Sketching Contours

Johannes Zimmermann [a,b], Andrew Nealen [a], Marc Alexa [a,b]

[a] *TU Berlin*

[b] *Walt Disney Animation Studios*

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.

Key words: User Interfaces, Sketching, Sketch Based Modeling, Deformations

1 Introduction

The process of generating 3D shapes in engineering or content creation typically goes through several design cycles, with the team meeting to do *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. If the shape is represented digitally it seems natural to use the sketches as input for a system that computes the necessary deformation automatically.

In [1] we have presented such a tool for surface mesh editing 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. It builds on mesh deformation tools [2,3], and the main challenge is to solve the feature extraction and correspondence problem *automatically* and in *real-time*.

Laplacian Surface Editing [2], but also most other recent mesh deformation techniques (e.g., [4,5]) 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. Our system computes this information from the over-sketched feature line in fractions of a second. The steps described below comprise our system (see also Fig. 1):

(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.

In steps 1 and 2 the user sketch is identified with a segment of a feature line in the current view of the model. These steps are identical to our earlier version of this work and allow modifying a parameter that represents a trade-off between matching based on proximity and similarity. We have verified in a user study that there is no "right" answer to how this parameter should be chosen. Also, we discuss more background material, better explaining and motivating our choices.

In steps 3, 4, and 5 we compute the necessary input for shape deformation, and we have experimented with variants for all of them since our earlier version. We have significantly improved the algorithm for step 3, mostly for the case of many vertices being projected close to the feature line (see section 4.3). Several reviewers and experts had suggested an alternative assumption for the fourth step, namely not simply assuming constant depth for the modified vertex positions, but intelligently extending the features of the model. We have implemented this variant and performed a user study, comparing the two versions. We have found that most users actually prefer constant depth (see section 6). Also for the fifth step we have experimented with another variant, however, we are most satisfied with the original version.

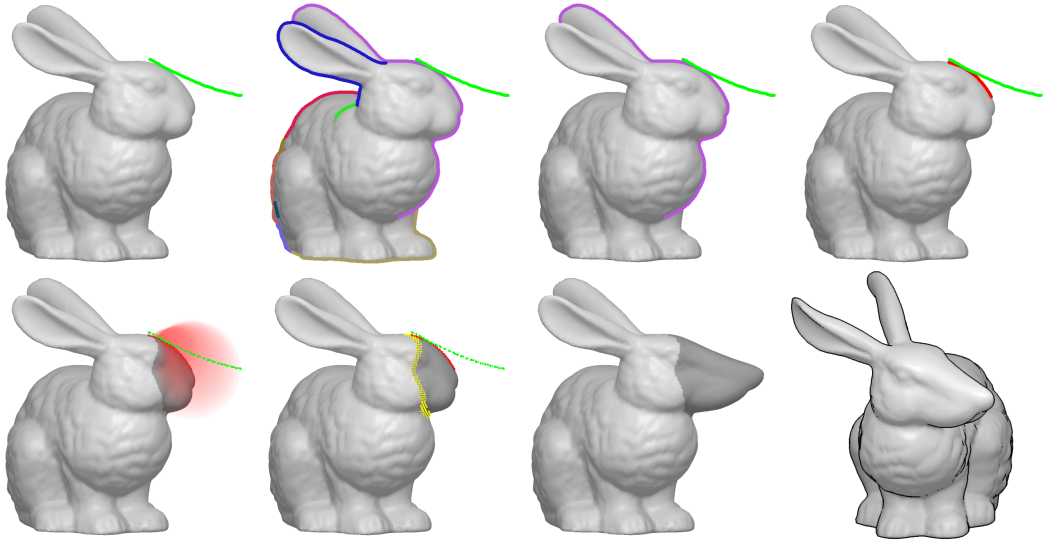

Fig. 1. Algorithm pipeline. Top row, left to right: a) user-sketch, b) image-space silhouettes, c) retained silhouettes after proximity culling, d) handle estimation; Bottom row, left to right: e) correspondences and ROI estimation by bounding volumes, f) setup for Laplacian Surface Editing, g) and h)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

2 Related Work and System Design

In this chapter, we briefly discuss techniques related to our system and explain some of our design choices.

2.1 Sketch-based Interfaces for Mesh Editing

Sketch-based interfaces are a very popular method for creation and deformation of 3D surface meshes [6–8]. Deriving the parameters for mesh deformation from sketches only is not new: Kho and Garland [9] (see Fig. 2) 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 [10], 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. [3]. They have enhanced Lapla-

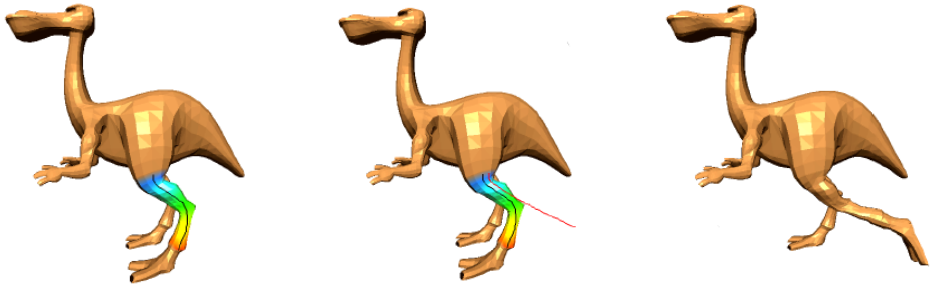

Fig. 2. Sketching Mesh Deformations by Kho and Garland [9]. Left to right: ROI with handle (black), suggested modification (red), and deformation result. Images courtesy of Youngihn Kho and Michael Garland.

cian Surface Editing techniques to work in the setting of prescribing new silhouettes (see Fig. 3). 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.

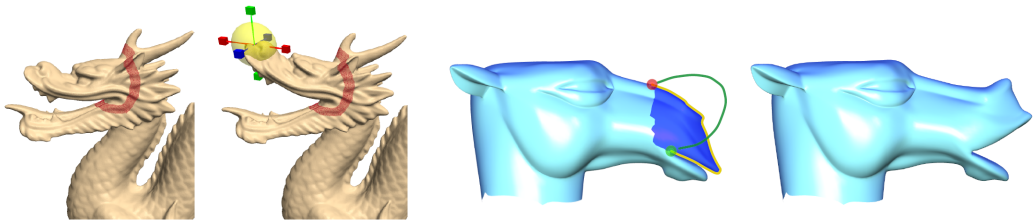

Fig. 3. Left: Laplacian Surface Editing (LSE) by Sorkine et al. [2]. Right: A Sketch-Based Interface for Detail-Preserving Mesh Editing by Nealen et al. [3].

In our system, all these selections are now automated; the user only provides a single stroke, from which handle and ROI are estimated (see Fig. 1).

2.2 Image-Space Silhouettes

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

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 silhouettes are usually obtained using edge detection filters on the depth map and/or normal map of the shape [11]. 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. 4).

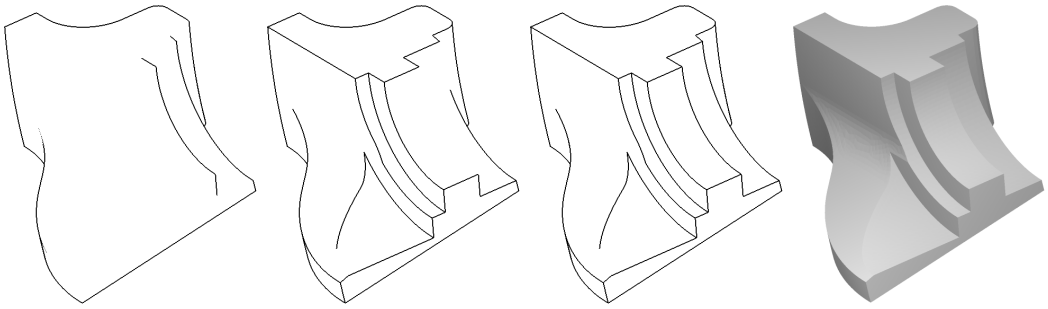

Fig. 4. Feature lines. From left to right: depth map discontinuities, normal map discontinuities, combined discontinuities, flat shaded scene.

2.3 Edge Detection

Considering an image as a function $f(x, y)$, edges are described by discontinuities of f, which can be detected by the examination of derivatives of f. Here the most common approach is to compute (approximations of) local derivatives by applying convolution filters and to classify edges based on some threshold θ.

