

Interactive Free-Form Level-Set Surface-Editing Operators

Abstract

We present a set of interactive, free-form editing operators for direct manipulation of level-set models. The mathematics, algorithms and techniques needed to implement numerous level-set modeling capabilities have been developed. The operators have been combined with an OpenGL interface and the VISPACK level-set library to create a preliminary interactive level-set modeling system. VISPACK's narrow-band data structures have been extended to localize all computations and updates to optimize running time. We describe the level-set speed functions that implement the surface-editing operators. A level-set surface can be edited through a click-sketch-and-pull interface that allows a user to identify a point or Region-of-Influence (ROI) to be modified on the surface. The user may then pull a point or a curve within the ROI to produce a free-form surface manipulation. Additional operators include surface detailing, carving and smoothing, as well as a sketch-based technique that pulls the surface toward a profile curve.

Key words: implicit modeling, level-set modeling, volume modeling, surface editing, free-form deformations

1. Introduction

Surface models, e.g. triangles meshes, NURBS and subdivision surfaces, have been the most widespread modeling representation used within computer graphics and visualization for several decades. In these models topologically 2D surfaces existing in 3D Cartesian space have been explicitly represented with 2D structures, such as triangles and spline patches. While these have been the predominant models for quite some time, implicit models, which represent surfaces as iso-surfaces of a 3D scalar field, are becoming more prevalent and important to such disparate disciplines as model processing in medicine/biology and special effects for movies.

As imaging technology continues to be rapidly deployed and utilized in medicine and science, an increasing number of volume datasets are generated, producing an overwhelming flood of raw volume data that must be processed, viewed and analyzed. Usually these datasets contain 3D surfaces that are of interest to physicians and scientists. In computer graphics many surface reconstruction algorithms generate volumetric, implicit representations



Figure 1: A cartoon bear created with free-form level-set surface editing operators.

before converting the objects into explicit surface models. Advanced special effects in movies utilize physical simulation to produce computer-generated fluid flows of floods, storms, pouring/splashing liquids, etc. These simulations require the initial specification of volumetric shapes, and produce dynamic volume datasets as output. In these instances, the scanning, computational or simulation process rarely

produces “perfect” models in the judgement of the physicians, scientists, modelers and directors who use them. Frequently the volume datasets and the surfaces embedded in them need to be fixed, adjusted and/or edited to meet the requirements of the user’s application.

Given the abundant sources of volumetric datasets and the paucity of editing tools for interactively modifying the implicit surfaces that they define, we are developing a suite of free-form editing operators for volumetric implicit surfaces. These editing operators have been developed for a specific class of implicit surface, level-set models, which are an important and powerful type of implicit model that offer numerous benefits. They combine a low-level volumetric representation with the mathematics of deformable implicit surfaces, and are based on formulating and solving a partial differential equation (PDE) [1, 2]. They have been devised within a well-developed mathematical framework that provides robust numerical techniques for evaluation and evolution [3]. They are guaranteed to define simple (non-self-intersecting) and closed surfaces. Level-set models easily change topological genus, making them ideal for representing complex structures of unknown or transforming genus. Additionally, they provide the advantages of implicit models, e.g. supporting straightforward solid modeling operations and calculations, while simultaneously offering a surface modeling paradigm. One drawback of using level-set techniques based on conventional distance fields is the inability to represent arbitrary sharp features. However, uniform and adaptive complete distance fields [4, 5] define fields that are not band-limited and are able to capture exact surface details. Additionally, Novotny et. al. [6] presents a method for voxelization of solid objects containing sharp details.

The mathematics, algorithms and techniques needed to implement numerous interactive, free-form level-set modeling capabilities have been developed. These capabilities have been implemented utilizing a pre-existing level-set library, and incorporated into an interactive modeling system. Key to making these free-form editing operators interactive are new data structures, which have been integrated into the li-

brary, that minimize running times by localizing all computations and surface updates. We have designed several level-set speed functions that yield flexible surface-editing operators. These operators provide the user an intuitive and straightforward way to interact with 3D level-set models using conventional input devices such as a mouse and keyboard.

The surface can be edited through a click-sketch-and-pull interface that allows a user to identify a point or Region-Of-Influence (ROI) to be modified on the surface. An ROI is specified by drawing a closed curve on the surface. If no ROI is specified, a superellipsoid or distance function is used to define what portion of the surface is to be edited. The user may then pull a point or a curve within the ROI to produce a free-form surface manipulation. Additional operators include surface detailing, carving and smoothing, as well as a sketch-based technique that pulls the surface toward a profile curve. These operators allow a model to be stretched and shaped, split in pieces, bent and merged smoothly. Topology changes occur naturally and automatically because of the properties of level-set modeling. A painting capability was also added to the system to allow the user to specify colors on the resulting level-set models. The cartoon bear in Figure 1 is created using the surface editing capabilities discussed in this paper.

Contributions. Our work has developed novel level-set modeling functionality and technology. Firstly, we have developed a set of free-form editing operators, which provide direct implicit surface manipulations, within a level-set framework. Since a level-set model can only be modified via solving the level-set equation, a speed function, which is the component of the equation that defines the surface’s evolution, has been devised for each editing operator. The specific mathematical form of these speed functions are detailed in Section 4 and summarized in Table 1. These mathematical details are the main contribution of our work, and furnish novel modeling/manipulation capabilities for level-set models. Secondly, the data structures that focus processing on only those portions of the model’s surface that are changing provide new computational level-set techniques that enhance interactive performance. As compared to previous PDE-based mod-

eling approaches, ours is the first that is able to interactively modify a volumetric implicit surface by solving a PDE. This new interactive level-set modeling capability has been utilized to implement an additional set of indirect, sketch-based operators, as described in [7].

2. Related Work

Volume Sculpting. Volume graphics [8] involves the synthesis, manipulation, and rendering of volumetric objects, which are stored as an array of voxels. Interactive sculpting tools for clay-like [9, 10] and solid [11] models represent the material by voxel data and define tools that can add/remove material, as well as perform a smoothing operations. Some sculpting metaphors utilize alias-free volume sampling [12] or uniformly sampled scalar fields [13] as the volume representation. These efforts are extended in [14] to achieve interactive editing speeds using resolution adaptive volume sculpting and also in [15] to create a real-time sculpting system using subdivision solids. Some volume sculpting applications use haptic feedback to give the user a sense of shaping a virtual material [16]. Volumetric models are frequently represented as uniform or adaptive 3D distance fields [17, 18, 19, 20]. Leu and Zhang [21] describe a method to convert volume sculpted models into distance fields and apply curvature-based smoothing using the level-set method. 3D Coat [22] is a recently released commercial system that seems to offer some of the capabilities of our operators. Currently no technical publications are available that describe its inner workings, so we are unable to compare it to our work.

Level-set models are implicit surfaces embedded in volume datasets; thus supporting straightforward solid modeling operations while providing surface properties such as normal and curvature that may be used during surface editing. Their main advantage over voxel-based models is their ability to provide a high-level surface paradigm based on a low-level volumetric representation.

Volume Deformations. Free-Form Deformations (FFDs) [23] place a lattice around a model. Moving

the lattice deforms the 3D space enclosed by the lattice, and therefore deforms the model. Different approaches to this metaphor utilize curves (wires) [24] that are placed in close proximity to a manipulated surface to act as handles that deform the surface locally, cellular automata [25] for transportation of mass through a 3D grid or evolving scalar fields that define deformations of polygonal models [26]. Space deformation techniques for interactive virtual sculpting [27, 28] create a deformation field with a volumetric tool. There also exists vector field based deformations [29, 30] and point-based techniques [31] for performing free-form deformations of polygonal meshes.

These editing systems use indirect spatial deformations to edit the underlying model. Level-set techniques work directly on the implicit surface and do not deform the space around the model. They provide more intuitive and straightforward control over the deforming surfaces.

Implicit Modeling. Implicit models [32, 33] are a widely used representation for geometric modeling applications. Soft Objects were one of the first successful systems based on implicit models [34, 35, 36]. This original work was extended to create an implicit modeling system that combines CSG operations with Barr deformations [37], as well as blending and warping [38]. The system's interactive performance was improved in [39, 40]. More techniques for generating 3D implicit sweep volumes compatible with these systems are described in [41] and is extended with a sketch-based editing framework [42] and with a curve-based free-form deformation capability [43]. Desbrun and Cani [44, 45] present a hybrid model that combines implicit surfaces with a particle system, a rigid solid or a mass-spring network for animation of soft inelastic substances which undergo topological changes. They also use a volume-based implicit representation to animate iso-surfaces defined within a 3D grid that stores a potential field [46]. Some other modeling systems use skeletons defined as a graph of interconnected subdivision curves and surfaces [47, 48, 49], interpolating [50] or variational implicit surfaces [51, 52], convolution surfaces [53] or spherical implicit functions [54], to represent 3D models.

Arbitrary deformations to analytical implicit surfaces require a skeleton structure to create a volumetric model. Level-set models have volumetric representations by definition and do not require any additional operators like skeleton calculations or volume sweeping.

Sketch-Based Modeling. Sketching communicates ideas rapidly with approximate input, no need for precision or specialized knowledge, and easy low-level correction and revision. Sketch-based modeling tools allow the user to sketch the salient features of a 3D primitive and the system produces the corresponding 3D model in the scene. Our free-form editing operators extend several existing sketch-based techniques for meshes [55, 56, 57, 58, 59, 60], parametric surfaces [61], procedural surfaces [62], volumetric models [63] and implicit surfaces [42, 43, 51, 52, 53, 54] to level-set models.

PDE Models. Level-set methods have been used for volume sculpting [64], CSG-based surface editing, automatic blending and curvature-based smoothing [65, 66]. Our new editing operators provide a significantly more expressive and flexible editing capability to level set modeling. The interactive level-set modeling features described in this paper have been employed to produce additional indirect, sketch-based editing operators [7]. Mullen et al. [67] propose a mass-preserving variational approach for geometry processing of volumetric implicit surfaces and foliations using an Eulerian formulation. Zhang and Lihua [68] developed a geometric modeling framework based on partial differential equations (PDEs) that incorporates geometric constraints and functional requirements into PDEs. PDE-based volumetric sculpting as implemented in [69, 70, 71, 72] defines smooth surfaces as a solution to a fourth order elliptic PDE with geometric and physical boundary conditions such as curvature and normals. Lawrence and Funkhouser [73] propose a painting paradigm for specifying surface deformations for level-set surfaces and triangle meshes.

In contrast to previous work, our operators and processing techniques provide new methods for interactive direct, free-form modifications of level set mod-

els. We also compared our results to previous PDE-based modeling work in terms of model resolution, processor speed and running times. The most recent related study [72] uses a maximum resolution of $65 \times 65 \times 65$ and an approximately three times slower CPU. Their results show that at this resolution it takes 16 seconds to 6 minutes for various operations to converge on a solution. The previous volume sculpting work [64] using an octree representation for the level-set model can edit volumes with a resolution of $1024 \times 1024 \times 1024$. They also solve the level-set equation on a sub-volume and achieve somewhat interactive running times. They show that on a $20 \times 20 \times 20$ sub-volume their average running time is 6 – 7 frames-per-second (fps). A similar operation runs 100 fps with our framework on an approximately 4 times faster CPU.

3. Level-Set Models

Level-set models are defined as an iso-surface, i.e. a level set, of a dynamic implicit function ϕ [1, 2],

$$S = \{x \mid \phi(x, t) = k\}, \quad (1)$$

where $k \in \mathbb{R}$ is the iso-value, $x \in \mathbb{R}^3$ is a point in space on the iso-surface and $\phi : \mathbb{R}^3 \rightarrow \mathbb{R}$ is a scalar function. The surface is deformed by solving a partial differential equation (PDE) on a regular sampling of ϕ , i.e. a volume dataset. The level-set PDE can be written as

$$\frac{\partial \phi}{\partial t} = -|\nabla \phi|F(x, D\phi, D^2\phi \dots), \quad (2)$$

where $F()$ is a user defined speed term which depends on a set of order-n derivatives of ϕ as well as other functions of x . Level-set methods provide the techniques needed to change the voxel values of the volume in a way that moves the embedded iso-surface to meet a user-defined goal [3]. The scalar field of a level-set model is represented by a volume dataset, i.e. a 3D array that stores scalar values in its elements.

$F()$ defines the speed of the level-set surface at point x in the direction of the local surface normal ($\nabla \phi / |\nabla \phi|$). The surface is deformed over time by

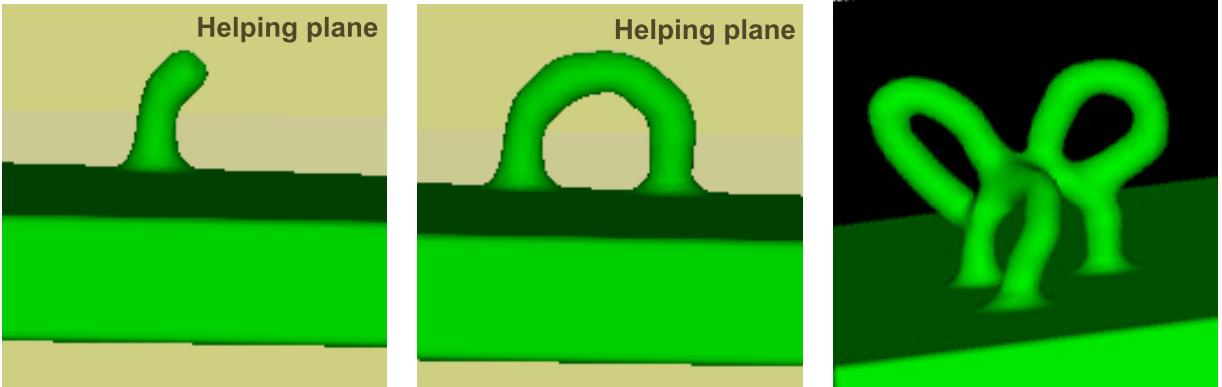


Figure 2: A loop is created by clicking and dragging a point on the surface ($\alpha = 2.0$). The first two frames demonstrate the use of the helping plane (the yellow background). The last frame shows several loops smoothly merged with each other.

moving either in or out in the direction of the local normal. The magnitude and direction of the movement is specified by the F function, which is defined over the entire volume. In order to apply level-set methods to a particular application, a custom F function must be designed and implemented. The following section describes in detail the F functions that have been devised for interactive free-form geometric modeling based on level-set models.

