Eurographics Symposium on Geometry Processing (2003)
L. Kobbelt, P. Schröder, H. Hoppe (Editors)

Mesh Forging: Editing of 3D-Meshes Using Implicitly Defined Occluders

G. H. Bendels† and R. Klein‡

University of Bonn,
Institute of Computer Science II, Computer Graphics,
Römerstrasse 164, 53117 Bonn, Germany

Abstract

In recent years the ease of use and the flexibility in the editing process shifted into focus in modelling and animation applications. In this spirit we present a 3D mesh editing method that is similar to the simple constrained deformation (scodef) method [9]. We extend this method to the so-called mesh forging paradigm by adding an occluder to the editing environment. Our method resembles and was in fact motivated by the forging process where an anvil is used to give the manipulated object the desired shape. While users perform the editing operation by directly manipulating the 3D-mesh, the occluder is defined implicitly. To enable fine detail edits even in sparsely triangulated areas, we propose an adaptive refinement method that also allows the creation of sharp features where desired. The functionality and ease of use of our editing approach is shown by several examples.

Categories and Subject Descriptors (according to ACM CCS): I.3.8 [Computer Graphics]: Applications; I.3.4 [Graphics Utilities]: Graphics Editors; I.3.5 [Computational Geometry and Object Modelling]: Object Modelling

1. Introduction and Previous Work

Traditionally, the goal of research in geometric modeling and design was to find representations of 3D objects and modeling tools enabling a possibly intuitive and efficient design process, the choice of the former usually influencing the range of choices for the latter and vice versa. A prominent example of such *free form surface design* methodologies are the piecewise polynomial tensor product surfaces. This representation reflects a balance of certain specific demands. On the one hand, there are the industrial requirements of enforcing some constraints on the result, e.g. smoothness, manufacturability, etc. On the other hand, the designer should be able to work efficiently, which calls for intuitivity and simplicity, especially in the early stages of design. As geometric modeling finds applications in a wide spectrum of computer graphics applications, the focus of much research effort shifts towards the latter. Originally, the needs of the industry imposed requirements on geometric design methodology that made it cumbersome and time consuming. In case of tensor product surfaces, even the changes appearing conceptually small are hard to execute. However, exact and provable geometric properties of the product are not relevant in numerous current application areas like film industry, computer games, etc. As the most relevant quality measure in these contexts is the visual appeal, much of the research focusses on finding new modeling metaphors and techniques according to this criterion [35].

In case of implicit deformations, the objects are embedded in a space. The warping of these spaces causes a corresponding deformation of the object. According to the most popular such approach called Free Form Deformations (FFD) [3, 37], the surrounding space is defined as a multidimensional spline. This technique has been further improved and generalized [16], moreover, it was adapted to generate animations by e.g. [16, 17]. The main advantage of these methods is that they enable a deformation that is independent of the complexity of the object being manipulated. However, as pointed out in

† bendels@cs.uni-bonn.de
‡ rk@cs.uni-bonn.de

[23], the placement and control of the lattice defining the deformation is non-trivial. The improvements of [23, 24] remedied these problems for the case of a single edit. In this system the user has to define only the initial control lattice, which is then modified transparently to the user as she edits the object by dragging a point on the surface. However, since the control points move according to the user's input, the result of a subsequent edit depends on a new control lattice, and may thus be contra-intuitive, cf. [19], as it is different from an identical edit in an untouched region.
A prominent example of techniques relying on piecewise polynomial representations is the classical NURBS modeling approach [18]. It reflects the idea of imposing a control mesh directly on the surface of the object, the desired modeling effect is then achieved by modifying the control points of this mesh. However, since there are few degrees of freedom in determining the scale of an edit and the topology of the control mesh, working with NURBS requires much expertise and is time intensive.

The algorithmic generalization of piecewise polynomial representations are the subdivision surfaces [15, 14], where a smooth surface is generated by iterative refinement of a control mesh according to certain subdivision rules. Since subdivision surfaces essentially comprise a representation of an object on different levels of scale, they may be utilized as a basis for Multiresolution Mesh Editing (MME) [41]. The idea of multiresolution editing is to use different levels of detail of the object to perform edits on different scales: Detail edits are performed on finer meshes and large scale edits on coarser meshes representing the same object. Saving the finer meshes as details with respect to the coarser meshes provides for detail preservation during large scale edits. Unfortunately, since the one-ring of the edited vertex defines the region of influence (ROI) of an edit, the subject of the edit and the ROI are completely connectivity-defined and cannot be chosen arbitrarily. Kobbelt et al. [27, 28] provided a solution to this problem by abandoning the idea of defining the multiresolution representations as a coarse-to-fine hierarchy of nested meshes, and rather utilizing a hierarchy of scales in terms of smoothness. The levels of detail are generated by a hierarchical mesh smoothing scheme. During the editing process the designer picks a region of interest defining the scale of the edit, and a handle, i.e. another region within the ROI that undergoes the user-defined transformation. Each time the user transforms the handle, the mesh within the ROI transforms according to a constrained energy minimization principle. This latter modeling metaphor stems from the area of variational modeling [39] where the shape of a region of the surface is a solution to a constrained optimization problem. Thus, the basis function of the deformation is completely determined by the shape of the handle and the boundary of the ROI.
In the above algorithms the user picks and specifies a transformation for parts of the object. Into this type of *direct editing*, further research has been done. Lee [29] proposes a method where the user picks a set of handle vertices in the mesh and specifies modifications for these. For vertices in the editing region, which is defined by the user beforehand, the transformations of the handle vertices are interpolated using multilevel B-Splines. These are parameterized over a 2D embedding of the editing region, making this method suitable especially in flat regions. The influence of the handle vertices' transformation on neighboring vertices is determined by the size of the coarsest control lattice used in the B-spline interpolation.
Recently, Pauly et al. [33] presented a modelling technique, transferring multiresolution results to the case of point-sampled geometry. In their setting, shapes are modified by defining a so-called *zero-region* and a *one-region*. The one-region undergoes the full user-defined transformation (translation or rotation), whereas the zero-region remains fixed and a predefined blending function is used to create a smooth transition between the two regions. Also focussing on point-sampled geometry Zwicker et al. [42] generalize standard 2D image editing techniques to 3D, reconstructing well-known pixel editing tools.
While the methods mentioned so far mainly apply to modifying existing shapes, SKETCH[40], SKIN [30] and TEDDY [25], which were later extended in [26], are among those methods generating models from scratch. Modifying shapes in these approaches is realised using a method called oversketching, i.e. drawing parts of the silhouette of the shape anew.

Although interactive display of *implicit surfaces* [5] is still a challenge, implicit modeling has gained more and more research attention in recent years. Distance surfaces like *blobs* [4], *meta-balls* [10], *soft objects* [7], and *convolution surfaces* [6] are popular in Computer Animation since the geometric "skeleton", with respect to which they are defined, can be used as an internal structure to control the animation [11] and even for LOD-representations[13, 1]. Several methods have been presented to tackle the so-called "unwanted blending problem", e.g. [2]. Furthermore, the availability of inside-outside information allows for efficient collision detection and response [32].

