Interactive Freeform Editing Techniques

for

Large-Scale, Multiresolution Level Set Models

A Thesis

Submitted to the Faculty

of

Drexel University

by

Manolya Eyiyurekli McCormick

in partial fulfillment of the

requirements for the degree

of

PhD in Computer Science

2012

Dedications

Canim ablama, huzur icinde yat. [1]

[1] To my beloved sister, may you rest in peace.

Acknowledgements

Sincere thanks to my advisor Dr. David E. Breen for paving the road and guiding me through it with great dedication. Further gratitude is expressed to all members of my thesis committee, Dr. David E. Breen, Dr. Ali Shokoufandeh, Dr. William Regli, Dr. Paul Diefenbach, Dr. Ross Whitaker and Dr. Ko Nishino for their ideas and opinions that shaped and reshaped my research and this document.

Special thanks to Dr. Ross Whitaker for the use of and his assistance with the VISPACK library.

I would also like to extend my deepest gratitude to those friends and family who helped me keep going against all obstacles on the way. Heartfelt thanks to my mom, dad, sister and brothers for never losing their hope, and always rekindling mine. Special thanks to John McCormick, Yelena Kushleyeva, Walt Mankowski, Nadya Sultanik and Linge Bai for taking the time to proof read this work, and for being such great friends.

Last but certainly not least, I am grateful for having the love and support of John and Sherlock, who shared a home with a cranky PhD student and never complained.

This research was financially supported by NSF grant CCF-0702441

Table of Contents

List of Tables

List of Figures

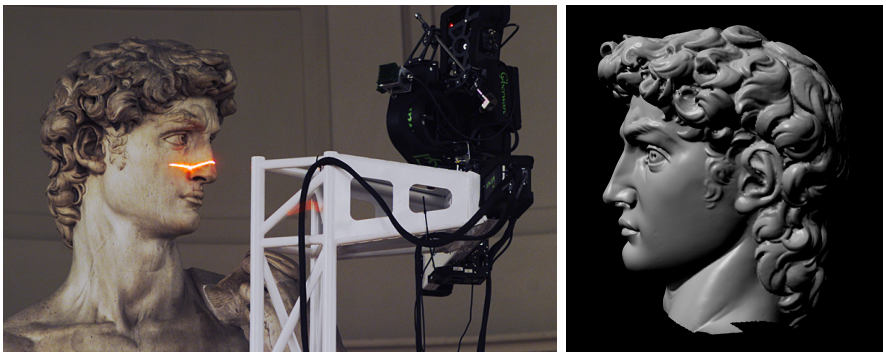
Abstract

Level set methods provide a volumetric implicit surface representation with automatic smooth blending properties and no self-intersections. They can handle arbitrary topology changes easily, and the volumetric implicit representation does not require the surface to be re-adjusted after extreme deformations. Even though they have found some use in movie productions and some medical applications, level set models are not highly utilized in either special effects industry or medical science. Lack of interactive modeling tools makes working with level set models difficult for people in these application areas.

This dissertation describes techniques and algorithms for interactive freeform editing of large-scale, multiresolution level set models. Algorithms are developed to map intuitive user interactions into level set speed functions producing specific, desired surface movements. Data structures for efficient representation of very high resolution volume datasets and associated algorithms for rapid access and processing of the information within the data structures are explained. A hierarchical, multiresolution representation of level set models that allows for rapid decomposition and reconstruction of the complete full-resolution model is created for an editing framework that allows level-of-detail editing. We have developed a framework that identifies surface details prior to editing and introduces them back afterwards. Combining these two features provides a detail-preserving level set editing capability that may be used for multi-resolution modeling and texture transfer. Given the complex data structures that are required to represent large-scale, multiresolution level set models and the compute-intensive numerical methods to evaluate them, optimization techniques and algorithms have been developed to evaluate and display the dynamic isosurface embedded in the volumetric data.

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 isosurfaces of a 3D scalar field, are becoming more prevalent and important to such disparate disciplines as special effects and medicine/biology. The scalar fields of these models are frequently represented by volume data sets, i.e. 3D rectilinear grids that store scalar values at grid crossings.



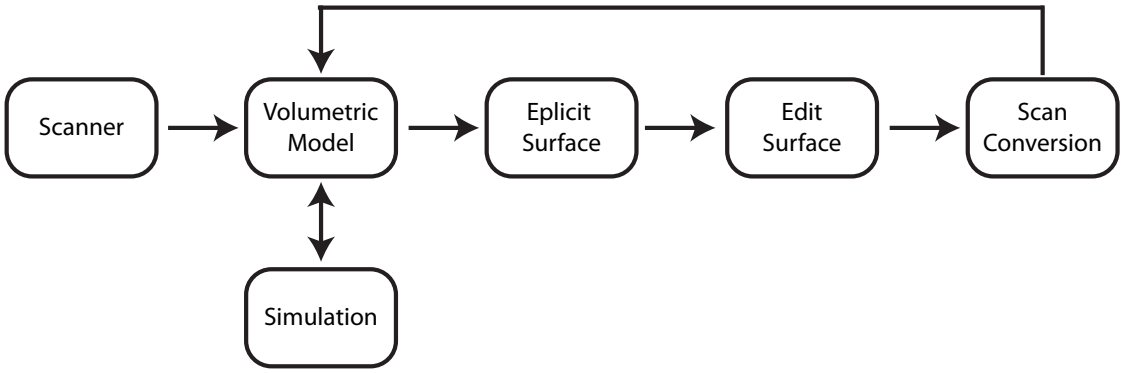
(a) (b)

Figure 1.1: (a) 3D laser scanner, scanning statue of David. (b)The 3D model reconstructed from the laser scan data. (Images from the Digital Michelangelo Project)

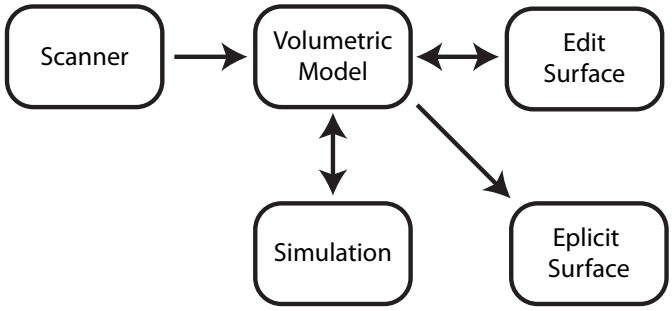
As imaging technology continues to be rapidly deployed and utilized in medicine and science, an increasing number of volume datasets are generated, producing an

overwhelming flood of raw volume data that must be processed, viewed and analyzed (Figure 1.1). Usually there are 3D surfaces embedded in these volume datasets that are of interest to doctors and scientists. In the field of computer graphics, laser scanning technology is used to acquire high resolution models of complex objects. The raw data from this process are large point sets or numerous distance maps that must be reconstructed to produce the final surface model. Many of the reconstruction algorithms generate volumetric, implicit representations of the objects before converting them into explicit surface models. Many of the advanced special effects in movies utilize physical simulation to produce computer-generated fluid flows of floods, storms, pouring/splashing liquids, etc. The computational fluid dynamic (CFD) calculations for these effects are usually done on a regular 3D grid and produce dynamic volume datasets as output.

In all three of these examples complex surfaces are acquired/produced and are represented implicitly with large-scale volume datasets. In medicine and science the implicit surface models of an object of interest must be extracted from the volume dataset. For most datasets these objects of interest are quite complex and cannot be automatically identified and extracted, and require significant user input and manipulation to produce the final desired result.. Laser-scanning-based model acquisition is not an error-free technology. Frequently, models automatically produced by this process must be manually fixed. In addition, designers may wish to edit/modify the model once it has been obtained. During special effects sequences simulations sometimes require complex initial condition configurations and mid-sequence corrections/redirections, as well as post-simulation clean-ups. In all of these cases large-scale implicit, volumetric models need to be modified and edited to meet scientific and artistic goals. Unfortunately, general editing capabilities for volumetric surface models do not exist, and frequently it becomes necessary to extract explicit surface



(a) Current pipeline for using volumetric models



(b) Pipeline for future modeling and simulation systems using volumetric models

Figure 1.2

models from the volumes, edit the surfaces, and re-convert the edited surface back into a volumetric, implicit form. This is an inefficient and cumbersome approach to the problem of editing these types of models.

In summary, there are several applications in computer graphics and medical science that use volumetric models. Usually, these models are converted into explicit surface representations before they can be utilized by such applications. The conversion and reconstruction algorithms are cumbersome and the raw data may have a significant amount of noise/errors, requiring user manipulation and clean up prior to further processing and analysis. Furthermore, drastic deformations of complex

explicit models generate a series of problems such as cracks and rough patches where surfaces meet. They require remeshing at the areas that get thinned or expanded too much. The topological errors caused by self intersections are nontrivial to correct. To the best of our knowledge there is currently a paucity of adequate volumetric editing tools capable of high resolution and high level surface manipulations. To address these shortcomings we have developed techniques and algorithms for interactive freeform editing of large-scale, multiresolution level set models. We believe that creating tools for directly editing volumetric, implicit models is the most logical and advantageous approach to modifying these models, rather than relying on conversion techniques and explicit surface editing capabilities. Figure 1.2 shows the current pipeline for using volumetric models as well as the new pipeline we have devised using our approach as explained in this thesis.

Level set models combine a low-level volumetric representation, the mathematics of deformable implicit surfaces, and robust numerical techniques to produce a powerful approach to geometric modeling. Level set model manipulations are based on formulating and solving a partial differential equation (PDE). They are guaranteed to define simple (non-self-intersecting) and closed surfaces, and they easily change topological genus, making them ideal for representing complex structures of unknown or transforming genus. These volumetric models support straightforward solid modeling operations and calculations, while simultaneously offering a surface modeling paradigm. The benefits offered by these features provide the motivation for utilizing level set models to process and manipulate volumetric, implicit surfaces and make them unique for applications utilizing complex surfaces with dynamically changing topology such as "amorphous" characters moving freely in an environment while interacting with other solid or soft objects, cracking or exploding surfaces, fluid and smoke simulations, as well as representing surfaces acquired from medical scan data.

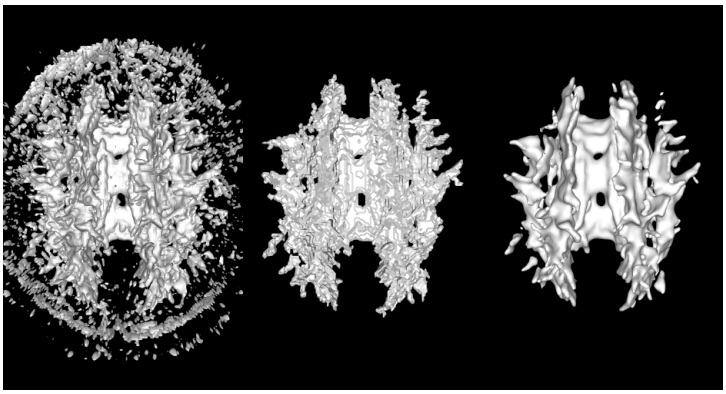
(a) (b)

(c)

(d) (e)

(f)

Figure 1.3: Screenshots of level set models utilized in special effects and animations. (a) The Sandman in Spider-Man 3. (b) The Tar Monster from Scooby Doo 2. (c) Terra cotta soldiers from The Mummy: Tomb Of The Dragon Emperor. (d) The giant maelstrom in Pirates of the Caribbean 3. (e) Dancers from Bacardi commercial. (f) Clouds from Puss in Boots.



(a)

Figure 1.4: Level set models are used in medical volume segmentation [Breen et al., 2005].

Even though they have found some use in special effects and animation (see Figure 1.3) as well as some medical applications such as volume segmentation (see Figure 1.4), level set models are not highly utilized in either special effects industry or medical science due to the memory requirements of storing the volumetric representation and the time consuming evaluations of the techniques and algorithms necessary for modifying such implicit surfaces at high resolutions. The space and time complexity of storing and deforming these models prevent them from being utilized in interactive modeling systems as well. A current state-of-the-art system uses models that contain one billion voxels and provides 25-30 frames-per-second(fps) evaluation time at these resolutions. Even the most advanced data structures and algorithms developed for level set models are not sufficient to support both of these requirements simultaneously.

Our work closes the gap between level set methods and interactive modeling applications by providing new techniques and algorithms to incorporate these models in state-of-the-art modeling frameworks. Aspects of another emerging modeling ap-

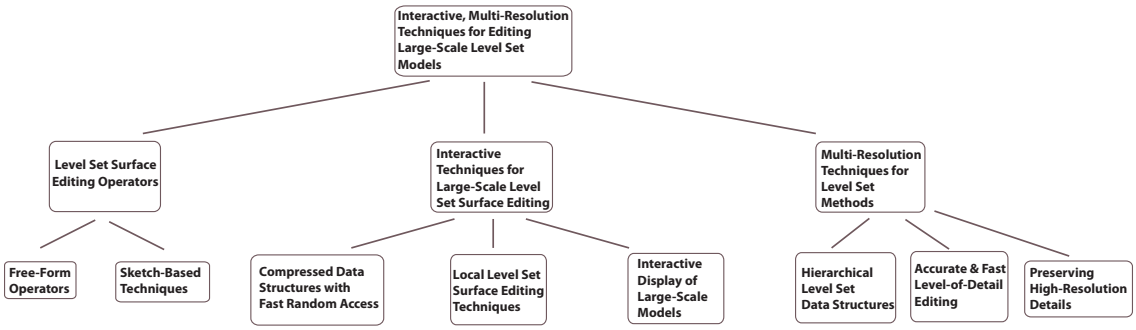


Figure 1.5: Research outline and objectives.

proach, point-based models, have been incorporated with level set models to provide enhanced, novel modeling capabilities. Algorithms and techniques needed to implement numerous level set modeling capabilities have been developed. A general outline of the research objectives is depicted in Figure 1.5. The contributions of this work are:

1. We have developed a set of free-form editing operators, which provide direct implicit surface manipulations, within a level set framework. These operators allow a user to add or remove surface detail from a level set model by interactively moving geometric handles attached to the surface. 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.

2. We have explored and evaluated data structures and techniques that enable for the first time the interactive editing and display of high resolution level set models. The high resolution models, if stored in a 3D array, would contain more than three billion voxels, and the interactive rates achieved by our level set modeling system is normally above 25 fps. As compared to previous

PDE-based modeling work, our system provides significantly faster processing speeds on much larger volumetric models, even when considering the difference in processor power.

3. We have developed a framework that identifies surface details prior to editing and introduces them back afterwards. Additionally we have developed techniques that allow a user to manipulate/edit a level set surface at different geometric scales and levels of detail. Combining these two features provides a detail-preserving level set editing capability that may be used for multiresolution modeling and texture transfer.

4. We have developed $2D$ schemes that provide a versatile, expressive and powerful localized curve editing capability for Catmull-Rom splines.

5. Given the complex data structures that are required to represent large-scale, multiresolution level set models and the compute-intensive numerical methods to evaluate them, optimization techniques and algorithms have been developed to evaluate and display the dynamic isosurface embedded in the data structure.

Application Areas: The outlined research provides benefits for a number of fields. Firstly, numerous fundamental advances for geometric modeling and Computer-Aided Design are produced, bringing new capabilities to implicit/level set modeling. These advances may find immediate use in two disparate application areas; special effects/animation and medical/biological imaging. Level set models are being used significantly to simulate fluid flow [Losasso et al., 2004, Enright et al., 2002b] and morphing effects [Breen and Whitaker, 2001, Wiebe and Houston, 2004] in animations and movies. Powerful modeling techniques will be useful for creating the initial conditions needed for the simulation of these effects. In medicine and biology, level set models are used for segmentation and processing of imaging datasets [Breen et al.,

2005, Yoo, 2004]. These techniques rarely provide a completely accurate solution; thus interactive modeling methods are needed, so doctors and biologists may fine-tune and correct mistakes made by the automated systems. Laser-scanning-based model acquisition is not an error-free technology. Frequently the volumetric models automatically produced by this process must be manually fixed, and designers may also wish to edit/modify the model once it has been obtained. Our editing framework provides the means to directly manipulate these volumetric data to correct and enhance the 3D reconstructions.

2. The Level Set Method

Level set models are defined as an isosurface (S), i.e. a level set, of a dynamic implicit function ϕ,

$$S = \{x \mid \phi(x,t) = k\}, \tag{2.1}$$

where $k \in \Re$ is the iso-value, $x \in \Re^3$ is a point in space on the isosurface and $\phi : \Re^3 \rightarrow \Re$ is a scalar function.

The scalar field (ϕ) of a level set model is represented by a volume dataset, each voxel of which stores a scalar value that represents the shortest distance to the implicit surface. Each voxel value has a sign, positive or negative, based on the position of the voxel with respect to the interface, i.e. inside or out. This scalar field is also called a signed distance field/function (SDF). Even though an SDF does not uniquely represent an isosurface, it is preferable due to the smoothness of the function near the surface and the simplifications it allows while approximating surface properties such as normals and curvatures. When ϕ is an SDF, the surface normal can be approximated by the gradient ($\nabla\phi$), and the curvature can be approximated by the Laplacian ($\Delta\phi$).

Level set methods [Osher and Sethian, 1988, Sethian, 1999, Osher and Fedkiw, 2002] provide the techniques needed to change the values of the implicit function in a way that moves the embedded isosurface. 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

$$\frac{\partial\phi}{\partial t} = -|\nabla\phi|F(x, D\phi, D^2\phi \ldots), \tag{2.2}$$

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.

Depending on the speed term $F()$, Equation 2.2 can either be a hyperbolic PDE (Equation 2.3) that defines a constant motion in the normal direction, or a parabolic PDE (Equation 2.4) that defines motion by mean curvature:

$$\frac{\partial \phi}{\partial t} = -|\nabla \phi| a, \tag{2.3}$$

$$\frac{\partial \phi}{\partial t} = -|\nabla \phi| b\kappa. \tag{2.4}$$

In Equation 2.3, $F()$ only depends on up to first derivatives of ϕ and 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 moving either inwards or outwards in the direction of the local normal with constant speed a. A variety of numerical methods exist to calculate spatial derivatives necessary to solve Equation 2.3. However, one must consider the direction in which the interface moves while picking grid points for gradient calculations. In one dimension, if the interface is moving from left to right, the points to the left of the interface must contribute to gradient calculations and vice versa. This method is called upwind differencing. A first-order accurate upwind differencing scheme approximates spatial derivatives by taking the difference between a grid point and its 1-ring neighbors. Higher order methods such as HJ-ENO [Harten et al., 1987] and HJ-WENO [Liu et al., 1994] provide up to fifth-order accurate approximations while using more neighboring grid points in calculations.

A second-order accurate central difference scheme can be used to approximate the mean curvature (κ) in the parabolic PDE in Equation 2.4. Here, b is a constant and the surface moves in the direction of convexity when $b < 0$ and in the direction of concavity otherwise.

The Forward Euler method provides a first-order accurate time discretization.

This method coupled with an upwind or central differencing scheme respectively is a consistent finite difference approximation to PDEs in Equations 2.3 and 2.4, since the approximation error converges to zero as $\Delta t \to 0$ and $\Delta x \to 0$. The approximation is convergent, i.e. the correct solution is obtained as $\Delta t \to 0$ and $\Delta x \to 0$, if and only if it is also stable. Stability guarantees that small errors in the approximation are not amplified as the solution is marched forward in time. Stability is enforced by using the Courant-Friedreichs-Lewy (CFL) conditions, which asserts that the numerical waves should propagate at least as fast as the physical waves. This leads to a time step restriction of

$$\Delta t < \frac{\Delta x}{\max(|F|)}. \tag{2.5}$$

Higher order methods also exist for a more accurate temporal discretization. TVD-RK schemes can be used to calculate up to third order accurate approximations to ϕ at time $t+\Delta t$ [Shu and Osher, 1988]. The Forward Euler method discussed above is a first order accurate TVD-RK scheme. Even though higher than third order schemes exist, they do not make a significant difference in practical applications.

A third type of motion advects the level set surface passively in an externally generated velocity field. Numerical techniques like upwind differencing and forward Euler can be used to solve these types of hyperpolic PDEs (Equation 2.6). The work described in this thesis does not use this type of motion to move the level set surface.

$$\frac{\partial \phi}{\partial t} = -|\nabla \phi| \vec{V} \tag{2.6}$$

As mentioned earlier, a number of simplifications can be made when ϕ is an SDF. It is also important that ϕ stays smooth enough to approximate its spatial derivatives with some degree of accuracy using finite difference schemes. As the interface evolves, ϕ will generally drift away from its initialized value as a signed distance and develop

noisy features and steep gradients. For these reasons, it is important to re-initialize ϕ as a signed distance to the interface. Some techniques exist to reinitialize ϕ as an approximate SDF. The SDF has the property

$$|\nabla\phi| = 1. \tag{2.7}$$

Equation 2.7 is called the Eikonal equation. The grid points adjacent to the interface are set initially and form the boundary conditions. The equation can be solved using upwind differencing to approximate the gradient. When the solution of the Eikonal equation reaches a steady state, ϕ is an SDF. Fast Marching Methods (FMM) [Sethian, 1995, Tsitsiklis, 1995] can mimic the solution to the Eikonal equation by marching distance values out from the interface to calculate the SDF at each grid point.

The sparse field method (SFM) [Whitaker, 1998] is an efficient algorithm that maintains ϕ as an approximate SDF. Even though it cannot be used to initialize a narrow-band just from a set of points on the interface, it can be used to maintain ϕ as an SDF during level set deformations once initialized by another method like FMM. SFM 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-initialize the entire distance field. The distance field is re-initialized on the fly as the values of ϕ are updated during the level set evolution.

3. Previous Work

3.1 3D Modeling

The related previous work in modeling and surface editing can be organized into five main categories: subdivision surfaces, volume sculpting, volume deformations, implicit modeling and PDE-based modeling. In this section we will describe each of these modeling approaches, present previous work on each topic and discuss why we believe a level set approach would be more beneficial over these approaches.

3.1.1 Subdivision Surfaces

Subdivision surfaces [Warren and Weimer, 2001, Peters and Reif, 2008, Andersson and Stewart, 2010] are polygonal mesh surfaces generated from a base mesh through an iterative process that subdivides the polygons; thus smoothing the mesh by increasing the density of polygons. Complex smooth surfaces can be derived in a reasonably predictable way from relatively simple base meshes. Many different schemes exist for the polygon subdivision process. The Catmull-Clark scheme [Catmull and Clark, 1978, Stam, 1998] is the most well-known, and is currently used in many high-end modeling and animation packages. This scheme generalizes bi-cubic uniform B-splines to produce a subdivision scheme. For arbitrary initial meshes, it generates limit surfaces that are C^2 continuous everywhere except at extraordinary vertices where they are C^1 continuous. Doo-Sabin subdivision surface [Doo and Sabin, 1978] is a type of subdivision surface based on a generalization of bi-quadratic uniform B-splines to produce C^1 limit surfaces with arbitrary topology for arbitrary initial meshes. Another popular scheme for surface subdivision is the Loop scheme [Loop, 1987], which works only on triangle meshes and generates C^2 continuous limit

surfaces everywhere except at extraordinary vertices where they are C^1 continuous.