Notice that Equation 2 has a temporal component t . The modifications to the level set are affected by this component and the resulting surface produced during interactive editing is a function of how the user moves the anchors, i.e. points and curves, as well as the speed at which they are moved.

4. Editing Operators

4.1. 3D Input

Our free-form editing operators require the definition of 3D locations and 3D curves both on and off the surface using a conventional 3-button mouse. For 3D points on the surface the display's Z-buffer is read at the 2D cursor location when the mouse is clicked. The 3D point in window coordinates is “un-projected” back into world coordinates to produce a point lying on the model. For specifying 3D points off of the surface, we offer two methods. The first method provides a *helping-plane*. During editing operations the 2D input produced by mouse strokes can

be mapped onto an arbitrary plane within the scene. If utilized, the plane can be added at a point of interest on the model and displayed in the scene with a translucent color. Initially, the plane's normal is set to face the user, however, it can be changed to an arbitrary orientation with a mouse interaction. 2D input is mapped onto the plane during editing operations. A helping-plane (displayed in translucent yellow) is used in the editing operations in Figures 2, 6, 7 and 11. When specifying a 3D curve, a second method is available. Here, the 3D curve is defined to lie on a plane perpendicular to the view direction, and begins at the point where the first mouse click intersects the level-set surface. We utilize an enhanced form of Catmull-Rom splines to specify curves [74] for the editing operators. These splines provide localized and multi-resolution editing capabilities for 3D curves.

4.2. Pulling a point, symmetric ROI

This operator allows a user to click on a point on the surface (x_s) and drag it. Clicking creates a tracker particle on the surface at x_s . A part of the surface, defined by a radius of influence around x_s , then moves outward. As the 3D cursor location (x_c) changes the tracker particle moves toward x_c , but is constrained to stay on the deforming surface; thus defining an updated x_s value for the surface evolution. The new position of x_s lies on the line connecting x_s to x_c , where this line intersects the surface. The evolution stops once the tracker particle reaches the 3D cursor

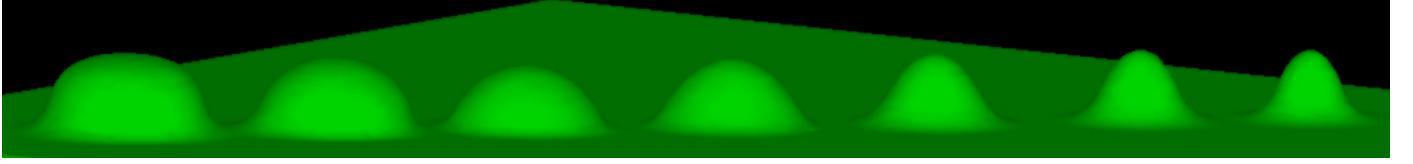


Figure 3: Effect of changing the α parameter in Equation 3. $\alpha = 0.25, 0.5, 0.75, 1.0, 2.0, 3.0, 4.0$.

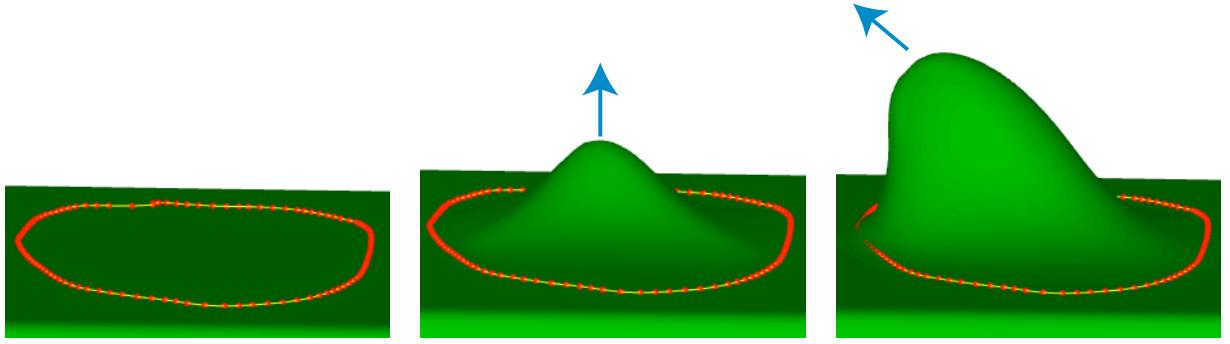


Figure 4: Defining and deforming a patch on the surface. $\alpha = 4$. $\epsilon = 5$ voxels.

location or the mouse button is released. The radius value may be changed any time during the movement. The operator produces a symmetric modification around x_s . While this is the simplest of editing operators, we did find it useful for designing and creating handle-like structures, e.g. in Figure 2.

The speed function that implements this operator is

$$F(\mathbf{x}) = \begin{cases} \cos^\alpha(\pi/2 * d_s(\mathbf{x})/r) & d_s(\mathbf{x}) \leq r \\ 0 & d_s(\mathbf{x}) > r, \end{cases} \quad (3)$$

where \mathbf{x} is a point on the surface being evaluated, and $d_s(\mathbf{x})$ is the geodesic distance from the point \mathbf{x} to the point being dragged, i.e. x_s . The geodesic distance between x_s and all the voxels within the ROI can be calculated using the sweeping algorithm explained in Section 4.9 and Algorithm 1. Equation 3 states that the speed of the surface evolution drops off with a cosine function depending on the distance from the dragged point. The speed goes smoothly to zero at distance r (the edge of the ROI). The shape of the drop-off may be controlled by the parameter α , which exponentiates the cosine function. Figure 2 shows a surface being edited using this operator ($\alpha = 2.0$). Figure 3 demonstrates the different bump shapes created by varying α .

4.3. Pulling a point, arbitrary ROI

For this operator the user first draws a closed boundary curve (C_b) on the surface to designate an ROI, then clicks and drags a point (x_s) within the ROI. All points in the ROI move outward, with the points closest to x_s moving the fastest. The ROI's surface speed gradually decreases to zero at the boundary curve. As with the previous operator the tracker particle x_s moves towards the cursor's 3D location (\mathbf{x}_c), but remains on the surface. The surface movement stops either when x_s reaches \mathbf{x}_c or when the user releases the mouse button. In Figure 4, a C_b curve, defined by the red control points, is drawn in yellow. The user clicks and drags a point upwards and then to the left to produce a surface modification.

The speed function for the operator is

$$F(\mathbf{x}) = f(d_{out}(\mathbf{x})) * \left(\frac{\max(d_{in}^{xs}(\mathbf{x})) - d_{in}^{xs}(\mathbf{x})}{\max(d_{in}^{xs}(\mathbf{x}))} \right)^\alpha \quad (4)$$

$$f(d) = \begin{cases} 1/2 - 1/2 * \cos(\pi * d(\mathbf{x})/\epsilon) & d \leq \epsilon \\ 1.0 & d > \epsilon, \end{cases} \quad (5)$$

where $d_{in}^{xs}(\mathbf{x})$ is the geodesic distance to x_s from \mathbf{x} , $\max(d_{in}^{xs}(\mathbf{x}))$ is the maximum of these values over

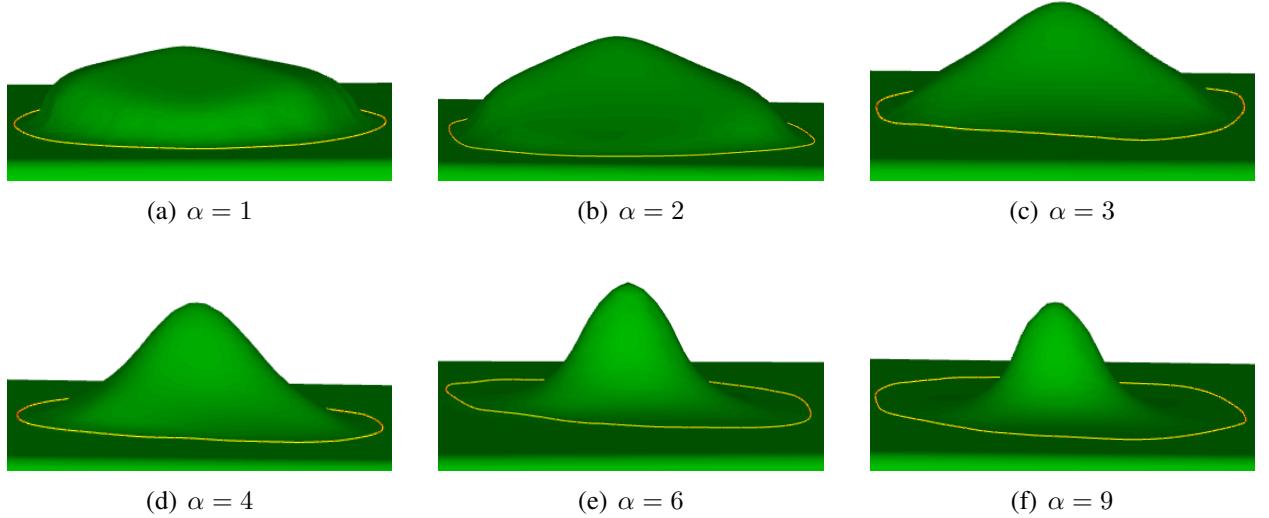


Figure 5: The α parameter in Equation 4 changes the shape of the modification.

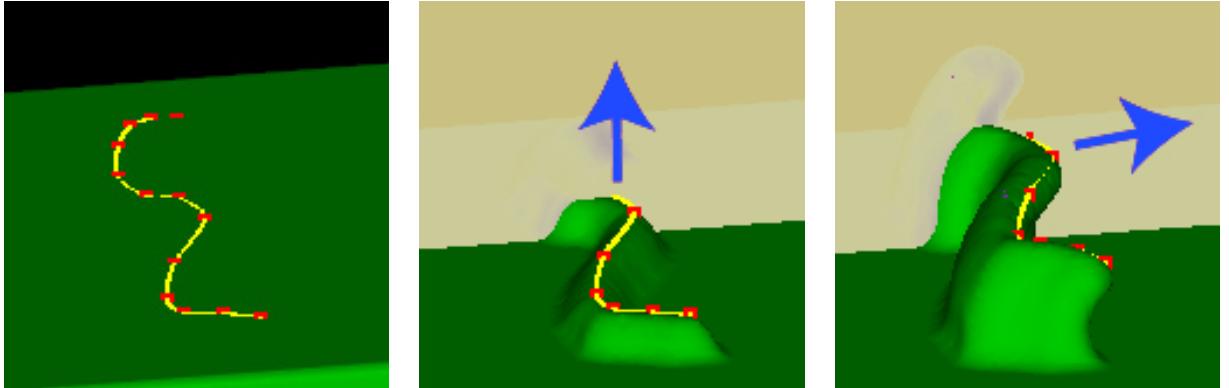


Figure 6: A curve is placed on the surface and pulled first upwards then towards the right.

all points in the ROI, and $d_{out}(\mathbf{x})$ is the geodesic distance to the boundary curve from \mathbf{x} . The first term, $f(d_{out}(\mathbf{x}))$, ensures that the speed smoothly goes to zero at the boundary curve starting at some distance ϵ from the boundary. ϵ is a user-defined parameter. The second term of the equation decreases the speed as the point on the surface (\mathbf{x}) gets further from the point being dragged (\mathbf{x}_s). α is once again a parameter that controls the shape of the “bump” pulled up from the surface. Increasing α produces a faster drop-off. Examples of varying the α parameter are given in Figure 5. The geodesic distance $d_{in}^{xs}(\mathbf{x})$ is calculated by sweeping out distance information from \mathbf{x}_s to voxels that lie on the surface using Algorithm 1 (see Appendix and Section 4.9).

4.4. Pulling a curve on the surface, symmetric ROI

With this operator the user draws an open curve C_s on the surface, clicks on the curve, and then drags it. The surface near the curve moves out to follow the curve. All points on the surface within a specified distance from C_s move with a speed that decreases proportionally to their distance from the curve. In Figure 6, the curve is first pulled up, then dragged slightly towards the right. Observe that the surface bends slightly as it follows the user’s input. The speed function for this operator is

$$F(\mathbf{x}) = \begin{cases} (1.0 - d_{in}^{cs}(\mathbf{x})/r)^\alpha & d_{in}^{cs}(\mathbf{x}) \leq r \\ 0 & d_{in}^{cs}(\mathbf{x}) > r, \end{cases} \quad (6)$$

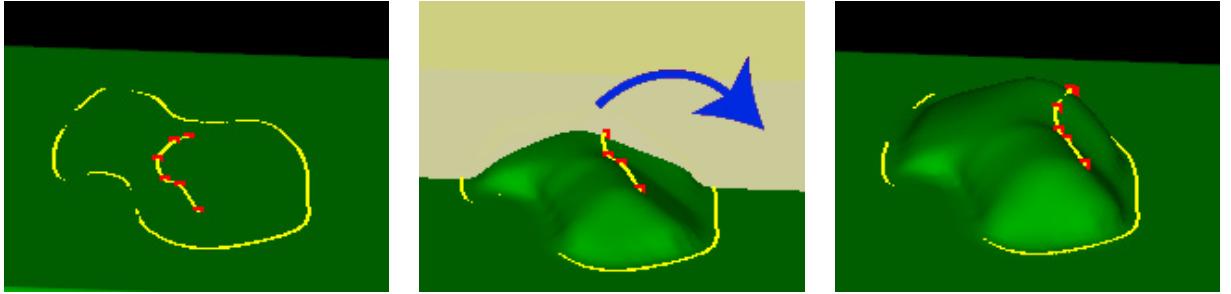


Figure 7: A patch on the surface is raised using a handle. The handle is pulled in an arc towards the right side of the window.