Of particular relevance for this paper is the work of Borrel et al. presented in [8, 9] and later extended in [36]. The algorithm proposed in this paper is similar in the way the displacement of vertices in the neighbourhood of user-defined handles are computed but is extended to incorporate sharp features where desired. In addition to that, we use geodesic distances to define an object-inherent parametrization for our shape functions, thus freeing the user of the need to adjust object-independent region of influence definitions as in e.g. [36].
Anisotropic parameterizations are feasible through multiple, handle-independent *anchors*. Moreover we introduce editing occluders, i.e. implicitly defined 3D-objects that influence the editing operations. With these editing occluders we define a novel editing paradigm resembling the forging process where an anvil is used to give an object the desired shape.

The main contributions of this paper are a closed formulation for the editing method including the occluder field influence. By representing the model to be deformed as a triangular mesh, while the occluder is defined implicitly, our approach takes advantage of both representations respectively, in particular the efficient collision detection. Moreover, we transfer the concept of *Precise Contact Modeling* (PCM) to the setting of triangular meshes.

The rest of this paper is organized as follows. In section 2, we give an overview over the editing process, describe the underlying mathematical formulation and the influence of shape functions. Section 3 introduces what we call the mesh forging process. The adaptive refinement method proposed for the shape function edits is described in section 4. Results and examples of models manipulated with our tool are given in section 5. This section also gives conclusions and discusses future directions of research.

2. Editing Process

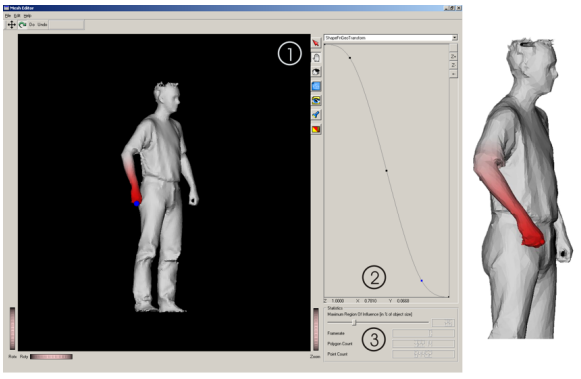

Figure 1: *The user interface with an exemplary arm movement. The model was reconstructed from a laser range scan. On the right hand side we see the modified model after applying a translation modification based on geodesic distances. The arm is lifted without affecting the torso. Note the detail preservation leading to the the realistic folds of the sleeve.*

In this section we will describe how the user's specification of the modification for a set of handle vertices is used to produce a smooth, detail-preserving and intuitive deformation of the whole mesh. Due to the similarity to our method and since it is not widely known, we briefly review the method presented in [36] in section 2.1. In section 2.2 we describe how this methodology can be extended to rotations, while the rest of the section focusses on defining suitable parameterizations of the editing region.

2.1. Translations

We first consider the case where a user specifies a translation for the handle vertices. The main idea is that every geometric modification of a 3D shape can be interpreted as a displacement function

$$\mathbf{d} : \mathbb{R}^3 \to \mathbb{R}^3 \tag{1}$$

that assigns to every point $\mathbf{p} \in \mathbb{R}^3$ a displacement vector $\mathbf{d}(\mathbf{p})$ such that the resulting point positions after the modification are given as

$$\mathbf{p}_{new} = \mathbf{p}_{old} + \mathbf{d}(\mathbf{p}_{old}). \tag{2}$$

In our setting the values of this displacement function (also referred to as *constraints* [9]) are defined at the handle vertices only. For all other vertices the mapping has to be determined.

The idea to solve this problem is to write the displacements of all vertices as a weighted sum of virtual displacement vectors for the handle vertices – which we call *partial displacements* $\mathbf{d}_j$ in contrast to the *total displacements* $\mathbf{d}(\mathbf{p})$

$$\mathbf{d}(\mathbf{p}) = \sum_{j=1}^{k} \alpha_j(\mathbf{p})\, \mathbf{d}_j. \tag{3}$$

Here k is the number of handle vertices and

$$\alpha_j : \mathbb{R}^3 \to \mathbb{R}, \quad j = 1, \ldots, k$$

are weight functions. Note that for each $\mathbf{p}$, $\alpha_j(\mathbf{p})$ can be interpreted as a special weight corresponding to the handle vertex $\mathbf{p}_j$.

The above formulation allows for the desired degree of freedom for choosing the set of handle vertices and the properties of the edit. Please note that simply setting $\mathbf{d}_j = \mathbf{d}(\mathbf{p}_j)$ is not a satisfying choice, as becomes clear when we consider the case where the ROI of different handle vertices overlap in a way such that $\alpha_j(\mathbf{p}_i) \neq 0$ for a pair of handle vertices $\mathbf{p}_i, \mathbf{p}_j$, $i \neq j$. In this case $\mathbf{p}_i$ undergoes (in addition to its own transformation) a transformation induced by the handle vertex $\mathbf{p}_j$. This would render the handle vertices moving to positions different than defined by the user, unless we impose strict normalization conditions on the weight functions, which, in turn, would prohibit the use of arbitrary and user-defined weight functions. Therefore, we calculate the partial displacements *according to* the weight functions.

Since, by user-definition, $\mathbf{d}(\mathbf{p}_j)$ is known for the handle vertices $\mathbf{p}_1, \ldots, \mathbf{p}_k$, equation (3) leads to a linear system of equations

$$\Big(\mathbf{d}(\mathbf{p}_j)\Big)_{j=1,\ldots,k} = \mathbf{A} \Big(\mathbf{d}_j\Big)_{j=1,\ldots,k} \tag{4}$$

with

$$\mathbf{A} = \Big(\alpha_j(\mathbf{p}_i)\Big)_{\substack{i=1,\ldots,k \\ j=1,\ldots,k}} \tag{5}$$

giving us $3k$ equations for the total displacements $\mathbf{d}(\mathbf{p}_j)$ with $3k$ unknowns $\mathbf{d}_j$, $1 \leq j \leq k$.

Of course, ill-conditioned weight function choices (such as $\alpha_j(\mathbf{p}_j) = 0$) might leave the matrix $\mathbf{A}$ singular or close

to singular. This can easily be avoided using either a SVD for detecting and prohibiting those ill-conditioned modifications of the shape function or ROI, or a pseudo-inverse as suggested in [34]. For a detailed discussion cf. [9].

After solving (4), (3) is used again to compute the total displacements for the other vertices.

Please note that for a single handle vertex $\mathbf{p}_0$, (3) reduces to

$$\mathbf{d}(\mathbf{p}) = \alpha(\mathbf{p})\,\mathbf{d}_0$$

with $\mathbf{d}_0 = 1/\alpha(\mathbf{p}_0)\,\mathbf{d}(\mathbf{p}_0)$, leading to a very efficient formulation for the frequent case of single handle vertex edits.