Subdivision surfaces are easy to implement, they can model surfaces of arbitrary topological type, and the continuity of the surface can be controlled locally. However, their use has been hindered by the lack of a closed form, i.e. they are defined only as the limit of an infinite procedure. Additionally, subdivision surfaces are explicit surface representations that cannot be used to represent and modify volumetric models.

3.1.2 Volume Sculpting

Volume graphics [Kaufman et al., 1993] involves the synthesis, manipulation, and rendering of volumetric objects, which are stored as an array of voxels. Interactive sculpting tools for clay-like [Galyean and Hughes, 1991, Perng et al., 2001] and solid [Wang and Kaufman, 1995] models represent the material by voxel data and define tools that can add/remove material, as well as perform smoothing operations. Some sculpting metaphors utilize alias-free volume sampling [Wang and Kaufman, 1994] or uniformly sampled scalar fields [Ferley et al., 2000] as the volume representation. These efforts are extended in Ferley et al. [2001] to achieve interactive editing speeds using resolution adaptive volume sculpting and also in McDonnell et al. [2001] 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 [Blanch et al., 2004]. Volumetric models are frequently represented as uniform or adaptive 3D distance fields [Frisken et al., 2000, Perry and Frisken, 2001, Frisken and Perry, 2006, Jones et al., 2006]. Leu and Zhang [2008] describe a method to convert volume sculpted models into distance fields and apply curvature-based smoothing using the level set method.

Level set models are implicit surfaces embedded in volume datasets; thus support-

ing 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.

3.1.3 Volume Deformations

Freeform Deformations (FFDs) [Sederberg and Parry, 1986] 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 cellular automata [Arata et al., 1999] for transportation of mass through a 3D grid or evolving scalar fields that define deformations of polygonal models [Hua and Qin, 2004]. Space deformation techniques for interactive virtual sculpting [Angelidis et al., 2004, 2006] create a deformation field with a volumetric tool. There also exist vector field based deformations [von Funck et al., 2006, 2007] and point-based techniques [McDonnell and Qin, 2007] for performing freeform 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.

3.1.4 Implicit Modeling

Implicit models [Bloomenthal and Wyvill, 1997, Velho et al., 2002] are a widely used representation for geometric modeling applications. Soft Objects were one of the first successful systems based on implicit models [Wyvill et al., 1986b,a, Wyvill and Wyvill, 1989]. Wyvill et al. [1999] created an implicit modeling system that combines constructive solid geometry (CSG) operations with blending and warping. They use

a tree-based representation, called the BlobTree, where leaves of the tree are the primitives and inner nodes are the operations, i.e. warp, blend, union, intersection and difference, as well as Barr deformations [Barr, 1984]. The system's interactive performance was improved in Bloomenthal and Wyvill [1990], Schmidt et al. [2005a]. More techniques for generating 3D implicit sweep volumes compatible with these systems are described in Schmidt and Wyvill [2005]. These techniques are extended with a sketch-based editing framework [Schmidt et al., 2005b] . ShapeShop [Schmidt et al., 2005b] uses BlobTrees as the underlying shape representation for a sketch-based editing framework. A curve-based primitive is introduced that may be "inflated" or extruded. Sketch-based models are produced by combining this primitive with CSG and blending operations. This work was extended by the addition of a curve-based freeform deformation capability [Sugihara et al., 2008].

Desbrun and Cani [Desbrun and Cani, 1995, Cani and Desbrun, 1997] 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 isosurfaces defined within a 3D grid that stores a potential field [Desbrun and Cani, 1998].

Some other modeling systems use skeletons defined as a graph of interconnected subdivision curves and surfaces [Angelidis and Cani, 2002, Angelidis et al., 2002, Hornus et al., 2003], interpolating [Turk and O'Brien, 2002] or variational implicit surfaces [Karpenko et al., 2002, Araujo and Jorge, 2003], convolution surfaces [Tai et al., 2004] or spherical implicit functions [Alexe et al., 2004], to represent 3D models.

Arbitrary large-scale deformations to analytical implicit surfaces require a skeleton structure to create a volumetric model. Level set models have volumetric representations by definition. In contrast to skeleton-based implicit models, our operators

have been developed for direct small-scale modifications of implicit surfaces that are not tied to skeleton manipulation. Large-scale level set model deformations could be performed with skeletons. This is a topic of future work.

3.1.5 PDE Models

Level set methods have been used for volume sculpting [Bærentzen and Christensen, 2002], CSG-based surface editing, automatic blending and curvature-based smoothing [Museth et al., 2002, 2005]. Mullen et al. [Mullen et al., 2007] propose a mass-preserving variational approach for geometry processing of volumetric implicit surfaces and foliations using an Eulerian formulation. Zhang and Lihua [2001] developed a geometric modeling framework based on partial differential equations (PDEs) that incorporates geometric constraints and functional requirements into PDEs. PDE-based volumetric sculpting [Du, 2003, Du and Qin, 2004, 2005, 2007] 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 [2004] 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, freeform modifications of level set models. They provide a significantly more expressive and flexible editing capability to level set modeling. We also favorably compare our results to previous PDE-based modeling work in terms of model resolution, processor speed and running times in Section 5.4.

3.2 Sketch-Based Techniques for 3D 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. Numerous sketch-based modeling techniques have been developed for meshes [Zeleznik et al., 1996, Igarashi et al., 1999, Igarashi and Hughes, 2003, Nealen et al., 2007, Mori and Igarashi, 2007, Zimmermann et al., 2007], parametric surfaces [Cherlin et al., 2005], procedural surfaces [Schmidt and Singh, 2008], volumetric models [Owada et al., 2003] and implicit surfaces [Schmidt et al., 2005b, Sugihara et al., 2008, Karpenko et al., 2002, Araujo and Jorge, 2003, Tai et al., 2004, Alexe et al., 2004]. Our sketch-based editing operators are inspired by these techniques and extend a number of them to level set models.

SKETCH [Zeleznik et al., 1996] introduced a gesture-based interface for the rapid modeling of CSG-like models consisting of simple primitives. The user sketches the salient features of a 3D primitive and the system instantiates the corresponding 3D model in the scene. An improved sketch-based modeling system, Teddy [Igarashi et al., 1999], uses 2D user strokes to construct 3D polygonal surfaces. This highly interactive system translates simple user strokes to actions such as paint, erase, extrude, cut and smooth to create 3D models. They later developed a framework to create visually smooth surfaces from their sketch-based modeling environment [Igarashi and Hughes, 2003]. This work was extended in FiberMesh [Nealen et al., 2007] to use a set of 3D curves to define the surfaces. For a given set of curves, the system automatically constructs a smooth surface by applying functional optimization. Another application created by the same approach is Plushie [Mori and Igarashi, 2007], an interactive design system for 3D plush toys. In all of these applications a relatively coarse mesh (1000-2000 vertices) is used to achieve interactive performance. Owada et al. [2003] use a binary volume dataset to overcome the topological restrictions of Teddy.

Wires [Singh and Fiume, 1998] is a deformation technique that uses curves (wires), placed in close proximity to a polygonal surface, as handles to deform the surface locally. This technique has been applied to animation of facial expressions, cloth animation and surface stitching. Lawrence and Funkhouser [2004] utilize a painting paradigm for local surface deformations, where user-applied "paint" defines instantaneous surface velocities. They initially implemented this technique using level set surfaces, but later switched to polygonal surfaces in order to achieve interactive rates and improve spatial resolution. Cherlin et al. [2005] use interpolating parametric surfaces in their sketch-based modeling framework. Layered procedural surfaces may be created and manipulated with Surface Trees [Schmidt and Singh, 2008], a hierarchical representation of surface patches and surface editing operations. This approach merges sketch-based interaction with a 3D analog of the intuitive layer-based metaphors found in 2D graphic design tools.

3.2.1 Curve Editing

Some of the editing capabilities we have developed require sketched curves to facilitate user interaction. The user should be able to draw and edit the curves interactively in order to create custom outlines for surface manipulations.

Catmull-Rom (C-R) splines [Catmull and Rom, 1974] offer many useful modeling properties, such as affine invariance, global smoothness, and local control. They are therefore of great interest to Computer Aided Design (CAD) users. C-R splines are easily evaluated and are a good choice for interactive applications because they interpolate their control points and therefore provide an intuitive way to represent and edit curves in these applications. In general it is more natural for points drawn by a user to end up on the curve, rather than defining control points that lie away from the actual curve.

Finkelstein and Salesin [1994] present the theory and methods for multiresolution curves, which are detail preserving, end-point-interpolating cubic B-splines that may be modified at different spatial resolutions. They describe a robust mathematical foundation based on wavelets that supports smoothing, editing and approximating these splines. Although this type of curve has desirable properties, e.g. multiresolution support, it is not easy to apply to an interactive application where a novice user would need to provide a set of parameters to create the complex framework. Elber and Gotsman [1995] extend this approach to non-uniform B-splines and provide the mechanism for local refinement and adaptive local curve manipulation. However, this method uses least-squares approximation for multiresolution decomposition of the freeform curve and is incapable of providing continuous resolution control. Later work [Elber, 2001] presents a scheme that combines multiresolution control with linear constraints into one framework, allowing one to perform multiresolution manipulation of non-uniform B-spline curves, while specifying and satisfying various linear constraints on the curves. The multiresolution control combined with linear constraints prescribes a precise freeform geometry, and creates a framework for interactive editing of non-uniform B-spline curves.

Compared to previous work, our approach provides a novel technique for direct manipulation of an interpolating spline that allows a user to easily modify a curve with an adjustable span of influence; thus overcoming the locality restriction of C-R splines. The relative simplicity of the method ensures that the modification will occur at interactive rates.

3.3 Representing and Rendering Large-Scale Volumetric Models

3.3.1 Advanced Level Set Data Structures

The original level set method [Osher and Sethian, 1988] has $O(n^3)$ time and space complexity, where n is the side length of the bounding volume in which the deforming isosurface is embedded. The time complexity can be reduced to $O(n^2)$ with a narrow-band scheme [Adalsteinsson and Sethian, 1995, Peng et al., 1999, Whitaker, 1998] that solves the PDE in a narrow-band only around the interface. In order to minimize the memory requirement of level sets while keeping the time complexity of the evaluation algorithms low, octree-based approaches have been employed [Bærentzen and Christensen, 2002, Losasso et al., 2004]. Octrees have also been utilized in other volumetric modeling systems [Frisken et al., 2000, Meagher, 1982, Perry and Frisken, 2001]. The space complexity of these methods is $O(n^2)$ and the random access time to the values is $O(\log n)$. The major drawback of these methods, as with all level set methods, is that they require a uniform refinement along the interface to use finite-difference based numerical methods; thus losing the adaptiveness benefit provided by the octrees.

Run-length encoding (RLE) is a simple form of lossless data compression in which runs of data (i.e. sequences in which the same data value occurs in consecutive data elements) are stored as a single data value and a count. The RLE sparse level set data structure [Houston et al., 2004] assigns either a very large positive or negative value to all voxels outside of the narrow-band, which are then compressed into runs on each side of the narrow band using RLE. It has $O(n^2 + R + D)$ space complexity, where R is the number of runs and D is the number of voxels in the narrow-band. The method uses a 2-D array to facilitate fast random access ($O(\log r)$, where r is the number of runs in a level set cross section. Nielsen and Museth [2006] created the DT-Grid (Dynamic Tubular Grid) data structure, which uses a hierarchical representation of the

data's dimensions to compress the volume and gain memory efficiency. DT-Grid provides constant access time to the grid's values and their immediate neighbors as long as all values are accessed sequentially. It also provides logarithmic ($O(\log n)$) random access to the sparse data by keeping the data lexicographically sorted. However, this data structure is unsatisfactory for interactive applications due to the additional steps required for keeping the data sorted and densely packed during insertion and deletion operations. The RLE sparse level set and DT-Grid were combined to create an improved data structure, Hierarchical RLE (H-RLE) level sets [Houston et al., 2006]. This method employs RLE in a dimensionally recursive fashion combined with a narrow-band scheme and provides $O(D)$ space complexity, i.e. $O(n^2)$. H-RLE also keeps the data sorted in linear arrays and suffers from the same drawbacks as DT-Grid when performing arbitrary modifications to the level set.

While narrow-band schemes effectively address the problem of time 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. For example, the cartoon bear model in Figure 4.3(a), which is represented with a $320 \times 320 \times 600$ dense volume, i.e. full Cartesian grid, requires 3 GB of memory during editing. The advanced data structures described in this section all reduce the space complexity from $O(n^3)$ to $\sim O(n^2)$. However, none of these data structures were designed for the rapid, random, local voxel accesses, updates and modifications that are essential for an interactive surface editing application. Data structures and algorithms we describe in Chapter 5 reduce the memory required for the bear model to 150MB while allowing the user to edit this model at 25 fps and higher.

3.3.2 Interactive Rendering of Large-Scale Dynamic Point Sets

The conventional method for viewing level set models is to extract a polygonal approximation of the isosurface from the volume using the Marching Cubes algorithm [Lorensen and Cline, 1987]. This approach has been shown to provide minimally acceptable display rates for low resolution models undergoing limited deformations [Museth et al., 2002, 2005], but it will not provide interactive rates for large-scale models.

Points on the surface can be used for interactive display [Stamminger and Drettakis, 2001] via rendering on a Graphics Processing Unit (GPU). Several recent advances have demonstrated that high quality interactive renderings can be generated from large-scale point-based models [Ren et al., 2002, Botsch et al., 2004, Chen et al., 2004, Botsch et al., 2005], using surface splatting [Zwicker et al., 2001]. Surface splatting is a point rendering and texture filtering technique that directly renders opaque and transparent surfaces from point clouds. Point rendering work on GPUs has produced interactive display rates for static models consisting of millions of points. The challenge here is to produce interactive rates for even larger dynamic point-based models and the main barrier to achieving this goal is the communications bottleneck between the CPU and GPU.

Recent research has demonstrated that level set models can be interactively evolved and displayed using GPUs [Cates et al., 2004, Lefohn et al., 2003, 2004, Rumpf and Strzodka, 2001]. These efforts, while impressive and ground-breaking, only implemented time-invariant, rudimentary speed functions capable of curvature-based flows that moved to match iso-values in volume data to perform segmentation. A major contribution of Lefohn et al. [2003, 2004] is the mapping and packing of a narrow-band embedded in a 3D grid into the limited 2D memory of the GPU. The data structures, speed functions and algorithms needed to implement freeform editing of large-scale,

multiresolution level set models are significantly more complex than those used in these segmentation examples, which makes them harder to implement for a GPU.

Hierarchical octree data structures are one of the most common choices to handle large point sets for interactive rendering [Coconu and Hege, 2002, Sainz and Pajarola, 2004]. While octrees provide a simple hierarchical organization of space, they can suffer from the fact that in general points on a 3D surface cannot be evenly partitioned into octants. This may lead to an unbalanced and suboptimal data structure. In contrast, k-d trees [Nievergelt and Widmayer, 1997, Samet, 1990] can guarantee a fully balanced hierarchical structure. Bounding volume hierarchies (BVHs) have also been used in rendering to efficiently support spatial queries such as visibility culling or ray-object intersections [Clark, 1976, Gross and Pfister, 2007, Rusinkiewicz and Levoy, 2000]. Unlike octrees and k-d trees, a BVH does not necessarily completely partition space; thus it allows for a more generic and efficient hierarchical organization of spatial data.

3.3.3 Optimized Spatial Hashing

Spatial hashing is a process by which a 3-D or 2-D domain space is projected into a 1-D hash table. The hash function takes a 2-D or 3-D data point and returns an index that corresponds to a 1-D entry in the hash table. Points on an object may be hashed and the locations can then be quickly queried. Spatial hashing has been utilized by several fast collision detection algorithms [Eitz and Lixu, 2007, Li et al., 2008] and has garnered much interest from the gaming industry [Hastings et al., 2005, Reynolds, 2006].

A "collision" occurs when two distinct elements are assigned into the same position in the hash table. A perfect hash function for a set S is a hash function that maps the elements of S into unique integers, with no collisions. Perfect hash functions are

rare in the space of all possible functions. Thus, one cannot expect to construct a perfect hash function for an arbitrary and dynamic set of points. Instead, a hash function must be developed that minimizes collisions. Lefebvre and Hoppe [2006] define a GPU-compatible minimal perfect hash function that is pre-computed on a static set of points. Using this hash function they can pack sparse data into a compact table while retaining efficient random access. While their GPU algorithm is efficient, it is not suitable for storing dynamic content. Teschner et al. [2003] proposed an algorithm for (self-)collision detection of dynamically deforming objects. Their algorithm employs a hash function for compressing a potentially infinite regular spatial grid. Although the hash function does not always provide a unique mapping of points to hash table positions, it can be generated very efficiently and does not require complex data structures, such as octrees or BSP trees.

3.4 Multiresolution Modeling

There exists a large body of multiresolution editing work based on mesh models [Zorin et al., 1997, Kobbelt et al., 1998, Guskov et al., 1999, 2002]. Additionally, multiresolution techniques have been used for mesh compression and simplifications [Khodakovsky et al., 2000, Laney et al., 2002]. Zorin et al. [1997] defines a set of editing operators on multiresolution triangular meshes produced via adaptive subdivisions. The editing is done locally on a subsection of the surface, which is then re-triangulated and rendered. The multiresolution adaptive representation provides an accurate yet efficient framework for editing triangulated surfaces. Schröder [2002] gives an overview of developments in subdivision surfaces and how these can help digital geometry processing. Another multiresolution mesh representation is described in Kobbelt et al. [1998]. A constrained mesh optimization is approximately solved in real time using multi-level techniques from numerical analysis. Mesh editing is done at a

coarse level and then mapped/translated to finer levels. Guskov et al. [1999] provide techniques for multiresolution mesh processing based on 2^{nd} order divided differences. They generate a mesh pyramid for irregular meshes through edge-collapse and vertex-split operations that guarantee minimal distortion. The algorithms are utilized in a number of applications including smoothing, enhancement, editing and texture mapping. The use of irregular meshes supports topology change throughout the hierarchy and approximates detailed features at multiple scales. However, regular refinements allow for more efficient data structures and processing algorithms.

A hybrid mesh is a multiresolution surface representation that combines advantages from regular and irregular meshes when processing topologically and geometrically complex surfaces [Guskov et al., 2002]. Hybrid meshes mostly use regular refinements when subdividing, but also allow occasional irregular operations to grow extra skin or change topology within the hierarchy. Guskov et al. [2000] introduce normal meshes, a representation inspired by differential geometry. A normal mesh is a multiresolution mesh where each level can be written as a normal offset from a coarser version. Normal meshes are very space- and bandwidth-efficient, describing a surface as a succinctly specified base shape plus a hierarchical normal map. Hence only a single float per vertex is needed to represent each level of the mesh. They can be useful in numerous applications such as compression, filtering, rendering, texturing, and modeling. Although mesh models provide well-designed surface editing frameworks, they suffer from slower re-meshing and self-intersection repair operations.

The previous work on multiresolution implicit modeling can be organized in two categories, the systems that utilize semi-Lagrangian methods to solve the hyperbolic level set equation on an adaptive grid, i.e. an octree [Strain, 1999, Min, 2004, Enright et al., 2005], and the systems that take a multiresolution approach to solve a specific problem like image segmentation [Law et al., 2008]. The methods developed in the

first category have been used in fluid simulations [Enright et al., 2002b] and medical image segmentation [Droske et al., 2001]. The second category consists of techniques that evolve an implicit surface in a multi-grid fashion. These techniques start with a low-resolution initial approximation to the implicit surface and search for a solution at the lowest resolution. They then subdivide the pixels around the propagating front, i.e. the implicit surface, and solve again at this increased resolution. This process continues until the final resolution, i.e. the resolution of the original image being segmented, is reached. A similar approach has also been used for producing implicit surface reconstructions from multiple depth maps and multiple views of objects [Sarti and Tubaro, 2001, Slabaugh and Schafer, 2002]. However, to the best of our knowledge, there exists no work on extending these multiresolution/multi-grid techniques to editing volumetric and level set models.

3.5 Detail Preserving Level Set Method

Level set methods are used to smoothly capture an evolving interface, but suffer an excessive amount of mass loss in under-resolved regions of the flow. This hinders the representation of thin interfacial filaments and regions of high curvature. Enright et al. [2002a] proposed the particle level set method (PLSM) for improving the mass conservation properties of the level set method when the interface is passively advected in a flow field. They later presented the semi-Lagrangian-based particle level set method for fast and accurate capturing of interfaces [Enright et al., 2005]. The particle level set method has been used during the animation and simulation of smoke and complex water surfaces [Enright et al., 2002b, Losasso et al., 2004]. One drawback of this method is that it requires advecting the particles along with the surface at every step of the simulation. This process is slow at higher resolutions, especially during substantial modifications to the surface, because large numbers of particles need to

be moved several times to reach their destination on the deforming surface. We believe that a method that can project these particles onto the final surface once the interactive editing is completed is more beneficial when dealing with larger models. Since the details are encoded locally, the challenge is to keep the details, i.e. the local neighborhood of particles, intact after the projection.

3.6 Geometric Texture Transfer

Geometric texture mapping is the 3-dimensional extension of traditional texture mapping using images. Here, surface characteristics of a 3D model are skinned and applied onto another model to create a variety of geometric details without the effort required to manually specify them. Elber [2005] and Zhou et al. [2006] use a stitching technique to create more geometrically complex surfaces by tiling patterns over thin shell triangle meshes. Bhat et al. [2004] present a volumetric approach to tiling patterns in order to create more complex textures. Lai et al. [2005] present an explicit texture transfer method based on geometry images [Gu et al., 2002]. Andersen et al. [2009] extend the height field texture representation by incorporating displacements in the tangential plane in the form of a normal tilt. Shell maps [Porumbescu et al., 2005] provide a mapping between shell space and texture space that can be used to generate small-scale features on surfaces using a variety of modeling techniques. The method is based upon the generation of an offset surface and the construction of a tetrahedral mesh that fills the space between the base surface and its offset. Schroeder et al. [2005] present a method capable of producing complex surface features based on displacement mapping and stochastic geometry. Their method generates a diverse set of surface models by stochastically defining offset values on triangular meshes in statistically-consistent patterns. Brodersen et al. [2008] extend these geometry mapping techniques to level set models. They can warp and blend geometric details

using level set surfaces. These details are represented either as a mesh or a level set surface themselves. The former representation facilitates a fast mapping, while the latter produces a higher quality surface. Both implicit and explicit techniques use a particle-based parametrization in 3-dimensions around the base surface, which adds to the time complexity. The implicit method is significantly slower due to the extensive number of particles and the use of compute-intensive global radial basis functions.