2.3.1 Convolution Filters

The 1st derivative can be obtained by convolution kernels that compute the horizontal and vertical gradient:

$$G_x = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \qquad G_y = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}. \tag{1}$$

Edge energy E and direction Θ are then defined by

$$E = \sqrt{{G_x}^2 + {G_y}^2}, \qquad \Theta = arctan\left(\frac{G_x}{G_y}\right). \tag{2}$$

The Sobel operator enhances the 2D gradient kernels by an implicit smoothing operation, where the gradient is computed by a weighted average over the 8-neighborhood to reduce the effect of noise:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \qquad S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}. \tag{3}$$

The discrete Laplace filter is an edge detection operator that computes an approximation of the local 2nd derivative (Laplacian) of image values – here based on the 4-neighborhood:

$$L_{xy} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{4}$$

Ballard and Brown [12] note that *"the Laplacian has fallen in disuse"*, since *"(a) useful directional information is not available, and (b) the Laplacian, being an approximation to the second derivative, doubly enhances any noise in the image"*. On the other hand, the signed Laplacian can be used to detect pixels in the foreground of discontinuities. Since these pixels are spanned only by the vertices we consider to be handle candidates, we chose to use the discrete Laplace filter on a 4-neighborhood to classify our silhouette pixels, and determine the silhouette orientation by computing gradient directions (see Eqn. 2). Finally, we want to stress that noise is not an issue, since we are dealing with synthesized scenes that virtually lack any noise.

2.3.2 Canny Algorithm

Edge energies retrieved by the application of convolution filters are commonly used to classify edge pixels by the application of a threshold θ. The Canny algorithm [13,14] applies gaussian smoothing, and computes edge energies and directions using the Sobel operator. Edge pixels are then classified by a hysteresis approach. In a first step, pixels whose energy exceeds θ_{high} are classified to be edge pixels. These initial edges are then expanded by tracing them in respect to their gradient direction until their energy falls below θ_{low}. Although

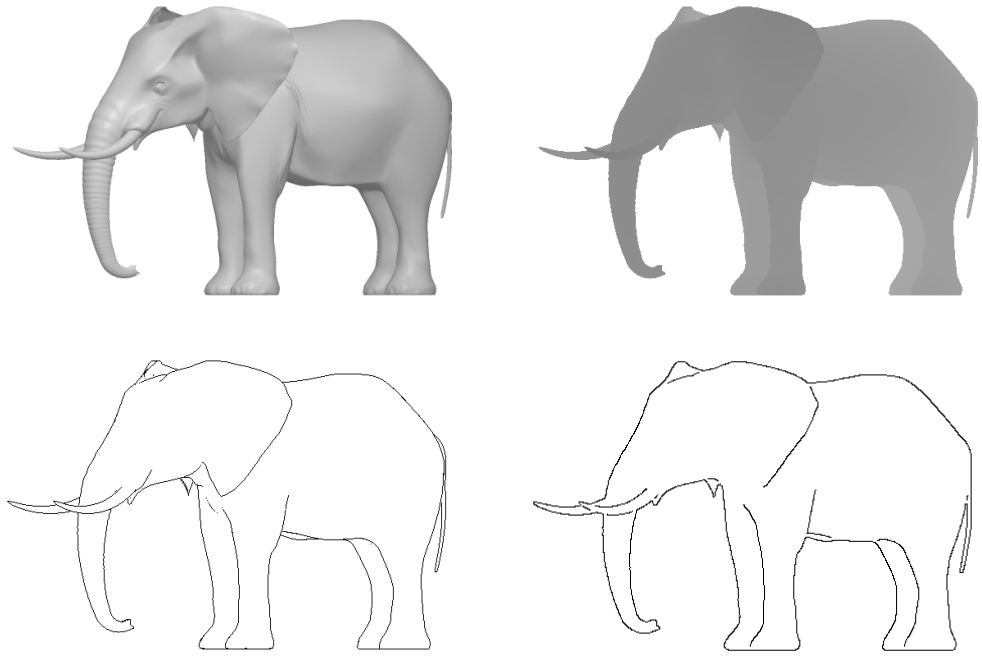

Fig. 5. Edge detection. Top row, from left to right: shaded model, depth map. Bottom row, left to right: result of the 4-neighborhood Laplace filter, and output of the Canny Algorithm. Please note the disconnected silhouettes at the ELEPHANT's forehead and ear in the output of the Canny Algorithm.

the Canny operator serves well in edge detection for generic images, it fails in extracting meaningful silhouettes from synthetic depth images: since edges are traced disregarding their depth continuity, the algorithm yields false connectivity for overlapping silhouettes (see Fig. 5).

Considering thresholds for the classification of edge pixels, we noticed that in our scenario the application of a single threshold already yields satisfying results. Following the idea to create polylines from pixels by tracing them, we developed a custom algorithm that is described in section 4.1.2.

2.4 Polyline Simplification

Using edge detection, we implemented a silhouette extraction algorithm that extracts silhouette polylines as strings of image space coordinates. We use simplified polyline delegates to reduce the complexity of the handle estimation. Therefore, we can drop the silhouettes' depth information, since the handle is estimated with the 2D user sketch. We use the Douglas-Peucker algorithm [15], that subsequently refines an initially coarse model of a polyline by adding vertices with respect to their displacement from the current approximation.

This algorithm has a runtime complexity of $O(n\ log_2 n)$ and terminates if a given error is met (see Fig. 6).

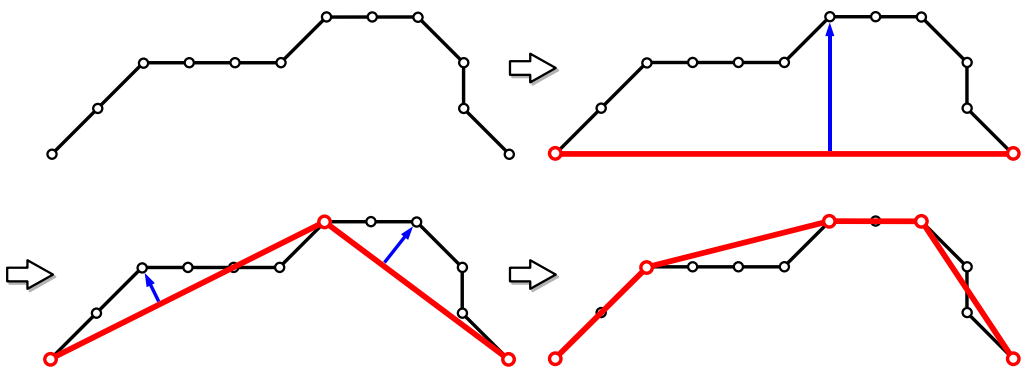

Fig. 6. Douglas-Peucker Algorithm [15]. Left to right, top to bottom: original polyline and first three levels of its recursive approximation.

2.5 Handle Retrieval

Matching a segment of a silhouette in image-space to the user-sketch requires a metric that defines 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. 7). By intrinsic shape we mean similarity regardless of position, (uniform) scale and orientation in space.

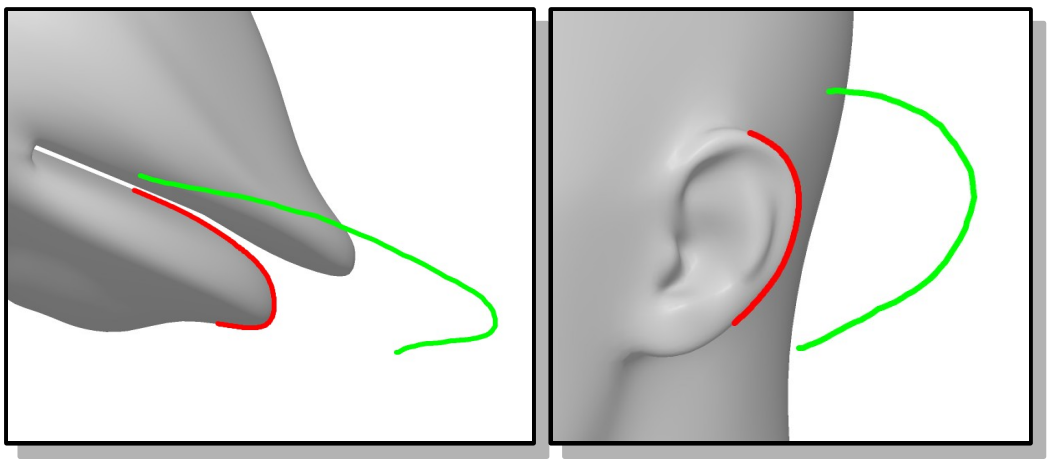

Fig. 7. Handle estimation due to the similarity of handle candidate (red) and targeted deformation (green).