2.2. Rotations

As stated above, every transformation can be interpreted as a displacement field, and thus even rotation-like modifications of the object (like turning a person's head) are in theory possible with the above formulation. In general, to achieve satisfying results, this would require a considerable number of handle vertices, though. Therefore we propose using a different kind of constraints for rotational editing operations. Instead of defining *total displacements* for the handle vertices, the user defines *total rotations* $\eta_1,\ldots,\eta_k$ with respect to an axis $\mathbf{n}$.

In our current implementation, the rotation axis is defined by the screen center and the viewing direction. Analogously to the displacement field in the translation case, we define a rotation field

$$\eta : \mathbb{R}^3 \to \mathbb{R}$$

that assigns to every point $\mathbf{p} \in \mathbb{R}^3$ a rotation angle $\eta(\mathbf{p})$ such that the resulting point positions after the modification are given as

$$\mathbf{p}_{new} = \mathbf{R}(\eta(\mathbf{p}_{old}),\mathbf{n})\,\mathbf{p}_{old}, \tag{6}$$

where $\mathbf{R}(\eta,\mathbf{n})$ is the rotation matrix that rotates the space by an angle of η about the axis $\mathbf{n}$. As in the translation context, the values of this rotation map are defined at the handle vertices only. For the other vertices the mapping has to be determined as above using

$$\eta(\mathbf{p}) = \sum_{j=1}^{k} \alpha_j(\mathbf{p})\eta_j$$

with *partial angles* η_j and *total angles* $\eta(\mathbf{p})$.

2.3. Parametrizations and Shape Functions

A very simple approach for a weight function could be for example $\alpha_j(\mathbf{p}_i) = \delta_{ij}$ leading to $\mathbf{d}(\mathbf{p}_j) = \mathbf{d}_j$ and $\eta(\mathbf{p}_j) = \eta_j$.

Note the similarity of this formulation to standard scattered data interpolation problems. The above choice of weight functions would give us the generally unsatisfying approach that moves the handle vertices as specified and leaves all other vertices unchanged. Instead, we define the weight functions to be a composition

$$\alpha_j(\mathbf{p}) = \varphi \circ \gamma_j(\mathbf{p})$$

of a *shape function*

$$\varphi : \mathbb{R}^{\geq 0} \to \mathbb{R} \tag{7}$$

and a *parametrization* of the object

$$\gamma_j : \mathbb{R}^3 \to \mathbb{R}^{\geq 0}. \tag{8}$$

The mathematical framework presented in the previous sections is applicable with any distance function $\gamma\colon \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}^{\geq 0}$ if we define

$$\gamma_j(\mathbf{p}) = \gamma(\mathbf{p}_j,\mathbf{p}). \tag{9}$$

In order to achieve intuitive results however, γ should be chosen such that it defines intuitive neighborhoods on the object.

Although appropriate in some occasions, choosing Euclidean distances

$$\gamma(\mathbf{p},\mathbf{q}) = \|\mathbf{p}-\mathbf{q}\|$$

as in [38] would make it virtually impossible, for example, to bend a person's index finger without interfering with the other fingers.

We propose using *geodesic* distances instead, i.e. the length of the shortest curve between $\mathbf{p}$ and $\mathbf{q}$ on the boundary of the object. Using definition (9), the object is thus parameterized via geodesic distance fields with respect to the handle vertices $\mathbf{p}_j$.

The impact of this choice on the resulting editing operations is visualized in figure 3(a) <see color section> that shows an editing operation applied to a simple triangle mesh representation of a hand. Pictures (1) and (3) show the original mesh with the region of influence colored in red. Pictures (2) and (4) show the modified meshes, after an identical edit has been performed (based on Euclidean distances in the top row and on geodesic distances in the bottom row). Note how the middle and ring finger have been deformed in picture (2), whereas in picture (4) only the index finger has been modified (as desired in this example).

Raffin et al. [36] propose user-definable *hulls of influence* surrounding the parts of the object that should be affected to achieve the above results, but we feel that the geodesic distances provide for a useful *object-inherent* parametrization for the modification of surfaces, and therefore lead to a more convenient user interface, freeing the user from the need to fit hulls of influence to the specific editing situation (which might be difficult, if the fingers e.g. are very close).

To efficiently compute the geodesic distances between the handle vertices and all other vertices, our implementation uses the algorithm from [31]. Nevertheless this is computationally nontrivial, but it does not prevent interactive response,

since it is performed only when the vertices are selected or deselected, not during dragging.

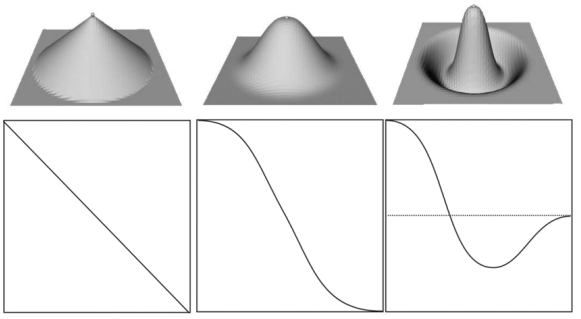

Figure 2: *Different shape function settings applied to the same editing operation*

Shape Functions (cf. figure 2) provide for more flexibility and degrees of freedom than e.g. the trivariate Bernstein polynomials used in FFD methods or the parameterized Gaussian functions used in [38].

In many cases, choosing smooth shape functions will be sufficient, but in other cases the user might want to introduce sharp features into the edited area. This can easily be achieved using our approach by employing the appropriate shape function, given that the triangulation of the underlying mesh is adequately fine (see section 4).

2.4. Separating Handles and Anchors

The method presented so far relies on an object parametrization with respect to the handle vertices. Whereas this results in an intuitive and easy-to-use tool, there is no theoretical obligation to identify the *handles* used to define the total transformations with the *anchors* used to define the object parametrization. In some occasions, it might be desirable to parameterize the object with respect to other vertices than the handles. As an example, one can think of turning a person's head (with a rigid head and a smoothly twisted neck) while the rest of the body remains unchanged down from the shoulders (see figure 3(b) in the color section). In this case it is useful to define the parametrization with respect to anchor vertices at the top of the head, while the handle vertex can be picked somewhere else on the mesh.

Additionally, separating handles from anchors has another advantage: We can extend this line of thought to a *multiple anchors - single handle*-approach, i.e. the parametrization of the object is defined with respect to a set of anchor vertices $\mathbf{a}_1^j,\ldots,\mathbf{a}_l^j$ rather than to a single anchor vertex. Thus we are able to define anisotropic distance fields on the object, freeing us from the limitation of rotationally symmetric parameterizations. In our current implementation, the distance field defining the parametrization corresponding to handle vertex $\mathbf{p}_j$ is then defined for every mesh vertex $\mathbf{p}$ as

$$\gamma_j(\mathbf{p}) = \min_{1\le i\le l} \gamma(\mathbf{a}_i^j, \mathbf{p}). \tag{10}$$

It is known that the iso-values of $\gamma_j(\mathbf{p})$ in equation (10) do not form smooth curves. But implementing known techniques from implicit modelling (e.g. following the ideas of Convolution Surfaces from [5]) into our setting is a straightforward extension leading to smooth distance fields.