4. Interactive Level Set Surface Editing

Level set models are used in special effects to perform physically-based simulations such as fluid flow, as well as to create animations of amorphous characters using morphing. The simulations often do not generate the desired results and may require post-processing and clean-ups. Furthermore, the intermediate steps of the simulation or morphing processes, i.e. keyframes, may also require control and redirection from the user in order to create specific outcomes. In medicine and science, level set models are used to perform volume segmentation. The process is dynamic, but not fully automated, and requires user input to create correct segmentations. The medical field would greatly benefit from an interactive tool that can provide the user with the ability to direct the segmentation process while it is underway. Such a capability would allow the user to rapidly create the correct outcome for the volume segmentation.

Figure 4.1 shows how an interactive editing framework can be used to provide user control and guidance during the morphing process. This process have been used to create the Tar Monster shown in Figure 1.3. The initial model is created using an explicit surface representation and one of the many commercial tools for geometric

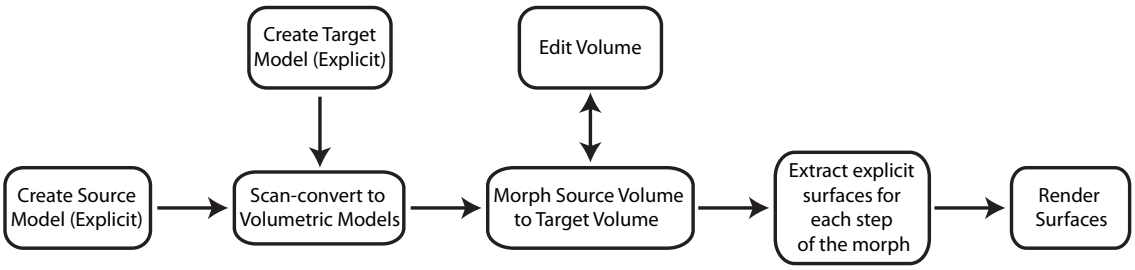


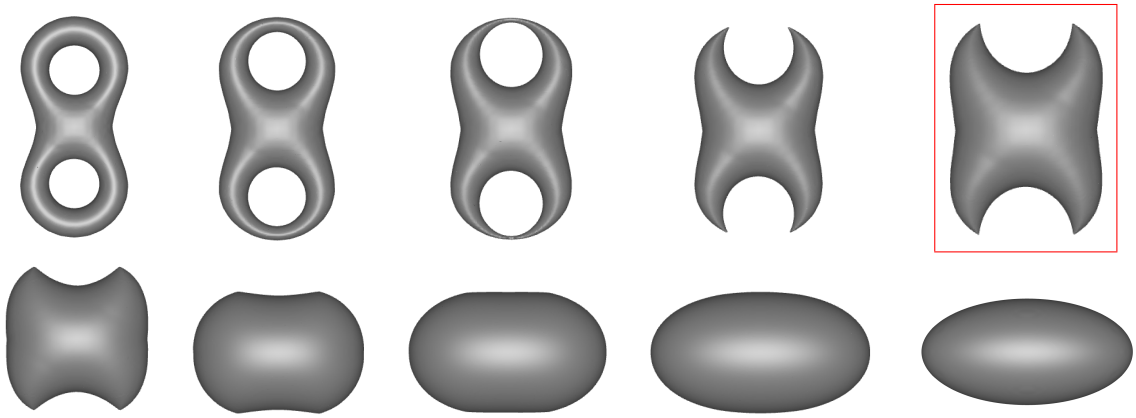
Figure 4.1: Flowchart of the animation pipeline for doing level set morphing

modeling. It is then scan-converted into a volumetric representation, in this case a level set model. Level set morphing [Breen and Whitaker, 2001, Breen et al., 2001] is used to create an animation sequence. Each frame in the animation sequence is processed to extract explicit surfaces that are then rendered to complete the pipeline.

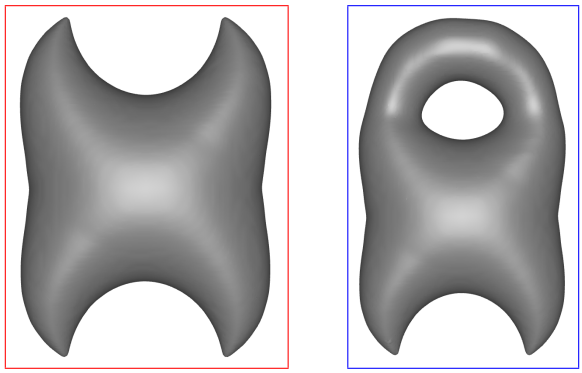
Figure 4.2 shows how an interactive editing framework can be used to provide user control and guidance in the morphing process. An initial morph creates an animation that morphs a double torus into an ellipsoid. The intermediate level set result shown in Figure 4.2(b) (within the red box) is then edited by the user to create an intermediate target shape (drawn within a blue box) that changes the shape of the object during the transition. Having level set editing capabilities allows the user to directly modify the intermediate level set model and removes the need for extracting a surface, importing it into a commercial surface modeling system, editing it, and re-scan-converting it back into a level set volume. Surface extraction and scan-conversion are slow and tedious processes that can introduce changes and errors into the model.

This chapter describes the techniques and algorithms we have developed for editing level set surfaces. The mathematical equations that map user interaction into level set deformations in order to implement numerous interactive, freeform and sketch-based level set modeling capabilities have been derived. These capabilities have been implemented utilizing a pre-existing level set library, and incorporated into an interactive modeling system. 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.

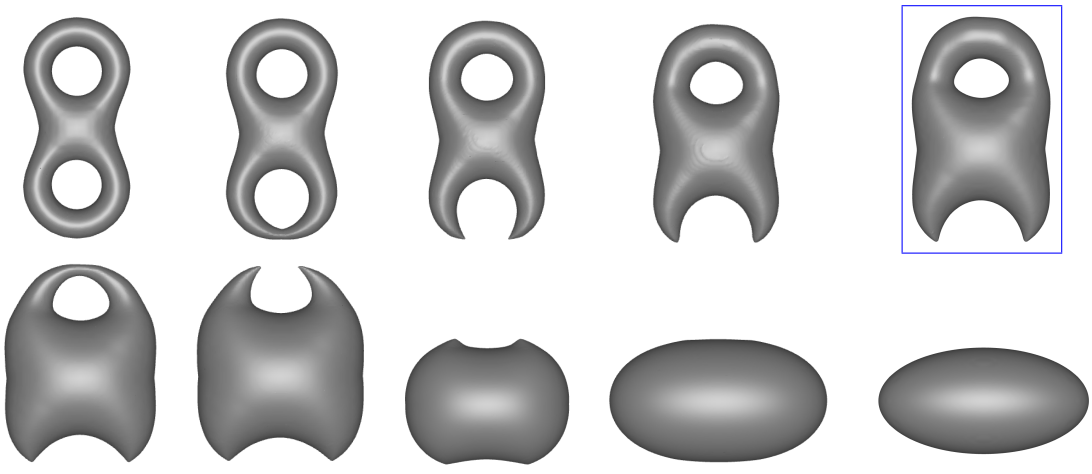
A level set surface can be edited through a click-sketch-and-pull interface that allows a user to identify a point or Region-Of-Influence (ROI) to be modified on the surface. The concept of an ROI is analogous to $3D$ brushes used in commercial



(a) The initial morph sequence

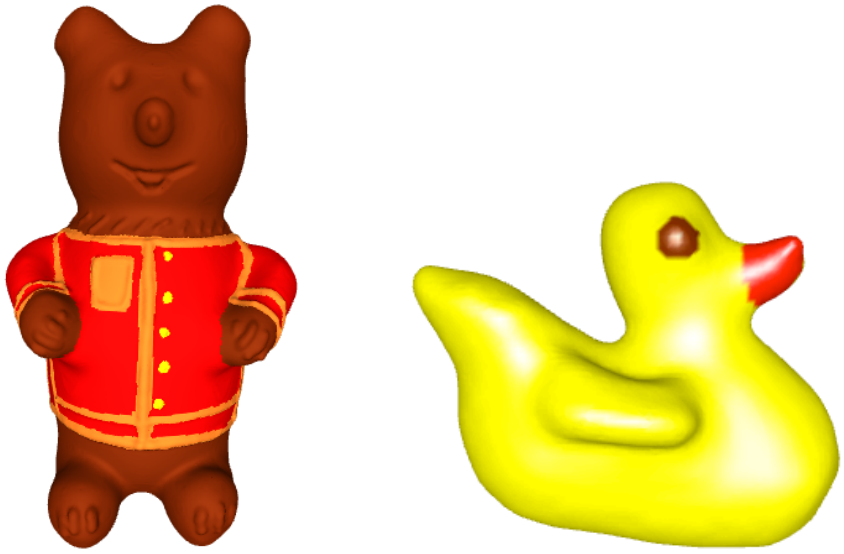


(b) The middle frame is modified



(c) The final morph sequence

Figure 4.2: Using interactive surface editing to control and direct level set morphing.



(a) (b)

Figure 4.3: (a) A cartoon bear created with freeform level set surface editing operators. (b) A rubber duck is created from a level set sphere and a set of sketched curves.

packages such as ZBrush. 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 freeform surface manipulation. Other operators include surface detailing, carving and smoothing. 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 4.3(a) is created using the surface editing capabilities discussed in Section 4.3.

Additionally, a variety of sketch-based level set modeling capabilities have been developed. These capabilities include local and global surface editing using single or multiple curves that are specified on or above a level set surface. The sketch-based approach allows a user to sketch a curve on, above or near a level set model. The

model then evolves in response to the user's curve-based input to create a surface that locally matches the shape of the curves. The curves may then be modified, with the level set surface adjusting to the curve changes. Section 4.2 explains the interactive techniques used to draw and edit sketch curves. Section 4.4 describes the sketch-based editing operators. Some of these operators are used to create a rubber duck in Figure 4.3(b) from a level set sphere and a set of sketch curves.

Our work has developed novel level set modeling functionality and technology. It provides a general, expressive and interactive set of editing operators for PDE-based implicit models. Previous work in level set and PDE-based modeling has primarily focused on volume sculpting and CSG operations. Ours are the first freeform and sketch-based editing operators developed for level set models. These operators allow a model to be stretched and shaped, split into pieces and merged smoothly. Topology changes occur naturally and automatically because of the properties of level set models.

4.1 3D User Interaction

Our 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 "unprojected" 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 4.13, 4.17, 4.18 and 4.24. 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 for the editing operators. These splines provide localized and multiresolution editing capabilities for 3D curves.

4.2 Localized Editing of Catmull-Rom Splines

There is a large body of research on curves from the 1960s to the present time [Cohen et al., 2001, Farin, 2002]. Curves are widely used in every aspect of computer graphics, especially splines, which are piecewise polynomial parametric curves. Splines are popular in CAD because of the simplicity of their construction, their ease and accuracy of evaluation, and their capacity to define complex shapes during interactive design. Catmull-Rom (C-R) splines [Catmull and Rom, 1974] are a family of cubic interpolating curves formulated such that the tangent at each control point is calculated using the previous and next control points on the spline. C-R splines have C1 continuity, local control, and interpolate their control points, but do not lie within the convex hull of their control points. We utilize this type of spline in an interactive surface modeler because of its ability to interpolate every control point. Direct control point manipulation has been recognized as a powerful computer graphics tool, and yet the strictly defined local influence of C-R control points, while useful from many perspectives, is limiting and may be a drawback for many freeform editing applications. In a single control point manipulation operation, one is unable to apply a modification to the spline shape that affects more than a small neighborhood on

the spline, the span between neighboring control points.

Our interactive surface-editing framework uses C-R splines for sketch-based editing. Mouse strokes are translated into splines that are used to define surface deformations. We employ C-R splines in this setting in order to provide an interactive and easy-to-use method for curve editing. C-R spline's ability to interpolate control points is an important feature, one that allows us to accurately translate user input into a mathematical representation. In order to overcome the immediate neighborhood limitation when modifying C-R splines, we describe an approach for the localized, interactive editing of C-R splines. To provide greater flexibility, control and expressiveness, we have developed techniques that expand and generalize the result of modifying one C-R control point. The techniques allow the user to define the range and type of influence that manipulation of a single control point may produce on a C-R curve, thus creating a versatile and powerful localized curve editing capability.

Localized editing gives the user more control over the shape of the spline when moving a single control point. An active window is defined around the control point selected by the user, and it limits the resulting modifications to a user-defined segment of the curve. The extent of the window can be changed by the user any time during editing. We provide a variable-resolution editing framework that lets the user specify the control point resolution within the active window. This enables the user to increase the density of the control points where more detail is needed, and makes that part of the curve more controllable. However, due to the locality of C-R splines, any change to a single control point only changes the portion of the curve between the control point and its neighboring control points. The dense sampling of the control points provides the means for creating fine details, but it also makes it difficult to specify changes at varying spatial scales. The active window therefore gives the user

control over the range of influence associated with a single control point editing stroke. The movement of this single control point is distributed to every control point within the window. The main issue to be addressed is how to transfer the modifications made by the user when dragging a single control point to the other control points within the active window. Our work proposes techniques for distributing the motion of a single C-R control point within an active window in order to produce an intuitive and expressive localized spline editing functionality. Several techniques are described and their results are compared in this section.

The curves in our editing system are drawn by the user either on a reference plane or directly on the surface. As the user clicks and drags the cursor, several control points are captured that follow the cursor's movements. The sampling of the control points is directly related to the speed of the user's strokes. In other words, slower, more detailed input creates a denser sampling and a more accurate representation of the curve, while a faster and free-handed drawing leads to a less detailed curve. A C-R spline is fit to these control points once the mouse button is released. The first and last control points are inserted multiple times to force the curve to interpolate both end points. A discrete sampling of the points on the curve is displayed to the user as a polyline. The sampling rate used to display the curve is user-controlled and can be changed during the editing session. The control points and the boundary of the active window are highlighted while drawing the curve to aid interactive editing. The curve can be further modified by the user through our localized editing interface. The editing interface provides the user with several options for modifying C-R splines. The user can set an active window size to ensure only a certain part of the curve is modified with each stroke. This window can also be interactively changed to fit the user's needs. Once the curve is drawn and a window size is set, editing is achieved by clicking on a control point and dragging it to a new position. The active window

need not be symmetric. The user can shift either end of the window to change its extent. All the control points within the active window around the dragged control point are moved following one of the user-defined schemes described in Sections 4.2.3 and 4.2.4. The curve is fit to the new set of control points after every editing step, and the user is provided with immediate feedback of the overall shape of the curve while editing. The number of control points within the active window can also be changed by the user in order to provide more detailed, higher resolution editing. The number of control points may also be reduced within the window as well.

The movement of the control points within the active window can be described through a set of schemes that modify all the other control points within the window as a function of the displacement of the selected control point. Two alternate ideas, interpolating the displacement of the selected control point within the active window, and interpolating the displacement vector with a vector orthogonal to the curve, are described in the following sections.

4.2.1 Active Window

Defining an active window gives the user control over the range of control points that are to be affected by a single editing stroke. The movement of a single control point is distributed to every control point within the window. The window size cannot be larger than the curve itself. The active window is set to be the start and end points of the curve in case the window size is greater than the curve. The window is centered at the control point being dragged, but the extent need not be symmetric.

The window size is defined as a number of control points n_R to the right and m_L to the left of the selected/modified control point. For example, the window can span 3 control points on the left and 4 control points on the right. This can either occur if the window size is set to four, but a control point that is 3 control points

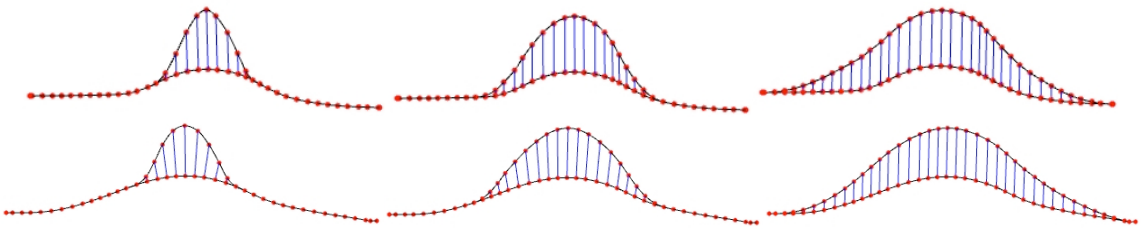


Figure 4.4: Changing the active window size. Top: Only the displacement is interpolated. Bottom: Both the displacement and the normals are interpolated. Left-to-right: Editing with window size = 5 control points. Middle: Editing with window size = 10. Right: Active window spans the whole curve. The blue lines show the movement of the control points within the active window. The red dots are the control points.

from a curve boundary is selected, or if the user adjusts the left or right end of the window interactively. At any time the boundary of the active window is highlighted and the user can adjust the window size on either side by moving these highlighted points. Figure 4.4 demonstrates editing with a symmetric active window, as well as the results from changing the window size.

4.2.2 Multiresolution Control

The resolution of the control points within the active window is user-defined. The control points derived from user input are non-uniform and their sampling resolution is directly proportional to the speed of the input strokes. We found it useful to work with a uniform sampling of control points in most cases, so we have incorporated an option to redefine the curve with a new set of control points that are evenly distributed over the curve. Once the initial control points are inputted, a C-R spline is fit to these points. If the option to uniformly sample the control points is chosen, a new set of control points is created by evenly sampling the curve, and a new curve is fit to these control points. The number of control points is kept the same during

this operation. However, the user may also increase/decrease the number of control points immediately after the curve is resampled.

In cases where a part of the curve is stretched or condensed, the resolution on the modified portion of the curve may need to be adjusted. The user can either choose to resample the curve within the active window with the same resolution as the rest of the curve or increase/decrease the resolution of the control points within the active window. This resampling is also uniform, but the resolution can be higher or lower than the rest of the curve. Once the active window is placed over another portion of the curve, the resolution in the previous active window stays fixed until it is once again edited by the user. This kind of variable-resolution control combined with a flexible, i.e. asymmetric, active window enables us to define very specific parts of the curve to be edited. Rough sketches can be created by lowering the resolution and working with a larger active window. Then more control points can be added where needed to provide additional detail and control.

4.2.3 Interpolating the Control Point Displacement

The first approach taken to solving the problem of distributing control point displacement within the active window sets the direction of the displacement to be the same for all control points within the window. The magnitude of the displacement monotonically decreases from the control point being modified to the boundaries of the window. There are two properties needed for the function that implements this kind of drop off in order to create an acceptable result. The function should be smooth and it should go to zero at the boundaries to avoid discontinuities. We have tested several such functions and found three that provide the desired smooth transitions (Equations 4.1, 4.2 & 4.3). These equations include a linear, Gaussian and a cosine function that all decrease from 1 to 0 within the window range. Equation 4.2 uses a

second function $f(x)$ to ensure proper boundary conditions, since the Gaussian does not go to zero in a finite range.

$$1 - \frac{d}{W} \tag{4.1}$$

$$f(W-d)e^{\frac{-d^2}{2\sigma^2}} \tag{4.2}$$

$$\sigma = W/5$$

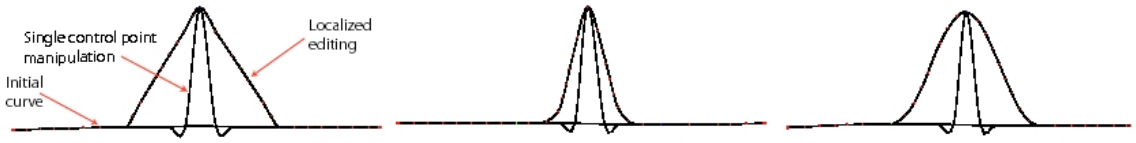
$$f(x) = \begin{cases} 1.0 & x > \epsilon \\ (x/\epsilon)^2 & x \leq \epsilon, \end{cases}$$

$$\frac{1}{2} + \frac{1}{2}\cos^{\alpha}\left(\frac{d}{W}\pi\right) \tag{4.3}$$

d is the distance (over the curve) to the center of the window, i.e. the control point being modified by the user, and W is a window size defined for each side of the active window, either W_R or W_L. f is a function that ensures the whole equation goes to zero smoothly. It is equal to 1 up to a small distance from the boundary of the window ($\epsilon = \sigma$ in our examples) and goes to zero smoothly at the boundary.

Figures 4.5(a) and 4.5(b) present curves produced from interpolating the control point displacement and using the three drop-off functions. Figure 4.5(a) applies the change to the entire curve and Figure 4.5(b) uses a symmetric active window of size 5. Equation 4.3 created the best results in terms of a smooth and intuitive interaction, and we use this function in the final examples presented in this section (Figures 4.10-4.12). Figure 4.6 shows a case where the curve is edited twice, once pulled towards the left and once towards the right. Although the results are good, moving all control points in the same direction is not always satisfactory for our purposes. Section 4.2.5

(a) Distributing the displacement of the control point on the entire curve.



(b) Distributing the displacement of the control point within the active window. Window size=5.

Figure 4.5: Left to right: Linearly decreasing function in Equation 4.1, exponentially decreasing function in Equation 4.2, sinusoidal function in Equation 4.3 ($\alpha = 1.0$). Three curves are drawn in each case: Initial curve, local effect of moving one control point, and the curve after distributing the movement to all control points.

describes another scheme for calculating the offset direction for all control points in the active window.

The sinusoidal interpolant may be further adjusted to produce a variety of results. The parameter α is the exponent of the cosine in Equation 4.3 . Setting the parameter to something other than 1 changes the shape of the curve in the active window. As seen in Figure 4.7 , a value of α less than 1 begins to square off the modified section of the curve. Increasing α's value above 1 creates a faster drop-off in the displacement and a sharper bump in the curve.

4.2.4 Interpolating the Curve Normal

We further investigated distributing the direction of the control point movement within the active window, and tested two other solutions that led to another approach to localized editing of Catmull-Rom splines. One way to move the control points is to let the points move in the direction normal to the curve. Although this appears to

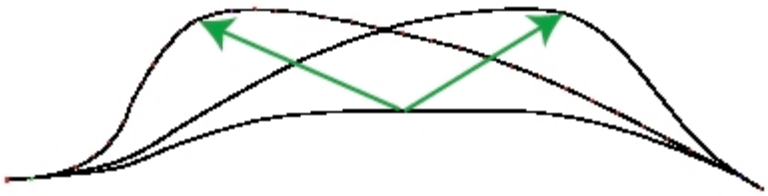


Figure 4.6: The result of distributing the displacement of the control point. The same curve is modified twice by pulling the same control point in two different directions. Green arrows show displacement of one control point during editing.