We use a method by Cohen and Guibas [16] to maximize this intrinsic shape similarity. They perform partial matching of two planar polylines under similarity transformations (isotropic scaling, rotation, and translation) by minimizing differences in their turning angle over arc length representation. Although they manage to reduce their matching algorithm's runtime complexity from $O(m^3n^3)$ to $O(m^2n^2)$ (here: for m silhouette polyline vertices and n user-sketch vertices), it can not achieve real-time performance in our setting. Using their scoring function, we relax our solution by testing only a discrete subset of all possible matches in $O(m+n)$ per match.

Finally, we retrieve 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.

2.6 Mesh Segmentation

We consider defining the ROI as a form of mesh segmentation, for which various geometry-based methods are described (see [17,18]), and even image-based approaches are conceivable (see [19,20]). Whereas image-based approaches obviously suffer from occlusion, geometry-based methods are only restricted by our requirement for interactive response times.

Katz and Tal [17] describe a geometry-based method for binary mesh segmentation that is extended to a hierarchical k-way case. Their segmentation uses the criterions of *geodesic and angular distance* between faces of a mesh. Based on these criterions they compute probabilities for the mesh faces to belong to a certain segment. According to these probabilities the faces are then classified to belong to either a certain segment, or a fuzzy region between some segments. The final segment boundaries are determined within the respective fuzzy regions using an undirected flow network graph. Unfortunately, this approach implies an *all-pairs shortest path* problem over all vertices of a mesh, which is why this method is too complex to be applied in real-time.

The method described by Ji et al. [18] (see Fig. 8) grows an initial segmentation, which can be implemented efficiently. The growing is constrained by an improved isophotic metric, that determines the distance between two points on the surface with respect to their *geodesic distance*, the *difference of their normals*, as wells as an augmented *directional curvature* along edges. They use this mesh segmentation approach to implement a framework for feature preserving mesh editing using Laplacian Surface Editing (LSE) [2], where the user selects the ROI by marking fore- and background with strokes, and sketches its deformation relative to the foreground stroke. However, it is unclear how to

supply an initial segmentation solely based on the mesh and the user-sketch.

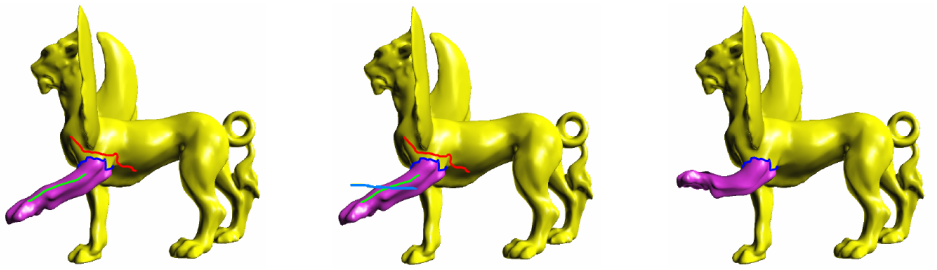

Fig. 8. ROI selection using Easy Mesh Cutting by Ji et al. [18]. Left: ROI (purple, with blue boundary) selection by a foreground stroke (green), and background stroke (red). Middle: Deformation hint (light blue). Right: Deformation result. Images courtesy of Ji et al.

In our application we consider topologically growing the ROI from the handle vertices to be a feasible method, and we use a specific set of bounding volumes (spheres) to constrain this process both spatially and topologically.

2.7 Mesh Deformation

Once we have defined handle vertices, their transformed target positions and the region of interest, the application of Laplacian Surface Editing is straightforward (please see [2,21] for details). 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. In other words, we retain handle vertex positions in depth, even if their transformed target positions refer to a 2D isotropic scaling operation that might be extended onto the third dimension. Interestingly, several of the refinements of Laplacian Surface Editing (such as [2]) favor isotropic scaling. For this reason we are currently using an approach in the spirit of Lipman et al. [22], where local transformations of each frame are estimated a priori (please see Fig. 9). 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 a 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

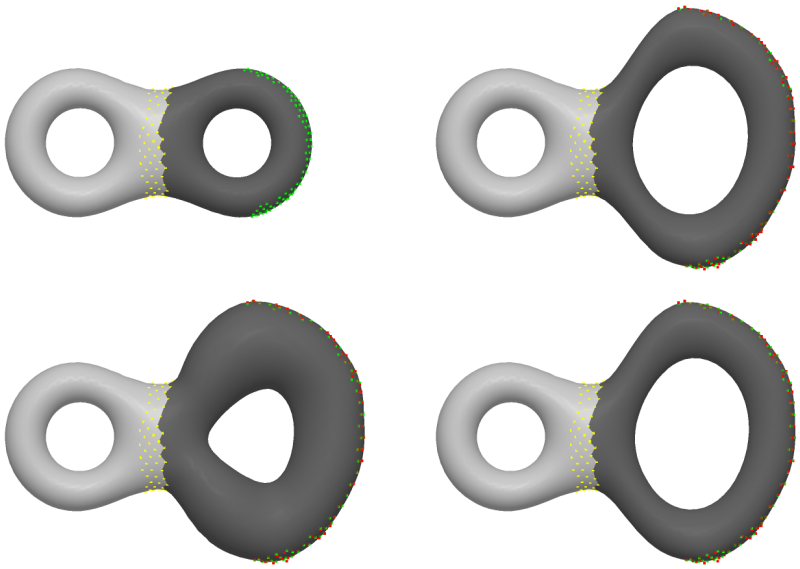

Fig. 9. LSE refinements. Top left: deformation setup with ROI (dark gray), static anchors (yellow), and handle vertices (green). Top right: straightforward application of LSE (with target vertices in red). Bottom left: refined deformation according to Sorkine et al. [2] (by estimating rotation and isotropic scale introduced to the Laplacians). Bottom right: refined deformation according to Lipman et al. [22].

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 to obtain a simplified version (see Section 2.4). In the following sections we detail the steps of our algorithm.

4.1 Silhouette Retrieval

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 noise-free synthetic scene to speed up our algorithm, rather than relying on well established methods like the Canny edge detector [13] 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:

$$sil(p) := L_{xy}[depth(p)] > \theta_p \tag{5}$$

Note that we retrieve only edge pixels that describe the foreground of a discontinuity by using a signed comparison. Depending on the choice of θ_p, the binary images retrieved consist of continuous silhouette paths (Fig. 10, left).

In our implementation, the depth range of the scene is mapped to $[0, 2]$, while the object's bounding box is scaled to have unit diagonal and centered at unit distance. With these values we have found $\theta_p \approx 0.01$ to be a good choice.

Note that the silhouette paths can be more than a single pixel wide, especially in areas of high curvature.

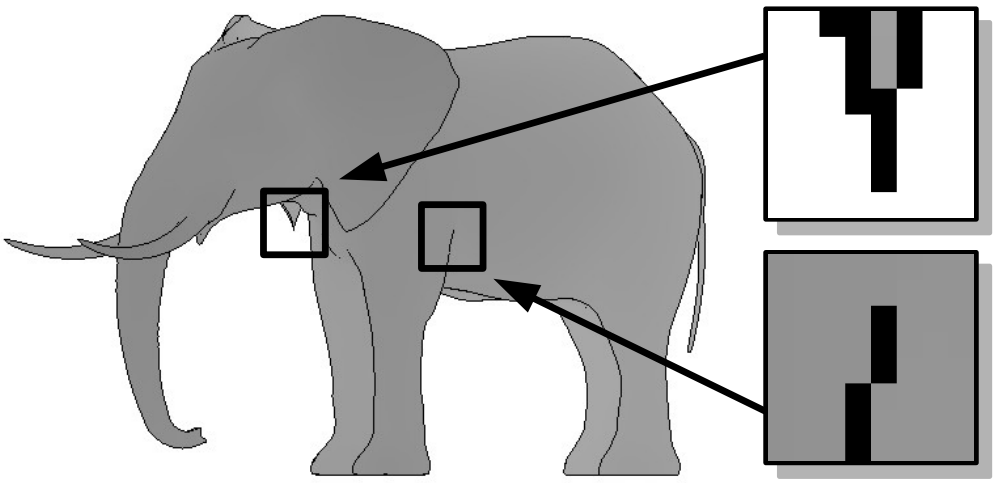

Fig. 10. Depth map with binary overlay from Eqn. 5 (left), degenerated silhouette feature (top, right), silhouette caused by a surface crease (bottom, right)

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

$$cont(a, b) := \|depth(a) - depth(b)\| < \theta_n. \tag{6}$$

The silhouette pixels form a path that could be wider than a single pixel, making the conversion to a poly-line 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. 11b: the silhouette terminates on the center 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. 11 and 12: 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.