3. Mesh Forging Process

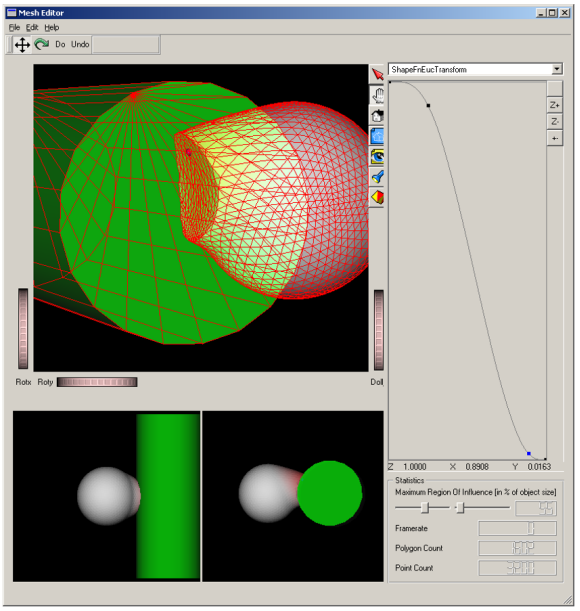

Figure 3: *Example of a mesh forging operation. A vertex of the editing object (the grey coloured ball) is picked and dragged. The occluder (the green cylinder) induces a force field that superposes the displacement field and drives the transformed vertices around it.*

The basic idea of mesh forging is to add an *occluder* (the anvil) to the editing space in form of a force field, thereby replicating the *Precise Contact Modelling* (PCM) methodology (see below) in the context of mesh editing. Here, the occluder field controllably superposes (and thereby modifies) the transformation applied to the vertices of the mesh (cf. figure 3 for an example).

In implicit modelling, contact situations between two surfaces

$$S_i = \{\mathbf{p}\in\mathbb{R} \mid f_i(\mathbf{p}) = c\}, \qquad i\in\{1,2\}$$

are easily detected by checking for points $\mathbf{p}$ satisfying both $f_1(\mathbf{p})\le c$ and $f_2(\mathbf{p})\le c$. For these points in the interpenetration region, a *compression term* is added to the field function

f_i. If only S_1 is deformable and S_2 rigid, $f_1(\mathbf{p})$ is replaced by

$$c + (c - f_2(\mathbf{p}))$$

for all $\mathbf{p}$ in the interpenetration region.
In order to mimic volume preservation in the contact regions, a *dilation term* $b(\mathbf{p})$ is added for points $\mathbf{p}$ in the propagation region

$$\{\mathbf{p} \in \mathbb{R} \mid \tilde{c} \geq f_2(\mathbf{p}) > c\}$$

with some constant $\tilde{c}$. For an in-depth description of the PCM methodology, see [12, 32].

3.1. The Algorithm

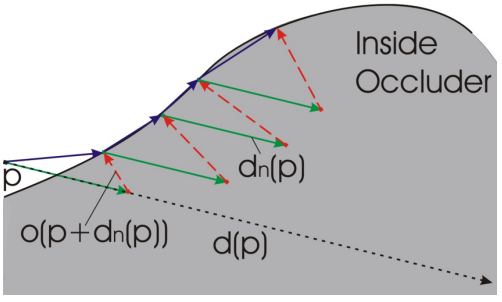

Figure 4: *Successive occluder influence. The stippled vector indicates the vertex transformation* $\mathbf{d}(\mathbf{p})$ *as defined by the editing operation, the green vectors represent the n-th part* $\mathbf{d}_n(\mathbf{p}) = 1/n\ \mathbf{d}(\mathbf{p})$ *of this transformation. The dashed red vectors indicate the repelling force due to the occluder field. Note how the considered vertex* $\mathbf{p}$ *moves along the boundary of the occluder leading to the expected editing behaviour.*

Let us start with an example to motivate our algorithm. Suppose that in a mesh editing environment, we have a vertex $\mathbf{p} \in \mathbb{R}^3$ in the mesh and, defined by some modelling operation, a total displacement $\mathbf{d}(\mathbf{p})$ for this vertex. In order to detect collisions even for larger edits, we first subdivide $\mathbf{d}(\mathbf{p})$ into N pieces – otherwise it would be possible to move *through* the occluder or through occluder details. Our algorithm then leads to the following sequence of transformations (see figure 4):

$$\begin{aligned}
\mathbf{p} &\mapsto \mathbf{p}^1 = \mathbf{p} + \tfrac{1}{N}\,\mathbf{d}(\mathbf{p}) + \mathbf{o}(\mathbf{p} + \tfrac{1}{N}\,\mathbf{d}(\mathbf{p})) \\
\mathbf{p}^1 &\mapsto \mathbf{p}^2 = \mathbf{p}^1 + \tfrac{1}{N}\,\mathbf{d}(\mathbf{p}) + \mathbf{o}(\mathbf{p}^1 + \tfrac{1}{N}\,\mathbf{d}(\mathbf{p})) \\
&\vdots \\
\mathbf{p}^{N-1} &\mapsto \mathbf{p}^N = \mathbf{p}^{N-1} + \tfrac{1}{N}\,\mathbf{d}(\mathbf{p}) + \mathbf{o}(\mathbf{p}^{N-1} + \tfrac{1}{N}\,\mathbf{d}(\mathbf{p}))
\end{aligned} \tag{11}$$

Here, $\mathbf{o}(\mathbf{p})$ is the occluder field and corresponds to the compression term in the implicit modelling context. To describe the editing process in general, we get the following recursive algorithm:

$$\mathbf{p}^{i-1} \mapsto \mathbf{p}^i = \mathbf{p}^{i-1} + \tfrac{1}{N}\,\mathbf{d}(\mathbf{p}) + \mathbf{o}(\mathbf{p}^{i-1} + \tfrac{1}{N}\,\mathbf{d}(\mathbf{p})) \tag{12}$$

Our new transformation equation can then be written as

$$\mathbf{p} \mapsto \mathbf{p} + \mathbf{d}(\mathbf{p}) + \sum_{i=1}^{N} \mathbf{o}(\mathbf{p}_i). \tag{13}$$

This is a natural generalization of equation (2).

Although the above formulation would allow for a complete tracking of the editing path as indicated by the mouse movement, we still evaluate the displacement field $\mathbf{d}$ at $\mathbf{p}$ only.