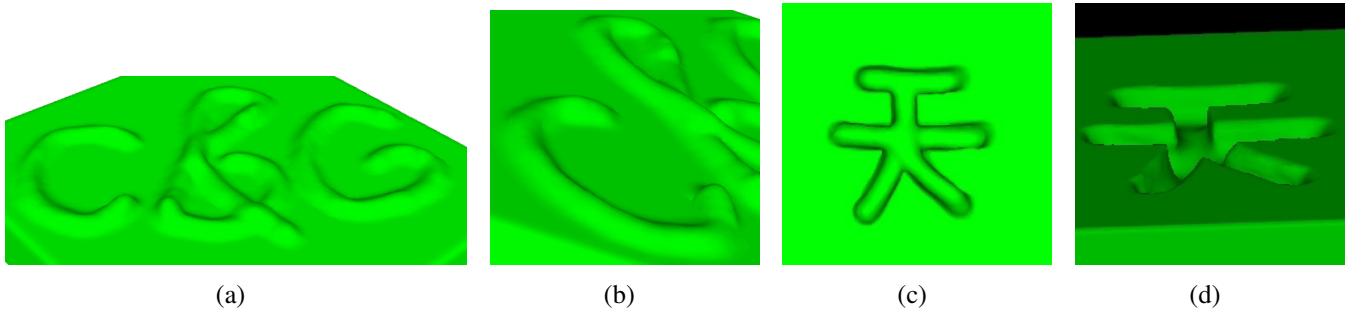


Figure 8: An example of the surface detailing tool. The two images on the left consist of offsets on the surface created by continuous cursor strokes (a, b). The two images on the right demonstrate interactive carving of the Chinese character for sky (c, d).

where r is the width of the ROI and $d_{in}^{cs}(\mathbf{x})$ is the geodesic distance between \mathbf{x} and the curve C_s . Points near the curve move the fastest and the speed decreases to zero at distance r from the curve. The evolution stops once the user releases the curve. α can be used to further control the shape as explained in the previous operators.

4.5. Pulling a curve on the surface, arbitrary ROI

The user first draws a closed curve (C_b) on the surface to define an ROI and another curve (C_s) on the surface to be used as a handle. Clicking and dragging a point on the curve moves the handle, and deforms the surface within the ROI. Figure 7 shows a sequence where the user drags the handle in an arc towards the right side of the window. The speed function for this operator is

$$F(\mathbf{x}) = f(d_{out}(\mathbf{x})) * \left(\frac{\max(d_{in}^{cs}(\mathbf{x})) - d_{in}^{cs}(\mathbf{x})}{\max(d_{in}^{cs}(\mathbf{x}))} \right)^\alpha, \quad (7)$$

where $d_{in}^{cs}(\mathbf{x})$ is the geodesic distance to curve C_s and $d_{out}(\mathbf{x})$ is the geodesic distance to the boundary curve C_b from \mathbf{x} . $\max(d_{in}^{cs}(\mathbf{x}))$ is the maximum over all points in the ROI. $f(d_{out}(\mathbf{x}))$ is defined in Equation 5. The points closest to the handle move the fastest, and the points on or outside of the boundary curve do not move. The speed decreases as the distance to the handle curve increases, and goes to zero at curve C_b . The evolution stops once the user releases the curve. α can be used to further control the shape as explained in the previous operators.

4.6. Surface Detailing/Carving

With this operator the user picks a superellipsoid-shaped tool and moves it over the surface to add or subtract features. The surface is locally modified in the general shape of the chosen tool. The size of the tool can be changed interactively and the height/depth of the features depends on the speed of the strokes. The faster the cursor is moved over the surface the lesser the detail that is added and vice

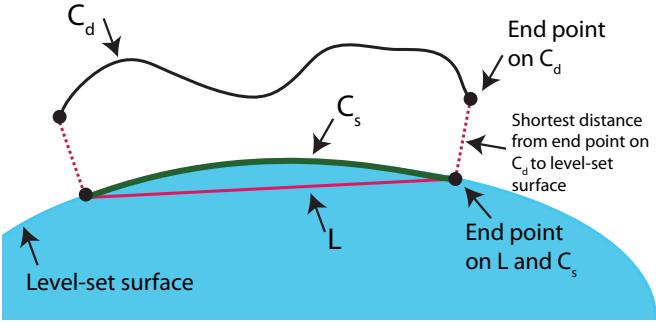


Figure 12: Projecting the cross section curve onto the level-set surface. The line L in 3D space is created using the closest points to the end points of C_d on the surface. Points starting from L move towards C_d and stop once they reach the surface, creating the projected curve C_s .

versa. A spherical tool is used to create the example in Figure 8 a-b.

The speed function for this operator is

$$F(\mathbf{x}) = \begin{cases} 0 & f_{se}(\mathbf{V}) > 0 \\ \beta * f_{se}(\mathbf{V}) & f_{se}(\mathbf{V}) \leq 0 \end{cases} \quad (8)$$

$$\mathbf{V} = \mathbf{x} - \mathbf{x}_c$$

The tool is centered at the cursor point \mathbf{x}_c . $f_{se}(\mathbf{V})$ takes in the relative position of \mathbf{x} with respect to \mathbf{x}_c and evaluates the superellipsoid implicit inside-outside function [75]. f_{se} is negative inside the superellipsoid, therefore setting $\beta = -1$ drives the surface outwards until it reaches the superellipsoid's boundary where $f_{se} = 0$. Changing the superellipsoid's shape parameters (ϵ_1 and ϵ_2) will also change the shape of the surface detail.

Interactive carving is implemented in the same manner, only with $\beta = 1$. The user clicks and moves the cursor over the surface, pushing the surface in. This can be interpreted as carving out material. The speed of the strokes determines the depth of the carving. A fast movement creates a shallow mark on the surface while constant slow strokes create deeper indentations. Figure 8c-d shows a sample carving using the spherical tool.

We have also utilized the carving operator as an interactive eraser tool to remove some parts of a model. A feature can easily be removed by adjusting the size of the tool and moving it over the unwanted portion

of the model. Figure 9 demonstrates the use of this operator to remove the spout from the Utah teapot.

4.7. Sketching Cross-sections

With this operator a curve may be used to define a cross-section of a local shape change. The user draws a closed boundary curve (C_b) on the surface to define an ROI and another curve (C_d) that comes up out of the surface and defines a desired cross-section of the desired shape. Every point within the ROI moves outwards with a speed defined in Equation 9 until the surface reaches C_d . We created a “mountain” on the surface with C_d defining the peak and C_b defining the extent of the foothills (Figure 10). Once the evolution starts, a third curve (C_s) is created on the surface. This curve is represented as a dense set of points and is the projection of C_d onto the surface.

The projection curve is created in two steps. First, the closest points on the surface to both end points of C_d are found. A 3D line segment L is created using these two closest points. L and C_d are both represented with the same number of dense points and a one-to-one correspondence is defined between each pair of points on the curves. Next, all points on L move to the level-set surface either towards or away from their corresponding points depending on their position with respect to the surface and C_d . The points stop when they reach the surface creating a projection curve C_s of the cross-section curve C_d on the surface. Figure 12 shows C_d , C_s and L around the level-set surface. At every step of the evolution, C_s moves toward C_d and the surface grows to meet the new C_s , until C_s (and the surface) reaches C_d . We recognize that there are cases where ordering/pairing produced by the projection may result in inconsistencies as the surface deforms. We have not experienced this in our work to date. If problems do arise in the future, a more robust projection method [76] may be utilized.

The red dots on the cross-section curve in Figure 10 are control points that can be manipulated by clicking and dragging. New control points can also be added to the curve. After a control point is moved or added, the curve is recalculated and the level-set

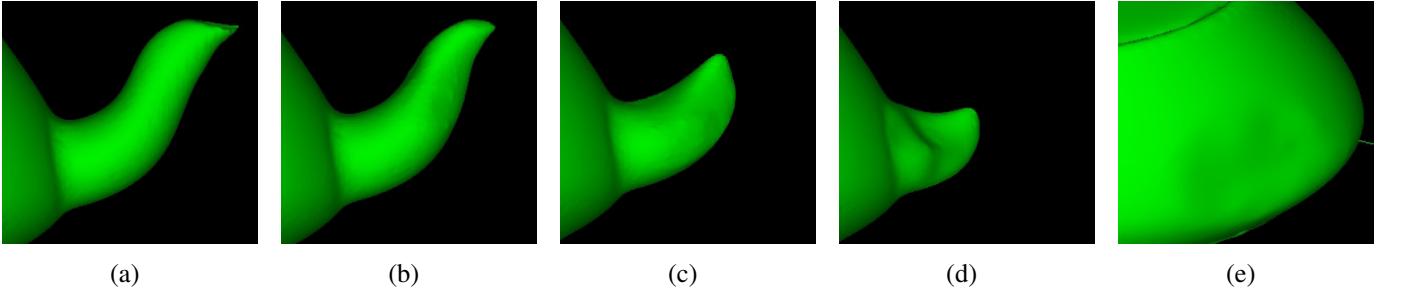


Figure 9: Interactive carving as an erasing tool. Frames (a-d) demonstrates the spout being erased and the last frame (e) shows the final result.

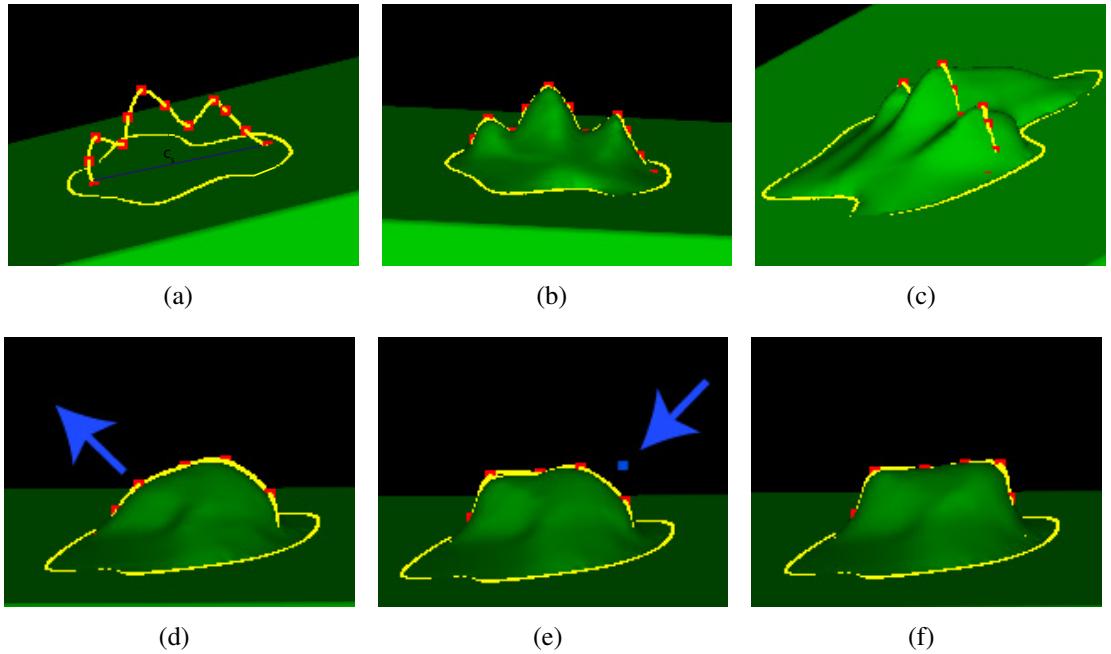


Figure 10: Two curves are defined (a), and the surface is grown to fit to both. The final result is shown from a different point of view(b, c). (d) a control point is modified. (e) another control point is added. (f) the final result.

equation is solved once more to fit to the new cross-section. In Figure 10 (bottom), an initial bump is shown in the leftmost frame. The control point on the left is pulled towards the top-left corner first and then a new control point is added on the right hand side of the bump. Both modifications result in an immediate change in the shape of the surface, as it fits to the new cross-section curve.