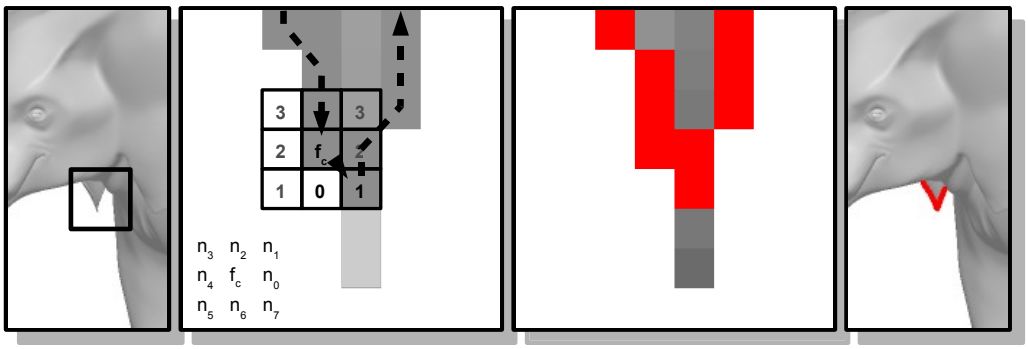

Fig. 11. Tracing the silhouette path near a degenerate feature (from left to right): a) Elephant's ear, b) tracing step ($f_c \rightarrow n_7$) with priority map, neighborhood index (bottom left) and a degenerate feature in light grey (which is removed in a pre-processing step), c) final silhouette path, d) extracted silhouette.

However, correctly identifying endpoints of silhouette paths requires extra attention. A silhouette path ends in surface creases and it also ends in image-space when the silhouette is occluded by a part of the surface that is significantly closer to the viewer. Our basic tracing algorithm correctly identifies these two cases, however, it might also classify sharp corners as endpoints (Fig. 10). 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. 10). 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. 11, second image). Furthermore, the basic algorithm will arbitrarily select silhouette paths if the respective neighboring pixels happen to have almost similar depth (Fig. 12, left). Therefore, as an additional criterion for identifying connected silhouette pixels we use consistency of the surface normals along the silhouette. As we are only interested in the image-space orientation of the normals, it is sufficient to consider the gradients of the depth map (Fig. 12).

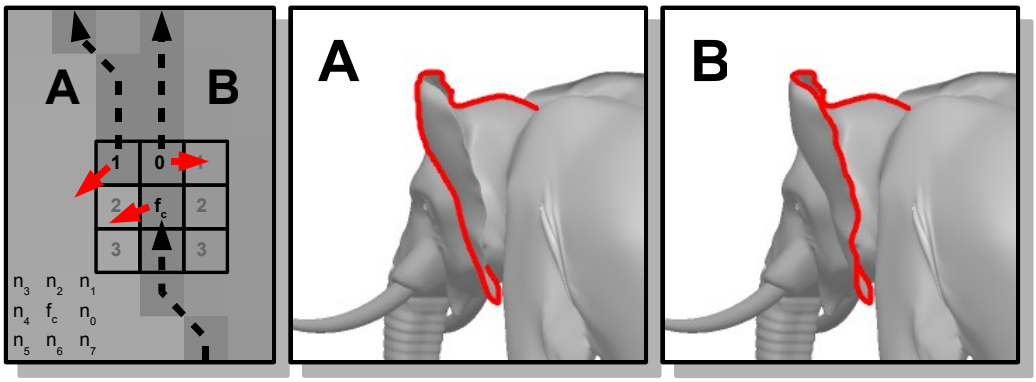

Fig. 12. Maintaining depth map gradient orientation. Path A shows how our tracing algorithm maintains depth map gradient orientation with respect to the tracing direction (left image: gradients shown as arrows per pixel). If we disregard these gradients, the tracing algorithm will trace a non-intuitive silhouette path, 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.

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.6,
  c) f_n maintains the orientation of depth map gradients w.r.t. the current tracing direction (see Fig. 12), 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. 5, and 6 respectively, whereas c) ensures continuity of the normals along the silhouette paths (see Fig. 12). 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. Fig. 13 shows some results of our algorithm.

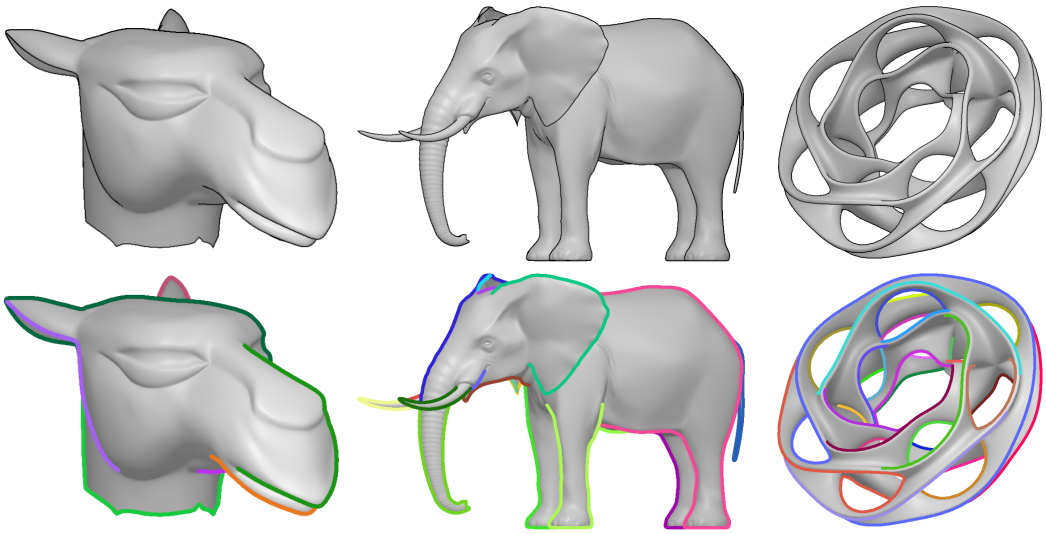

Fig. 13. Top row: models with their binary edge images, bottom row: extracted and segmented silhouette paths (shown wider than one pixel for illustration purposes).

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 [15]), and reduce the silhouettes by culling according to a proximity criterion.

The criterion on similarity is derived from the *Polyline Shape Search Problem (PSSP)* described by Cohen and Guibas [16]. First, we compute Turning Angle Summaries (TASs) $\{(s_0, t_0), ..., (s_n, t_n)\}$ from the edges $\{e_0, ...e_n\}$ of the target and silhouette polylines by concatenating tuples of edge lengths s_i and cumulative turning angles t_i, where

$$s_i = \| e_i \|, \quad t_i = \begin{cases} \measuredangle(e_0, 0) & \text{if } i = 0 \\ \measuredangle(e_{i-1}, e_i) + t_{i-1} & \text{if } i > 0 \end{cases} \tag{7}$$

Please note that these summaries lack the representation of absolute coordinates, but they do retain the polyline arc length. 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 arc length axis (see Fig. 14).

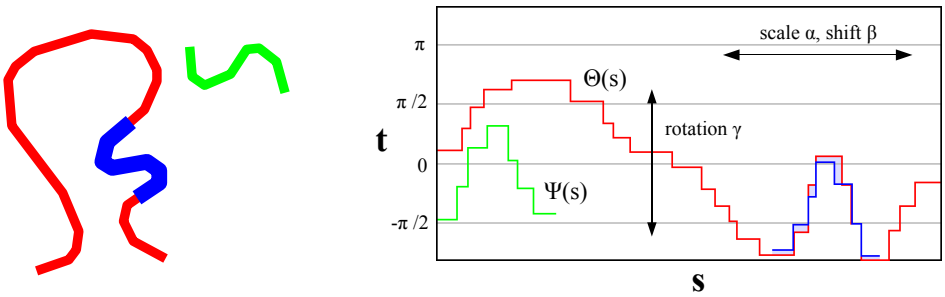

Fig. 14. Left: the short, green target polyline, red silhouette, and blue best-match shown as a subset of the red silhouette polyline. Right: arc length vs. cumulative turning angle representations of target $\Psi(s)$, silhouette $\Theta(s)$, and best-match polylines.

