Fisher R. B.: An experimental comparison of range image segmentation algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence 18*, 7 (1996), 673–689.

[HS97] Hoffman D. D., Singh M.: Salience of visual parts. *Cognition*, 63 (1997), 29–78.

[IMT99] Igarashi T., Matsuoka S., Tanaka H.: Teddy: A sketching interface for 3D freeform design. In *Proceedings of SIGGRAPH* (1999), pp. 409–416.

[JLCW06] Ji Z., Liu L., Chen Z., Wang G.: Easy Mesh Cutting. *Computer Graphics Forum 25*, 3 (2006), 283–291.

[KG05] Kho Y., Garland M.: Sketching mesh deformations. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games* (2005), pp. 147–154.

[KS07] Kara L. B., Shimada K.: Sketch-based 3d-shape creation for industrial styling design. *IEEE Computer Graphics and Applications 27*, 1 (2007), 60–71.

[KSvdP07] Kraevoy V., Sheffer A., van de Panne M.: Contour-based modeling using deformable 3d templates. Tech Report TR-2007-13, CS, 2007.

[KT03] Katz S., Tal A.: Hierarchical mesh decomposition using fuzzy clustering and cuts. In *ACM SIGGRAPH* (2003), pp. 954–961.

[LSCOL04] Lipman Y., Sorkine O., Cohen-Or D., Levin D.: Differential coordinates for interactive mesh editing. In *International Conference on Shape Modeling and Applications* (2004), pp. 181–190.

[NSACO05] Nealen A., Sorkine O., Alexa M., Cohen-Or D.: A sketch-based interface for detail-preserving mesh editing. *ACM Trans. Graph. 24*, 3 (2005), 1142–1147.

[PP93] Pulli K., Pietikäinen M.: Range image segmentation based on decomposition of surface normals. In *8th Scandinavian Conference on Image Analysis (SCIA'93)* (Tromso, May 1993).

[SLCO*04] Sorkine O., Lipman Y., Cohen-Or D., Alexa M., Rössl C., Seidel H.-P.: Laplacian surface editing. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry processing* (2004), pp. 179–188.

[YZX*04] Yu Y., Zhou K., Xu D., Shi X., Bao H., Guo B., Shum H.-Y.: Mesh editing with Poisson-based gradient field manipulation. *ACM Trans. Graph. 23*, 3 (2004), 644–651.