The speed function for this operator is

$$F(\mathbf{x}) = \frac{d_{up}(\mathbf{x})}{\max(d_{up}(\mathbf{x}))} * f(d_{out}(\mathbf{x})) * \left(\frac{\max(d_{in}^{cs}(\mathbf{x})) - d_{in}^{cs}(\mathbf{x})}{\max(d_{in}^{cs}(\mathbf{x}))} \right)^\alpha, \quad (9)$$

where $d_{out}(x)$ is the geodesic distance from x to C_b , and $d_{in}^{cs}(x)$ is the geodesic distance from x to C_s . The first step of calculating d_{up} for point x involves finding the closest point in the point set representing C_s from x , called x_{cs} . Recall that x_{cs} has a corresponding point on C_d , called x_{cd} . $d_{up}(x)$ is the Euclidean

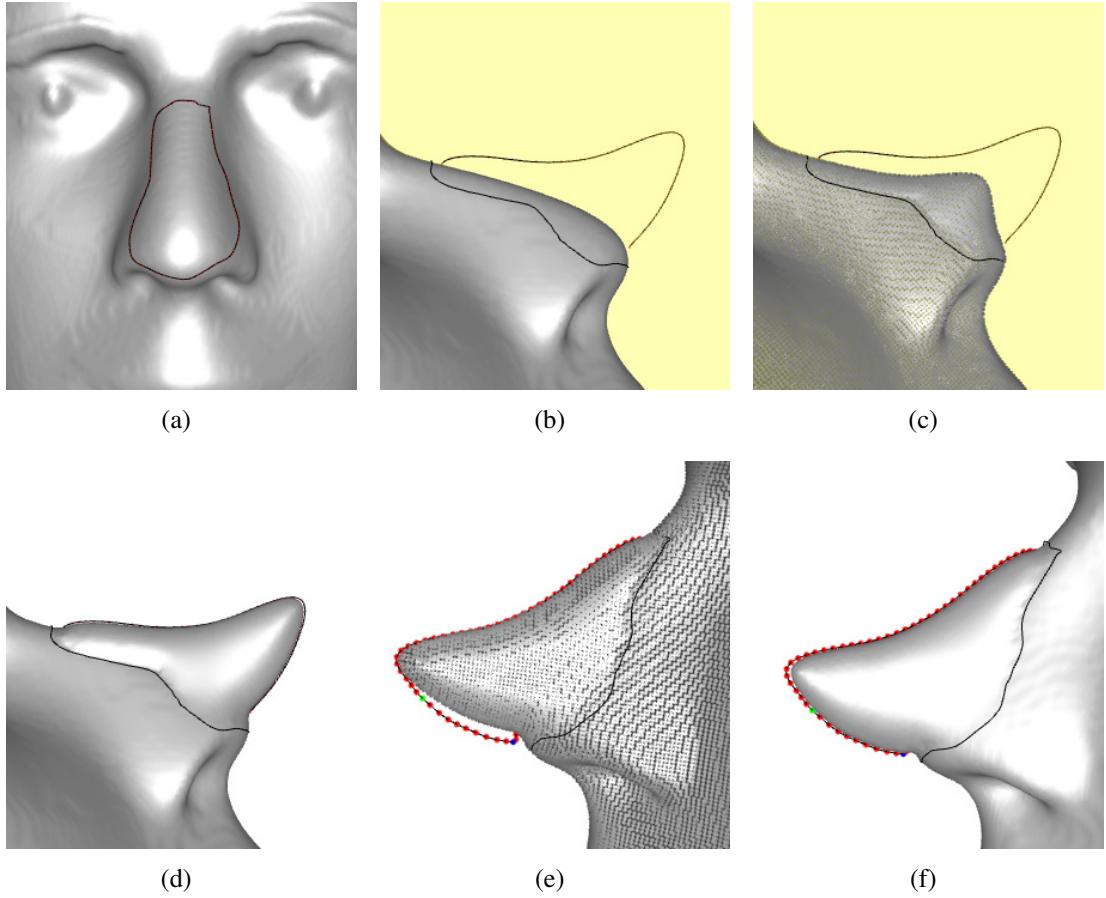


Figure 11: (a-b) Two curves define the new shape of the nose. (c-d) The surface fits to these curves. (e) The cross-section curve is modified for further refinement of the final shape. (f) The final result. (Some curvature-based smoothing is applied later on to produce the final shape of the nose in Figure 23). A point representation of the surface is used in (c) and (e) to provide a clearer view of the curves and control points. $\alpha = 2.0$.

distance between x_{cs} and x_{cd} . Both max functions are taken over all points in the ROI. $f()$ is defined in Equation 5, and ensures that the speed function goes to zero (and therefore the evolution stops) within a distance ϵ to boundary curve C_b . α can be used to further control the shape as explained in the previous operators.

The first term of Equation 9 ensures that the evolution will stop once the surface reaches the C_d curve. Together the last two terms define the speed function as a decreasing function of geodesic distance from the projected curve C_s to the boundary curve C_b . We used this operator for the edits on the nose of the mannequin head in Figure 23. A close-up showing how the new nose is created is in Figure 11.

4.8. Interactive Smoothing

We found it useful to add an interactive smoothing tool in our editing system. This tool is a flexible and interactive version of the local smoothing operator described in [65]. The user clicks on the surface and moves the cursor over the area that needs smoothing. A curvature-based speed function then modifies the surface following the user’s strokes. The smoothing speed function is described in Equation 10. The operator works within a fixed radius of influence. The size of the tool and the amount of smoothing to be applied can be adjusted by the user. The smoothing speed function is

$$F(\mathbf{x}) = \gamma * g(d_g(\mathbf{x})) * \kappa(\mathbf{x}) \quad (10)$$

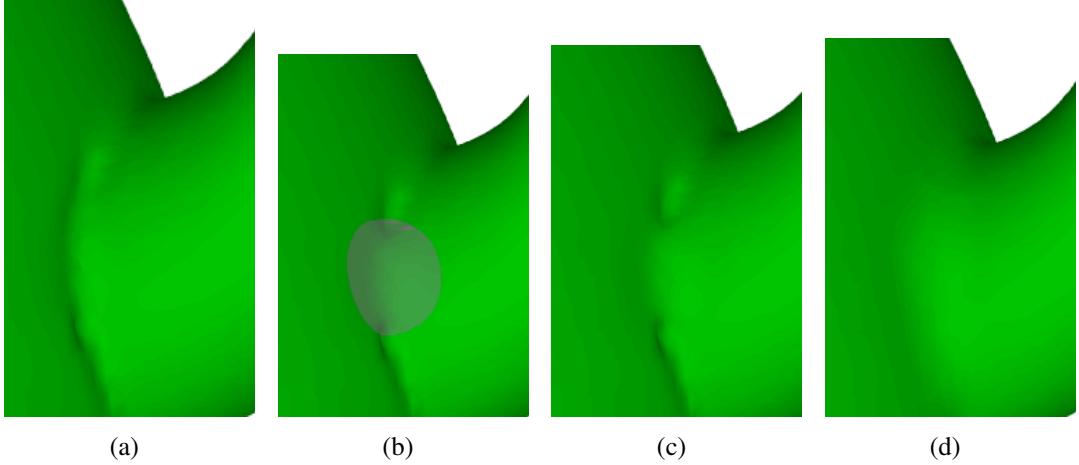


Figure 13: Interactive smoothing on the spout of the Utah teapot. (a) The initial scan converted model. (b) Smoothing tool is placed over rough region. (c) Smoothing has been locally applied. (d) Smoothing completed around the area where spout meets the teapot.

$$g(d) = \begin{cases} 1.0 & d \leq r - \epsilon \\ 1/2 + 1/2 * \cos(\pi * (d - r + \epsilon) / \epsilon) & r - \epsilon < d \leq r \\ 0.0 & d > r, \end{cases} \quad (11)$$

where γ is a constant that controls the amount of smoothing, $d_g(\mathbf{x}) = |\mathbf{x}_c - \mathbf{x}|$ is the distance from the point \mathbf{x} to the cursor \mathbf{x}_c , κ is mean curvature, r is the radius of the smoothing tool, and ϵ defines a transition region near the edge of the ROI. $g(d)$ is a function that ensures that the amount of curvature-based smoothing drops off smoothly to zero at the boundary of the smoothing tool at a user-specified distance r . Figure 13 demonstrates the use of smoothing on the spout of the Utah teapot with $\gamma = 0.3$.

4.9. Voxels inside an ROI

The voxels within the ROI are calculated using a sweeping algorithm (Algorithm 1). For symmetric operators, like pulling on a point with symmetric ROI, the algorithm first adds the point on the surface that the user clicked (\mathbf{x}_s) to the list of voxels within the ROI. All immediate “surface-crossing” neighboring voxels of \mathbf{x}_s are added to the list representing the ROI along with their Euclidean distances to \mathbf{x}_s . A “surface-crossing” voxel is one that has the level set between it and one of its immediate neighbors on the grid. This is easily identified via a sign flip between

the ϕ values of the voxel and its neighboring voxels. Next, all immediate neighbors of these newly added voxels which are also “surface-crossing” are added to the ROI list. Their distance to \mathbf{x}_s is calculated by adding the distance of the intermediate voxel in the list and the Euclidean distance between the new voxel and the intermediate voxel. We make sure that this distance is updated in case a shorter route is discovered further along in the computations. If the operator uses a curve on the surface instead of a point, the algorithm uses the sample points on this curve as initial points. The same sweeping algorithm is used with multiple \mathbf{x}_s as initial points. All possible paths to the curve are considered and the shortest path to the curve is used to calculate the geodesic distances.

For those operators that employ a user-drawn boundary curve to define an arbitrary ROI, the algorithm first marks all the surface-crossing voxels adjacent to the boundary curve. Sweeping out from the initial ROI voxels adds all encountered surface-crossing voxels to the ROI voxel list unless marked by the boundary curve. The algorithm visits all surface-crossing voxels enclosed by the boundary curve in a radially symmetric fashion and creates a list of voxels along with their geodesic distances to the initial curve or point defined on the surface. Similarly distance information may be swept in from the boundary curve voxels to all the surface voxels inside the ROI to calculate distances to this boundary curve.

Geodesic distance fields for non-convex shapes might have C^1 discontinuities at points equidistant to multiple points on the boundary or anchor curves. Schmidt and Wyvill [41] address this issue and suggest techniques to fit smoothed approximate distance fields to these surfaces. Non-smooth distance fields used by the speed functions may result in discontinuities in the resulting surface. We apply curvature-based smoothing locally within the ROI at regular intervals to overcome these artifacts.

When an editing operator is based on purely spatial Euclidean relationships, unwanted surface movements may be produced on nearby but unselected portions of the model, as seen in Figure 14a and 14b. Here, since the ROI is based on Euclidean distance, the bottom surface bulges out toward the protrusion being created by pulling the blue point, an unwanted modification that will eventually produce a topology change not specified by the user input. Basing the operators on geodesic distances allows us to properly localize the editing operation, as seen in Figure 14c and 14d. The blue points represent x_s and the pink points highlight the ROI in Figure 14a and 14d.

5. Modeling System

The free-form surface-editing operators have been implemented in an interactive level-set modeling system. The system consists of four major components depicted in Figure 15, (1) the level-set library that solves the level-set PDE on a narrow band, (2) the OpenGL user interface (UI), (3) the data structures that hold the volume and the narrow-band information, and (4) the routines that translate user input into speed functions for the level-set PDE.

The first component of the framework utilizes the VISPACK level-set library [77] to efficiently solve the level-set PDE. We have developed an editing user interface (UI) within an OpenGL application, as described in Section 5.3, which has been integrated with the VISPACK library. The application accepts specific user actions and translates them into speed functions for the level-set equation. The actions include mouse clicks and strokes, as well as keyboard input, that are designed to provide the user with an

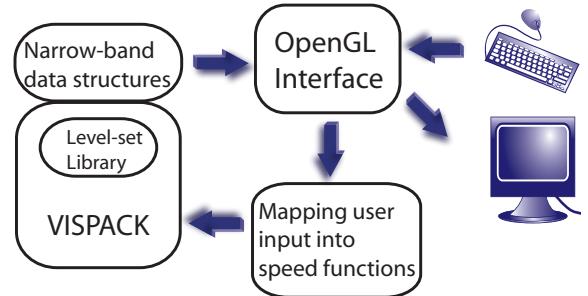


Figure 15: *Level-set surface-editing framework.* User input is translated into level-set speed functions. The level-set PDE is solved on a portion of the narrow band by the VISPACK library, and the resulting edited model is displayed in the UI.

intuitive and straightforward way to interact with the model and underlying library functions. We have also enhanced the narrow-band implementation¹ in VISPACK to further improve its computational performance. Section 5.4 describes the additional data structures used to achieve real-time evaluation of the level-set equation in a subset of the narrow band.

5.1. Computational Pipeline

The steps demonstrated in Figure 16 are processed every time a surface operator is invoked by the user. An ROI is calculated based on a selected point or a curve on the surface and either the size of the symmetrical tool or the arbitrary boundary curve drawn by the user. Speed functions explained in Sections 4.2-4.7 are used to calculate the change of ϕ values at all voxels within this ROI. In the next step, the scalar (ϕ) values at these voxels are updated. This update implicitly moves the level set based on the speed function. In the final step the new implicit surface is extracted from the updated scalar field and displayed on the screen.

5.2. Numerical Techniques

The speed functions described in Sections 4.2-4.8 create hyperbolic PDEs that require upwind differencing schemes for calculating spatial derivatives.

¹Rather than track all the level sets, the narrow-band method focuses computation on those voxels which are located in a narrow band around the zero level set.

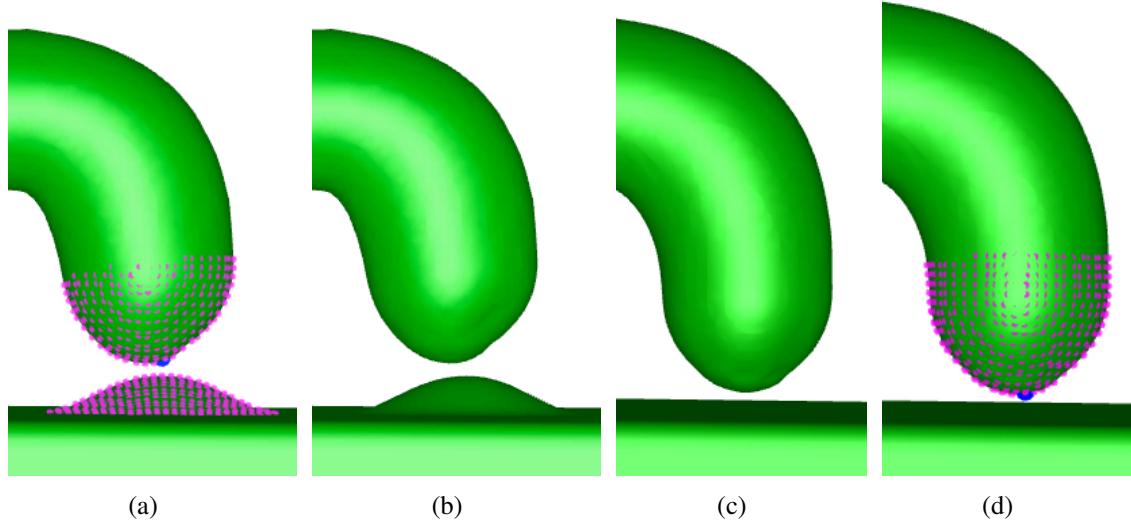


Figure 14: Pulling on a point, symmetric ROI with a 15 voxel radius. (a-b) The ROI is calculated by checking every voxel within a 15^3 bounding box centered at x_s (shown in blue in (a) and (d)). All voxels with a Euclidean distance of 15 or less to x_s are added to the ROI. (c-d) The ROI is calculated using the sweeping algorithm (Algorithm 1). The pink points in (a) and (d) represent the voxels in the ROI for each case. Using geodesic instead of Euclidean distance ensures that only the selected portions of the model are modified.