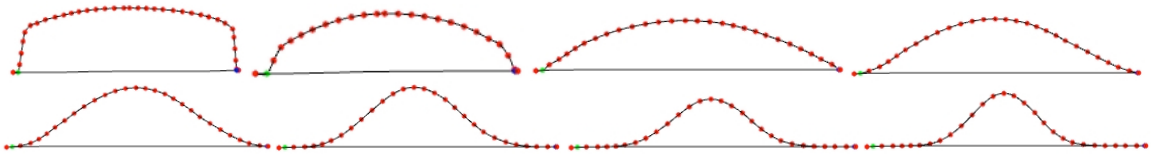
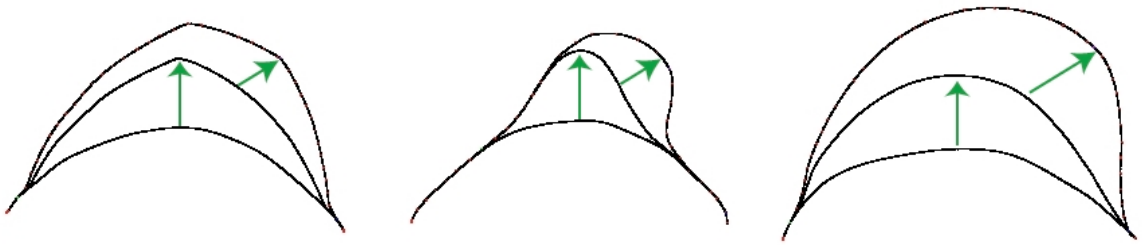
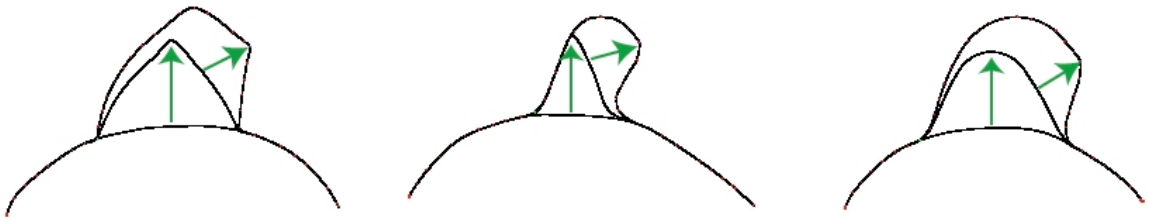


Figure 4.7: The result of changing α in Equation 4.3. Left to right: $\alpha = 0.1$, 0.25, 0.5, 0.75, 1.0, 2.0, 3.0, 4.0.

be a natural way to move the control points, it did not produce acceptable results. This approach leads to unwanted deformations because it gradually expands/shrinks the portion of the curve within the window. The second approach interpolates the selected control point displacement with the curve's normal at the endpoints of the active window, while decreasing the magnitude of the offset using either Equation 4.1, 4.2 or 4.3. We use spherical linear interpolation (slerp) [Shoemake, 1985] to calculate intermediate vectors between the displacement of the selected control point and the curve normals at the boundaries of the window. All vectors are initially unit vectors that are then scaled according to functions described in Section 4.2.3 to create a set of displacement vectors ($\vec{D}_i$) for control points in the active window:



(a) Interpolating control point displacement with the curve normal at curve end points using spherical linear interpolation. The active window spans the whole curve.



(b) Interpolating control point displacement with the curve normal at the window boundary using spherical linear interpolation. Window size=5.

Figure 4.8: The green arrow represents the editing stroke. Left to right: Linearly decreasing function in Equation 4.1, exponentially decreasing function in Equation 4.2, sinusoidal function in Equation 4.3.

$$\vec{D}_i = \left|\vec{D}_0\right| \times S(d_i) \times \text{slerp}\left(\vec{D}_0, \vec{N}_n, d_i\right). \tag{4.4}$$

$\vec{D}_0$ is the displacement of the selected control point. $\vec{N}_n$ is the curve normal at one of the boundaries of the active window, where n is the window size. The normals at n_R or n_L are used depending on which side of the window is being processed. Both of these vectors are normalized before being interpolated. d_i is the distance along the curve from control point i to the center of the window. $S(d_i)$ is one of the scaling functions in Equations 4.1 to 4.3. This approach created the best results in terms of smoothness of the resulting curve as well as the most intuitive response to moving a single control point. Figures 4.8(a) and 4.8(b) show results from this method using the three drop-off functions and two active window sizes.

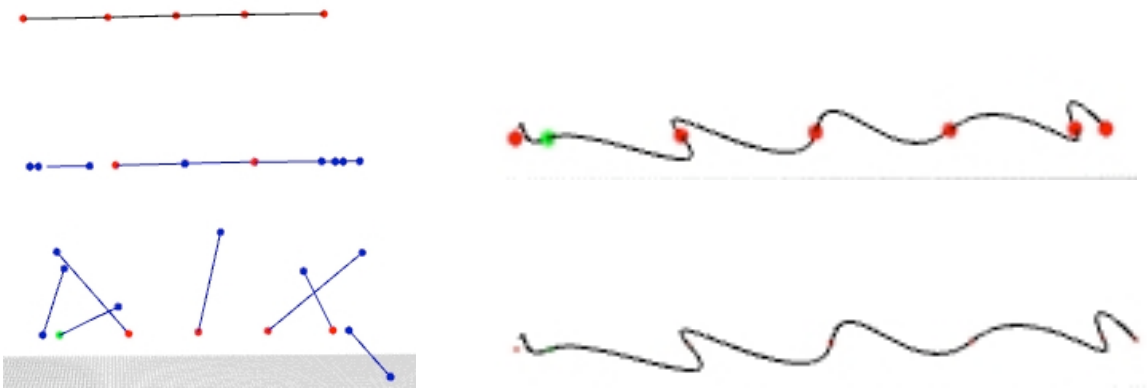


Figure 4.9: Editing tangent vectors at control points. The control points are highlighted in red and the tangent vectors are drawn as blue lines. The vectors are drawn at each control point, and the blue points at the end of each vector can be picked and modified, resulting in new tangent vectors. Left: Top to bottom: The original C-R curve, original tangents, modified tangents. Right: The final curve after modifying the tangents, drawn with and without the control points highlighted.

4.2.5 Editing Tangent Vectors At Control Points

We also explored the effects of editing tangent vectors at control points as an additional means for manipulating the curves. The tangents are initially calculated by averaging control points as explained in Catmull and Rom [1974] for C-R curves. The user then can put the system in tangent editing mode, and only the control points with their tangent vectors are displayed instead of the curve itself. Clicking and pulling on the displayed lines representing the tangents can manipulate these vectors. Both the direction and size of the tangents can be changed. Figure 4.9 shows an example of changing a C-R curve's tangents to edit its shape.

4.2.6 Results

Figures 4.10 through 4.12 demonstrate the variable-resolution, localized editing capabilities of the curve-editing approach. They show intermediate steps from an

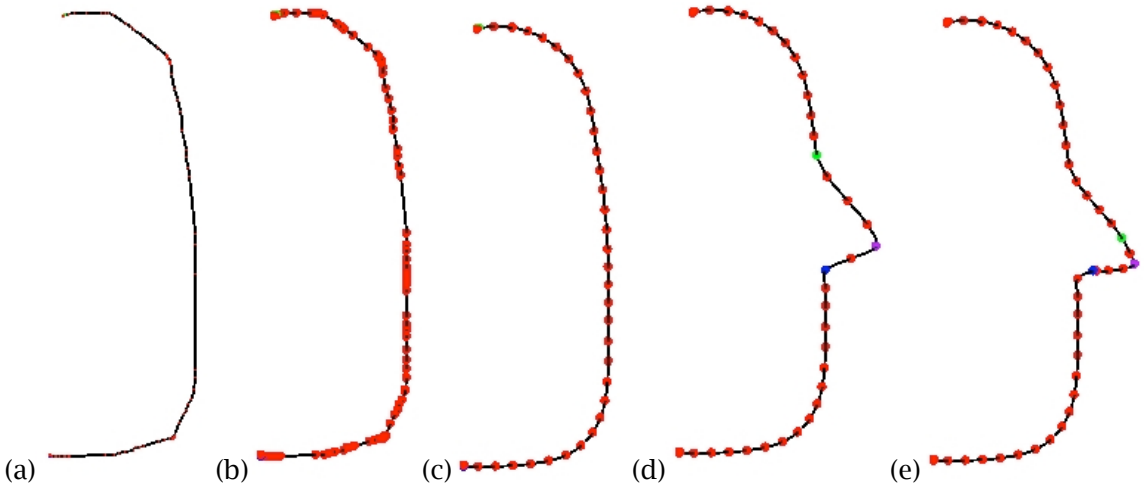


Figure 4.10: The rough, non-uniform user input (a-b) is sampled uniformly and smoothed (c). Pulling the point highlighted in purple outwards creates the nose (d). The resolution in the active window is increased for further editing. The tip of the nose is pulled down to create the nose in (e). The active window boundaries are highlighted in green and blue. The editing is applied to the point highlighted in purple

editing session that defines a facial profile. Figure 4.11(f) presents the final result of the session. The initial user input (Figures 4.10(a-b)) is neither smooth nor uniform. First a uniform sampling of the control points at a user-defined resolution is produced (Figure 4.10(c)). Pulling the point highlighted in purple in Figure 4.10(d) outwards creates the nose. This operation stretches the area around the tip of the nose. The user chooses to resample this part of the curve to increase the resolution for further editing. Figures 4.11(a-e) show how variable-resolution control and localized editing is utilized to create the mouth and chin. In order to create a rounder nose tip the user limits the active window to a small part of the curve around the nose and increases the resolution (Figure 4.12(a)). Figure 4.12(b) and 4.12(c) show the editing of the nose. We added 3 more curves to roughly define the eye. Similar techniques are used to create and modify these additional curves.

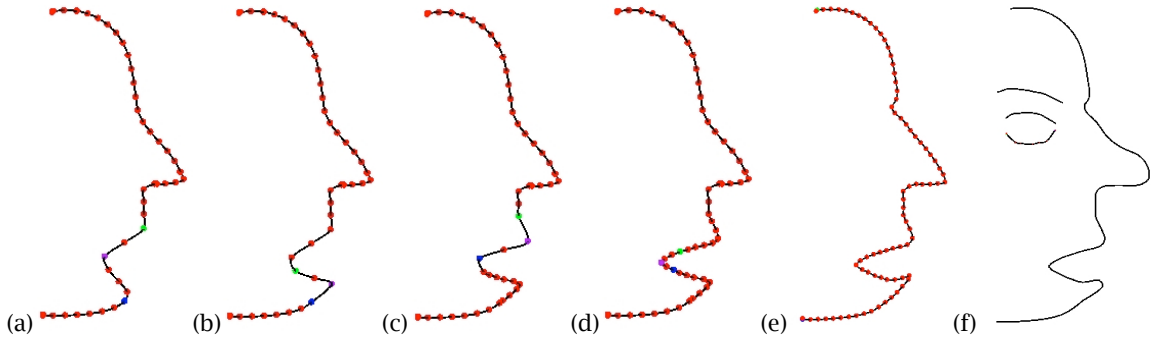


Figure 4.11: The user works at different resolutions to create the mouth. The active window boundaries are highlighted in green and blue. The editing is applied to the point highlighted in purple. The entire curve is resampled in (e). The final result from sketching a profile is shown in (f).

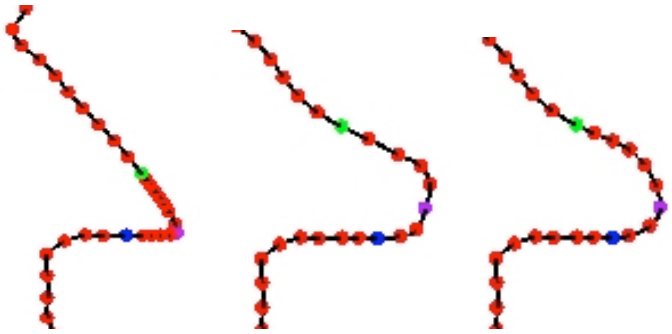


Figure 4.12: The user works at different resolutions to edit the nose. The active window boundaries are highlighted in green and blue. The editing is applied to the point highlighted in purple.

4.3 Freeform Editing Operators

Our freeform editing operators draw inspiration from many previous modeling techniques, and bring their editing capabilities to level set models. For example, our point-based operators are similar to those in Ferley et al. [2001], McDonnell et al. [2001], Angelidis et al. [2006], Owada et al. [2003], Bærentzen and Christensen [2002]. Our operator that allows a user to pull a curve attached to a surface is comparable to an editing capability found in the Wires system [Singh and Fiume, 1998]. Carving and detailing tools can also be found in Perng et al. [2001], Wang and Kaufman [1995], Ferley et al. [2000, 2001], Angelidis et al. [2006]. This section describes how to design custom speed functions based on user interaction in order to create freeform deformations for level set surfaces.

4.3.1 Pulling a point, symmetric ROI

This operator allows a user to click on a point on the surface ($\mathbf{x_s}$) and drag it. Clicking creates a tracker particle on the surface at $\mathbf{x_s}$. A part of the surface, defined by a radius of influence around $\mathbf{x_s}$, then moves outward. As the 3D cursor location ($\mathbf{x_c}$) changes the tracker particle moves toward $\mathbf{x_c}$, but is constrained to stay on the deforming surface; thus defining an updated $\mathbf{x_s}$ value for the surface evolution. The new position of $\mathbf{x_s}$ lies on the line connecting the old position of $\mathbf{x_s}$ to $\mathbf{x_c}$, where this line intersects the surface. The evolution stops once the tracker particle reaches the 3D cursor 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 $\mathbf{x_s}$. While this is the simplest of editing operators, we did find it useful for designing and creating handle-like structures, e.g. in Figure 4.13.

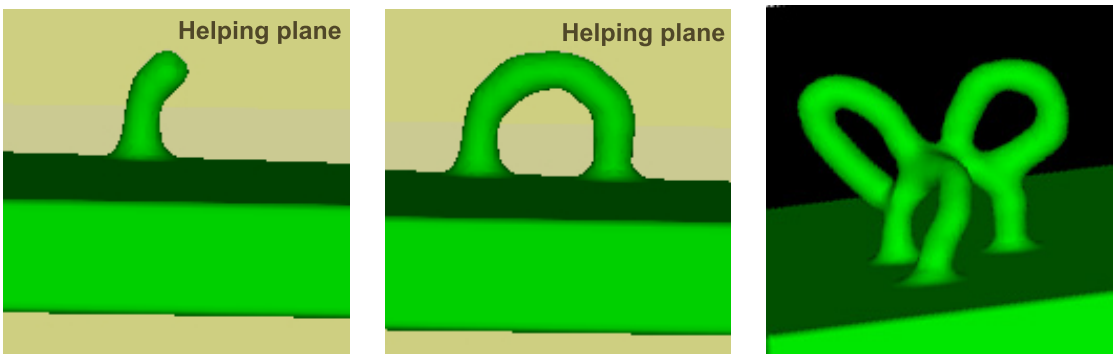


Figure 4.13: A loop is created by clicking and dragging a point on the surface ($\alpha = 2.0$). The first two frames demonstrate the use of the helping plane (the yellow background). The last frame shows several loops smoothly merged with each other.

The speed function that implements this operator is

$$F(\mathbf{x}) = \begin{cases} \cos^{\alpha}(\frac{\pi}{2} * \frac{d_s(\mathbf{x})}{r}) & d_s(\mathbf{x}) \leq r \\ 0 & d_s(\mathbf{x}) > r, \end{cases} \tag{4.5}$$

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. $\mathbf{x_s}$. The geodesic distance between $\mathbf{x_s}$ and all the voxels within the ROI can be calculated using the sweeping algorithm explained in Section 4.5 and Algorithm 1. Equation 4.5 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 4.13 shows a surface being edited using this operator ($\alpha = 2.0$). Figure 4.14 demonstrates the different bump shapes created by varying α.

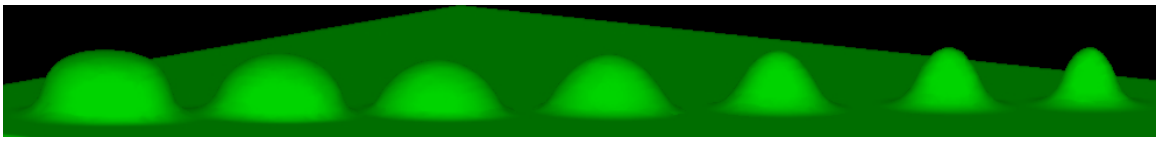


Figure 4.14: Effect of changing the α parameter in Equation 4.5. $\alpha = 0.25$, 0.5, 0.75, 1.0, 2.0, 3.0, 4.0.

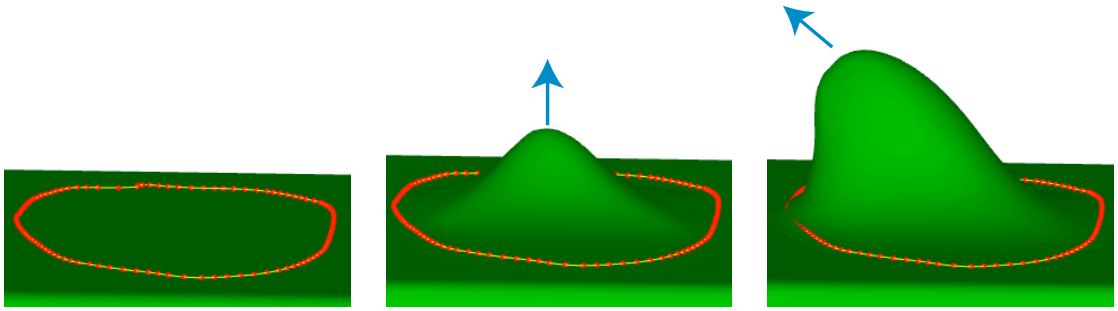


Figure 4.15: Deforming a patch on the surface by defining an ROI with a boundary curve and pulling a point. $\alpha = 4$. $\epsilon = 5$ voxels.

4.3.2 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 ($\mathbf{x_s}$) within the ROI. All points in the ROI move outward, with the points closest to $\mathbf{x_s}$ moving the fastest. The ROI's surface speed gradually decreases to zero at the boundary curve. As with the previous operator the tracker particle $\mathbf{x_s}$ moves towards the cursor's 3D location ($\mathbf{x_c}$), but remains on the surface. The surface movement stops either when $\mathbf{x_s}$ reaches $\mathbf{x_c}$ or when the user releases the mouse button. In Figure 4.15, 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

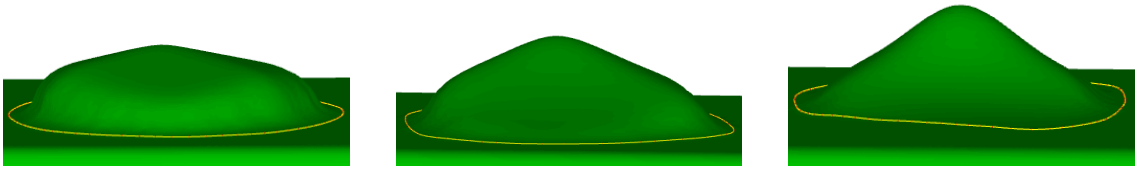
$$F(\mathbf{x}) = f(d_{out}(\mathbf{x})) * \left(\frac{\max(d_{\text{in}}^{xs}(\mathbf{x})) - d_{\text{in}}^{xs}(\mathbf{x})}{\max(d_{\text{in}}^{xs}(\mathbf{x}))} \right)^{\alpha} \tag{4.6}$$

$$f(d) = \begin{cases} 1/2 - 1/2 * \cos(\pi * d(\mathbf{x})/\epsilon) & d \leq \epsilon \\ 1.0 & d > \epsilon, \end{cases} \tag{4.7}$$

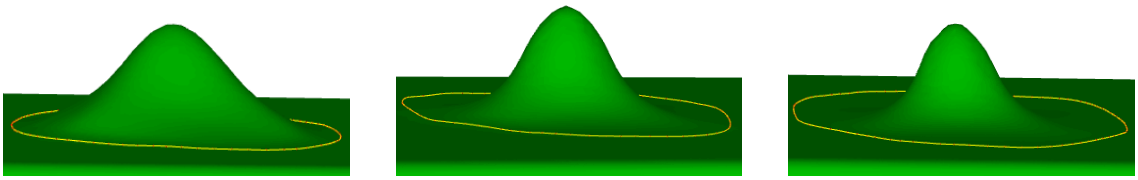
where $d_{\text{in}}^{xs}(\mathbf{x})$ is the geodesic distance to $\mathbf{x_s}$ from $\mathbf{x}$, $\max(d_{\text{in}}^{xs}(\mathbf{x}))$ is the maximum of these values over all points in the ROI, and $d_{out}(\mathbf{x})$ is the geodesic distance to the boundary curve from $\mathbf{x}$. The first 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 4.16. The geodesic distance $d_{\text{in}}^{xs}(\mathbf{x})$ is calculated by sweeping out distance information from $\mathbf{x_s}$ to voxels that lie on the surface using Algorithm 1 (see Section 4.5).

4.3.3 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 4.17, 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.



(a) $\alpha = 1$ (b) $\alpha = 2$ (c) $\alpha = 3$



(d) $\alpha = 4$ (e) $\alpha = 6$ (f) $\alpha = 9$

Figure 4.16: The α parameter in Equation 4.6 changes the shape of the modification.

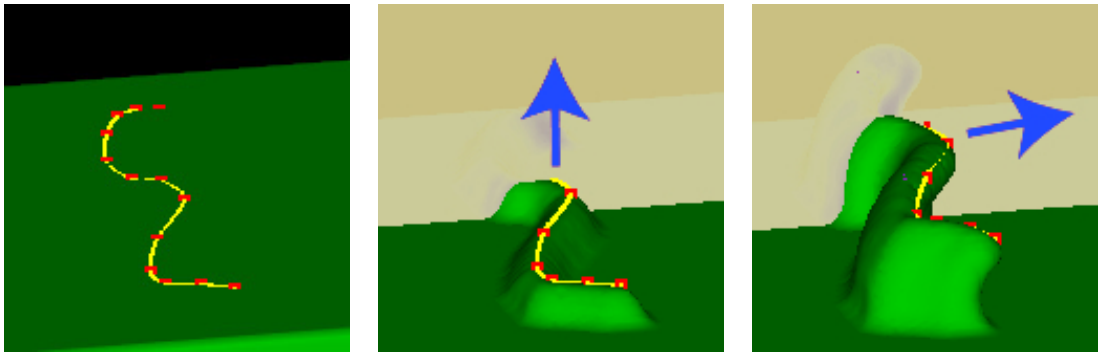


Figure 4.17: A curve with a symmetric ROI is placed on the surface and pulled first upwards then towards the right.

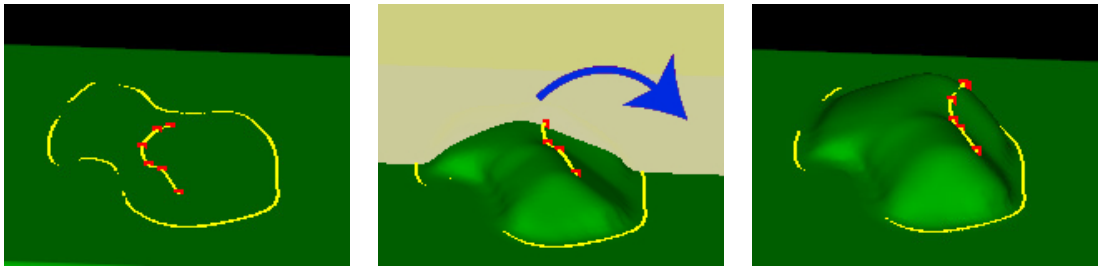


Figure 4.18: A patch on the surface, defined by a boundary curve, is raised using a curve handle. The handle is pulled in an arc towards the right side of the window.