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. 14). 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 [16]. 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 sihouette polylines (Fig. 14):

$$\begin{aligned} I_1(\alpha,\beta) &= \int_{s=\beta}^{\beta+\alpha} \Big(\Theta(s) - \Psi\Big(\frac{s-\beta}{\alpha}\Big)\Big) ds, \\ I_2(\alpha,\beta) &= \int_{s=\beta}^{\beta+\alpha} \Big(\Theta(s) - \Psi\Big(\frac{s-\beta}{\alpha}\Big)\Big)^2 ds. \end{aligned} \tag{8}$$

Given the arc length l of the target polyline, we compute optimal rotation

$$\gamma = \gamma_*(\alpha,\beta) = \frac{I_1}{\alpha l}, \tag{9}$$

and matching score

$$R_{*mod}(\alpha,\beta) = \frac{1}{\alpha l}\left(\frac{I_2(\alpha,\beta)}{\alpha l} - \left(\frac{I_1(\alpha,\beta)}{\alpha l}\right)^2\right). \tag{10}$$

Cohen and Guibas retrieve matches for all segments (α,β) by using a topological sweep algorithm [23] 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 decided to probe only a discrete number of sample segments in Eqn. 10 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: the scale parameter α is varied between 0.25 and 2 at intervals of 0.05; the translation parameter β is sampled at intervals of 0.01 of the screen dimension.

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:

$$R := 1/(1 + w_1 Prox_{near} + w_2 Prox_{far} + w_3 R_{*mod})^2 \tag{11}$$

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

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

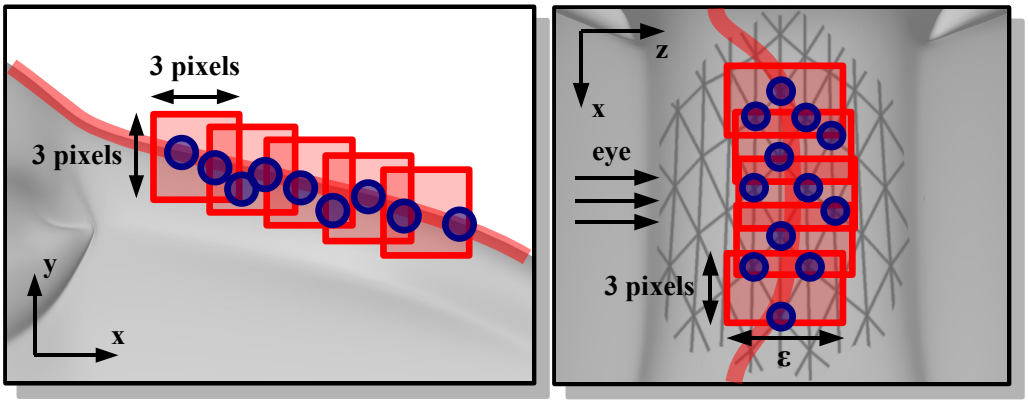

Fig. 15. Mesh vertices that are classified as handle members (blue circles) using one bounding volume (red box) for each image-space handle pixel. Left: view from the editor, right: view from top (silhouette indicated as a red line in both views).

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. 15). A mesh vertex is classified as a handle vertex if it lies in the union of these bounding volumes.

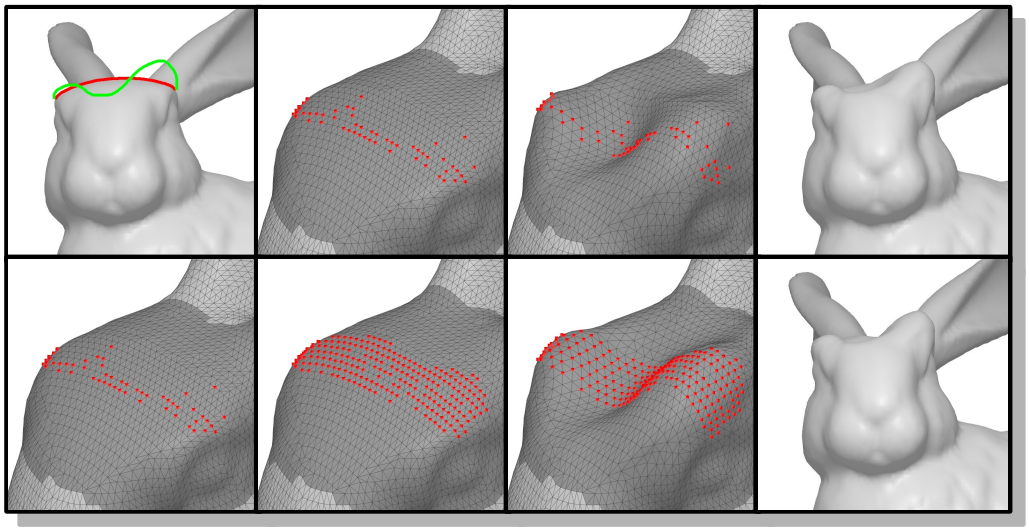

Fig. 16. Instead of deforming meshes solely based on the handle vertices that we retrieved by the bounding volumes (top row), we grow the set of handle vertices to select more vertices of the region described by the handle polyline (bottom row, please see text for details).

If a user wants to deform the profile of regions of low curvature or even planes, this approach only selects vertices in the foreground of such regions, since the bounding volumes introduce a tolerance ϵ in depth. It is clear that this issue cannot be solved by just using 2D bounding shapes (by omitting ϵ), since then we also might select vertices that are disconnected. Thus we grow the set of initially selected handle vertices by adding each vertex that a) is adjacent to a handle vertex, and b) whose position in image space lies within the respective projection of the bounding volumes (see Fig. 16).

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 arc length 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.

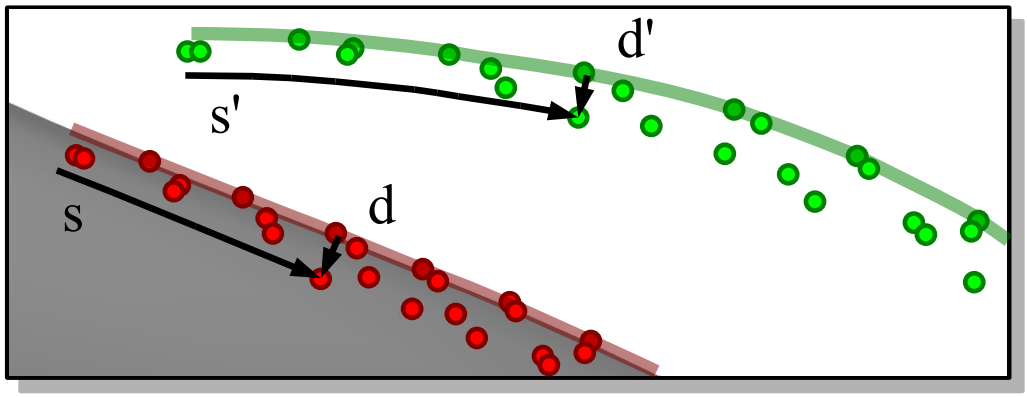

Fig. 17. Mapping of handle relative arc length position s and diplacement d (red) onto the target polyline (green).

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

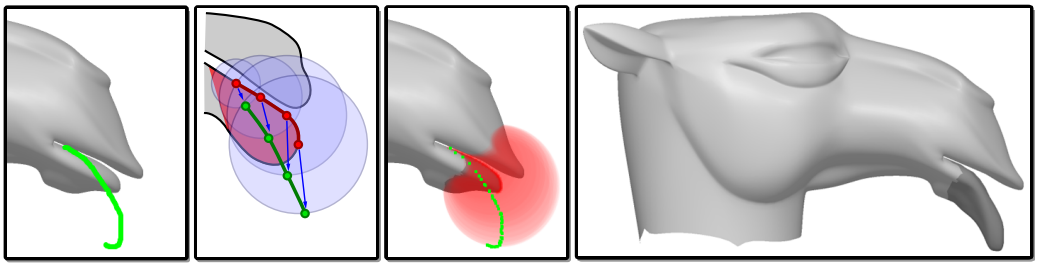

Fig. 18. Automatic ROI selection (from left to right): a) After the user places a sketch, the handle is estimated and correspondences are established. b) From these correspondences, the ROI is grown within the union of spheres, starting from the handle vertices (dark/red region, lower lip). c) Shows 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.

deformable material (Fig. 18), 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.

5 Results

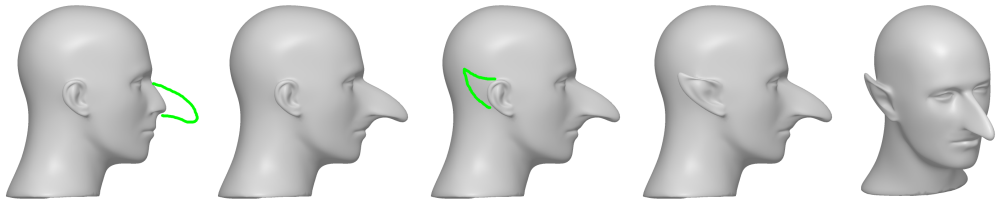

Fig. 19. The MANNEQUIN modeling session.