3.2. Defining the Occluder Field

For the efficient detection of contact situations we define the occluder implicitly as a signed distance vector field, i.e. for every point $\mathbf{p} \in \mathbb{R}^3$, in addition to the signed distance

$$\begin{aligned}
\delta: \quad \mathbb{R}^3 &\to \mathbb{R} \\
\mathbf{p} &\mapsto \text{signed distance of } \mathbf{p} \\
&\quad\ \text{to occluder surface,}
\end{aligned} \tag{14}$$

we store the direction

$$\begin{aligned}
\Delta: \quad \mathbb{R}^3 &\to \mathbb{R}^3 \\
\mathbf{p} &\mapsto (\mathbf{p}_0 - \mathbf{p})/\|\mathbf{p}_0 - \mathbf{p}\|,
\end{aligned} \tag{15}$$

to the closest point $\mathbf{p}_0$ on the occluder surface. We discretionarily choose $\delta(\mathbf{p}) < 0$ iff $\mathbf{p}$ is inside the occluder.

The well-known Adaptively Sampled Distance Fields (ADFs) [20] serve well for our purpose here since we can use the sample density of the ADF as a hint for the sampling distance for the editing paths. We propose using the voxel width as a local path sampling rate. This inherently allows for feature detection in the occluder field.

Although our current implementation makes use of analytically defined occluders, a future toolbox will contain a set of predefined signed distance fields. The distance fields to user-defined occluders have to be calculated in a preprocessing step. However, this does not prevent user interaction with the occluder: Resizing, translating, rotating are all trivially available without changing the actual values in the distance field. Evaluating the "transformed" distance field at a position $\mathbf{p}$ simply requires the evaluation of the original distance field after applying the inverse transformation to $\mathbf{p}$.

Having access to the distance values and to the closest point on the occluder surface at any position in space, we have now all the ingredients to formulate our occluder force field. We define

$$\begin{aligned}
\mathbf{o}: \quad \mathbb{R}^3 \times \mathbb{R}^3 &\to \mathbb{R}^3 \\
(\mathbf{p}, \mathbf{d}(\mathbf{p})) &\mapsto \mathbf{o}(\mathbf{p}, \mathbf{d}(\mathbf{p}))
\end{aligned} \tag{16}$$

as follows:

$$\mathbf{o}(\mathbf{p}, \mathbf{d}(\mathbf{p})) = -\psi(\delta(\mathbf{p} + \mathbf{d}(\mathbf{p}))) \cdot \Delta(\mathbf{p} + \mathbf{d}(\mathbf{p}))$$

where $\psi : \mathbb{R}^3 \to \mathbb{R}$ is an *influence function* that can be thought of as a kind of shape function for the occluder and

that determines the effective impact of the occluder field. Depending on the actual editing circumstances, different influence functions are appropriate, e.g.

$$\psi(\delta(\mathbf{p})) = \begin{cases} \delta(\mathbf{p}) & : \quad \delta(\mathbf{p}) \leq 0 \\ \exp(-\delta^2(\mathbf{p})) & : \quad \delta(\mathbf{p}) > 0. \end{cases} \tag{17}$$

This influence function guarantees that vertices penetrating the occluder are transferred to the occluder surface, regardless of the underlying editing operation, and vertices coming close to the occluder surface but not penetrating it are also repelled. This prevents (to some extent) the fingers from flattening in figure 2 <see color section>.

The rationale behind formulating $\mathbf{o} = \mathbf{o}(\mathbf{p}, \mathbf{d}(\mathbf{p}))$ instead of $\mathbf{o} = \mathbf{o}(\mathbf{p})$, i.e. making the occluder field not only dependent on the *locus* $\mathbf{p}$ but also on the editing *direction* $\mathbf{d}(\mathbf{p})$ is that this leads to a more flexible approach. The most significant benefit from this formulation is that we are able to assure that the occluder has no bigger effect than the originating displacement and therefore to restrict the occluder influence to the editing region of influence. This can easily be done by including $\|\mathbf{d}(\mathbf{p})\|$ as a factor into equation (17).

4. Adaptive Refinement

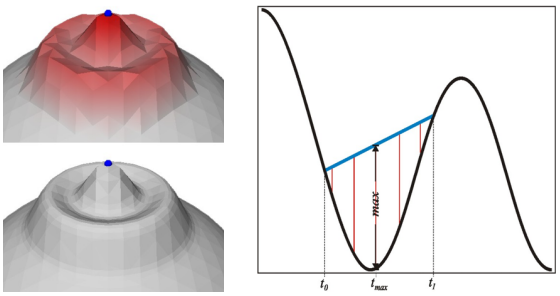

Figure 5: *Left: For some edits, the triangles might be too large to represent fine details as defined by the shape function (top). Result after refinement (bottom). Right: The error induced by applying a transform to a polygonal mesh as specified by the shape function* without *refining the mesh corresponds to the error induced by linearly interpolating the shape function between two adjacent vertices in the mesh.*

Editing operations change the geometric properties of the underlying mesh. Edits are likely to add small details that might not be representable by the current triangulation. Therefore an adaptive refinement method has to be applied, e.g. as proposed in [22] or [21]. Here, local curvature information (by midpoint subdivision or by vertex normal deviation, resp.) is used to decide if further refinement is required.

In addition to this general refinement technique, we propose exploiting the shape function information to decide if and *where* edges have to be subdivided in order to be able to incorporate sharp feature edits. Therefore, we consider the error induced by linearly interpolating the shape function between adjacent vertices (see figure 5). For a handle vertex $\mathbf{p}$ and an edge $(\mathbf{v}_0, \mathbf{v}_1)$ we define

$$\varepsilon^{\mathbf{p}}_{\mathbf{v}_0,\mathbf{v}_1} = \max_{t_0 \leq t \leq t_1} \left| \varphi(t) - \varphi(t_0) + \frac{t - t_0}{t_1 - t_0} (\varphi(t_1) - \varphi(t_0)) \right| \tag{18}$$

with $t_0 = \gamma(\mathbf{p}, \mathbf{v}_0)$, $t_1 = \gamma(\mathbf{p}, \mathbf{v}_1)$ and φ and γ as defined in (7) and (8) respectively. For multiple handle vertices $\mathbf{p}_1, \ldots, \mathbf{p}_k$ we define

$$\varepsilon_{\mathbf{v}_0,\mathbf{v}_1} = \max_{\mathbf{p}=\mathbf{p}_1,\ldots,\mathbf{p}_k} \varepsilon^{\mathbf{p}}_{\mathbf{v}_0,\mathbf{v}_1}.$$

Edges are subdivided if ε exceeds a user-controllable threshold.