The speed function for this operator is

$$F(\mathbf{x}) = \begin{cases} (1.0 - d_{\text{in}}^{cs}(\mathbf{x})/r)^{\alpha} & d_{\text{in}}^{cs}(\mathbf{x}) \leq r \\ 0 & d_{\text{in}}^{cs}(\mathbf{x}) > r, \end{cases} \tag{4.8}$$

where r is the width of the ROI and $d_{\text{in}}^{cs}(\mathbf{x})$ 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.

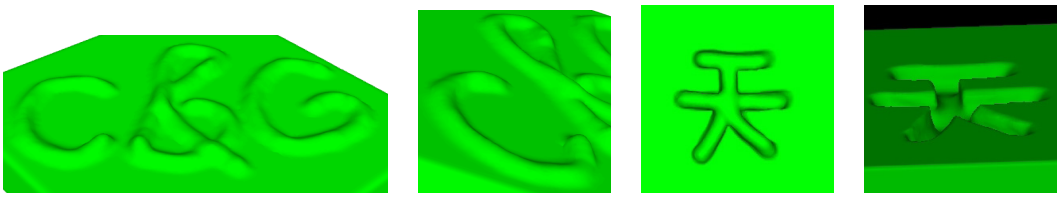
4.3.4 Pulling a curve on the surface, arbitrary ROI

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 4.18 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

$$F(\mathbf{x}) = f(d_{out}(\mathbf{x})) * \left(\frac{\max(d_{\text{in}}^{cs}(\mathbf{x})) - d_{\text{in}}^{cs}(\mathbf{x})}{\max(d_{\text{in}}^{cs}(\mathbf{x}))} \right)^{\alpha}, \tag{4.9}$$

where $d_{\text{in}}^{cs}(\mathbf{x})$ is the geodesic distance to curve C_s and $d_{out}(\mathbf{x})$ is the geodesic distance to the boundary curve C_b from $\mathbf{x}$. $\max(d_{\text{in}}^{cs}(\mathbf{x}))$ is the maximum over all points in the ROI. $f(d_{out}(\mathbf{x}))$ is defined in Equation 4.7 and ensures that the speed function goes smoothly to zero at the boundary curve. 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



(a) (b) (c) (d)

Figure 4.19: 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).

the shape as explained in the previous operators.

4.3.5 Surface Detailing/Carving

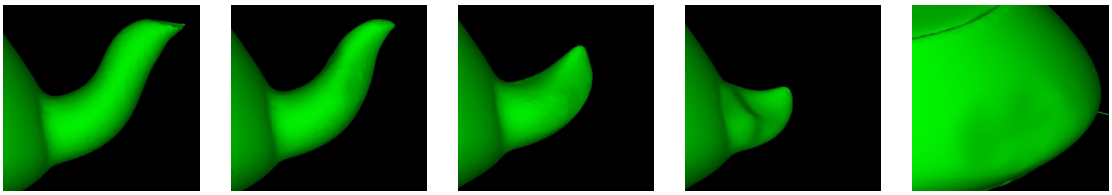
With this operator the user picks a superellipsoid-shaped tool and moves it over the surface to add or subtract features. The surface is locally modified in the general shape of the chosen tool. The size of the tool can be changed interactively and the height/depth of the features depends on the speed of the strokes. The faster the cursor is moved over the surface the lesser the detail that is added and vice versa. This kind of operator is analogous to displacement maps in explicit surface modeling paradigms. A spherical tool is used to create the example in Figure 4.19 a-b.

The speed function for this operator is

$$F(\mathbf{x}) = \begin{cases} 0 & f_{se}(\mathbf{V}) > 0 \\ \beta * f_{se}(\mathbf{V}) & f_{se}(\mathbf{V}) \leq 0 \end{cases} \tag{4.10}$$

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

The tool is centered at the cursor point $\mathbf{x_c}$. $f_{se}(\mathbf{V})$ takes in the relative position



(a) (b) (c) (d) (e)

Figure 4.20: Interactive carving as an erasing tool. Frames (a-d) demonstrate the spout being erased and the last frame (e) shows the final result.

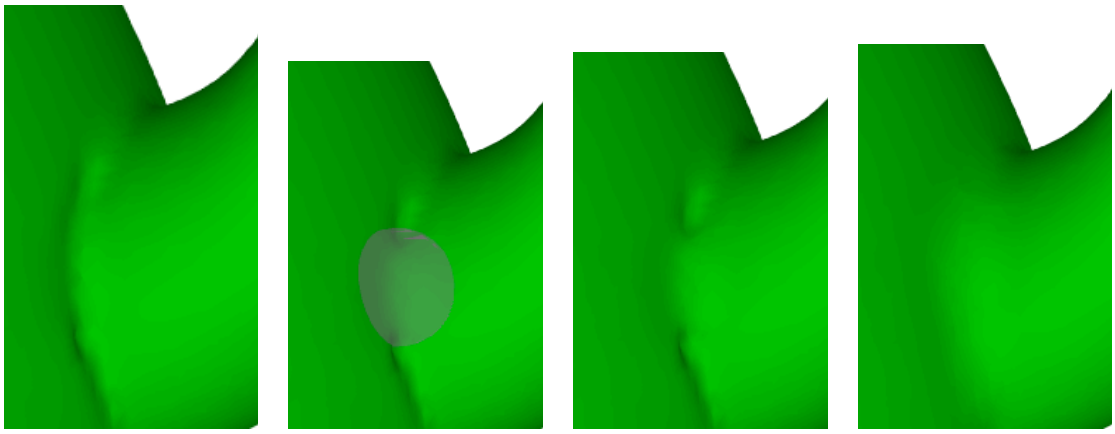
of $\mathbf{x}$ with respect to $\mathbf{x_c}$ and evaluates the superellipsoid implicit inside-outside function [Barr, 1981]. f_{se} is negative inside the superellipsoid, therefore setting $\beta = -1$ drives the surface outwards until it reaches the superellipsoid's boundary where $f_{se} = 0$. Changing 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 4.19c-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 4.20 demonstrates the use of this operator to remove the spout from the teapot model.

4.3.6 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



(a) (b) (c) (d)

Figure 4.21: Interactive smoothing on the spout of the teapot model. (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 the spout meets the teapot.

Museth et al. [2002]. 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 4.11. 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

$$F(\mathbf{x}) = \gamma * g(d_g(\mathbf{x})) * \kappa(\mathbf{x}) \tag{4.11}$$

$$g(d) = \begin{cases} 1.0 & d \leq (r - \epsilon) \\ 0.5 + 0.5 * \cos(\frac{\pi * (d - r + \epsilon)}{\epsilon}) & (r - \epsilon) < d \leq r \\ 0.0 & d > r, \end{cases} \tag{4.12}$$

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 4.21 demonstrates the use of smoothing on the spout of the teapot model with $\gamma = 0.3$.

Table 4.1 shows a summary of editing operators explained so far.

4.4 Sketch-based Editing Operators

Several sketch-based techniques have been developed to edit level set surfaces. The editing operations can be applied locally to a user-defined region or globally to the entire surface. A single closed curve on the surface can be used to identify a specific region of interest/influence (ROI) to be deformed. The user then draws one or more curves on or over the surface to define the outlines of the final model. The surface within the ROI moves towards these curves with the speed functions described below. The curves may then be modified to further shape the surface.

4.4.1 Sketching a Single Cross section

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 (B) on the surface to define an ROI and another curve (C_d) that defines a cross section of the desired shape. Every point within the ROI moves in the general direction of C_d with a speed function defined in Equation 4.13 until the surface reaches C_d. We created a "mountain" on the surface with C_d defining the peak and B defining the extent of the foothills (Figure 4.23). 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.

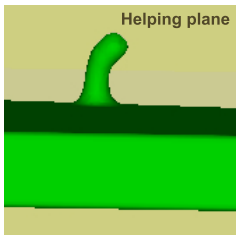
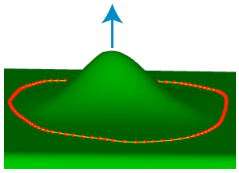
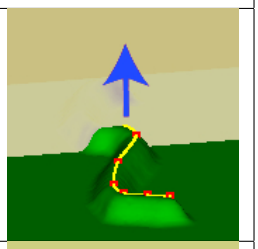
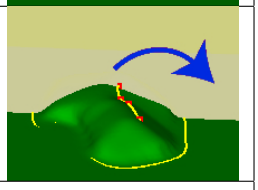
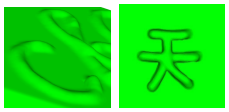
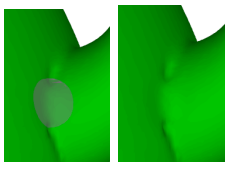
Operator	Speed Function	Result
Pulling a point, symmetric ROI	$F(\mathbf{x}) = \begin{cases} \cos^{\alpha}(\pi/2 * d_s(\mathbf{x})/r) & d_s(\mathbf{x}) \leq r \\ 0 & d_s(\mathbf{x}) > r \end{cases}$ $\mathbf{x}$: point on the surface being evaluated $d_s(\mathbf{x})$: geodesic distance from the point $\mathbf{x}$ to the point being dragged α: user defined parameter that can be used to further control the shape	
Pulling a point, arbitrary ROI	$F(\mathbf{x}) = f(d_{out}(\mathbf{x})) * \left(\frac{\max(d_{\text{in}}^{xs}(\mathbf{x})) - d_{\text{in}}^{xs}(\mathbf{x})}{\max(d_{\text{in}}^{xs}(\mathbf{x}))}\right)^{\alpha}$ $f(d) = \begin{cases} 1/2 - 1/2 * \cos(\pi * d(\mathbf{x})/\epsilon) & d \leq \epsilon \\ 1.0 & d > \epsilon \end{cases}$ $d_{\text{in}}^{xs}(\mathbf{x})$: 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.	
Pulling a curve on the surface, symmetric ROI	$F(\mathbf{x}) = \begin{cases} (1.0 - d_{\text{in}}^{cs}(\mathbf{x})/r)^{\alpha} & d_{\text{in}}^{cs}(\mathbf{x}) \leq r \\ 0 & d_{\text{in}}^{cs}(\mathbf{x}) > r \end{cases}$ r: width of the ROI $d_{\text{in}}^{cs}(\mathbf{x})$: geodesic distance between $\mathbf{x}$ and the curve C_s α is defined above	
Pulling a curve on the surface, arbitrary ROI	$F(\mathbf{x}) = f(d_{out}(\mathbf{x})) * \left(\frac{\max(d_{\text{in}}^{cs}(\mathbf{x})) - d_{\text{in}}^{cs}(\mathbf{x})}{\max(d_{\text{in}}^{cs}(\mathbf{x}))}\right)^{\alpha}$ $d_{\text{in}}^{cs}(\mathbf{x})$, $d_{out}(\mathbf{x})$, $f()$ and α are defined above	
Surface Detailing/Carving	$F(\mathbf{x}) = \begin{cases} 0 & f_{se}(\mathbf{V}) > 0 \\ \beta * f_{se}(\mathbf{V}) & f_{se}(\mathbf{V}) \leq 0 \end{cases}$ The tool is centered at the cursor point $\mathbf{x_c}$ $\lvert V \rvert = \lvert x - x_c \rvert$ $f_{se}(\mathbf{V})$ evaluates the superellipsoid inside-outside function around the cursor location. $\beta = -1$ surface detailing, $\beta = +1$ carving	
Interactive Smoothing	$F(\mathbf{x}) = \gamma * g(d_g(\mathbf{x})) * \kappa(\mathbf{x})$ $g(d) = \begin{cases} 1.0 & d \leq r - \epsilon \\ 1/2 + 1/2 * \cos(\pi * (d - r + \epsilon)/\epsilon) & r - \epsilon < d \leq r \\ 0.0 & d > r \end{cases}$ 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	

Table 4.1: A summary of the freeform editing operators

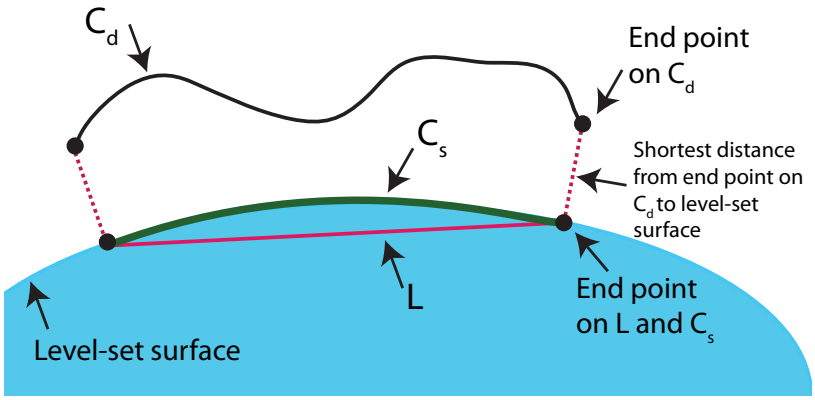


Figure 4.22: Projecting the cross section curve onto the level set surface. The line L in 3D space is created using the closest points to the end points of C_d on the surface. Points starting from L move towards C_d and stop once they reach the surface, creating the projected curve C_s.

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 4.22 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.

The red dots on the cross section curve in Figure 4.23 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 equation is solved once more to fit to the new cross section. An initial bump defined by two sketched curves is shown in Figure 4.23 (top). A control point on

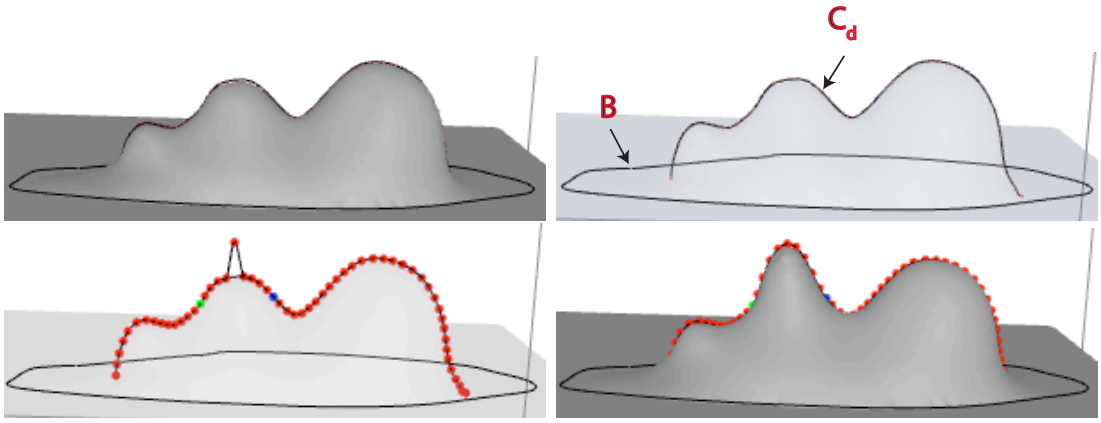


Figure 4.23: Top: Two curves are sketched, one on and one above the surface. The surface grows to fit to both cross sections. The final result is displayed with a surface drawn translucently on the right. Bottom: A control point is modified (left). The surface grows to fit to the modified curve (right).

the upper curve is pulled upwards. Figure 4.23 (bottom-right) presents the resulting cross section curve and the surface that evolves to fit to the curve.

The speed function for this operator is

$$F(x) = \frac{d_{up}(x)}{\max(d_{up}(x))} * f(d_{out}(x)) * \frac{\max(d_{in}(x)) - d_{in}(x)}{\max(d_{in}(x))}, \tag{4.13}$$

$$f(d) = \begin{cases} 1.0 & d > \epsilon \\ (d/\epsilon)^2 & d \leq \epsilon, \end{cases} \tag{4.14}$$

where x is a point on the surface, $d_{out}(x)$ is the geodesic distance from x to B, and $d_{in}(x)$ 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 simply the Euclidean distance between x_{cs} and x_{cd}. Both max functions are taken over all points

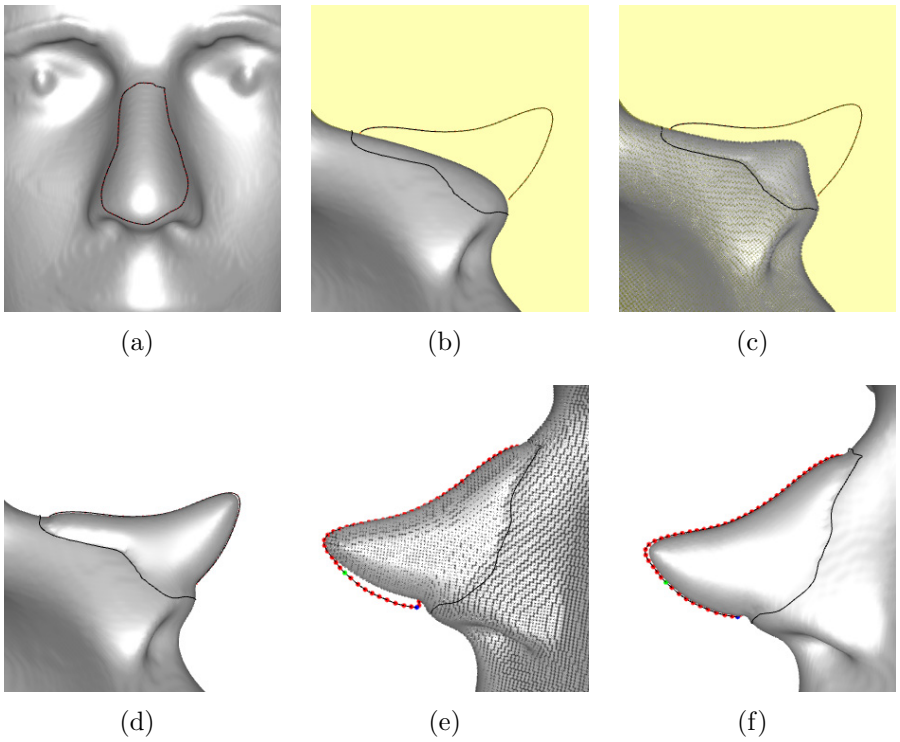


Figure 4.24: (a-b) Two curves define the new shape of the nose. (c-d) The surface fits to these curves. (e) The cross section curve is modified for further refinement of the final shape. (f) The final result. (Some curvature-based smoothing is applied later on to produce the final shape of the nose in Figure 4.36). A point representation of the surface is used in (c) and (e) to provide a clearer view of the curves and control points. $\alpha = 2.0$.

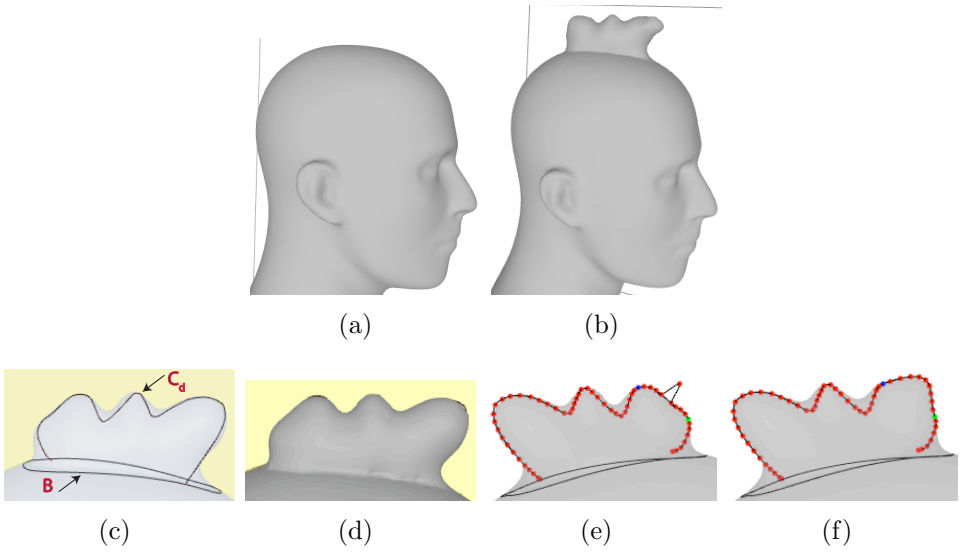


Figure 4.25: One cross section curve is used to create a mohawk for the mannequin head. (a-b) The initial and the final model. (c-d): Two curves define the shape of the mohawk. The surface fits to these curves. (e-f): The cross section curve is modified to further refine the final shape. The surface is drawn translucently in (c,e,f) to provide a clearer view of the curves and control points.

in the ROI. $f()$ is defined in Equation 4.14, and ensures that the speed function (and therefore the deformation) goes to zero within a distance ϵ to boundary curve B.

The first term of Equation 4.13 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 cross section curve C_d to the boundary curve B.

The edits on the nose of the mannequin head in Figure 4.24 and the mohawk in Figure 4.25 are created using this operator.

4.4.2 Multiple Cross Section Curves

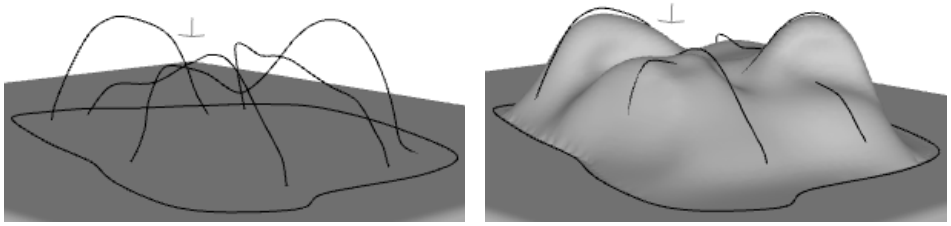
The operator from the previous section was extended to create editing capabilities that use several curves to define a 3D shape. Given a set of cross sectional curves,

we developed two approaches to create a surface that conforms to the shape of these curves. These two techniques were implemented for editing a level set surface by sketching multiple curves with a conventional 2D mouse, and are described in Sections 4.4.3 and 4.4.4. In both approaches the user first draws a closed curve on the surface to define an ROI. Multiple cross section curves are then sketched to define the 3D shape. Once the initial curves are placed the user then can modify them to specify further details. The surface is drawn in translucent colors in this mode to facilitate curve editing. Once curve sketching is completed, the system may be placed in surface evolution mode. In this mode the surface evolves within the ROI after each curve modification. This mode may also be toggled off to once again allow multiple edits of the curves before the surface is updated.