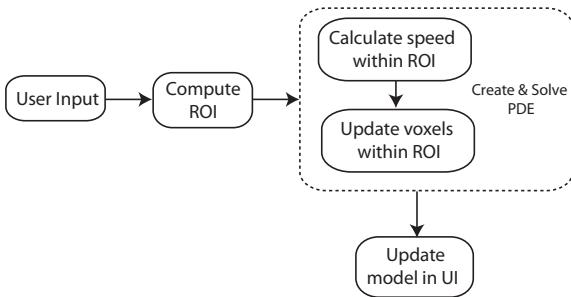


Figure 16: The computational pipeline.

VISPACK implements first and second order accurate upwind schemes, while third or fifth order schemes like ENO and WENO can be used to achieve more accurate solutions. Calculating the higher order numerical schemes incurs a higher performance cost, presenting a tradeoff between time and accuracy. Curvature-based smoothing explained in Section 4.8 creates a parabolic PDE. Second order accurate central differences can be used for this kind of PDE, as well as higher order upwind methods. Time integration is achieved using second order Eulerian techniques. There also exist third and fifth order TVD-Runge Kutta methods for time integration. These higher order finite difference methods

are explained in more detail in [3].

VISPACK utilizes the sparse-field algorithm [78] that uses an approximation to the distance transform. This algorithm makes it feasible to recompute the neighborhood of the level-set model at each time step without the need to stop the evolution and re-normalize the distance field. The distance field is re-normalized on the fly as the values get updated during the level-set evolution.

5.3. The OpenGL Interface

The interface supports two interaction modes. The first one is the *view* mode. In this mode, the user is able to change the view of the object by applying rotate, zoom and pan via mouse input. The user may choose one of the surface-editing operators (Section 4) while in the *edit* mode. The user interactions for these operators include drawing curves on/over the surface and/or clicking on and dragging points on the surface. A user draws a curve by clicking and moving the cursor over the surface. The cursor positions are tracked and define a list of control points. An enhanced Catmull-Rom spline [74] is fit to these control points once the mouse button is released. The

user can switch between *view* and *edit* modes using a single key stroke.

Our editing interface includes techniques from sketch-based modeling [42, 58], volume sculpting [64], spatial deformations [28], and level-set modeling [65], as well as user interfaces from previous free-form deformations research [24]. Although our interaction techniques are not unique, they provide new, intuitive and expressive means for modifying level-set models utilizing our novel surface editing operators.

The level-set model may be displayed either with point or polygon rendering. The VISPACK library can return a set of points lying on the level-set surface. These are displayed using OpenGL’s point rendering capability. The surface may also be displayed as a set of polygons which can be extracted from the level-set volume using a polygon extraction algorithm [34, 79, 80]. Point rendering is much faster and allows more interactive editing feedback, while polygon rendering gives a higher quality result and can be utilized to export mesh models. We have found it useful to be able to switch between the two types of viewing, allowing the user to choose either responsiveness or quality when rendering. Normally, point rendering is used to maximize interactivity during a modeling session. The user may then switch to polygon rendering to produce a higher quality model in order to more closely evaluate the session’s results.

We have also incorporated an interactive painting tool into our modeling framework. A paint brush with an adjustable size can be moved over the surface to apply different colors. Only the surface voxels within the tool’s extent are painted. The user can pick any of the pre-defined colors. Every color has an ID number and the color ID for every voxel is stored in the 3D grid. A tri-linear interpolation is used to determine the color of the actual surface points while displaying the model. Some examples can be seen in our final results in Figures 19, 20, 21, 22, 23 and 24.

5.4. Data Structures

VISPACK contains a narrow-band implementation for efficiently solving the level-set equation [78].

The technique localizes computation to only those voxels that lie within a narrow band of the level-set surface. The narrow-band data structure is implemented with a set of doubly linked lists, each storing a single layer of voxels within a certain distance of the surface. The layer may either be inside/outside of the surface or contain the surface itself. Each voxel is only stored in the list associated with the layer within which it resides, and the order of voxels within each list is arbitrary. While this implementation provides an efficient way to store and update the narrow band, it requires that the level-set PDE be solved over the whole surface. This is inefficient when an editing operation only affects a subset of the voxels, which requires traversing only a small portion of the doubly linked lists.

Since most of our surface-editing operators are designed to produce local surface modifications, it is advantageous to add another layer of data structures over the existing one in VISPACK. These additional data structures give the system the ability to limit computation to the subset of the level-set surface that is actually being modified. The new data structures are implemented as C++ vectors of pointers that point to entries within the VISPACK linked lists. These pointer data structures create an easy way to access a subset of voxels that are spatially contiguous. The elements in the vectors are created using a flood fill algorithm when the user first clicks on a point of interest. All the voxels on the surface up to a certain distance from the click point or within a region of influence (ROI) are added to these new vectors. Entries in the new vectors are updated, added and/or deleted during surface editing as voxels are added or removed from the original VISPACK narrow-band lists. An additional 3D array of pointers provides constant-time access to any narrow-band element. Each element in the 3D array points to the corresponding element in the VISPACK narrow-band list. These pointers are kept up-to-date with every change to the narrow-band data structures.

Figure 17 shows the VISPACK narrow-band data structures within the dotted green box, as well as the new data structures for further localizing the level-set computations. VISPACK keeps a linked list of all

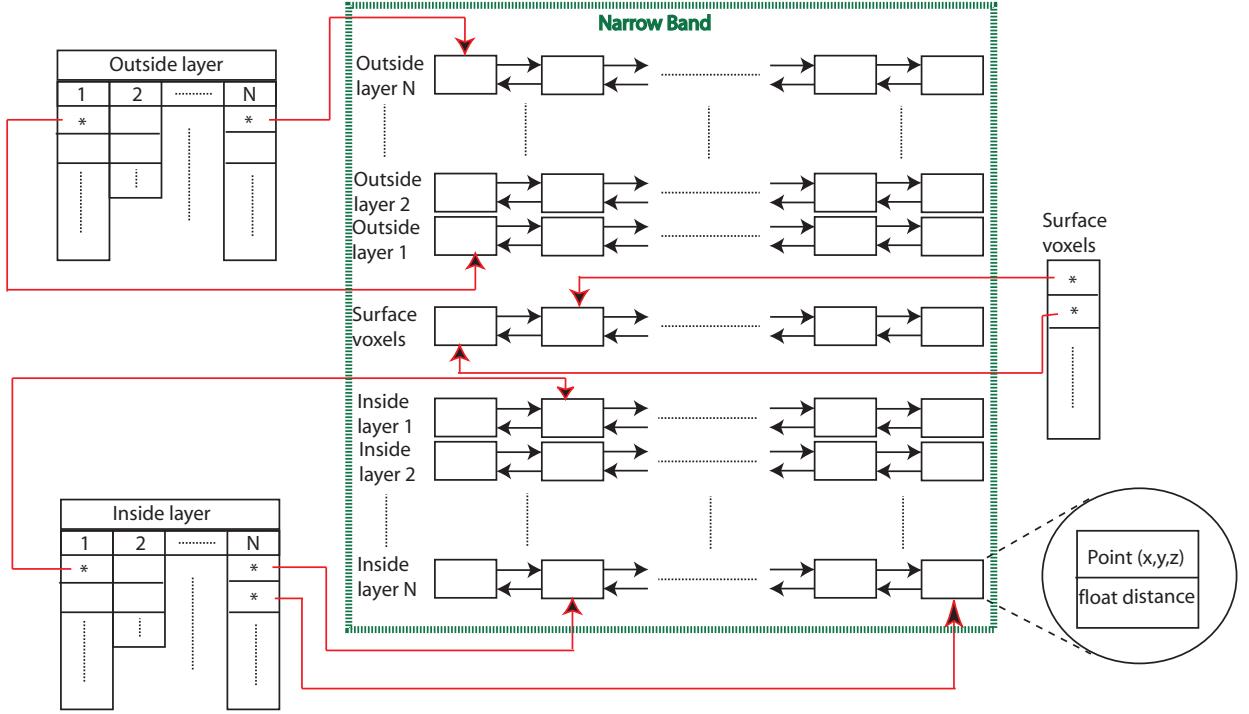


Figure 17: Three additional data structures (*Surface voxels* vector, *Outside layer* vectors and *Inside layer* vectors) are added to the narrow-band VISPACK data structure. The new data structures support interactive update rates by identifying the subset of voxels needed to solve the level-set PDE during an editing operation.

the voxels on the model’s surface, as well as separate linked lists for the voxels that are 1 to N layers away from the surface, both inside and outside. The elements of the linked lists store the 3D coordinates of the center of a voxel and the floating point distance to the surface, as seen in the lower right corner within a circle. We have added a collection of pointers to the original VISPACK linked-list elements that point to the subset of voxels involved in a surface editing operation. The pointers are kept in related vectors, e.g. vector *Outside Layer 1* keeps pointers to voxels just outside the surface and *Outside Layer N* keeps pointers to the voxels at the outer boundary of the narrow band. Similar information is stored for *Inside Layer 1* through N. The *Surface Voxels* vector is a single list that keeps pointers to the surface voxels. Since these layers do not necessarily have the same number of voxels we utilized C++ vectors to represent each collection.

Note that there is no prescribed order or structure to the pointers to the VISPACK list elements kept in each vector. The first pointer in the *Surface Vox-*

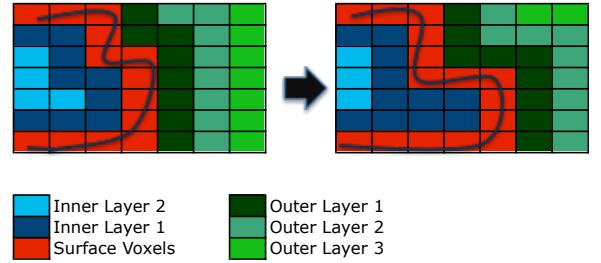


Figure 18: Changes in the narrow-band linked lists as the curve on the left evolves into the curve on the right.

els vector may point to the second element of the VISPACK Surface Voxels list. During editing several updates to any of these lists may happen. As the surface grows outwards some voxels are added to the Surface Voxels list, while some are dropped and possibly added to Inner Layer 1. The shift happens in every layer, i.e. a voxel in the 2nd inner layer could shift to the 3rd and so on. On the boundaries of the narrow band some voxels are dropped from Inner Layer N and some new voxels are added to Outer Layer N. The pointers in the relevant vectors are kept

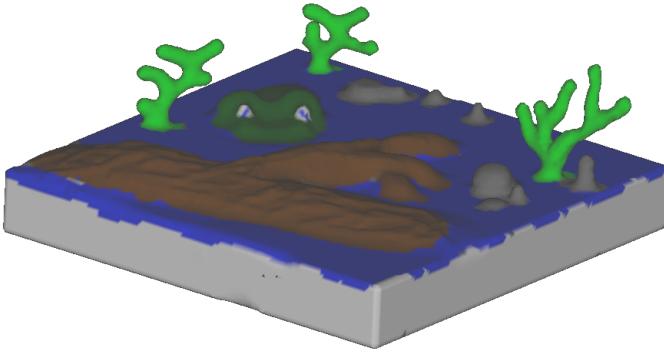


Figure 19: Lake with unusual inhabitants. The model is created on one side of a box using several of the level-set editing operators.

up-to-date with every change made to the original VISPACK lists, e.g. the pointer to the voxel that is dropped from VISPACK Inner Layer N also is also removed from the associated vector. The scenario is similar when adding a new voxel to any list. Figure 18 presents a 2D example that demonstrates the changes in the narrow-band linked lists as the surface evolves. The layers of the narrow band are color coded such that the red cells represent the surface voxels, blue cells represent inner and green cells represent outer layers. A legend is also provided at the bottom of the figure.