Let t_{max} be the parameter in $[t_0, t_1]$ for which the right hand side in (18) becomes maximal. A new vertex will then be inserted into this edge at the *temporary* position

$$\mathbf{v}_{max} = \mathbf{v_0} + \frac{t_{max}}{t_1 - t_0} (\mathbf{v}_1 - \mathbf{v}_0).$$

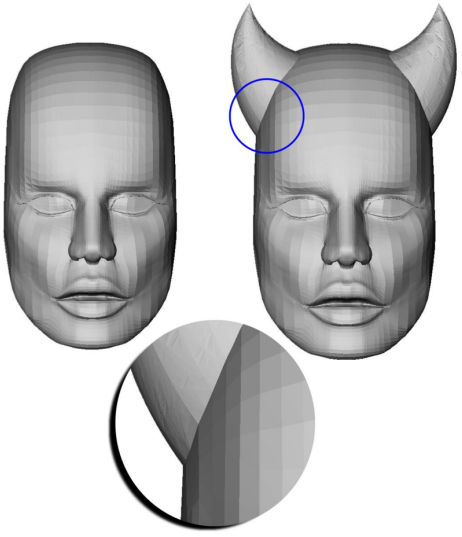

Figure 6: *Sharp features can be modelled because the mesh is adaptively refined according to an edit applied to the mesh.*

After the insertion of new vertices, the new edges are checked if further refinement is required. Finally, the new positions for all vertices in the mesh are calculated. In this step we take advantage of the fact, that we can easily approximate the geodesic distance for every new vertex $\mathbf{v}_{max}$ on the edge $(\mathbf{v}_0, \mathbf{v}_1)$ from the distance values of the two adjacent vertices. To this end, let δ_0 and δ_1 be the distances of the vertices $\mathbf{v}_0$ and $\mathbf{v}_1$ to the handle resp. We compute a virtual origin for the distance calculation on the intersection of the two spheres with radii δ_0 and δ_1 and centers $\mathbf{v}_0$ and $\mathbf{v}_1$. Since

the intersecting circle is orthogonal to the edge $(\mathbf{v}_0, \mathbf{v}_1)$, every point on it has the same distance to $\mathbf{v}_{max}$ which is used as the desired approximation of the geodesic distance in $\mathbf{v}_{max}$ (cf. [31]).

Whereas the above refinement strategy allows for sharp feature editing by recursively subdividing edges where indicated by the shape function, further refinement might be required to model contact situations, as can be seen in the rightmost picture in figure 2 <see color section>. We are currently working on an approach to extend our adaptive refinement strategy in this direction.

5. Results and Discussions

As an example for the power of our rotation scheme for animation purposes, we consider the famous Stanford Bunny. Figure 1 <see color section> shows how the bunny's ears can be transformed with one single editing operation consisting of as few as four editing steps. Firstly, the tips of the two ears are selected as handles; secondly, we make a preliminary choice for the ROI of the edit (we can always change that at later stages of the edit, the preliminary choice only improves the visual feedback during the following steps). In our current implementation, the rotation axis is defined by the screen center and the viewing direction. Hence we position the bunny accordingly and drag the tips of the ears to the desired position (figure 1 (4) <see color section>). After the transformation for the tips of the ears has been specified, we can interactively modify the shape function in order to achieve a realistic look of the ears. It would have been very hard to achieve this result using the translation scheme only.

Figure 3(b) <see color section> is an example for the use of the anisotropic rotational editing scheme. The rotation axis was chosen in this example approximately parallel to the spine. Note that only the head has turned, the shoulder region remained fixed. In order to have the head remain rigid while the neck is twisted, few anchor vertices have been chosen on the top of the head, and a single handle vertex was picked on the nose. The multiple anchor vertices are necessary because otherwise it had been difficult to adjust the region of influence such that only the neck is twisted (a head is not perfectly round, so choosing only one anchor vertex at the top will not lead to a satisfying region of influence behavior – with just one anchor vertex either the shoulder region will be influenced or "outer regions" as the chin will not move rigidly with the rest of the head).

The possibility of choosing multiple anchor vertices is also feasible to solve problems arising with the use of *geodesic* distances as the basis of our object parametrization: Certain object surface features (imagine a ring with a prominent gem on the finger in the example depicted in figure 3(a) <see color section>) can cast "geodesic shadows". We currently examine in how far surface smoothing before the calculation of the geodesic distances diminishes or solves this problem.

Nevertheless, as geodesic distances reflect properties inherent to the surface, they yield in general more intuitive editing results than Euclidean distances, without having to define object-independent hulls of influence as proposed in [36].

Our current implementation is based on the superposition of distance fields computed with respect to the single anchor vertices. As pointed out in section 2 this does not lead to a smooth distance field. However, implementing results of implicit modelling (e.g. convolution surfaces[6]) will lead to a convenient method to define smooth distance fields with respect to the anchor points as skeletons, where desired. Moreover, allowing other primitives like edges or triangles to be used for these skeletons will provide for a powerful tool to define regions of influence for the editing operations. This could be incorporated into our approach without even changing the user interface.

With our adaptive refinement method we are able to model sharp features even in sparsely triangulated regions of the mesh, as can be seen from figure 6. However, our refinement strategy is based on the shape function edit only. It does not take into account the underlying geometric properties of the object in the editing region. Therefore, even in cases in which the edit actually *reduces* details on the mesh, new vertices might be inserted. We're currently working on a method to prohibit the insertion of vertices in these cases, yet maintaining the efficiency of analyzing only the shape function.

So far, our implementation is purely vertex-based, the user can only choose vertices as handles and anchors. It would be desirable (and a straight forward extension) to allow for arbitrary points on the object surface to be picked.

In many cases picking one or more vertices on the mesh and dragging them to the desired position is sufficient and leads to satisfying results. In some occasions, though, that might not give users enough flexibility. Therefore we allow users to specify the parametrization of the object *independently* from the handles (cf. section 2.4). Among other benefits, this simplifies the editing operation in rotation cases where the distance field is defined on (or close to) the rotation axis. In these cases, it is convenient to grab a different point on the object to specify the rotation angles.

Our mesh editing approach can not only be used for animation but also for modelling purposes as it can be seen from figure 7. Starting from a sphere, we create a complete teapot. Note how the shape function can be used to model details on the teapot's handle with a rotational edit.

References

1. A. Angelidis and M.-P. Cani. Adaptive implicit modeling using subdivision curves and surfaces as skeletons. In *Solid Modelling and Applications*. ACM, June 2002. Saarbrucken, Germany.