4.4.3 Sketching over the surface

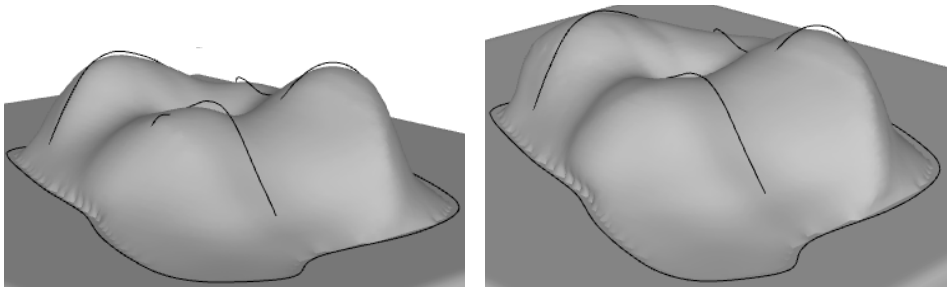
This method allows the user to edit and deform a level set model by sketching planar curves over the surface. We have explored several methods that will fit a surface to a given set of curves in 3D space and have found two reasonable approaches. In the first approach, the surface grows locally until it reaches one of the curves, and then stops at the first curve. The second approach involves blending the influence of each cross section curve at each point within the ROI. Both methods utilize the speed function defined in Equation 4.13 and require the calculation of d_{up} and d_{in} relative to each cross section curve at every point in the ROI.

In the first approach the speed function at point x on the surface is calculated with Equation 4.13 using the closest curve, where "closest" is defined to be the one with the lowest associated d_{in} value. The shape in Figure 4.26 (top right) is produced with this method. The second approach uses a blending function to calculate the speed of a point on the surface by combining contributions from multiple individual speed



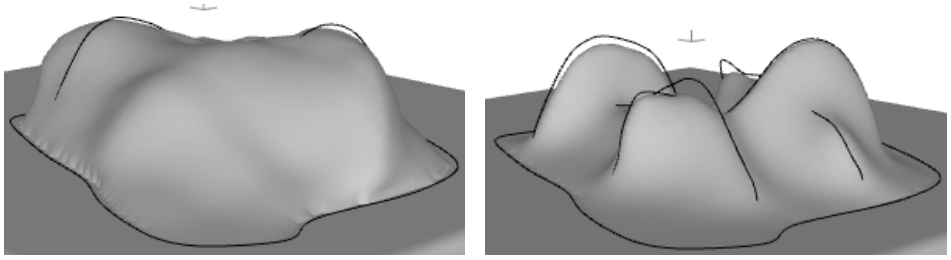
(a) The input curves

(b) Fitting the surface to the closest curve



(c) Blending with α_c^1

(d) Blending with α_c^2



(e) Blending with α_c^3

(f) Blending with α_c^4

Figure 4.26: Sketching cross section curves over the surface.

functions. In general this approach drives the surface to a location between non-intersecting cross section curves. We found this approach to produce more pleasing results and it has been used for the remaining examples. Equation 4.15 describes the general structure of the speed function for multiple curves.

$$F(x) = \sum_{c=1}^{N_c} \alpha_c(x) * F_c(x), \tag{4.15}$$

where N_c is the number of cross section curves, $\alpha_c(x)$ is one of the four blending

functions in Equation 4.16, and $F_c(x)$ is defined by Equation 4.13. For both terms the subscript c indicates that the calculation is done relative to cross section curve c.

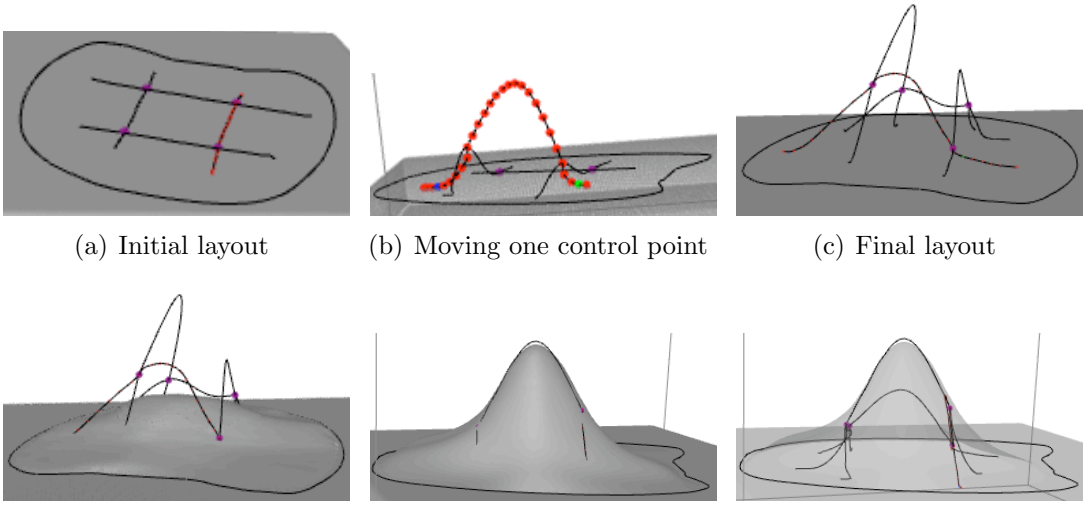
$$\begin{aligned}
\alpha_c^1(x) &= \tfrac{1}{2} + \tfrac{1}{2} * \cos(\tfrac{d_{in}(x)}{\max(d_{in}(x))}\pi) \\
\alpha_c^2(x) &= 1 - \tfrac{d_{in}(x)}{\max(d_{in}(x))} \\
\alpha_c^3(x) &= \tfrac{1}{N_c} \\
\alpha_c^4(x) &= \tfrac{1}{d_{in}(x)}
\end{aligned} \tag{4.16}$$

where $d_{in}(x)$ is the shortest distance between point x and the C_s curve associated with cross section curve c, and $\max(d_{in}(x))$ is computed over all cross section curves.

Figure 4.26 shows how the surface deforms given the input curves in the top left and the different blending functions defined in Equation 4.16. Although all blending functions generate reasonable results, α_c^1 generates the closest fit to all cross section curves while producing smooth transitions on the surface in between curves. α_c^2 also produces a close fit to the curves but the transitions between curves are not as smooth as those produced with α_c^1. α_c^3 is unable to fit to some of the curves closer to the surface and α_c^4 produces a rather sharp drop-off from the cross section curves. We use α_c^1 for the results shown in Section 4.7.

4.4.4 Sketching on the Surface

We have also developed a third approach for modifying a 3D surface from a set of sketched curves by giving the user control over curve-curve intersections. This method requires all cross section curves to be drawn on the surface initially. A 3D line intersection algorithm is then used to calculate intersection points between two curves. Short line segments connecting two consecutive points on each curve are tested against each other for intersections. A bounding box optimization technique is utilized to improve running time. Approximate 3D intersection points are calculated



(a) Initial layout (b) Moving one control point (c) Final layout

(d) Surface evolving to fit the cross section curves (e) Final shape (f) Final shape, surface drawn translucently to better see the curves

Figure 4.27: Sketching cross section curves on the surface.

using a closest point algorithm. Once the intersection points are calculated the two curves are bound together, and the two curves stay attached to each other at these intersection points during curve editing operations.

Once all curves are drawn and all the intersection points are calculated, the user can modify the curves by pulling on the control points. Any of the cross section curves can be selected by switching between curves using the arrow keys. Once a control point on a curve is moved, the curve is modified using techniques described in Section 4.2. When a single curve is edited the change is propagated to intersecting curves via intersection points. When an intersection point on one curve is moved to a new location, the movement is interpreted as an editing operation performed on the connected neighboring curves at the shared intersection points. Once curve sketching is complete, the system may be placed in surface evolution mode, and the surface moves within the ROI to fit to the cross section curves using the speed function in

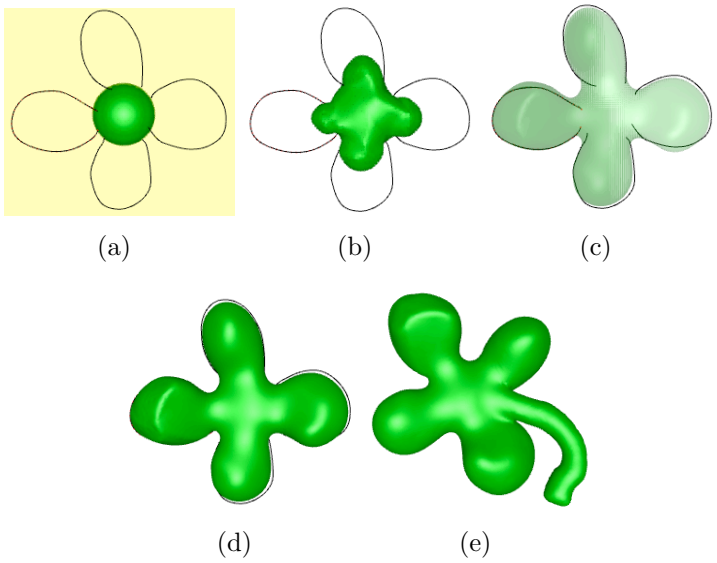


Figure 4.28: Global editing example. The sphere is modified with 4 curves to create a shamrock. An intermediate step during evolution is shown in top middle frame and the final result is drawn translucently in top right. The model is further modified to add a stem in bottom right with a point-based editing operator.

Equations 4.15 and 4.16. Figure 4.27 presents a surface editing session using this approach.

4.4.5 Global Deformations

All editing operators described so far are local, i.e. they are applied only in a user-defined portion of the surface. These operators can be extended to act globally. In this mode, the operators are applied to the entire surface. The general shapes of a number of models have been specified with this operator, e.g. the petals of the shamrock in Figure 4.28, and the initial definition of the shark and duck in Figures 4.40 and 4.39. The speed function for this operator is also defined in Equations 4.13, 4.15 and 4.16. Note that the function $f()$ (Equation 4.14) will always be 1 during global editing, since there is no ROI boundary.

4.5 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.

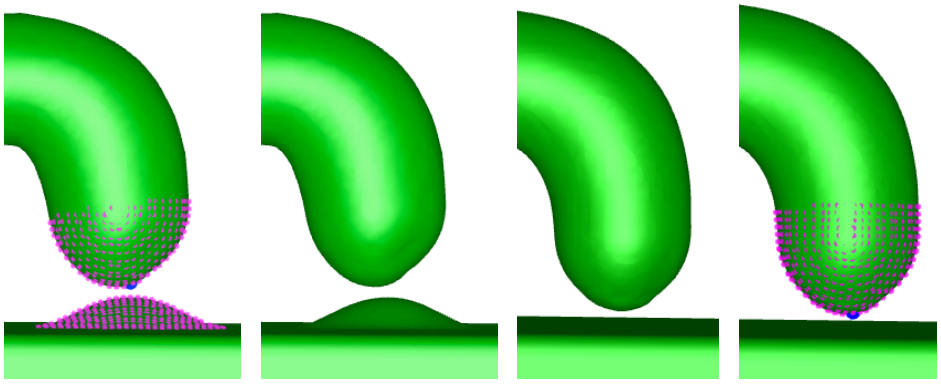
For those operators that employ a user-drawn boundary curve to define an arbitrary ROI, the algorithm first marks all the surface-crossing voxels adjacent to the boundary curve and inside the ROI. Sweeping out from the initial ROI voxels adds all encountered surface-crossing voxels to the ROI voxel list unless marked by the boundary curve. Starting at the handle (point or points on the handle 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 handle. Similarly distance information may be swept in from the boundary curve

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 $\mathbf{x_s}$.

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

for all voxels V in 1-Neighborhood of $\mathbf{x_s}$ **do**
  add V to LIST
  add $\|V - \mathbf{x_s}\|$ 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



(a) (b) (c) (d)

Figure 4.29: 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.

voxels to all the surface voxels inside the ROI to calculate distances to the 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 [2005] 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 4.29a and 4.29b. Here, since the ROI is based on Euclidean distance, the bottom surface bulges out toward the protrusion being created

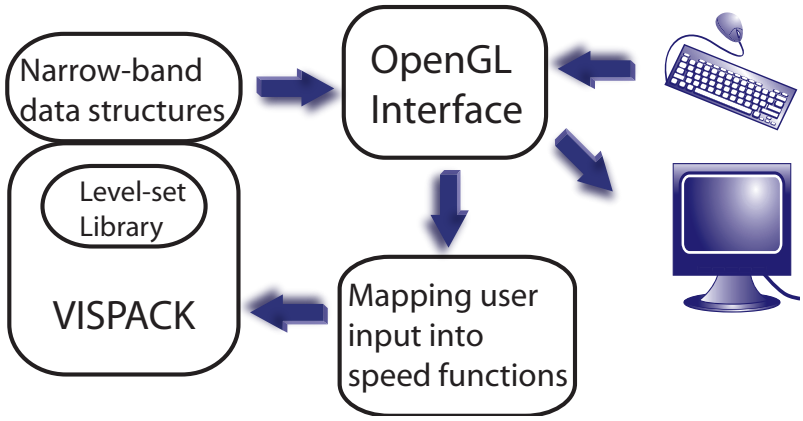


Figure 4.30: Level set surface-editing framework. User input is translated into level set speed functions. The level set PDE is solved on a portion of the narrow-band by the VISPACK library, and the resulting edited model is displayed in the UI.

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 4.29c and 4.29d. The blue points represent x_s and the pink points highlight the ROI in Figure 4.29a and 4.29d.

4.6 Modeling System

The level set surface editing operators have been implemented in an interactive level set modeling system. The system consists of four major components depicted in Figure 4.30, (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 [Whitaker, 2008] to efficiently solve the level set PDE. We have developed an editing user in-

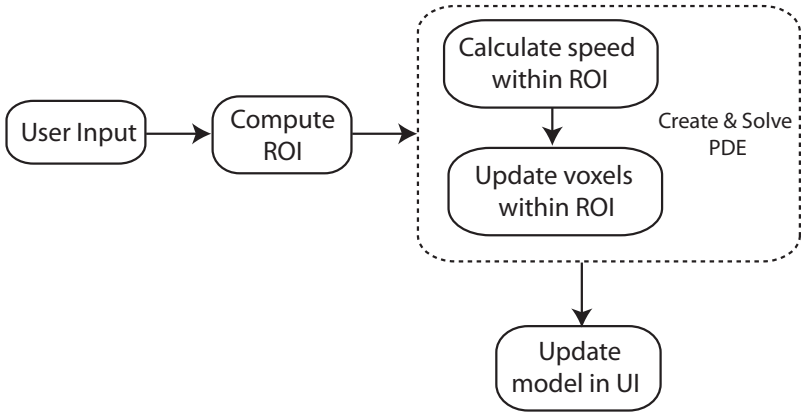


Figure 4.31: The computational pipeline.

terface (UI) within an OpenGL application, as described in Section 4.6.3, 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 intuitive and straightforward way to interact with the model and underlying library functions. We have also enhanced the narrow-band technique[1] in VISPACK to further improve its computational performance. Section 5.1 describes the additional data structures used to achieve real-time evaluation of the level set equation in a subset of the narrow-band.

4.6.1 Computational Pipeline

The steps demonstrated in Figure 4.31 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 4.3 and 4.4 are used

[1]Rather 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.

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 (either as points or polygons) and displayed on the screen.

4.6.2 Numerical Techniques

The speed functions described in Sections 4.3 and 4.4 create hyperbolic PDEs that require upwind differencing schemes for calculating spatial derivatives. VISPACK implements first and second order accurate upwind schemes, while third or fifth order schemes like ENO and WENO [Osher and Fedkiw, 2002] can be used to achieve more accurate solutions. Calculating the higher order numerical schemes impacts interactivity, presenting a tradeoff between time and accuracy. Curvature-based smoothing explained in Section 4.3.6 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 Osher and Fedkiw [2002].

VISPACK employs the sparse-field algorithm [Whitaker, 1998] 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 voxel values are updated during the level set evolution.

4.6.3 The User 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 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 as described in Section 4.2 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.

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 [Wyvill et al., 1986b, Lorensen and Cline, 1987, Bloomenthal, 1994]. 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 4.32, 4.33, 4.34, 6.7, 4.36 and 4.37.

4.7 Results

We created several examples to demonstrate the modeling capabilities of our editing operators. This section shows models created using the freeform and sketch-based editing operators described in this chapter.

Figure 4.32 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 mixture of editing operators listed in Table 4.2. Surface detailing was used to create the log, pulling a point was utilized for the plants and rocks. The animal's body was defined with the sketched cross section and its eyes were created by pulling on a curve with a symmetric ROI.

Figure 4.33 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 general shape of the head was defined with a cross section curve. The nose and arms were drawn out from the body with a point-based operator, and the eyes and the mouth were carved into the head.

We erased the spout and top handle of the teapot model and added new decorative handles with the point-pulling operator to create the model in Figure 4.34. The dimensions of the teapot model is $156 \times 232 \times 124$.

In the next example we edited a $200 \times 250 \times 200$ superellipsoid within a $401 \times$

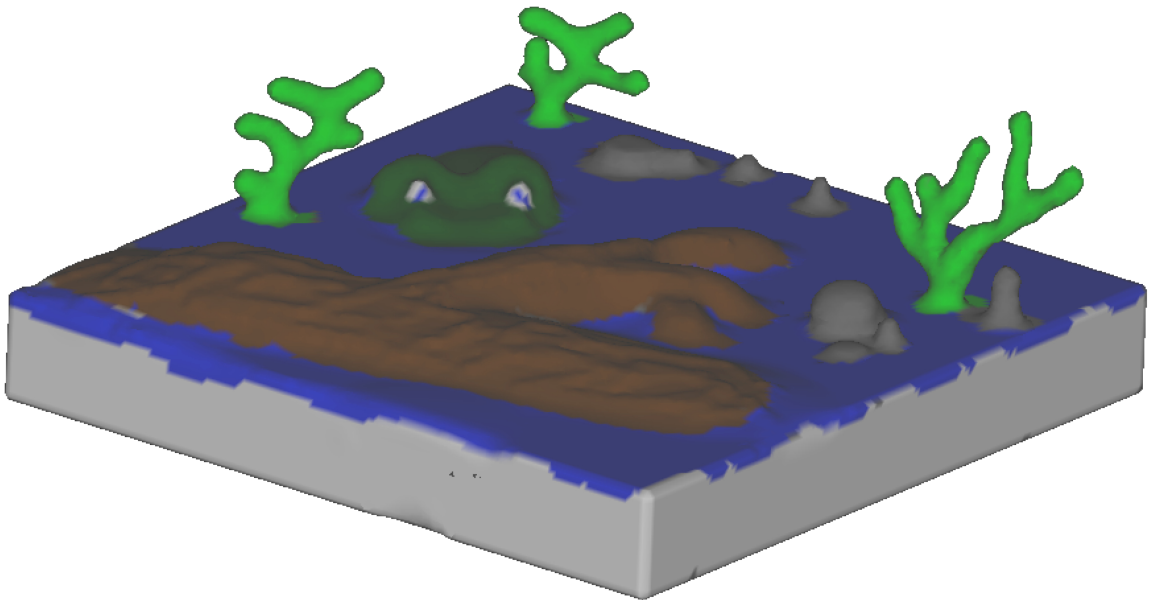


Figure 4.32: Lake with unusual inhabitants. The model is created on one side of a box using several of the level set editing operators.

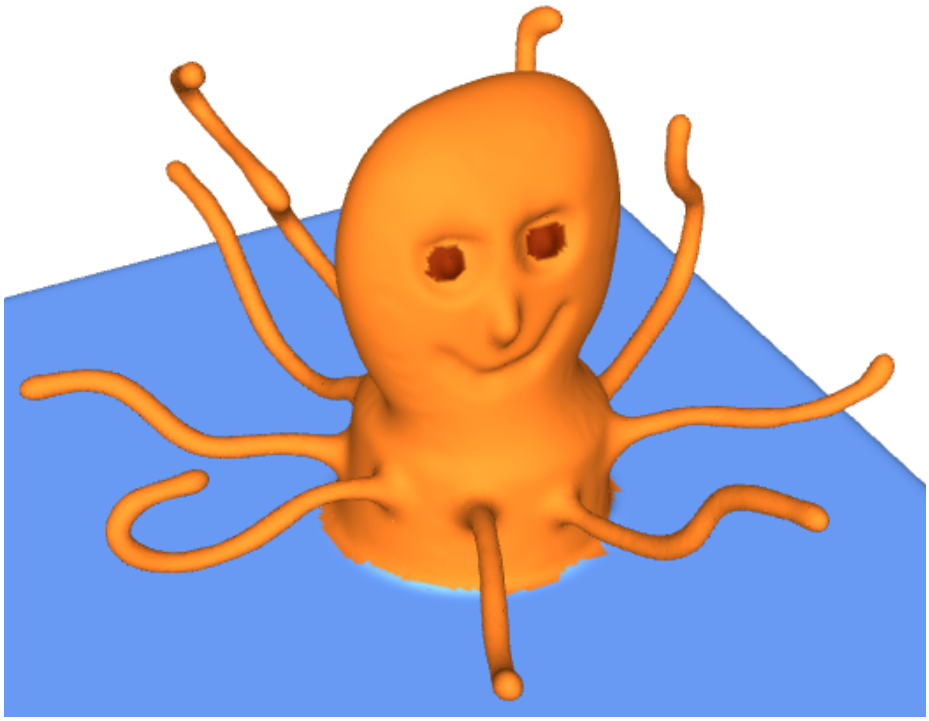


Figure 4.33: Cartoon octopus. The body of the octopus is created on one side of a box using the sketch-based editing operator. The head and the arms are grown from the body by pulling on points using a symmetrical ROI and the eyes are carved into the head.

Figure 4.34: The teapot model 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 4.35: 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.

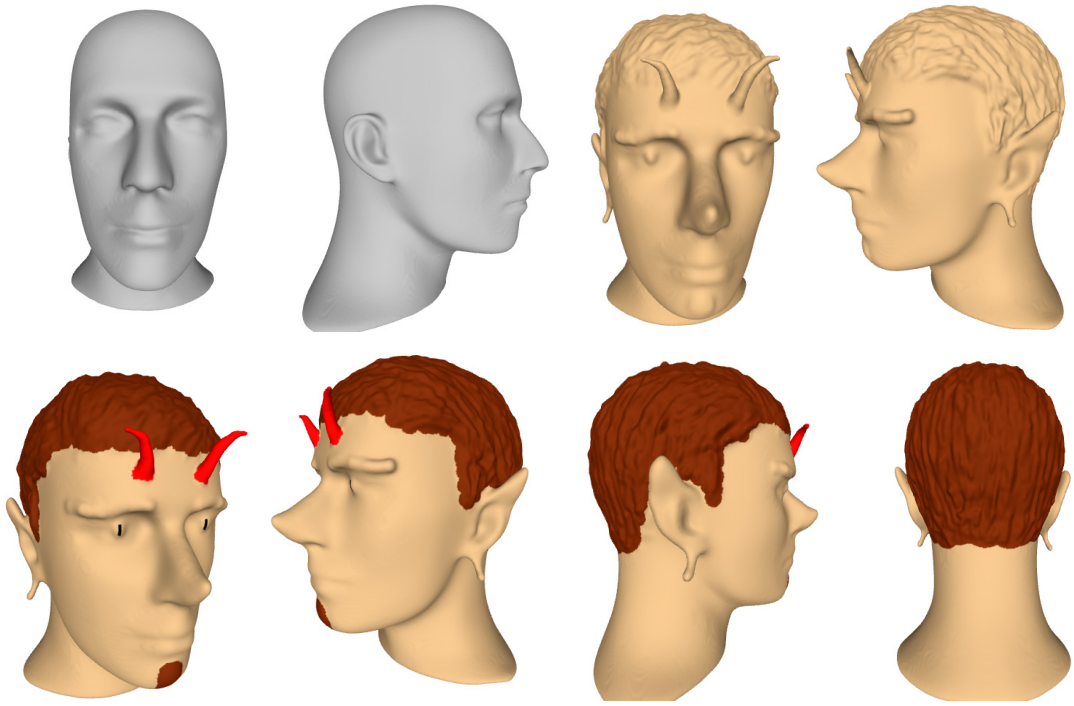


Figure 4.36: A fantasy character is created by adding horns and pointy ears to the mannequin model. The chin, eyes and nose are also modified and hair detail is added.