We also keep a 3D array of pointers at the same resolution of the volume to gain constant time access to any element in the VISPACK linked lists. These pointers point to the entries in the linked lists associated with the (i,j,k) locations in the 3D grid. This array is used to initialize the pointer vectors mentioned above when the user clicks on an arbitrary point on the surface. The size of the vectors, i.e. Inside/Outside Layer 1 through N and the Surface Voxels, is directly proportional with the size of the surface area that is being modified and is negligible for the operators discussed in this paper in comparison with the overall memory usage. For example, the combined size of these vectors is $(2 * N + 1) * M$, when editing M voxels on the surface. $N = 3$ is the width of the narrow band in our implementation. These additional data structures produce a considerable speed-up when working with high resolution volumes. We achieved approximately 10 times faster running times using the additional pointer vector data

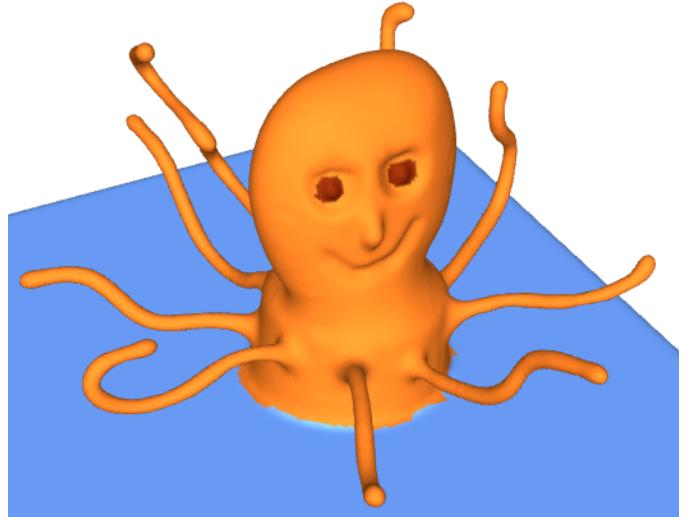


Figure 20: Cartoon octopus. The body of the octopus is created on one side of a box using the sketch-based editing operator. The head and the arms are grown from the body by pulling on points using a symmetrical ROI and the eyes are carved into the head.

structured compared to using just the original VISPACK narrow-band data structures for a 128^3 volume with an ROI that has a 20 voxel radius with the editing operators from Section 4.2. The surface embedded in this volume has approximately $128^2 \sim 16K$ voxels. There are approximately $\pi r^2 = \pi 20^2 \sim 1.2K$ voxels within the 20 voxel wide symmetric ROI. Since the running time is linear with the number of voxels processed, our data structures provide more than 10 times speed up by restricting the computations to a small area on the surface. Therefore, the extra memory usage allows us to create an interactive modeling framework.

6. Results

We created several examples to demonstrate the modeling capabilities of our editing operators. Figure 19 is a lake with plants, rocks, a floating log and an imaginary animal (a cross between a frog and a hippo). The initial model is a $140 \times 140 \times 20$ box within a $161 \times 161 \times 101$ volume. The model is created on one side of the box using a mix of editing

operators explained in Section 4 and Table 3. Figure 20 also uses the box model to create the body of the octopus. In this case we doubled the resolution of the box to $322 \times 322 \times 202$. The head, nose and the arms are grown from the body and the eyes and the mouth are carved into the head. We erased the spout and top handle of the Utah teapot and added new decorative handles to create the model in Figure 21. The dimensions of the teapot model is $156 \times 232 \times 124$. In the next example we edit a $200 \times 250 \times 200$ superellipsoid within a $401 \times 401 \times 401$ volume. The eyes are added by pulling the surface up and the mouth is modeled using interactive carving (See Figure 22). Figure 23 shows a fantasy character created from the mannequin head. The hair, horns and eyebrows are added, and the ears, eyes, nose and chin are modified. The resolution of this model is $360 \times 435 \times 510$. The body of the cartoon bear in Figures 1 and 24 is designed as the union of two superellipsoids in a $320 \times 320 \times 600$ volume. The arms, claws and the coat detail are added using the surface detailing tool. The legs and the nose are pulled out from the body symmetrically and the eyes are carved in. We applied our editing operators to a dataset obtained from a vasculature MRI scan (Figure 25). This dataset contains topological errors inherent to 3D scanning. Interactive carving is used to separate two blood vessels that were incorrectly merged. A blood vessel was interactively connected that was incorrectly split during reconstruction by pulling on one end of the vessel and merging it with the other end. The dimension of the model is $621 \times 371 \times 346$.

Table 3 summarizes all of the editing operators used to create these results along with the running times in frames per second (fps) on an Apple MacPro desktop computer with an Intel 8-core 3.2GHz CPU and 14 GB of memory running MacOS 10.5. Running times demonstrate interactive rates for a variety of volume resolutions.

The evaluation time for the speed functions is linear with the number of voxels within the ROI. This number is directly related to the radius of the ROI, which is defined in voxels. In other words, changing the resolution does not change the running time of a certain operator, however, one would need to double the ROI radius to cover



Figure 21: The Utah teapot is modified to create a decorative two-handle teapot. The spout and top handle are erased and new handles are added. Edits are made to one side of the model and a volumetric reflection operator is used to create the symmetric result.



Figure 22: Cartoon frog. A superellipsoid is used as the initial head model. The eyes are added by pulling the surface up and the mouth is modeled using interactive carving.

the same spatial area represented at half the resolution. The local editing operations run independent from the actual volume resolution. A good proof to this claim can be observed in Figures 19 and 23. As stated in Table 3, the operation for creating the plants and rocks on the lake is the same one used to create the horns on the mannequin head. The second model is approximately 10 times larger in surface area than the first, however, both editing operations run around 200 fps as we create the tip of the horns which have the same size ROI as the plants on the lake. Table 2 shows running times in terms of frames-per-second for a sphere model with given radius and

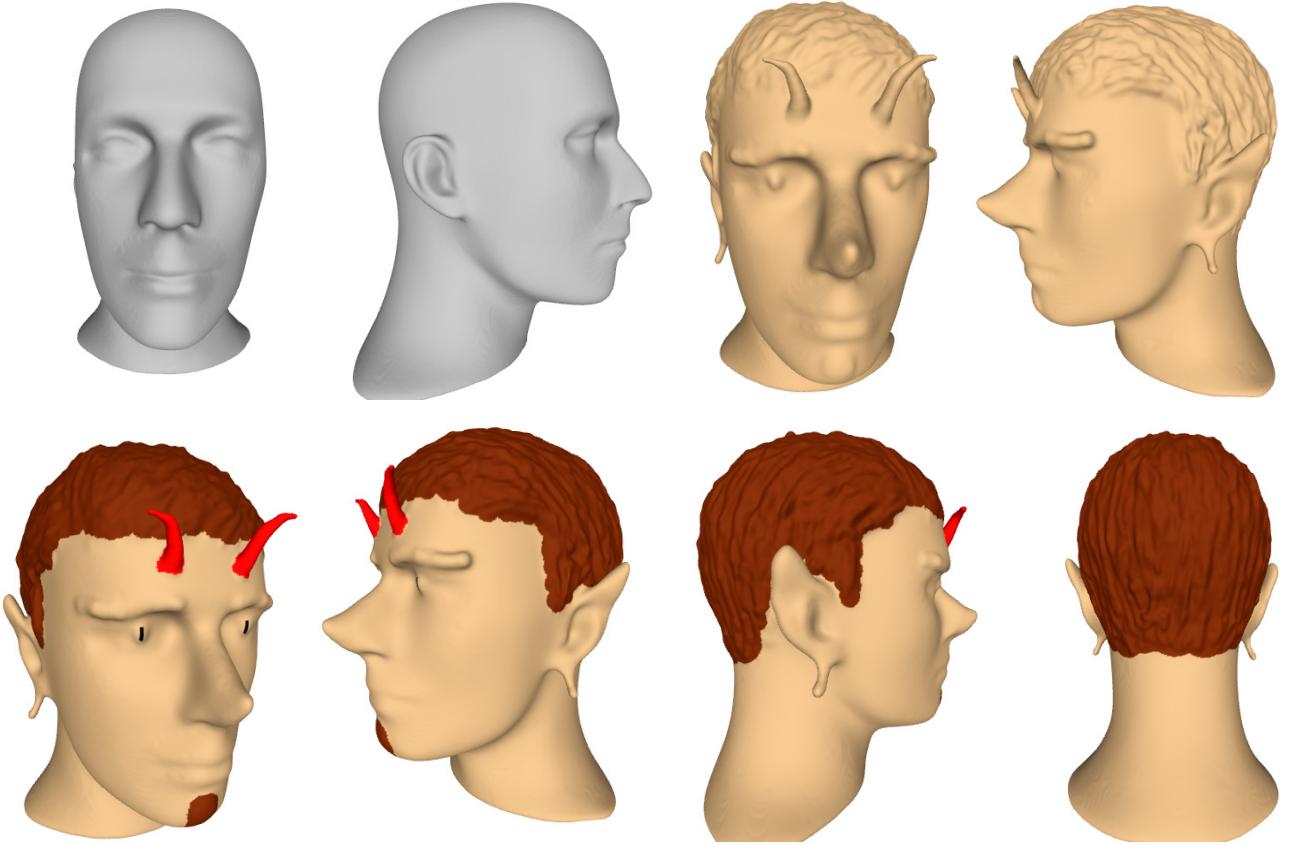


Figure 23: A fantasy character is created by adding horns and pointy ears to the mannequin model. The chin, eyes and nose are also modified and hair detail is added.

the same operator, i.e. pulling a point with symmetric ROI. The number of voxels within the ROI gets four times bigger every time the ROI radius doubles. The comparison of running times as the resolution increase shows that the run times are linear with the number of voxels processed for that operation. A comparison of the last two rows of Table 2 also shows that the running times are independent from the volume resolution or the surface area of the model.

While narrow-band schemes effectively address the problem of computational complexity in the original level-set formulation, they explicitly store a full Cartesian grid and use additional data structures to identify the narrow-band grid voxels. More advanced data structures have been proposed to address level-set memory issues, i.e. octrees [81], DT-Grid [82], and run-length encoding (RLE) [83]. These data structures reduce the memory requirements from $O(n^3)$ to $\sim O(n^2)$. However, they all

suffer from logarithmic random access times. Octrees are also used in adaptive sampling of 3D distance fields [17] to create adaptive representations of 3D volumetric models. Adaptive Distance Fields (ADFs) are also not applicable to our work because the level-set equation must be solved on a uniform grid.

7. Conclusion and Future Work

We have created a set of interactive, free-form editing operators for level-set models. The mathematics, algorithms and techniques needed to implement numerous level-set modeling capabilities have been developed. An OpenGL interface is combined with the VISPACK level-set library to create an interactive level-set modeling system. VISPACK'S narrow-band data structures have been enhanced to localize all computations and updates to improve running times. We have designed several speed functions that

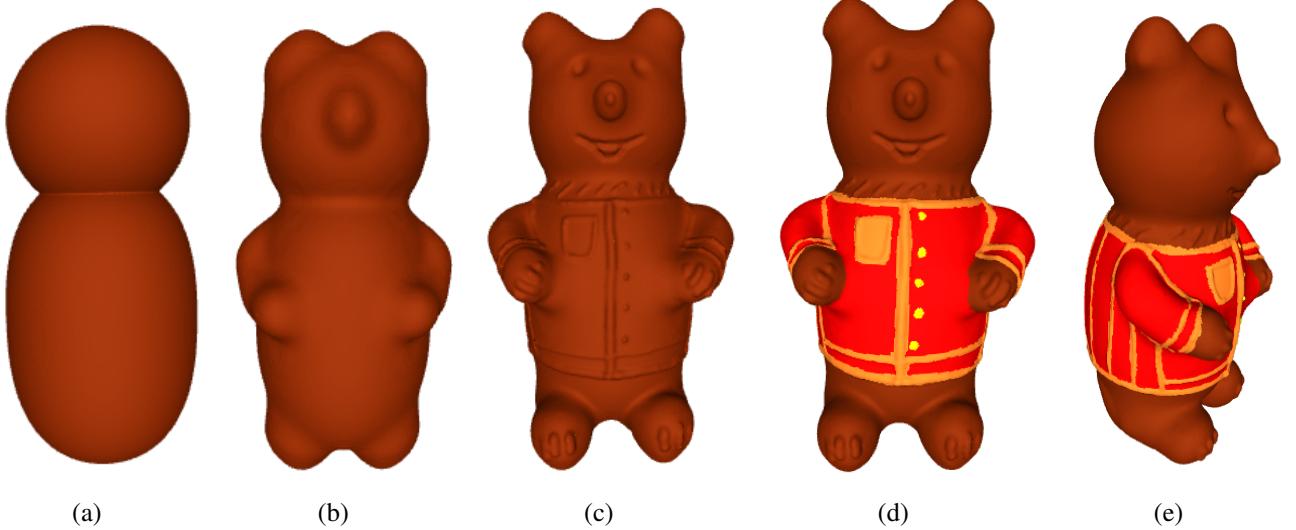


Figure 24: A cartoon bear is created using level-set surface editing operators. (a) The initial body is modeled with the union of two superellipsoids. (b-c) The bear is created using a collection of operators, e.g. surface detailing, carving, pulling on a point with symmetric ROI. (d-e) The painted final model is shown from two different angles.

implement novel level-set surface-editing operators. A 3D painting capability has also been added to the system. These operators allow a level-set model to be stretched and shaped, split into pieces, bent and merged smoothly. Topology changes occur naturally and automatically because of the properties of level-set models.

Our work stands apart from previous work in that it is the first to develop free-form editing operators within a level-set framework. As compared to previous PDE-based systems, ours is the first that is able to interactively modify and deform a volumetric implicit surface by solving a PDE. Several examples have been presented to demonstrate the flexibility and usefulness of the editing operators. These initial examples have been created with low-resolution volume datasets, which limit the amount of surface detail that can be specified.

A major limitation while editing level-set models is the memory storage costs associated with the increasing model resolutions. Now that the mathematics of the operators have been designed, tested and validated, the next phase of our work will implement these editing capabilities within a high-resolution level-set framework, e.g. DT-grids [82]. It

is expected that additional data structures will once again need to be developed to localize computation and expedite data access. This will allow us in the future to create level-set models with much finer surface structures and shapes. One of the advantages of level-set models, the ability to automatically handle topology changes, turns out to be a liability when a priori knowledge of the topology exists for the object to be segmented. [84, 85, 86] address this issue by introducing restrictions to the evaluation of the level-set function. It is crucial and also very straightforward to incorporate their work into our modeling system to give the user control over topology changes during surface editing. We also intend to make several improvements to the editing application itself. An undo capability will be added using an editing history, as well as additional mirroring and handle reuse capabilities.