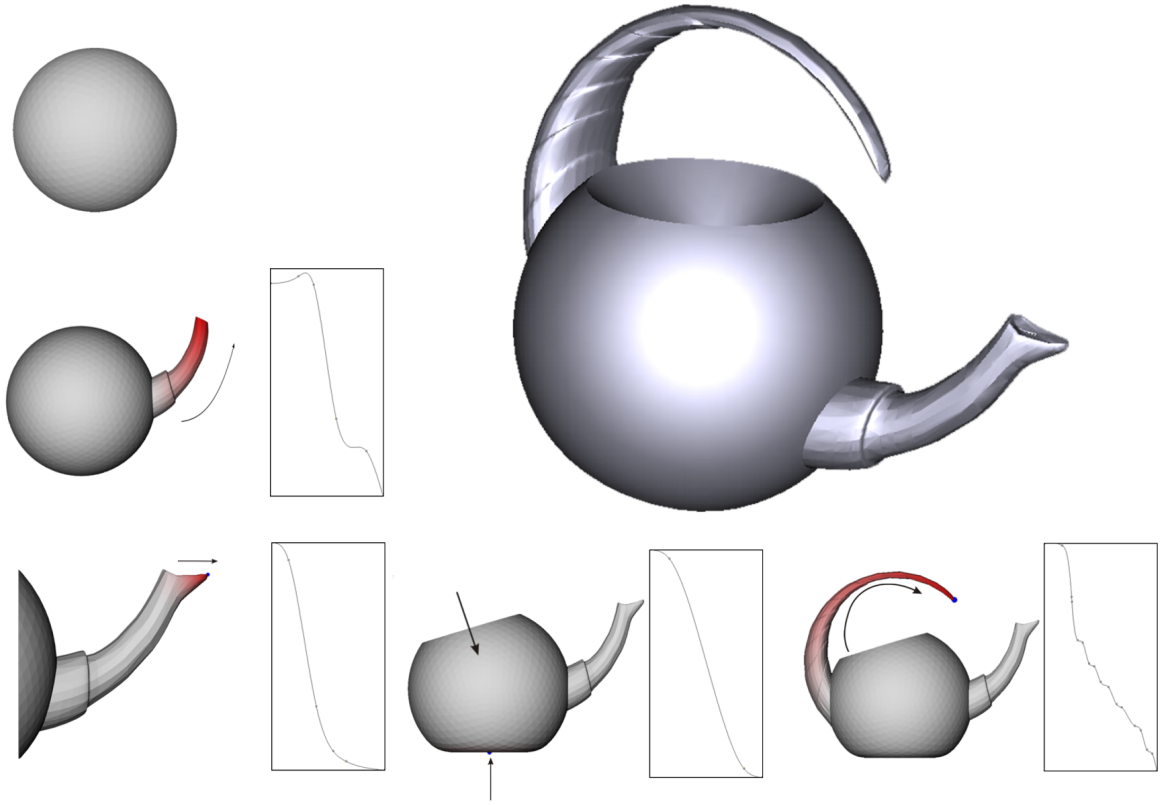

Figure 7: *Creating a teapot (top right) from a primitive (top left) with just a few editing operations. The basic editing operations are depicted together with the corresponding shape functions (left and bottom). The arrows indicate the modification applied to the handles. Note how the shape function can not only be used to adjust the overall shape of the edit but also to add details to the model (see the teapot's handle in the last editing step and in the final result).*

2. A. Angelidis, P. Jepp, and M.-P. Cani. Implicit modeling with skeleton curves: Controlled blending in contact situations. In *Shape Modeling International*. ACM, IEEE Computer Society Press, 2002. Banff, Alberta, Cananda.

3. A. H. Barr. Global and local deformations of solid primitives. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 21–30, 1984.

4. J. F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics (TOG)*, 1(3):235–256, 1982.

5. J. Bloomenthal. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers, Inc., 1997.

6. J. Bloomenthal and K. Shoemake. Convolution surfaces. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 251–256. ACM Press, 1991.

7. J. Bloomenthal and B. Wyvill. Interactive techniques for implicit modeling. *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, 24(2):109–116, 1990.

8. P. Borrel and D. Bechmann. Deformation of n-dimensional objects. In *Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications*, pages 351–369. ACM Press, 1991.

9. P. Borrel and A. Rappoport. Simple constrained deformations for geometric modeling and interactive design. *ACM Transactions on Graphics (TOG)*, 13(2):137–155, 1994.

10. B.Wyvill, C.McPheeters, and G.Wyvill. Data structure for soft objects. *The Visual Computer*, 2(4):227–234, 1986.

11. M.-P. Cani. Implicit representations in computer animation : a compared study. In *Proceedings of Implicit*

Surface '99, Sep 1999. Invited paper.

12. M.-P. Cani and M. Desbrun. Animation of deformable models using implicit surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 3(1), Mar 1997. Published under the name Marie-Paule Cani-Gascuel.

13. M.-P. Cani and S. Hornus. Subdivision curve primitives: a new solution for interactive implicit modeling. In *Shape Modelling International*, Italy, May 2001.

14. E. Catmull and J. H. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10:350–360, November 1978.

15. G. Chaikin. Short note: An algorithm for high-speed curve generation. *Computer Graphics and Image Processing*, 3:346–349, 1974.

16. S. Coquillart. Extended free-form deformation: a sculpturing tool for 3d geometric modeling. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 187–196. ACM Press, 1990.

17. P. Faloutsos, M. van de Panne, and D. Terzopoulos. Dynamic free-form deformations for animation synthesis. *IEEE Transactions on Visualization and Computer Graphics*, 3(3):201–214, /1997.

18. G. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*. Academic Press Inc., 1993.

19. N. Frisch and T. Ertl. Deformation of finite element meshes using directly manipulated free-form deformation. In *Proceedings of the seventh ACM symposium on Solid modeling and applications*, pages 249–256. ACM Press, 2002.

20. S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones. Adaptively sampled distance fields: a general representation of shape for computer graphics. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 249–254. ACM Press/Addison-Wesley Publishing Co., 2000.

21. J. E. Gain and N. A. Dodgson. Adaptive refinement and decimation under free-form deformation. In *Eurographics UK '99*, 1999.

22. J. Greissmair and W. Purgathofer. Deformation of solids with trivariate b-splines. In *Computer Graphics Forum*, pages 137–148, 1989.

23. W. M. Hsu, J. F. Hughes, and H. Kaufman. Direct manipulation of free-form deformations. *Computer Graphics*, 26(2):177–184, 1992.

24. S.-M. Hu, H. Zhang, C.-L. Tai, and J.-G. Sun. Direct manipulation of ffd: efficient explicit solutions and decomposible multiple point constraints. *The Visual Computer*, 17, 2001.

25. T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: a sketching interface for 3d freeform design. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 409–416. ACM Press/Addison-Wesley Publishing Co., 1999.

26. O. Karpenko, J. F. Hughes, and R. Raskar. Free-form sketching with variational implicit surfaces. *Computer Graphics Forum*, 21(3), September 2002.

27. L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. Interactive multi-resolution modeling on arbitrary meshes. *Computer Graphics*, 32(Annual Conference Series):105–114, 1998.

28. L. P. Kobbelt, T. Bareuther, and H.-P. Seidel. Multiresolution shape deformations for meshes with dynamic vertex connectivity. *Computer Graphics Forum*, 19(3), August 2000.

29. S. Lee. Interactive multiresolution editing of arbitrary meshes. *Computer Graphics Forum*, 1999.

30. L. Markosian, J. M. Cohen, T. Crulli, and J. Hughes. Skin: a constructive approach to modeling free-form shapes. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 393–400. ACM Press/Addison-Wesley Publishing Co., 1999.

31. M. Novotni and R. Klein. Computing geodesic distances on triangular meshes. In *The 10-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, pages 341–347, 2002.