(a) (b) (c) (d) (e)

Figure 4.37: 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.

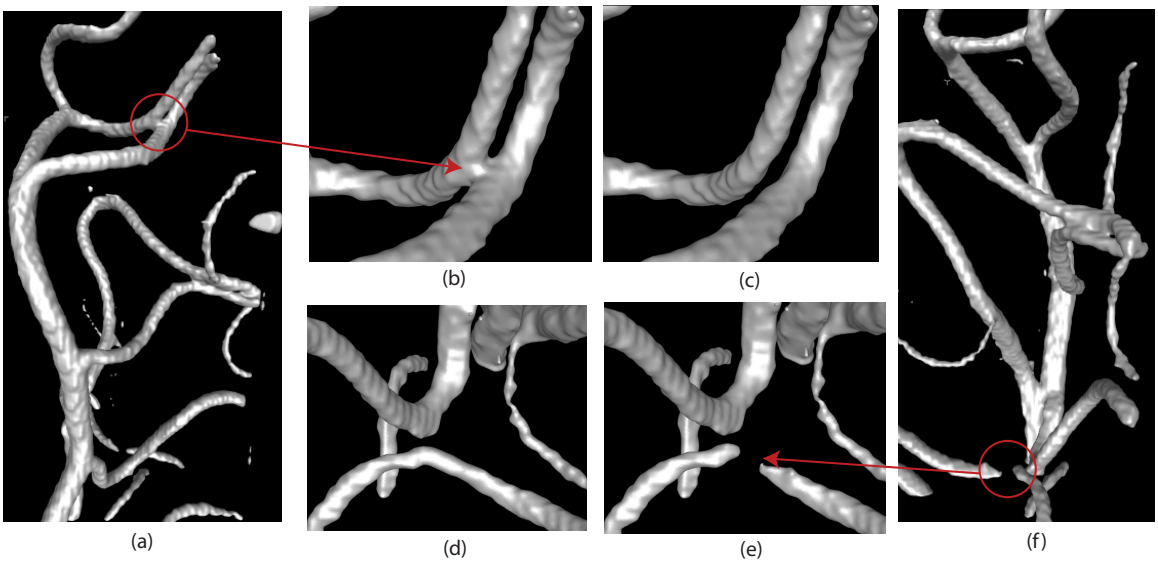


Figure 4.38: Topological repair of a vasculature data set. (a and f) The original model. (b-c) The volume is manipulated using interactive carving to separate two vessels that were merged due to errors in 3D scanning. (d-e) The volume is manipulated to recover lost data by connecting a vessel that was separated.

401×401 volume. The eyes of the Kermit-like figure are added by pulling the surface up and the mouth is modeled using interactive carving (See Figure 6.7).

Figure 4.36 shows a fantasy character created from the mannequin head. The hair, horns and eyebrows are added using surface detailing and point-pulling, and the eyes, ears, nose and chin are modified with surface detailing, curve cross sections and curve-pulling. The resolution of this model is $360 \times 435 \times 510$.

The body of the cartoon bear in Figures 4.3(a) and 4.37 is initially defined as the union of two superellipsoids in a $320 \times 320 \times 600$ volume. The legs and the nose are pulled out from the body symmetrically and the eyes are carved in. The general shape of the arms is created with surface detailing with a large tool. The arms, claws and the coat detail are added via surface detailing with a smaller tool.

We applied our editing operators to a dataset obtained from a vasculature MRI

scan (Figure 4.38). This dataset contains topological errors inherent to 3D scanning and reconstruction. Interactive carving is used to separate two blood vessels that were incorrectly merged. A blood vessel was interactively connected where it 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 $621 \times 371 \times 346$.

Figures 4.23, 4.26 and 4.27 begin with a box model with resolution $161 \times 161 \times 101$. Figure 4.25 uses the scan-converted mannequin head model at resolution $134 \times 160 \times 186$. Figures 4.40, 4.28 and 4.39 all start with a $20 \times 20 \times 20$ sphere within a $150 \times 150 \times 150$ resolution volume. The final resolution of the bounding box around each model is given in Table 4.2.

We created three "plastic toys" using the sketch-based modeling system. The shark model in Figure 4.39 is created using eight curves and a combination of local and global editing operators. The initial body is created from the sphere using a single cross section curve and the global editing operator. Three additional curves further define the head and the tail and create a slightly curved body. Three fins are added similarly by using a closed curve to define a ROI and a cross section curve for the fin shape. A painting capability allows color to be placed on the model. The painted and shaded final model can be seen from different views in Figure 4.39.

The duck (Figure 4.40) and the shamrock (Figure 4.28) models are similarly created using sketched curves and the level set sphere model. An outline curve for the duck is sketched and the sphere deforms to fit this curve. In order to create the wing the user draws one closed curve on the surface that identifies where the wing is going to be placed and another curve over the surface to define a cross section of the wing. Details like the eyes on the duck and the stem of the shamrock are created using point-based freeform editing operators.

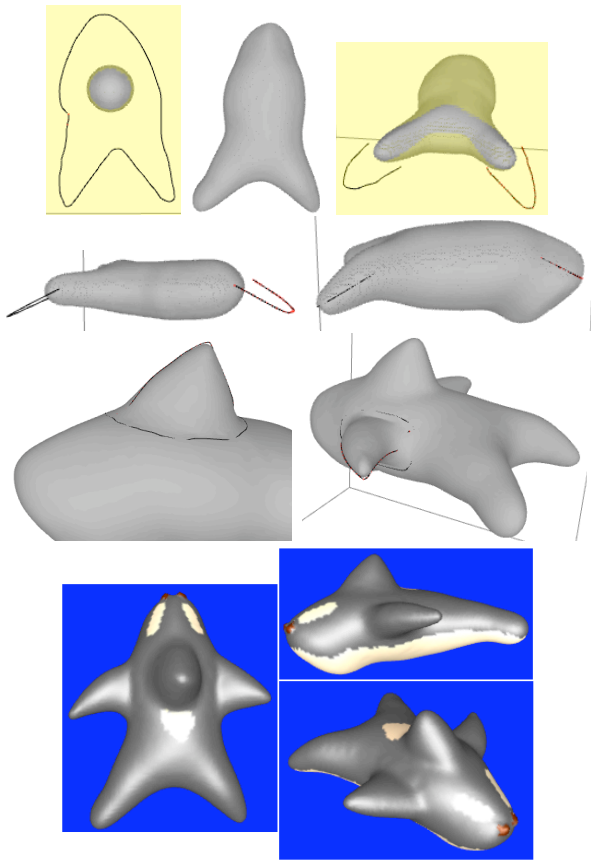


Figure 4.39: A sphere and a cross section curve is used to create the initial shark body. The tail and head are modified using additional curves. The fins are added by locally editing the shark body. The final painted model is shown from three different views.

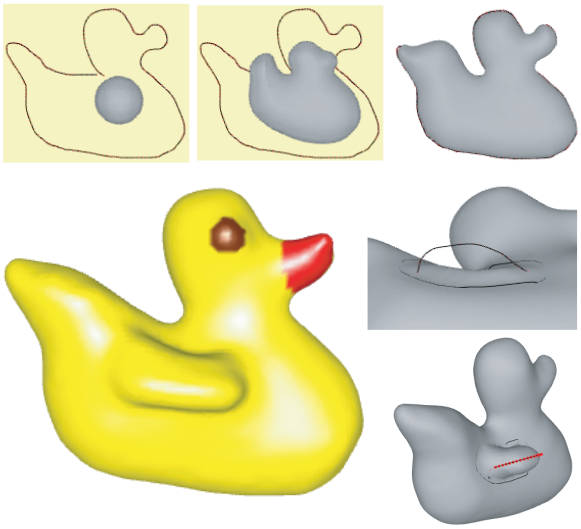


Figure 4.40: A duck is created from a sphere and a cross section curve. A wing is defined with a sketch-based editing operation.

4.8 Discussion

We have used the following techniques to measure the accuracy and effectiveness of our operators in comparison with the state-of-the-art surface manipulation frameworks. Table 4.2 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 dual Intel quad-core 3.2GHz CPUs and 14 GB of memory running Mac OS X 10.5. Even though the modeling application is not multi-threaded and is only running on one core, running times demonstrate interactive rates for a variety of volume resolutions.

The time needed to evaluate the speed functions is linear with the number of voxels within the ROI. This number is directly related to the ROI's radius, which is measured in voxels, for an operator with a symmetric ROI. In other words, changing the model's resolution does not change the running time of a certain operator for a given ROI. However, if one doubles each dimension of a model's volumetric representation, the radius of the ROI would also need to double in order to cover the same region on the model, thus increasing the number of voxels in the ROI (and the associated running time) by a factor of 4.

Since the computation of our editing operators is completely localized to the ROI, the time needed to compute them is not necessarily tied directly to the resolution of the model. Section 5.2 describes how additional data structures give the system the ability to limit computation to the subset of the level set surface that is actually being modified and increase performance. A good example of this can be observed in Figures 4.32 and 4.36. As stated in Table 4.2, the operator used to create 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, since the area of the ROI

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
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
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
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
Shamrock, Dimensions: $45 \times 50 \times 35$		
General shape	Sketch-based editing (global)	83
Stem	Pulling on a point, symmetric ROI	200
Rubber duck, Dimensions: $79 \times 67 \times 37$		
General shape	Sketch-based editing (global)	66
Wings	Sketch-based editing	90
Beak	Pulling on a point, symmetric ROI	200
Toy shark, Dimensions: $85 \times 99 \times 54$		
General shape	Sketch-based editing (global)	66
Fins	Sketch-based editing	90

Table 4.2: Editing details and running times for the final results. Speed is in frames-per-second (fps).

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

used to create the horns is approximately equal to the area of the ROI used to create the plants on the lake.

Results of another run time experiment are presented in Table 4.3. The table contains running times in frames-per-second for editing a sphere model with a given radius and the same operator, i.e. pulling a point with a symmetric ROI. The resolution of the underlying volumetric model was successively doubled in each dimension; thus increasing the total number of voxels by a factor of 8 for each expansion. For each model the ROI was also extended (by doubling its radius) to ensure that the same region of the sphere was modified for each experiment. The number of voxels within the ROI increases by a factor of 4 every time the ROI radius doubles. The comparison of running times as the resolution increases shows that the run times are linear with the number of voxels processed for that operation. The average increase in run time is approximately a factor of 4. A comparison of the last two rows of Table 4.3 also shows that the run times are independent from the volume resolution or the surface area of the model.

A major limitation of level set model editing is the memory required to store the underlying volumetric representation. While narrow-band schemes effectively address

the problem of time 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. The initial examples presented here have been created with low-resolution volume datasets, which limit the size of the feature that can be specified on the surface. Chapter 5 explains how we overcame this limitation by utilizing sparse volume datasets, as well as techniques for localized surface editing.

5. Representing High Resolution Level Set Models for Interactive Editing and Rendering

In Chapter 4, we described a set of level set surface editing operators and sketch-based techniques that may be used to modify level set surfaces. For these techniques to provide value to the user, they must operate in a framework that can effectively process high resolution volumetric models. This chapter describes utilizing sparse volume data structures in order to reduce the memory requirements of volumetric implicit models, as well as localized editing techniques, i.e. processing only the voxels within the ROI during an edit, in order to reduce running times that facilitate interactive editing of high resolution surfaces.

5.1 Efficient and Dynamic Data Structures for High Resolution Level Set Models

A major limitation to editing a level set model is the memory required to store its underlying volumetric representation. The space and time complexity of representing and deforming a level set model have prevented these models from being utilized in interactive modeling systems. An additional problem common to interactive systems for editing large-scale models is rendering the surfaces at interactive rates. A current state-of-the-art volumetric modeling system should support models containing at least one billion voxels and provide 25-30 frames-per-second (fps) evaluation and rendering rates at these resolutions. Even the most advanced data structures and algorithms developed to date for level set models are not able to meet both of these requirements simultaneously. This section describes a novel approach to storing a level set model in a compact spatial hash table, as well as the utilization of k-d trees to optimize

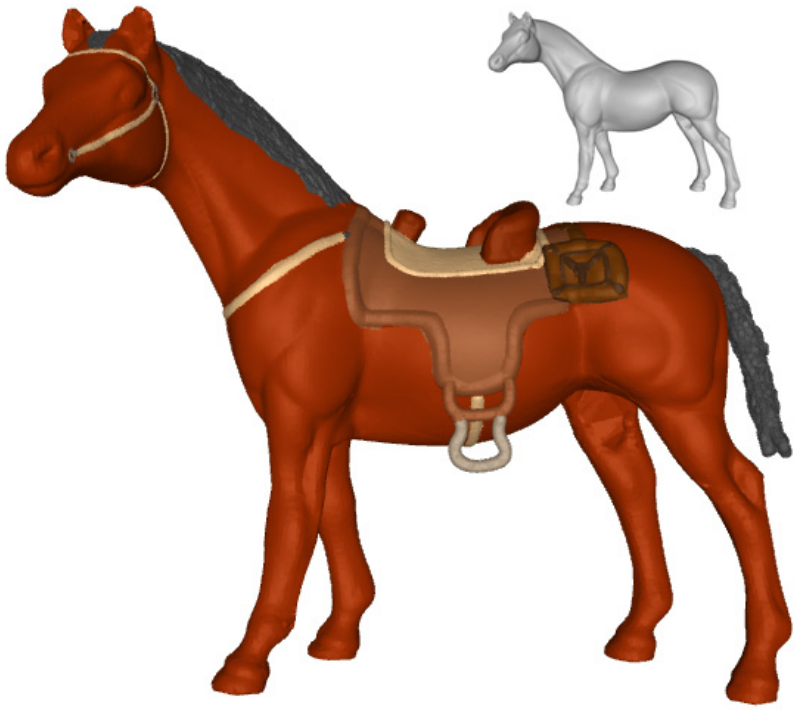


Figure 5.1: A scan converted level set model of a horse (upper right) is edited to add surface details.

rendering times. Our work closes the gap between level set methods and interactive modeling applications by providing new techniques that allow these models to be incorporated into current modeling systems. A level set model interactively created with our new data structures is shown in Figure 5.1.

The resolution of the models shown in Section 4.7 is limited by the memory required to store the volume dataset that represented the surface. For example, the largest level set volume dataset that we were able to store and interactively manipulate on a computer with 14GB of memory contained approximately 80 million voxels. This represents a cubic volume dataset with approximate dimensions of 430^3 voxels.

Higher spatial resolutions are needed in order to make detailed models with small, fine structures. Creating a volumetric model where n approaches 2,000 (8 billion

voxels if stored in a 3-D array) would provide high resolution capabilities and would be desirable for current volumetric modeling applications. A sparse data structure that only stores the data associated with the voxels lying in the narrow-band is the key to representing high resolution level set models. The challenge when creating volumetric models of these sizes is to keep the model processing time *interactive*, i.e. providing model and display updates at a rate of 25-30 frames-per-second (fps). While current level set data structures can represent models at these high resolutions [Houston et al., 2006, Nielsen and Museth, 2006, Nielsen et al., 2007], they do not support the rapid, arbitrary access and update operations required for interactive editing applications. To address this deficiency, this chapter presents data structures that enable interactive editing of large-scale level set surface models. The new approach utilizes spatial hashing to represent a narrow-band of voxels around the level set interface, as well as a k-d tree to hold the model's display points that lie on the surface itself. This sparse representation of voxels and surface points lets us create and modify high resolution level set models with modest memory requirements, while allowing fast data access/modifications and interactive graphics updates (normally above 25 fps). It also supports out-of-the-box editing, i.e. no bounding box limits the surface editing region, a restriction common when utilizing 3-D arrays. As compared to previous PDE-based modeling work, our system provides significantly faster processing speeds on much larger volumetric models, even when considering the difference in processor power.

5.1.1 Voxel Representation

For our level set editing system, we pack sparse data into a dense 1-D hash table using a hash function $H(P)$ for a set of data with 3-D coordinates P. The hash values should be uniformly distributed in order to minimize collisions and to guarantee

adequate performance. Furthermore, the evaluation time for the hash function should be compatible with the frame rates of the interactive application. To meet these requirements, we use spatial hashing to store data associated with narrow-band voxels in a 1-D hash table of size T_S. The data includes the 3-D position of the voxel, distance values, a gradient vector, a color index, a pointer into the narrow-band data structure, and indices into the arrays storing display data. Hash values are computed for all discrete vertex positions P using a hash function $H(P)$.

We use the hash function described in Teschner et al. [2003]

$$H(P) = (P_x \times C_1 \hat{} \; P_y \times C_2 \hat{} \; P_z \times C_3) \,\%\, T_S,$$

where P_x, P_y, and P_z are 3-D coordinates, and C_1, C_2, and C_3 are three constants, $\hat{}$ is the *bitwise exclusive or* operator and % is the *modulus* operator. The three constants, $C_1 = 73,856,093, C_2 = 19,349,663$, and $C_3 = 83,492,791$, are large prime numbers. We assume that the modulus operator always returns positive numbers, i.e. $-1 \,\%\, 5 = 4$. Teschner et al. [2003]'s analysis concludes that the function can be evaluated very efficiently and produces a comparatively small number of collisions with large hash tables.

For a level set model with $O(n^3)$ voxels, where n is the resolution in one dimension, the number of voxels on the surface is $O(n^2)$. We need at least three neighboring voxels on each side of the interface for the finite difference methods that calculate surface normals and curvature. We set the size of the hash table to be $7 * n^2$, which is approximately equal to the number of voxels inside the narrow-band. This drastically reduces the space complexity for large n, i.e. $n \sim 2^{10}$, while keeping the number of entry collisions low. For non-square volumes, the hash table size is set, at the beginning of an editing session, to $7 * I * J$, where I and J are the two largest dimensions of the bounding box around the model to be edited.

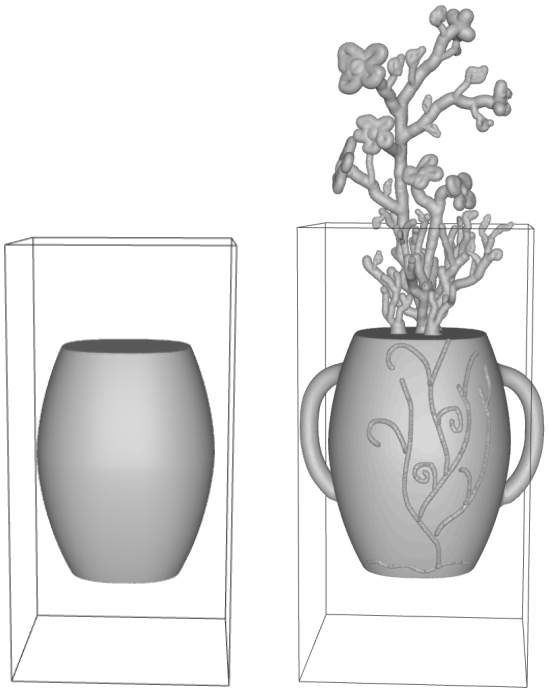


Figure 5.2: Left: Scan converted initial model with its bounding box. Right: Level set editing operators create a new model that extends outside of the bounding box.

In addition to the benefits of low memory requirements and fast access/modification times, spatial hash tables provide the extra advantage of implementing an unbounded grid; thus enabling "out-of-the-box" computing. In the past storing level set values in a 3-D array has constrained the model to lie within the bounds of the array. Similar to the data structures of Houston et al. [2006] and Nielsen and Museth [2006], spatial hash tables allow level set models to evolve outside of the bounds of their initial model. Figure 5.2 presents the bounding box of a scan-converted model and demonstrates how editing operators can create a new model that extends out of this bounding box.

5.1.2 Display Representation

We employ OpenGL Vertex Buffer Objects (VBOs) to display the points lying on the dynamic level set surface. Every time the surface changes the buffer holding the point data is remapped and transfered to the GPU. This remapping process may slow down an editing application if a large-scale model is displayed with one buffer, and the entire buffer must be updated for small changes in the model occurring every frame. Since our editing operators usually modify the surface locally we spatially divide the surface amongst several VBOs. Only the VBOs associated with the modified portion of the model are remapped during an editing operation. This distribution of the surface over several VBOs and the selected updates of a subset of the VBOs significantly improve display performance for small, localized surface modifications.

It is important to partition the surface evenly between the VBOs to optimize update times. One can utilize binary space partitioning to accurately distribute the vertices between VBOs. This method ordinarily requires that the data first be sorted when building the partition, which requires $O(N \log N)$ time for N data points. However, the surface is likely to change between each frame, thus requiring that the data structure be completely recomputed in order to keep the data sorted and the binary tree balanced. Therefore, it is more advantageous to use a simpler algorithm with smaller time complexity that subdivides the set of vertices into approximately equal segments. We have loosened the requirement of having a strictly balanced BSP tree in order to minimize the amount of time spent subdividing and balancing the tree that holds the display vertices. Therefore, we utilize a k-d tree and a separation plane calculation that does not require sorting to produce an approximately balanced tree. We employ a divide-and-conquer method that recursively finds the centroid of the display vertices and subdivides the set into two parts around an axis-aligned plane passing through the centroid. Finding the centroid is linear in both space and time

complexity.

Display vertices are added to and dropped from the surface as it changes from application of the editing operators. Enhancements to the VISPACK library (see Section 5.2) return the lists of voxels that are added to and removed from the narrow-band. Once a vertex is added, the VBO that should display this vertex is located by traversing the k-d tree and inserting the vertex into the vertex array associated with the particular VBO. If a vertex is dropped because it is no longer on the surface, its location in the VBO arrays can be retrieved from a hash table, and then removed from the VBO's vertex array. Only the VBOs associated with the modified nodes of the tree are remapped at each frame.

The algorithm does not guarantee to evenly partition a set of points on a level set; however, it is straightforward to calculate and does not require sorting of the vertices. As demonstrated later in Section 5.4 (see Figure 5.8), it creates a fair partitioning where each VBO will be assigned a set of vertices to display. Being a binary tree, it also answers vertex location queries in $O(L)$ time, where L is the level of subdivision, providing a fast way to update the vertex data and identify which VBOs need to be remapped at every frame.