Our framework is capable of performing interactive surface manipulations. These manipulations are currently limited to adding and removing surface details as well as locally reshaping level-set models. It is still an open question to do more structural deformations such as bending a tube at a fixed point using level-set methods. There are skeleton-based implicit techniques to achieve

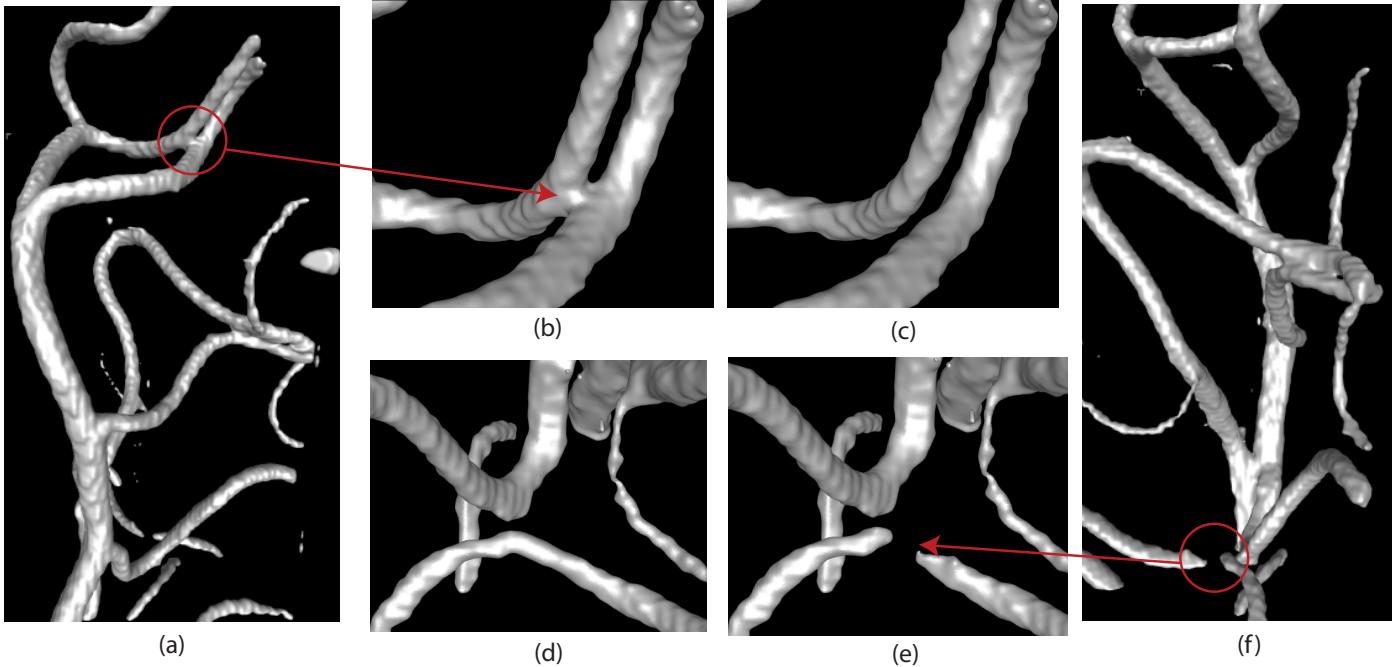


Figure 25: Topological repair of a vasculature data set. (a and f) The original model. (b-c) The volume is manipulated using interactive carving to separate two vessels that were merged due to errors in 3D scanning. (d-e) The volume is manipulated to recover lost data by connecting a vessel that was separated.

these kinds of deformations [47, 48, 87].

Acknowledgements. The authors would like to thank Ross Whitaker for the use of and assistance with the VISPACK library, Karan Thaker for his contributions to Figures 1, 23 and 24, and Linge Bai for her contributions to Figure 8. This research was supported by NSF grant CCF-0702441.

References

- [1] S. Osher, J. Sethian, Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations, *Journal of Computational Physics* 79 (1988) 12–49.
- [2] J. Sethian, *Level Set Methods and Fast Marching Methods*, 2nd Edition, Cambridge University Press, Cambridge, UK, 1999.
- [3] S. Osher, R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*, Springer, Berlin, 2002.
- [4] J. Huang, Y. Li, R. Crawfis, S. C. Lu, S. Y. Liou, A complete distance field representation, in: Proc. IEEE Visualization, 2001, pp. 247–254.
- [5] J. Huang, R. Crawfis, Adaptively represented complete distance fields, in: *Geometric Modeling for Scientific Visualization*, Springer, 2008, pp. 225–240.
- [6] P. Novotny, L. Dimitrov, M. Sramek, Enhanced voxelization and representation of objects with sharp details in truncated distance fields, to be published in *IEEE Transactions on Visualization and Computer Graphics* (2009).
- [7] M. Eyiurekli, C. Grimm, D. Breen, Editing level-set models with sketched curves, in: *Proceedings of Eurographics/ACM Symposium on Sketch-Based Interfaces and Modeling*, 2009, pp. 45–52.
- [8] A. Kaufman, D. Cohen, R. Yagel, Volume graphics, *Computer* 26 (7) (1993) 51–64.
- [9] T. Galyean, J. Hughes, Sculpting: An interactive volumetric modeling technique, in: Proc. SIGGRAPH, 1991, pp. 267–274.
- [10] K. L. Perng, W. T. Wang, M. Flanagan, M. Ouhyoung, A real-time 3D virtual sculpting tool based on modified marching cubes, *International Conference on Artificial Reality and Tele-Existence* 11 (2001) 64–72.
- [11] S. Wang, A. Kaufman, Volume sculpting, in: Proc. Symposium on Interactive 3D Graphics, 1995, pp. 151–156.
- [12] S. Wang, A. Kaufman, Volume-sampled 3D modeling, *IEEE Computer Graphics and Applications* 14 (5) (1994) 26–32.
- [13] E. Ferley, M.-P. Cani, J.-D. Gascuel, Practical volumetric sculpting, *The Visual Computer* 16 (8) (2000) 469–480.
- [14] E. Ferley, M.-P. Cani, J.-D. Gascuel, Resolution adaptive volume sculpting, *Graphical Models* 63 (6) (2001) 459–478.
- [15] K. McDonnell, H. Qin, R. Wlodarczyk, Virtual clay: A real-time sculpting system with haptic toolkits, in: Proc.

- Symposium on Interactive 3D Graphics, 2001, pp. 179–190.
- [16] R. Blanch, E. Ferley, M.-P. Cani, J.-D. Gascuel, Non-realistic haptic feedback for virtual sculpture, Tech. Rep. RR-5090, INRIA, U.R. Rhone-Alpes (January 2004).
- [17] S. Frisken, R. Perry, A. Rockwood, T. Jones, Adaptively sampled distance fields: A general representation of shape for computer graphics, in: Proc. SIGGRAPH, 2000, pp. 249–254.
- [18] R. Perry, S. Frisken, Kizamu: A system for sculpting digital characters, in: Proc. SIGGRAPH, 2001, pp. 47–56.
- [19] S. F. Frisken, R. N. Perry, Designing with distance fields, in: ACM SIGGRAPH 2006 Course #2 Notes, 2006, pp. 60–66.
- [20] M. Jones, J. Baerentzen, M. Sramek, 3D distance fields: A survey of techniques and applications, *IEEE Transactions on Visualization and Computer Graphics* 12 (4) (2006) 581–599.
- [21] M. C. Leu, W. Zhang, Virtual sculpting with surface smoothing based on level set method, *CIRP Annals - Manufacturing Technology* 57 (2008) 167–170.
- [22] [link].
URL <http://www.3d-coat.com>
- [23] T. Sederberg, S. Parry, Free-form deformation of solid geometric models, in: Proc. SIGGRAPH, 1986, pp. 151–160.
- [24] K. Singh, E. Fiume, Wires: a geometric deformation technique, in: Proc. SIGGRAPH, 1998, pp. 405–414.
- [25] H. Arata, Y. Takai, N. Takai, T. Yamamoto, Free-form shape modeling by 3D cellular automata, in: Proc. International Conference on Shape Modeling and Applications, 1999, pp. 242–247.
- [26] J. Hua, H. Qin, Scalar-field guided adaptive shape deformation and animation, *The Visual Computer* 20 (1) (2004) 47–66.
- [27] A. Angelidis, G. Wyvill, M.-P. Cani, Sweepers: Swept user-defined tools for modeling by deformation, in: Proc. International Conference on Shape Modeling and Applications, 2004, pp. 63–73.
- [28] A. Angelidis, M. Cani, G. Wyvill, S. King, Swirling-sweepers: constant-volume modeling, *Graphical Models* 68 (4) (2006) 324–332.
- [29] W. von Funck, H. Theisel, H.-P. Seidel, Vector field based shape deformations, *ACM Transactions on Graphics (SIGGRAPH 2006)* 25 (3) (2006) 1118–1125.
- [30] W. von Funck, H. Theisel, H.-P. Seidel, Explicit control of vector field based shape deformations, in: Proc. Pacific Conference on Computer Graphics and Applications, 2007, pp. 291–300.
- [31] K. McDonnell, H. Qin, PB-FFD: A point-based technique for free-form deformation, *Journal of Graphics Tools* 12 (3) (2007) 25–41.
- [32] J. Bloomenthal, et al. (Eds.), *Introduction to Implicit Surfaces*, Morgan Kaufmann, 1997.
- [33] L. Velho, J. Gomes, L. H. de Figueiredo, *Implicit Objects in Computer Graphics*, Springer, 2002.
- [34] G. Wyvill, C. McPheevers, B. Wyvill, Data structures for soft objects, *The Visual Computer* 2 (4) (1986) 227–234.
- [35] B. Wyvill, C. McPheevers, G. Wyvill, Animating soft objects, *The Visual Computer* 2 (4) (1986) 235–242.
- [36] B. Wyvill, G. Wyvill, Field functions for implicit surfaces, *The Visual Computer* 5 (1&2) (1989) 75–82.
- [37] A. Barr, Global and local deformations of solid primitives, in: Proc. SIGGRAPH, 1984, pp. 21–30.
- [38] B. Wyvill, E. Galin, A. Guy, Extending the CSG tree. warping, blending and boolean operations in an implicit surface modeling system, *Computer Graphics Forum* 18 (2) (1999) 149–158.
- [39] J. Bloomenthal, B. Wyvill, Interactive techniques for implicit modeling, *Computer Graphics (Proc. 1990 Symposium on Interactive 3D Graphics)* 24 (2) (1990) 109–116.
- [40] R. Schmidt, B. Wyvill, E. Galin, Interactive implicit modeling with hierarchical spatial caching, in: Proc. International Conference on Shape Modeling and Applications, 2005, pp. 104–113.
- [41] R. Schmidt, B. Wyvill, Generalized sweep templates for implicit modeling, in: Proc. 3rd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia (GRAPHITE '05), 2005, pp. 187–196.
- [42] R. Schmidt, B. Wyvill, M. Sousa, J. Jorge, Shapeshop: Sketch-based solid modeling with blobtrees, in: Proc. 2nd Eurographics Workshop on Sketch-Based Interfaces and Modeling, 2005, pp. 53–62.
- [43] M. Sugihara, E. de Groot, B. Wyvill, R. Schmidt, A sketch-based method to control deformation in a skeletal implicit surface modeler, in: Proc. 5th Eurographics Workshop on Sketch-Based Interfaces and Modeling, 2008, pp. 65–72.
- [44] M. Desbrun, M.-P. Cani, Animating soft substances with implicit surfaces, in: Proc. SIGGRAPH, 1995, pp. 287–290.
- [45] M.-P. Cani, M. Desbrun, Animation of deformable models using implicit surfaces, *IEEE Transactions on Visualization and Computer Graphics* 3 (1) (1997) 39–50.
- [46] M. Desbrun, M.-P. Cani, Active implicit surface for animation, in: Proc. Graphics Interface, 1998, pp. 143–150.
- [47] A. Angelidis, M.-P. Cani, Adaptive implicit modeling using subdivision curves and surfaces as skeletons, in: Proc. Seventh ACM Symposium on Solid Modeling and Applications, 2002, pp. 45–52.
- [48] A. Angelidis, P. Jepp, M.-P. Cani, Implicit modeling with skeleton curves: Controlled blending in contact situations, in: Proc. Shape Modeling International Conference, 2002, pp. 137–144.
- [49] S. Hornus, A. Angelidis, M.-P. Cani, Implicit modelling using subdivision curves, *The Visual Computer* 19 (2-3) (2003) 94–104.
- [50] G. Turk, J. O'Brien, Modeling with implicit surfaces that interpolate, *ACM Transactions on Graphics* 21 (4) (2002) 855–873.
- [51] O. Karpenko, J. Hughes, R. Raskar, Free-form sketching