32. A. Opalach and M. Cani-Gascuel. Local deformations for animation of implicit surfaces. In W. Straßer, editor, *13th Spring Conference on Computer Graphics*, pages 85–92, 1997.

33. M. Pauly, R. Keiser, L. Kobbelt, and M. Gross. Shape modeling with point-sampled geometry. In *Proceedings of the 30th annual conference on Computer graphics and interactive techniques (to appear)*, 2003.

34. R. Penrose. A generalized inverse for matrices. In *Proc. Cambridge Philos. Soc.*, pages 406–413, 1955.

35. R. N. Perry and S. F. Frisken. Kizamu: a system for sculpting digital characters. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 47–56. ACM Press, 2001.

36. R.Raffin, M. Neveu, and F. Jaar. Curvilinear displacement of free-form-based deformation. *The Visual Computer*, 16(1):38–46, 2000.

37. T. W. Sederberg and S. R. Parry. Free-form deformation of solid geometric models. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 151–160. ACM Press, 1986.

38. S.Yoshizawa, A. G. Belyaev, and H. Seidel. A simple approach to interactive free-form shape deformations. In *Pacific Graphics 2002 Proceedings*, pages 471–474, 2002.

39. W. Welch and A. Witkin. Free-form shape design using triangulated surfaces. *Computer Graphics*, 28(Annual Conference Series):247–256, 1994.

40. R. C. Zeleznik, K. P. Herndon, and J. F. Hughes. Sketch: an interface for sketching 3d scenes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 163–170. ACM Press, 1996.

41. D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. *Computer Graphics*, 31(Annual Conference Series):259–268, 1997.

42. M. Zwicker, M. Pauly, O. Knoll, and M. Gross. Pointshop 3d: an interactive system for point-based surface editing. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 322–329. ACM Press, 2002.

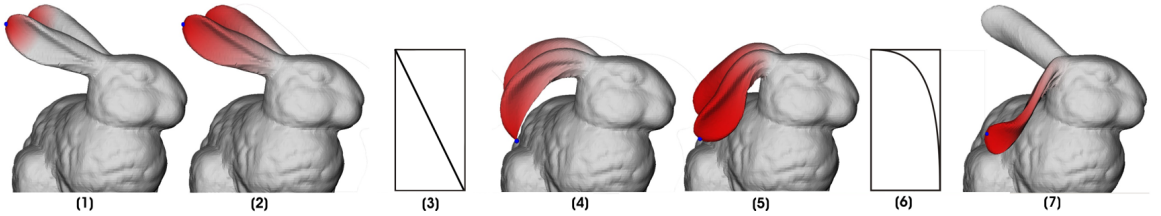

Figure 1: *Application of the rotational editing scheme. The tips of the bunny's ears are picked and dragged, while not only the region of influence (indicated by the red colored area (1) and (2)) but also the shape of the edit can interactively be modified and adjusted ((4) and (5)) using the shape function ((3) and (6) resp.) until the impression is visually satisfying. Picture (7) shows the corresponding edit using the translational scheme with a slightly adjusted shape function to produce a smooth changeover at the bunny's head.*

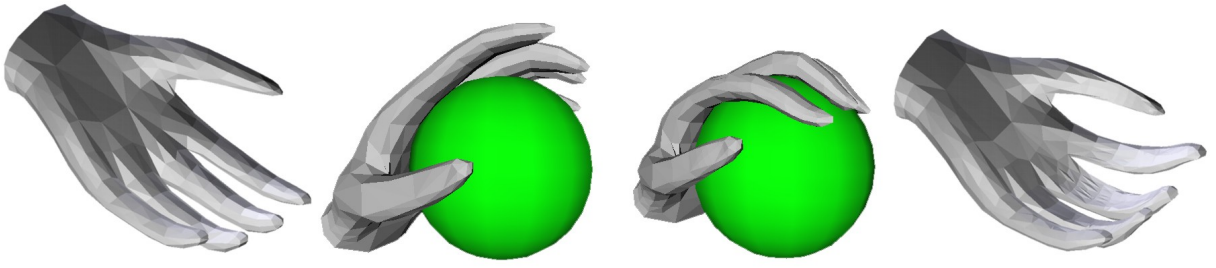

Figure 2: *Modelling a hand taking grip on a ball. The fingers of the hand are transformed by a simple drag on the finger tips. The force field induced by the occluder causes the fingers to be shaped around the ball instead of intruding into it.* Influence functions *prevent the fingers from flattening.*

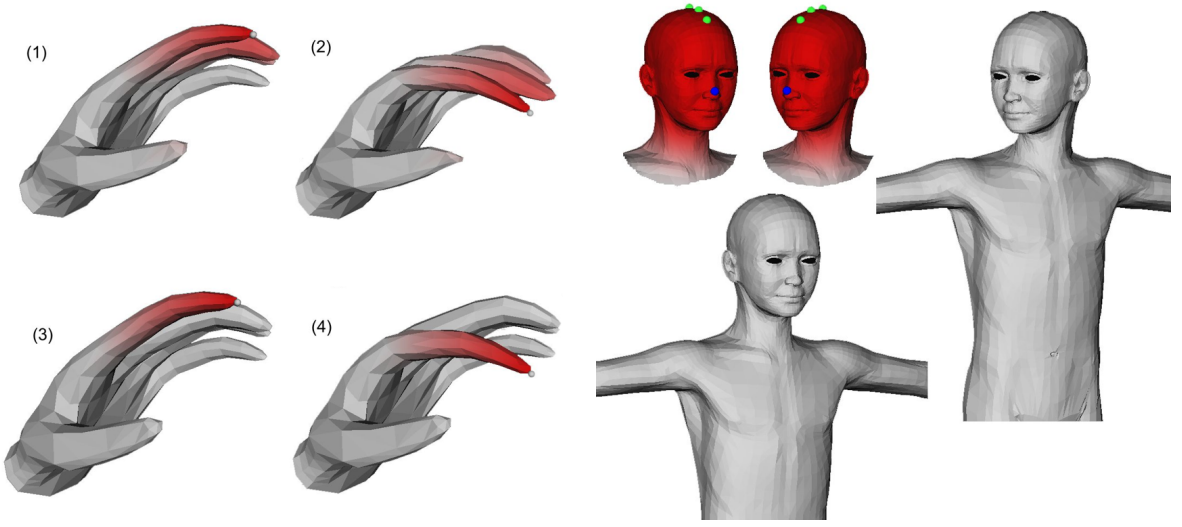

Figure 3: *(a) Editing operation with Euclidean (pictures 1 and 2) and geodesic (3 and 4) distances. The region of influence is indicated by red color, the little sphere at the tip of the index finger is the handle that is dragged during the edit. Note how the middle and ring finger are modified together with the index finger in the upper right picture. (b) Turning a model's head. As indicated by the red color, the shoulder region remains fixed, while the head is turned (using the blue handle on the nose). By using multiple (three) anchors (green spheres at the top of the head) we define an anisotropic ROI s.t. the head is turned rigidly, with a smooth changeover at the neck.*