The modeling sessions shown in Fig. 19 and 20 illustrate the ease of use: after the user places a stroke, the system responds interactively, presenting a deformation which generally corresponds to the user's intent. All algorithmic details, which are shown for illustration purposes in various figures in this paper, are absent from the actual user interface.

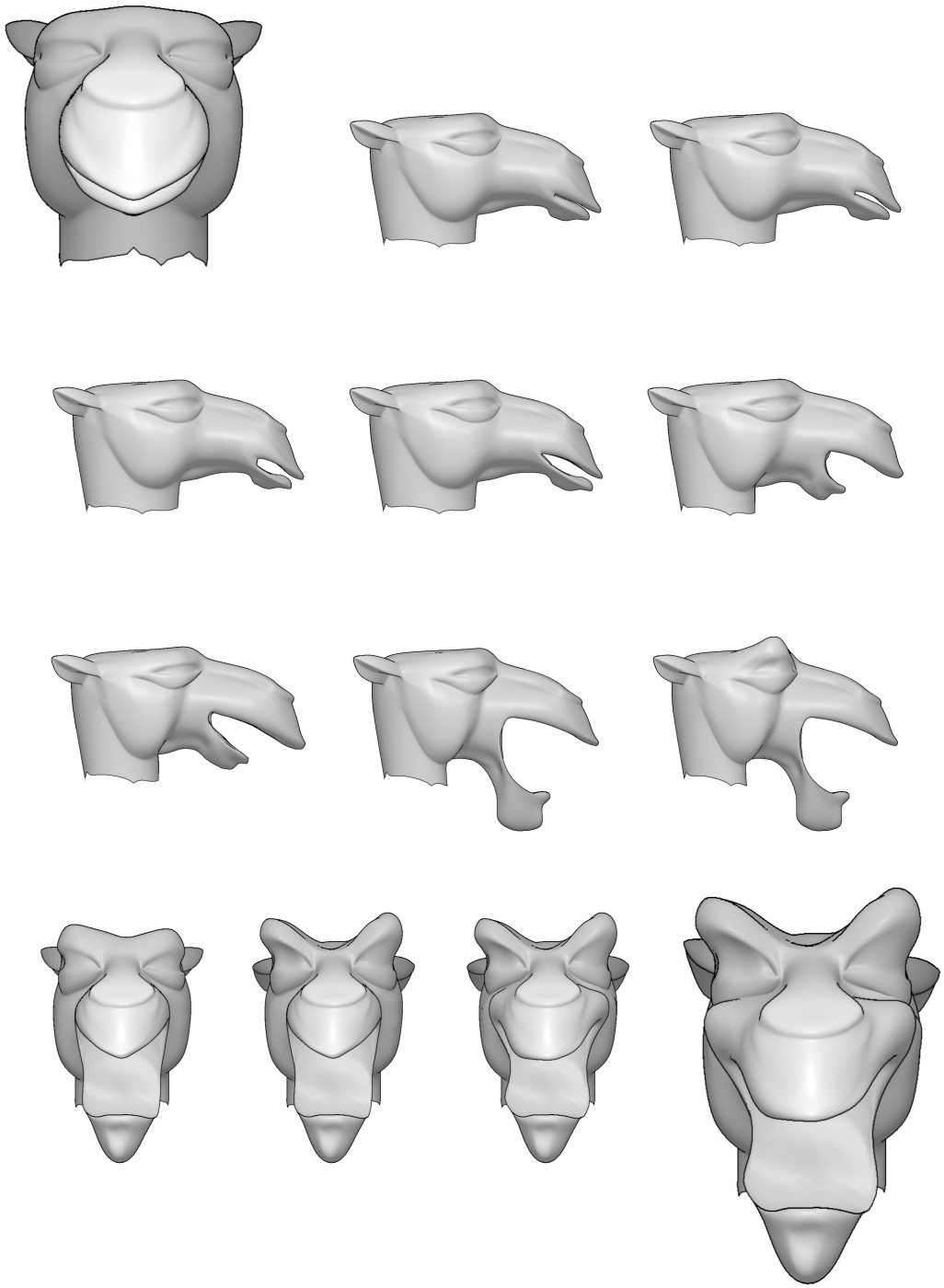

Fig. 20. Development of the FUNNY CAMEL. The modeling session took approx. 30 edits, which were applied from various viewing angles (not shown here).

Our first implementation used in [1] exploited the programmability of modern GPUs for edge extraction from the depth image. The resulting binary edge image was read back into system memory to perform the segmentation.

We have experimented with this part of the implementation and found that a CPU-only implementation has advantages: the access of data required in edge extraction and segmentation is easier, which enables optimizations in the segmentation code. The resulting implementation is faster and allows silhouette detection and extraction during navigation. Thus the user is always aware of the segmentation result, i.e. what are the potential feature lines to be modified with the stroke. Other minor improvements in the code have further decreased the computation time.

Model	Feature	Sil	Hnd	ROI	LSE	Sum	Size	Fig.
Bunny	Ear	6	261	9	365	641	5043^2	
CamelHead	Lip	4	113	2	98	217	1578^2	18
Mannequin	Nose	5	175	2	216	398	2040^2	19
	Ear	5	22	4	193	224	3786^2	19

Table 1
Timings of our system for several models. All times are in msecs, and refer to computing the silhouettes in the depth image, finding handle vertices, computing the region of interested, and then setting up and solving the LSE. Size refers to the system matrix.

Table 1 shows some timings obtained on a Intel Core 2 Duo 6600 processor with 2.4 GHz, 2GB memory and a two Nvidia 7900 GTX PCIe GPUs. To compare the performance of the current system to that of our first implementation, we remodeled roughly the same deformations as in [1], demonstrating a speed-up between 2.5 and 4.5. The improved extraction and segmentation of image-space silhouettes (column *Sil*) now takes only 1-5% of the processing time. Handle estimation and finding handle/target correspondence (column *Hnd*) depends on the density of silhouettes in the image, as well as the number of model vertices and is the dominating factor of the UI part of the approach. The column *Size* shows the dimensions of the sparse linear system, which is factored and solved every time the user places a new stroke. This is the other time consuming part and works interactively for ROIs up to a few thousand vertices. Of course we could also reuse the factorization as described in [3], however, this would require adding additional user interaction for choosing this option.

6 Discussion

Each of the steps in our approach presents a trade-off between fidelity and speed. While the requirement of interactive response certainly restricts the algorithmic possibilities, it is clear that almost all over-sketches are inherently ambiguous, even in the case of communication among humans – so it is unlikely that an algorithm will 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 normal discontinuities of of the surface, or even to more subtle feature lines such as suggestive contours [24] and apparent ridges [25]. 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.

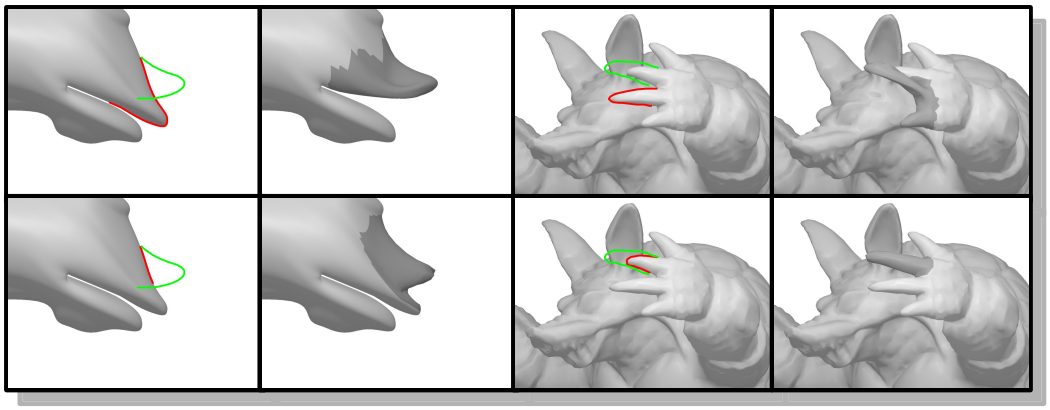

Fig. 21. Limitation of the handle estimation algorithm. Top row: handle estimations due to shape similarity and respective deformation results; Bottom row: handle estimations due to the proximity of polylines and respective deformation results.