- with variational implicit surfaces, Computer Graphics Forum 21 (3) (2002) 585–594.
- [52] B. Araujo, J. Jorge, Blobmaker: Free-form modelling with variational implicit surfaces, in: Proc. 12th Encontro Portugues de Computacao Graca, 2003, pp. 17–26.
- [53] C. L. Tai, H. Zhang, J. C.-K. Fong, Prototype modeling from sketched silhouettes based on convolution surfaces, Computer Graphics Forum 23 (1) (2004) 71–83.
- [54] A. Alexe, V. Gaildrat, L. Barthe, Interactive modeling from sketches using spherical implicit functions, in: Proc. AFRIGRAPH '04, 2004, pp. 25–34.
- [55] R. Zelenik, K. Herndon, J. Hughes, Sketch: an interface for sketching 3D scenes, in: Proc. SIGGRAPH, 1996, pp. 163–170.
- [56] T. Igarashi, S. Matsuoka, H. Tanaka, Teddy: A sketching interface for 3-D freeform design., in: Proc. SIGGRAPH, 1999, pp. 409–416.
- [57] T. Igarashi, J. Hughes, Smooth meshes for sketch-based freeform modeling., in: Proc. ACM Symposium on Interactive 3D Graphics, 2003, pp. 139–142.
- [58] A. Nealen, T. Igarashi, O. Sorkine, M. Alexa, Fibermesh: designing freeform surfaces with 3D curves, ACM Transactions on Graphics (SIGGRAPH 2007) 26 (3) (2007) 41.
- [59] Y. Mori, T. Igarashi, Plushie: an interactive design system for plush toys, ACM Transactions on Graphics (SIGGRAPH 2007) 26 (3) (2007) 45.
- [60] J. Zimmermann, A. Nealen, M. Alexa, Silsketch: automated sketch-based editing of surface meshes, in: Proc. 4th Eurographics Workshop on Sketch-based Interfaces and Modeling, 2007, pp. 23–30.
- [61] J. Cherlin, F. Samavati, M. Sousa, J. Jorge, Sketch-based modeling with few strokes, in: Proc. Spring Conference on Computer Graphics, 2005, pp. 137–145.
- [62] R. Schmidt, K. Singh, Sketch-based procedural surface modeling and compositing using surface trees, Computer Graphics Forum 27 (2) (2008) 321–330.
- [63] S. Owada, F. Nielsen, K. Nakazawa, T. Igarashi, A sketching interface for modeling the internal structures of 3D shapes, in: Proc. 4th International Symposium on Smart Graphics, 2003, pp. 49–57.
- [64] J. Baerentzen, N. Christensen, Volume sculpting using the level-set method, in: Proc. International Conference on Shape Modeling and Applications, 2002, pp. 175–182.
- [65] K. Museth, D. Breen, R. Whitaker, A. Barr, Level set surface editing operators, ACM Transactions on Graphics (Proc. SIGGRAPH) 21 (3) (2002) 330–338.
- [66] K. Museth, D. Breen, R. Whitaker, S. Mauch, D. Johnson, Algorithms for interactive editing of level set models, Computer Graphics Forum 24 (4) (2005) 821–841.
- [67] P. Mullen, A. McKenzie, Y. Tong, M. Desbrun, A variational approach to Eulerian geometry processing, ACM Transactions on Graphics (SIGGRAPH 2007) 26 (3) (2007) 66.
- [68] J. Zhang, Y. Lihua, Surface representation using second, fourth and mixed order partial differential equations, in: Proc. International Conference on Shape Modeling and Applications, 2001, pp. 250–256.
- [69] H. Du, Interactive shape design using volumetric implicit PDEs, in: Proc. ACM Symposium on Solid Modeling and Applications, 2003, pp. 235–246.
- [70] H. Du, H. Qin, A shape design system using volumetric implicit PDEs, Computer Aided Design 36 (11) (2004) 1101–1116.
- [71] H. Du, H. Qin, Dynamic PDE-based surface design using geometric and physical constraints, Graphical Models 67 (1) (2005) 43–71.
- [72] H. Du, H. Qin, Free-form geometric modeling by integrating parametric and implicit PDEs, IEEE Transactions on Visualization and Computer Graphics 13 (3) (2007) 549–561.
- [73] J. Lawrence, T. Funkhouser, A painting interface for interactive surface deformations, Graphical Models 66 (6) (2004) 418–438.
- [74] M. Eyiurekli, D. Breen, Localized editing of Catmull-Rom splines, Computer-Aided Design and Applications 6 (3) (2009) 307–316.
- [75] A. Barr, Superquadrics and angle-preserving transformations, IEEE Computer Graphics and Applications 1 (1) (1981) 11–23.
- [76] K. Singh, H. Pedersen, V. Krishnamurthy, Feature based retargeting of parameterized geometry, in: Proc. Geometric Modeling and Processing, 2004, pp. 163–172.
- [77] R. Whitaker, VISPACK, Tech. Rep. UUCS 08-0011, School of Computing, University of Utah (2008).
- [78] R. Whitaker, A level-set approach to 3D reconstruction from range data, International Journal of Computer Vision 29 (3) (1998) 203–231.
- [79] W. Lorensen, H. Cline, Marching Cubes: A high resolution 3D surface construction algorithm, in: Proc. SIGGRAPH, 1987, pp. 163–169.
- [80] J. Bloomenthal, An implicit surface polygonizer, in: Graphics Gems IV, Academic Press, 1994, pp. 324–349.
- [81] F. Losasso, F. Gibou, R. Fedkiw, Simulating water and smoke with an octree data structure, ACM Transactions on Graphics (SIGGRAPH 2004) 23 (3) (2004) 457–462.
- [82] M. Nielsen, K. Museth, Dynamic tubular grid: An efficient data structure and algorithms for high resolution level sets, Journal of Scientific Computing 26 (3) (2006) 261–299.
- [83] B. Houston, M. Nielsen, C. Batty, O. Nilsson, K. Museth, Hierarchical RLE level set: A compact and versatile deformable surface representation, ACM Transactions on Graphics 25 (1) (2006) 151–175.
- [84] X. Han, C. Xu, J. Prince, A topology preserving level set method for geometric deformable models, IEEE Transactions on Pattern Analysis and Machine Intelligence 25 (6) (2003) 755–768.
- [85] S. Bischoff, L. Kobbelt, Sub-voxel topology control for level set surfaces, Computer Graphics Forum (Proc. Eurographics 2003) 22 (3) (2003) 273–280.
- [86] F. Ségonne, J. Pons, E. Grimson, B. Fischl, Active contours under topology control - genus preserving level sets,

- in: Proc. International Workshop on Computer Vision for Biomedical Image Applications, 2005, pp. 135–145.
- [87] M. Sugihara, E. de Groot, B. Wyvill, R. Schmidt, A sketch-based method to control deformation in a skeletal implicit surface modeler, in: Proceedings of the 5th Eurographics Workshop on Sketch-Based Interfaces and Modeling, 2008, pp. 65–72.

8. APPENDIX

Algorithm 1 SweepGeodesic ($x_s, LIST, DIST$) :
This algorithm computes a list of voxels within a ROI along with their geodesic distances to a point x_s .

{LIST is the list of voxels within the ROI.}
 {DIST keeps the geodesic distances between the voxels within the ROI and x_s .}

```

for all voxels  $V$  in 1-Neighborhood of  $x_s$  do
  add  $V$  to LIST
  add  $\|V - x_s\|$  to DIST
end for

start = LIST.begin(), end=LIST.end()
for all voxels  $V$  in LIST [ start : end ] do
  for all voxels  $V_N$  in 1-Neighborhood of  $V$  do
    if  $V_N$  is a surface crossing voxel within ROI
    then
      if  $V_N$  is NOT in LIST then
        add  $V_N$  to LIST
        add  $DIST[V] + \|V - V_N\|$  to DIST
      else
        if  $DIST[V] + \|V - V_N\| < DIST[V_N]$ 
        then
           $DIST[V_N] = DIST[V] + \|V - V_N\|$ 
        end if
      end if
    end if
  end for
  start = end, end=LIST.end()
end for
```

| Operator | Speed Function | Result |
|---|--|--------|
| Pulling a point, symmetric ROI | $F(\mathbf{x}) = \begin{cases} \cos^\alpha(\pi/2 * d_s(\mathbf{x})/r) & d_s(\mathbf{x}) \leq r \\ 0 & d_s(\mathbf{x}) > r \end{cases}$ <p>\mathbf{x}: point on the surface being evaluated $d_s(\mathbf{x})$: geodesic distance from the point \mathbf{x} to the point being dragged α: user defined parameter that can be used to further control the shape</p> | |
| Pulling a point, arbitrary ROI | $F(\mathbf{x}) = f(d_{out}(\mathbf{x})) * \left(\frac{\max(d_{in}^{xs}(\mathbf{x})) - d_{in}^{xs}(\mathbf{x})}{\max(d_{in}^{xs}(\mathbf{x}))} \right)^\alpha$ $f(d) = \begin{cases} 1/2 - 1/2 * \cos(\pi * d(\mathbf{x})/\epsilon) & d \leq \epsilon \\ 1.0 & d > \epsilon \end{cases}$ <p>$d_{in}^{xs}(\mathbf{x})$: geodesic distance to \mathbf{x}_s from \mathbf{x} $d_{out}(\mathbf{x})$: geodesic distance to the boundary curve from \mathbf{x} ϵ defines a transition region near the edge of the ROI.</p> | |
| Pulling a curve on the surface, symmetric ROI | $F(\mathbf{x}) = \begin{cases} (1.0 - d_{in}^{cs}(\mathbf{x})/r)^\alpha & d_{in}^{cs}(\mathbf{x}) \leq r \\ 0 & d_{in}^{cs}(\mathbf{x}) > r \end{cases}$ <p>r: width of the ROI $d_{in}^{cs}(\mathbf{x})$: geodesic distance between \mathbf{x} and the curve C_s α is defined above</p> | |
| Pulling a curve on the surface, arbitrary ROI | $F(\mathbf{x}) = f(d_{out}(\mathbf{x})) * \left(\frac{\max(d_{in}^{cs}(\mathbf{x})) - d_{in}^{cs}(\mathbf{x})}{\max(d_{in}^{cs}(\mathbf{x}))} \right)^\alpha$ <p>$d_{in}^{cs}(\mathbf{x}), d_{out}(\mathbf{x}), f()$ and α are defined above</p> | |
| Surface Detailing/Carving | $F(\mathbf{x}) = \begin{cases} 0 & f_{se}(\mathbf{V}) > 0 \\ \beta * f_{se}(\mathbf{V}) & f_{se}(\mathbf{V}) \leq 0 \end{cases}$ <p>The tool is centered at the cursor point \mathbf{x}_c $V = \mathbf{x} - \mathbf{x}_c$ $f_{se}(\mathbf{V})$ evaluates the superellipsoid inside-outside function around the cursor location. $\beta = -1$ surface detailing, $\beta = +1$ carving</p> | |
| Sketching Cross-sections | $F(\mathbf{x}) = \frac{d_{up}(\mathbf{x})}{\max(d_{up}(\mathbf{x}))} * f(d_{out}(\mathbf{x})) * \left(\frac{\max(d_{in}^{cs}(\mathbf{x})) - d_{in}^{cs}(\mathbf{x})}{\max(d_{in}^{cs}(\mathbf{x}))} \right)^\alpha$ <p>d_{up} distance (see Section 4.7 for calculation) to cross section curve $d_{out}(\mathbf{x}), d_{in}^{cs}(\mathbf{x}), f(), \epsilon$ and α are defined above.</p> | |
| Interactive Smoothing | $F(\mathbf{x}) = \gamma * g(d_g(\mathbf{x})) * \kappa(\mathbf{x})$ $g(d) = \begin{cases} 1.0 & d \leq r - \epsilon \\ 1/2 + 1/2 * \cos(\pi * (d - r + \epsilon)/\epsilon) & r - \epsilon < d \leq r \\ 0.0 & d > r \end{cases}$ <p>γ: constant that controls the amount of smoothing $d_g(\mathbf{x})$: Euclidean distance from the point \mathbf{x} to the cursor \mathbf{x}_c κ: mean curvature. r: radius of the smoothing tool ϵ is defined above</p> | |

Table 1: Free-form Editing Operators

| Volume Resolution (voxels) | Sphere Radius (voxels) | ROI Radius (voxels) | Speed (fps) |
|-------------------------------|---------------------------|------------------------|----------------|
| 64^3 | 20 | 5 | 333 |
| 128^3 | 40 | 10 | 100 |
| 256^3 | 80 | 20 | 12.5 |
| 512^3 | 160 | 40 | 5 |
| 512^3 | 160 | 10 | 100 |

Table 2: Running times of a single operation at different resolutions. The number of voxels within the ROI gets four times bigger every time the radius of the ROI doubles. Running times are given in frames-per-second (fps).

| Editing Details | | Speed (fps) |
|--|-----------------------------------|----------------------|
| Lake model, Dimensions: $161 \times 161 \times 101$ | | |
| Plants and rocks | Pulling on a point, symmetric ROI | 200 |
| Surface on the rightmost plant | Surface detailing | 100-150 |
| Log | Surface Detailing | 20 |
| Animal (body) | Sketch-based editing | 12 |
| Animal (eyes) | Pulling on a curve, symmetric ROI | 34 |
| Animal (eyeballs) | Surface Detailing | 125 |
| Mannequin Head, Dimensions: $360 \times 435 \times 510$ | | |
| Hair, eyes and eyebrows | Surface Detailing | 100-200 |
| Horns | Pulling on a point, symmetric ROI | 100-200 |
| Nose and ears | Sketch-based editing | 40 |
| Chin | Pulling on a curve, symmetric ROI | 100 |
| Octopus, Dimensions: $322 \times 322 \times 202$ | | |
| Body and arms | Pulling on a point, symmetric ROI | 20 (body) 200 (arms) |
| Head | Sketch-based editing | 25 |
| Eyes | Interactive carving | 50 |
| Nose | Surface detailing | 100 |
| Teapot, Dimensions: $156 \times 232 \times 124$ | | |
| Erasing spout and top handle | Interactive carving | 25 |
| New handles | Pulling on a point, symmetric ROI | 100 |
| Cartoon frog, Dimensions: $401 \times 401 \times 401$ | | |
| Mouth | Interactive Carving | 10 |
| Tongue | Surface detailing | 50 |
| Eyes (balls) | Pulling on a point, symmetric ROI | 20 |
| Eyes (crosses) | Surface detailing | 50 |
| Cartoon bear, Dimensions: $320 \times 320 \times 600$ | | |
| Arms, claws and coat | Surface detailing | 50-150 |
| Legs, ears and nose | Pulling on a point, symmetric ROI | 50-75 |
| Eyes | Interactive carving | 50 |
| Aneurysm, Dimensions: $621 \times 371 \times 346$ | | |
| Splitting veins | Interactive carving | 100 |
| Connecting veins | Pulling on a point, symmetric ROI | 100 |

Table 3: Editing details and running times for the final results. Speed is in frames-per-second (fps).