We estimate the deformation handle by matching feature lines against the user-sketch using a scoring function that is a weighted sum of proximity and shape similarity terms. Although this works nicely in most cases, there are scenarios that show the limits of this approach with respect to the previously mentioned ambiguity of over-sketching. Consider Fig. 21: did the user intend to open the CAMEL's mouth by raising its upper lip, or simply want to extrude a new bumpy feature? Did the user intend to bend the longer finger or elongate the closer, shorter one in ARMADILLO's paw? In these cases, we might satisfy common intuition by using additional matching criteria (e.g. overshoots, rotation angles, etc.). Fig. 22 however illustrates that there are

scenarios that lack even any common intuition: we have performed a user study to find out if humans prefer matching based on proximity or on shape similarity in these ambiguous case. For the images presented in Fig. 22 we asked if they preferred matching the green stroke with the close straight segment or the angle, resulting in the deformations shown next to the illustration of the match. The surprising result is that our test subjects were undecided: roughly half preferred the extrusion (matching based on proximity) and the other half preferred the bending (matching based on shape similarity).

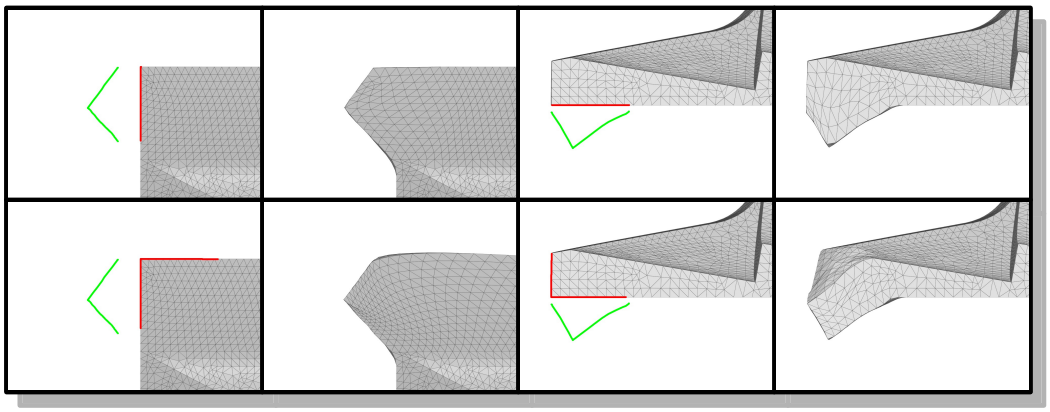

Fig. 22. Ambiguous user input: a user survey showed that one half of the interviewed users would expect an extrusion based on the proximity of the user-stroke and the silhouette of the model (top row), where the other half thought of a bending operation due to the similarity of the given shapes (bottom row). Interestingly, virtually all users applied their preference to both scenarios, so their decision can be considered to be independent of the saliency of the silhouettes in these scenarios.

Many experts have commented on our decision assuming that the depth values are constant while the positions in the image plane change based on the user's stroke: depth values should rather be consistent with the model's features than moved parallel to the image plane, which is arbitrary and not model-dependent.

We have experimented with the idea of approximating depth values for the handle vertices based on a PCA in the set of modified vertices. We assume a model-oriented deformation should retain the local relative co-variances, i.e. the ratios of eigenvalues of the co-variance matrix. In other words, features should rather scale than bend. This is in contrast to what we now call view-oriented deformation, where coordinates only change in the directions that are visible to the user, and not in depth. Figure 23 illustrates the two approaches.

We have performed a user study to evaluate which approach leads to deformation results that are perceived as natural given the user sketch. We first introduced the subjects to the navigation part of the interface and explained

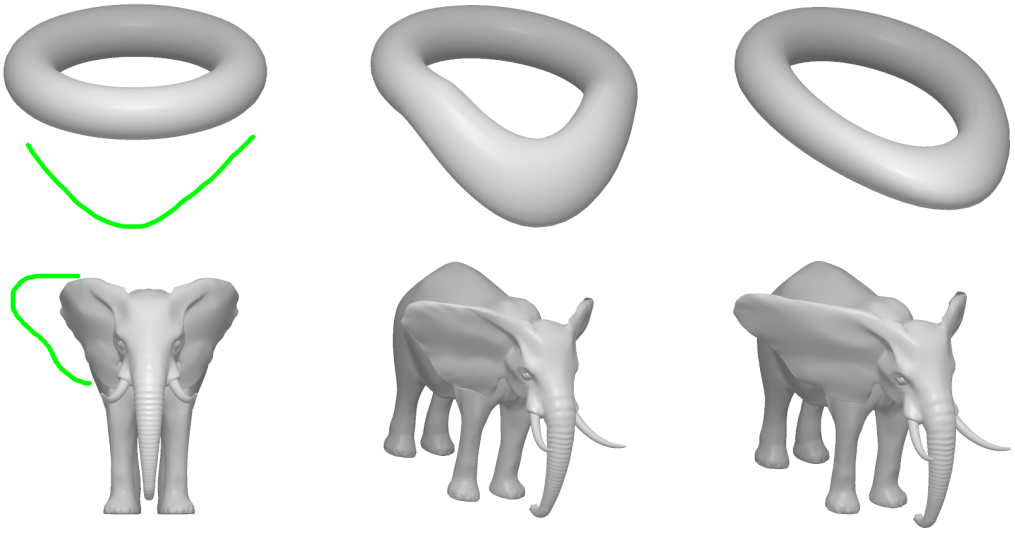

Fig. 23. View-oriented deformation vs. model-oriented deformation. Left column: deformation sketch; Middle column: view-oriented deformation; Right column: model-oriented deformation.

that the stroke they see leads to a deformation that is consistent in the view of the stroke. The we presented five different scenarios, each consisting of the picture of a model with a deformation sketch, as well as two 3D models as possible deformation results (see Fig. 23): one of these models was based on view-oriented deformation, the other on model-oriented deformation as described above. Without describing or explaining the idea behind the two different variants of the deformation, we asked the subjects to inspect the results and determine the ones they would expect given the stroke.

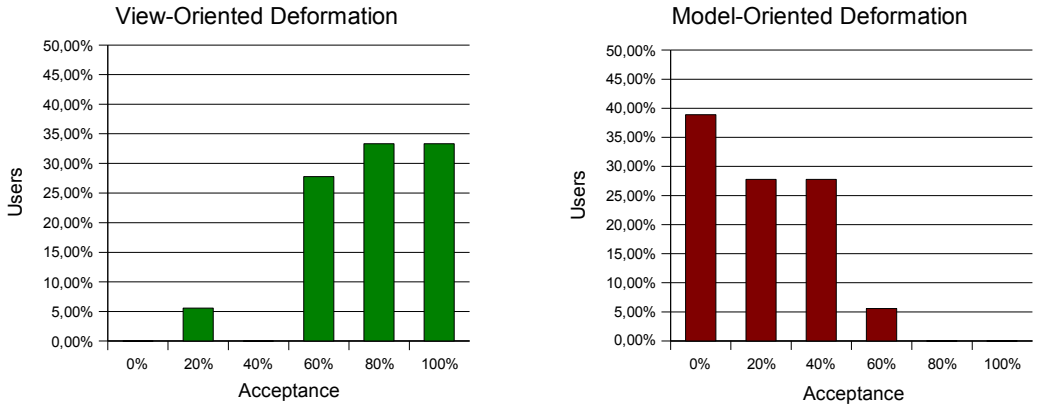

Fig. 24. Distribution of the user acceptance of the view-oriented deformation (left, mean acceptance $= 78 \pm 22\%$), and the model-oriented deformation (right, mean acceptance $= 20 \pm 19\%$).

Interestingly, the view-oriented deformation was preferred in 70 of 90 cases, whereas the model-oriented deformation was chosen only 18 times (no preference in 2 cases). For each subject we determine a preference for either one of the approaches and then consider the accumulated subject results (see Fig. 24): $78 \pm 22\%$ prefer the view-oriented deformation . A common statement during this user study was that "*I don't want to change the model in a direction that I am not aware of*", which explains the results and also shows that users intuitively assume a certain strategy for using the tool, namely first rotating the model so that they only need to change features within the image plane.

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.

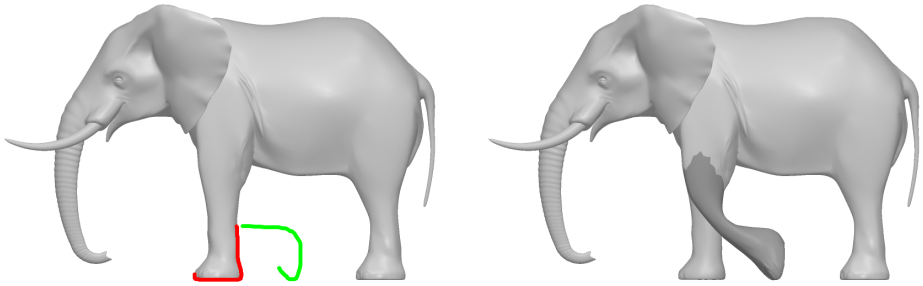

Fig. 25. Limitation of the deformation algorithm. Left: user-sketch of a bent leg. Right: unnatural deformation of the ELEPHANT's leg.

Looking at the deformation example in Fig. 25 (right), it is clear that LSE is not a universally applicable deformation tool. However, it should be feasible to incorporate the information gathered by the handle estimation (such as rotation and scale of the best handle match) in the deformation step. If – for instance – the best match corresponds to a considerable rotation of high similarity and a negligible change in scale (see Fig. 25, left), it might be feasible to deduce a rotation, whose axis describes a transition region between the static part of the mesh and the ROI that is to be rotated.

Our implementation of LSE is currently based on common weights for Laplacians, static anchors, and the transformed handle coordinates. Here, it might be interesting to introduce individual weights for each single constraint. These might be derived from additional information deduced from the user-sketch - e.g. stroke timings or even information on pressure of an input device along the stroke's path. Moreover, the system is almost generic with regard to the type of the deformation tool, and it would be very interesting to also try this approach in other settings.

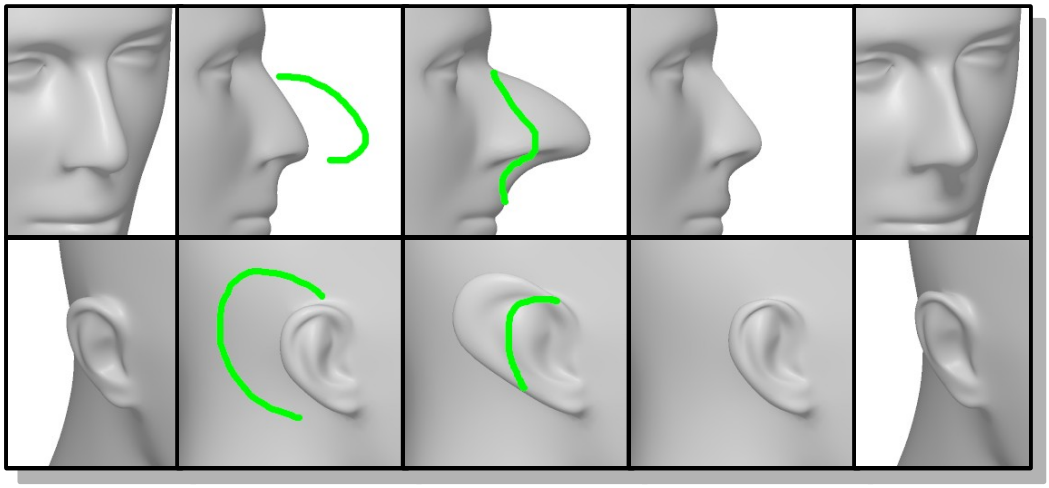

Fig. 26. Lack of idempotence. We tried to reconstruct the original shape of a previously deformed nose (top row), respectively ear (bottom row), by sketching its original silhouette: the system downsized both entities, but the original shape is not perfectly recovered (please see text for details).

Although our system is able to downsize model components, it lacks the property of idempotence for at least two reasons: even if a user manages to draw exactly the original silhouette of a previously deformed component, our system introduces errors by establishing correspondences between image- and object-space entities (see section 4.3). Secondly, errors introduced by subsequent applications of the mesh deformation tool [2] are more likely to accumulate than to neutralize each other (consider Fig. 26). Thus, it is highly recommended to incorporate an UNDO function into any implementation of our system.

Considering the whole system as a pipeline, we describe a series of fairly independent components. We believe that individual components might be refined or replaced using other approaches, however, the system as it is presented here is a proof-of-concept for automated sketch-based editing of surface meshes on the basis of image-space silhouettes.

References

[1] Johannes Zimmermann, Andrew Nealen, and Marc Alexa. Silsketch: Automated sketch-based editing of surface meshes. Accepted for publication in proceedings of the 4th Eurographics Workshop on Sketch-based Interfaces and Modeling, 2007.

[2] Olga Sorkine, Yaron Lipman, Daniel Cohen-Or, Marc Alexa, Christian Rössl, and Hans-Peter Seidel. Laplacian Surface Editing. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry processing*, pages 179–188, 2004.

[3] Andrew Nealen, Olga Sorkine, Marc Alexa, and Daniel Cohen-Or. A sketch-based interface for detail-preserving mesh editing. *ACM Trans. Graph.*, 24(3):1142–1147, 2005.

[4] Mario Botsch, Mark Pauly, and Markus Gross. PriMo: coupled prisms for intuitive surface modeling. In *Eurographics Symposium on Geometry Processing*, pages 11–20, 2006.

[5] Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Mesh editing with Poisson-based gradient field manipulation. *ACM Trans. Graph.*, 23(3):644–651, 2004.

[6] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A Sketching Interface for 3D Freeform Design. In *Proceedings of SIGGRAPH*, pages 409–416, 1999.

[7] Levent Burak Kara and Kenji Shimada. Sketch-Based 3D-Shape Creation for Industrial Styling Design. *IEEE Computer Graphics and Applications*, 27(1):60–71, 2007.

[8] Vladislav Kraevoy, Alla Sheffer, and Michiel van de Panne. Contour-based Modeling Using Deformable 3D Templates. Tech Report, 2007. `http://www.cs.ubc.ca/cgi-bin/tr/2007/TR-2007-13.pdf`.

[9] Youngihn Kho and Michael Garland. Sketching mesh deformations. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, pages 147–154, 2005.

[10] Donald D. Hoffman and Manish Singh. Salience Of Visual Parts. *Cognition*, (63):29–78, 1997.

[11] Aaron Hertzmann. Introduction to 3D Non-Photorealistic Rendering: Silhouettes and Outlines. In *Non-Photorealistic Rendering. SIGGRAPH 99 Course Notes.*, 1999.

[12] Dana H. Ballard and Christopher M. Brown. *Computer Vision*. Prentince-Hall Inc., 1982.

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

[14] Nicolai Petkov and Martijn B. Wieling. Canny edge detector (Canny filter) for image processing and computer vision. Web Application. `http://matlabserver.cs.rug.nl/cannyedgedetectionweb/web/index.html`.

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

[16] Scott D. Cohen and Leonidas J. Guibas. Partial Matching of Planar Polylines under Similarity Transformations. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, 1997.

[17] Sagi Katz and Ayellet Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. In *ACM SIGGRAPH*, pages 954–961, 2003.

[18] Zhongping Ji, Ligang Liu, Zhonggui Chen, and Guojin Wang. Easy Mesh Cutting. *Computer Graphics Forum*, 25(3):283–291, 2006.

[19] Adam Hoover, Gillian Jean-Baptiste, Xiaoyi Jiang, Patrick J. Flynn, Horst Bunke, Dmitry B. Goldgof, Kevin K. Bowyer, David W. Eggert, Andrew W. Fitzgibbon, and Robert B. Fisher. An Experimental Comparison of Range Image Segmentation Algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(7):673–689, 1996.

[20] Kari Pulli and Matti Pietikäinen. Range image segmentation based on decomposition of surface normals. In *8th Scandinavian Conference on Image Analysis (SCIA93)*, Tromso, May 1993.

[21] Uwe Hahne. *Weighting in Laplacian Mesh Editing*. May 2006. `http://www.cg.tu-berlin.de/fileadmin/webdav/hahne/diploma/diploma.pdf`.

[22] Yaron Lipman, Olga Sorkine, Daniel Cohen-Or, and David Levin. Differential Coordinates for Interactive Mesh Editing. In *International Conference on Shape Modeling and Applications*, pages 181–190, 2004.

[23] Herbert Edelsbrunner and Leonidas J. Guibas. Topologically sweeping an arrangement. In *STOC '86: Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 389–403, 1986.

[24] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. Suggestive contours for conveying shape. *ACM Trans. Graph.*, 22(3):848–855, 2003.

[25] Tilke Judd, Frédo Durand, and Edward H. Adelson. Apparent ridges for line drawing. *ACM Trans. Graph.*, 26(3):19, 2007.