We have found that having the unbalanced distribution of vertices in the VBOs produced by our partitioning method does not significantly impact graphics performance when editing and displaying our models. During editing, it is possible that the imbalance of vertices can increase significantly and slow the interactive display when updating and drawing the model. Given this situation, which occurs infrequently, the partitioning algorithm should be run again to improve the distribution of vertices within the VBOs. We have explored a number of methods for automatically triggering the repartitioning of the display vertices, based on frame rates and on quantifying the imbalance using the difference between or the ratio of the minimum and maximum

VBO sizes. If the difference or ratio changes significantly from the value calculated at initialization then the VBO data structure can be recomputed. Since graphics performance was normally not an issue when creating our models, we provide a manual method for repartitioning display vertices via user input in our system interface, which the user can execute if display times degrade during the editing session.

5.2 Local Surface Editing Techniques

VISPACK contains a sparse-field narrow-band implementation for efficiently solving the level set equation [Whitaker, 1998]. 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

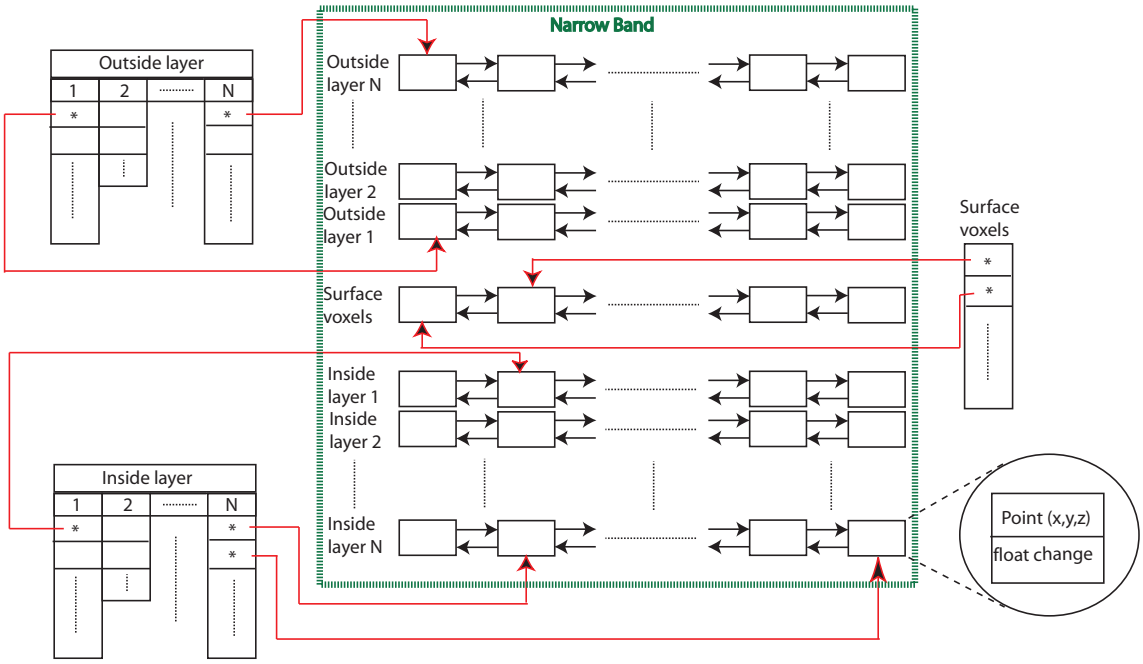


Figure 5.3: Three additional data structures (Surface voxels vector, Outside layer vectors and Inside layer vectors) are added to the narrow-band VISPACK data structure. The new data structures support interactive update rates by identifying the subset of voxels in the narrow-band needed to solve the level set PDE during an editing operation.

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 spatial hash table of pointers provides constant-time access to any narrow-band element. Each element in the hash table 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 5.3 shows the VISPACK narrow-band data structures within the dotted green box, as well as the new data structures that further localize the level set compu-

tations. VISPACK keeps a linked list of all 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 Voxels 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 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 5.4 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

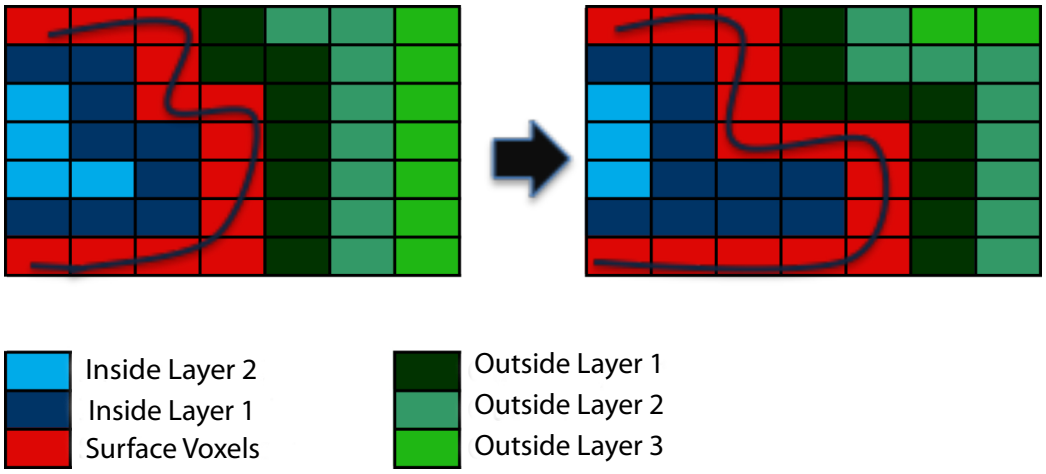


Figure 5.4: Changes in the narrow-band linked lists as the curve on the left evolves into the curve on the right.

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 spatial hash table of pointers at the same resolution as 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 hash table 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

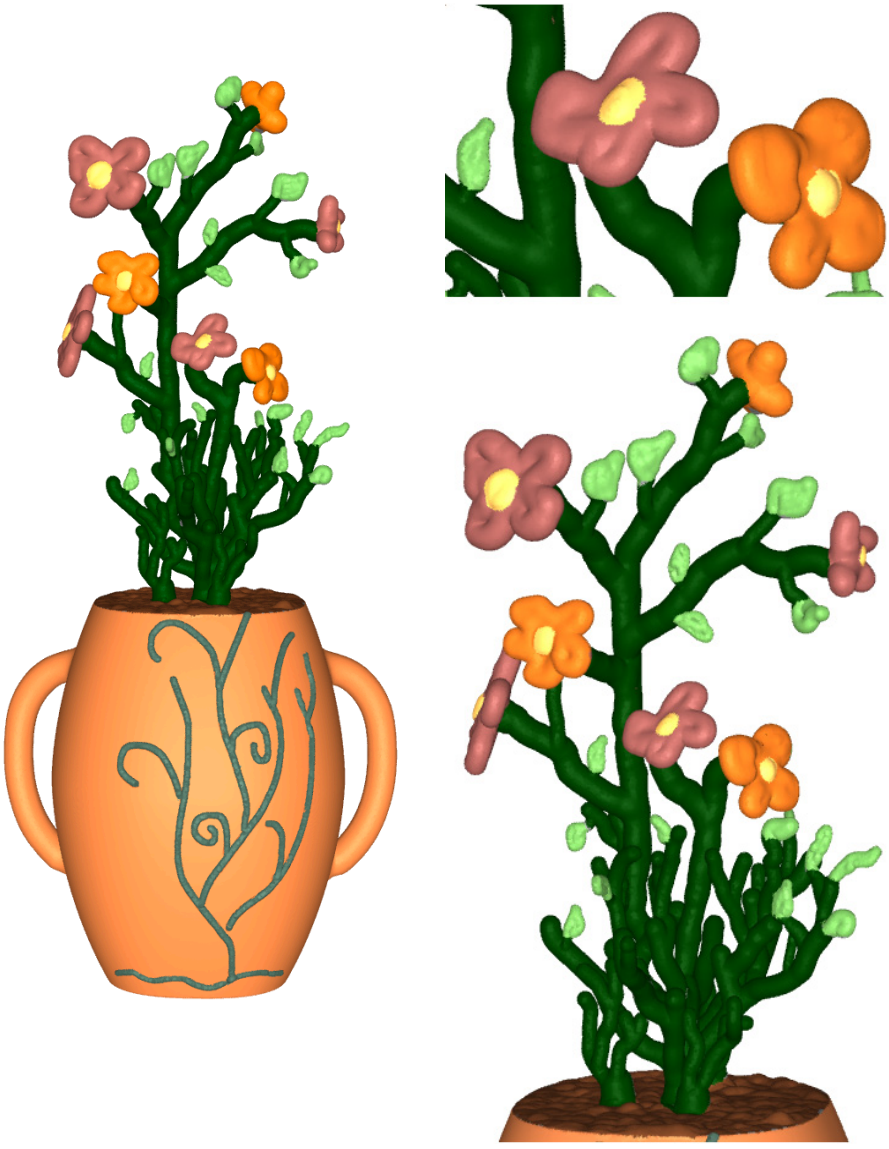
times using the additional pointer vector data structured compared to using just the original VISPACK narrow-band data structures for a 128^3 volume with an ROI that has a 20 voxel radius with the editing operators from Section 4.3.1. The surface embedded in this volume has approximately $128^2 \sim 16K$ 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.

5.3 Results

We created three models to demonstrate the capabilities of our level set editing system using sparse volume data structures. These models are significantly higher in resolution compared to those shown in Section 4.7.

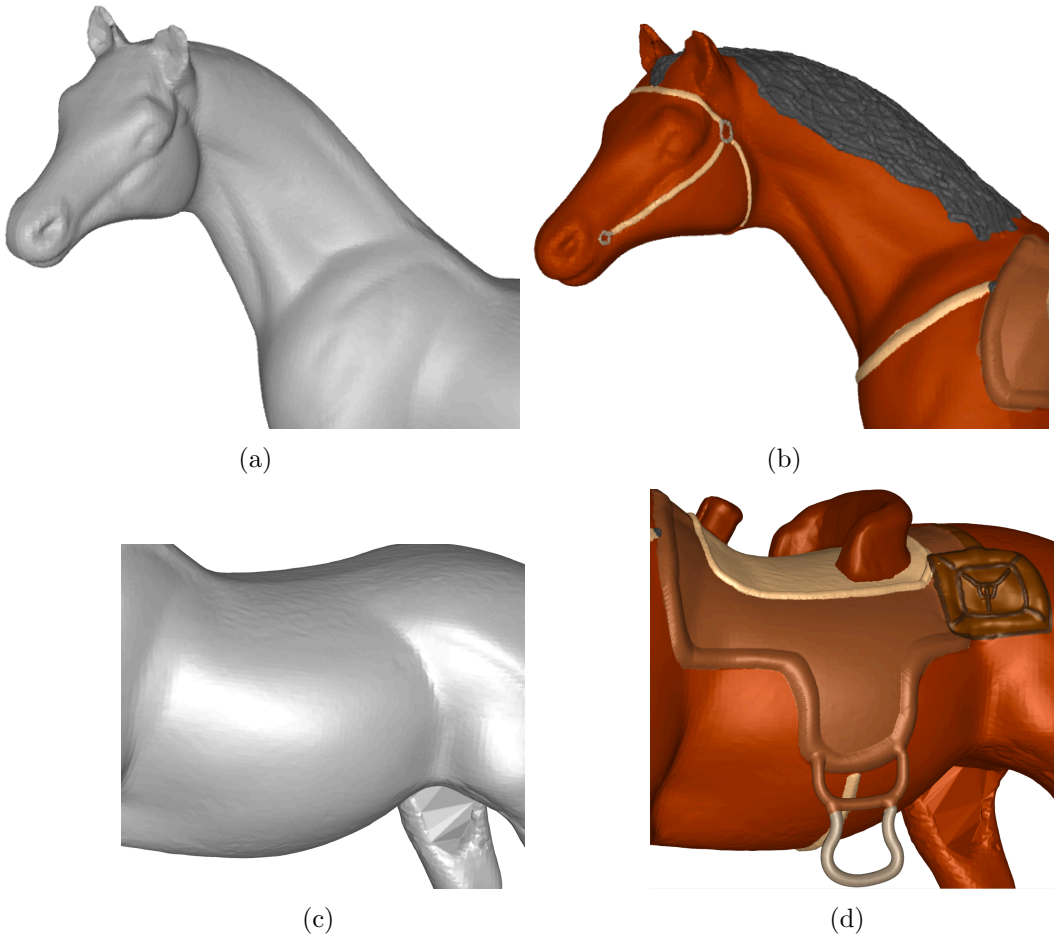
Figures 5.2 and 5.5 show a flower pot created from a simple initial model. The pot was defined as a scan-converted superellipsoid [Museth et al., 2005]. The stems, flowers and leaves are sculpted on top of the pot, and some soil and surface details are added to the pot itself with freeform surface editing operators. The pot was scan-converted into a $600 \times 600 \times 1200$ volume with $7,384,242$ voxels in the narrow-band and the final edited model moved out of this bounding box and has effective dimensions of $654 \times 600 \times 1794$, with $11,852,818$ voxels in the narrow-band.

Figures 5.1 and 5.6 show the editing of a horse model. We added a saddle, stirrups and a bridle to the model, as well as a tail and mane, using our editing tools. The initial and final models have the same effective dimensions, $944 \times 2048 \times 1709$. The initial model has $26,236,562$ voxels in the narrow-band and the final edited model has $28,582,546$ narrow-band voxels.



(a) (b)

Figure 5.5: A flower pot is modeled from a superellipsoid. (a) Handles and decorations on the surface are added to the initial model. Soil is added to the top of the pot. Stems, leaves and the flowers are then modeled above the soil. (b) Close-ups of the final painted model.



(a) (b)

(c) (d)

Figure 5.6: (a,c) Parts of the scan converted horse model. (b) A bridle, and mane are added. (d) A saddle, stirrups and saddlebag are added. (See Figure 5.1 for the final model)

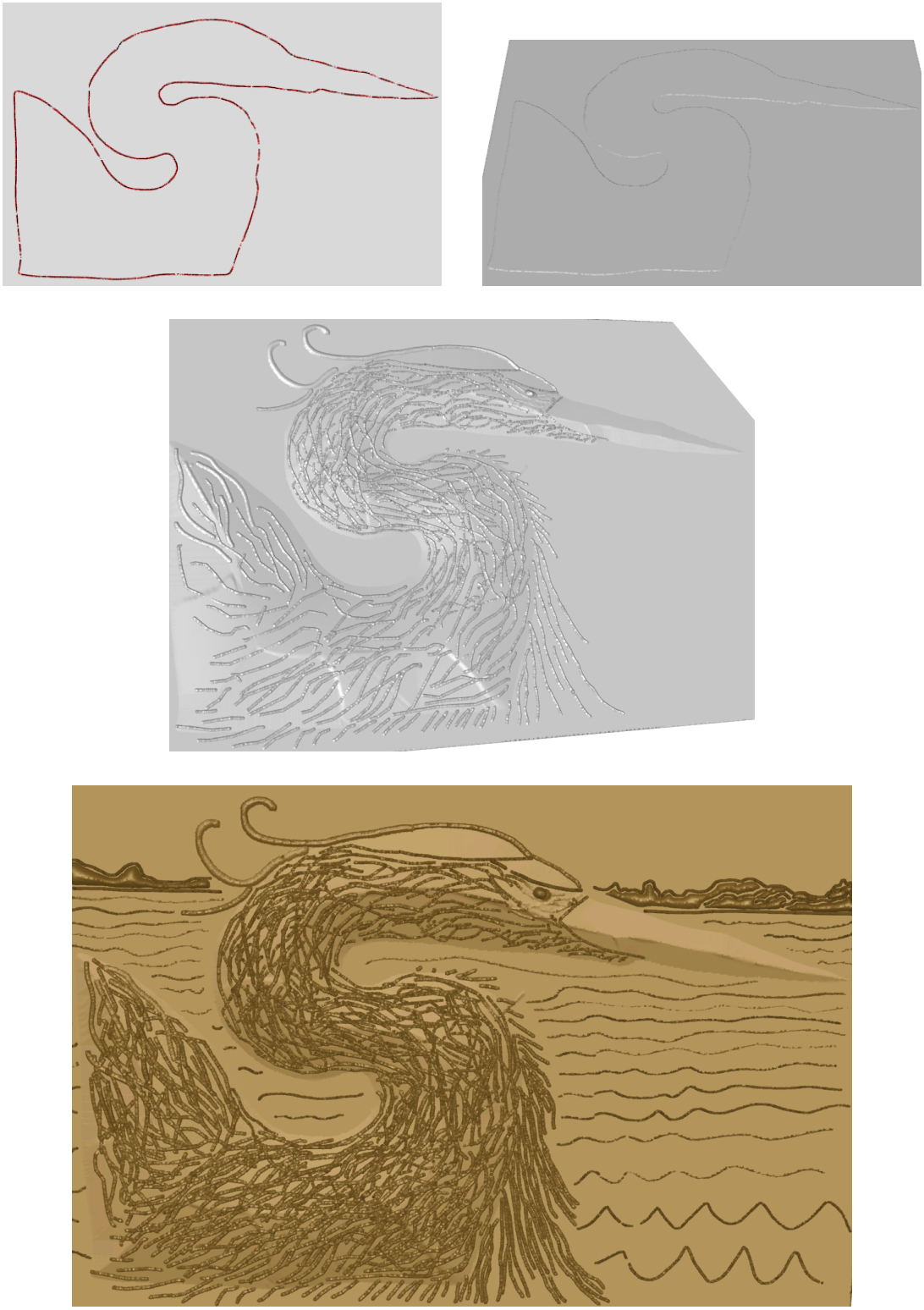


Figure 5.7: A bas relief model of a heron created with an open level set model.

Figure 5.7 presents a design of a heron. It is created with an open level set model that defines a single "flat" surface that can be modified in regions away from the boundary. Similar to unenclosed H-RLE level sets [Houston et al., 2006], our level set models do not need to be defined as closed solid objects. Given our flexible narrow-band representation, localization of PDE processing, and an explicit inside/outside categorization of the voxels, we are able to, for the first time, evolve a level set thin sheet. We can set the bounding box of the object and the Regions of Influence (ROIs) of the editing operators to be smaller than the extent of the model, thus avoiding unwanted numerical problems along the sheet boundary. Just modeling a single sheet surface allows us to make high resolution bas relief type models. The resolution of the initial volume representing a plane is $3081 \times 2057 \times 60$, and the effective resolution of the final model is $3081 \times 2057 \times 69$. The initial model has $44,091,388$ voxels in the narrow-band, and the final edited model has $44,552,586$ narrow-band voxels.

From the data presented in Tables 5.1 through 5.5, it can be seen that we have achieved the goal of creating a system that allows us to interactively edit high resolution level set models. Given that the operator functions and PDE processing takes more computation time than surface display, the timing results presented in these tables show that our system, with a spatial hash table for storing the level set surface and a k-d tree for storing the display points, provides an acceptable interactive frame rate for most of the editing operations. These frame rates compare favorably with previous volumetric PDE-based modeling systems, which either take many seconds to minutes to perform a low resolution surface modification [Du and Qin, 2007], or exhibit interactive rates approximately four times slower than ours [Bærentzen and Christensen, 2002]. The one model change that was not interactive is the offsetting operation performed on the heron body in Figure 5.7. This is a large-scale modifi-

Model	Dimensions	Table Size	Number of Voxels in Narrow-Band	Mean	Std. Dev.
Fig. 5.1	$944 \times 2048 \times 1709$	24,500,224	27,869,503	1.138	1.067
Fig. 5.5	$654 \times 600 \times 1794$	8,212,932	11,505,899	1.401	1.184
Fig. 5.7	$3081 \times 2057 \times 69$	44,363,320	44,552,586	1.003	1.002

Table 5.1: Statistics for the spatial hash function and hash table. Given are the number of entries in the hash table (size), and the mean and the standard deviation of the number of voxels stored in each entry.

cation that affects nearly half of the high resolution surface. The body was created by first defining the ROI with a boundary curve. The ROI area was then lifted with a speed proportional to the distance to the boundary curve. To perform this editing operation required processing almost 27 million voxels; thus the excessive run times.

5.4 Discussion

The goal of our work is to develop techniques that enable interactive editing of high resolution level set models. There are two potential bottlenecks that may interfere with the attainment of this goal. The first is the processing and evolution of the level set PDE and the second is the display of the constantly-changing large-scale model. These processes run sequentially, and we can only edit the model as fast as the slower of these two components.

We present techniques for accelerating the former in Section 5.2 by localizing the editing operations and reducing the size of the PDE domain that must be solved. Fast data access also affects the speed at which the PDE can be solved. Spatial hash tables, as described in Section 5.1.1, furnish acceptable access times, while also providing for the efficient storage of high resolution models and supporting unconstrained out-of-

the-box computing. Table 5.1 contains the average number and standard deviation of voxels per table entry with the given hash table sizes for the example models. The table shows that most of the entries in our hash tables contain 1 to 3 voxels, highlighting the effectiveness of the hash function to distribute data over the whole table and minimize data collisions. Table 5.1 also shows that the spatial hash tables are able to represent these models with an order of magnitude fewer voxels than a 3-D array. For example the full resolution model for Figure 5.1 would contain over 3.3 billion voxels, while our data structure only stores the approximately 28 million voxels of the narrow-band.

During our research we implemented and investigated a number of potential data structures for our level set modeling system, namely DT-grids [Nielsen and Museth, 2006] and run-length encoding (RLE) [Houston et al., 2004]. We found neither of them to provide the rapid random access and modification times required for an interactive editing application. DT-grids were developed for high resolution level set applications that process the complete level set surface, storing the narrow-band data in dense, lexicographically-sorted arrays. Therefore they do not adequately support random access, insertion and deletion of data elements, since they assume that the whole level set surface is processed sequentially for every time step. For example, inserting a new element into the (sorted) DT-grid is an O(N) operation, where N is the number of elements in the data structure. Because of this, during development we abandoned DT-grids as a data structure for our interactive level set modeling system. RLE was then investigated as a potential method for storing our models.

Table 5.2 presents a comparison of execution times required to perform a variety of editing operations on the example models using two different data structures, one based on run-length encoding and the other on spatial hashing of the model's voxel data. In general using spatial hash tables provides a 1.5 to 2 times speed-up over a

Model Detail	**Time (secs) with RLE**	**Time (secs) with Hashing**	**fps with Hashing**
Horse, Fig. 5.1. Dimensions: $944 \times 2048 \times 1709$			
Saddle	0.033	0.016	63
Seat	0.33	0.20	5
Mane/Tail	0.0090	0.0025	400
Tail	0.17	0.10	10
Bridle	0.010	0.0083	120
Stirrups	0.016	0.010	100
Flowers, Fig. 5.5. Dimensions: $654 \times 600 \times 1794$			
Pot handles	0.071	0.040	25
Pot details	0.011	0.0066	152
Plant roots	0.014	0.0059	169
Flowers	0.033	0.020	50
Leaves	0.0083	0.0040	250
Soil	0.033	0.018	56
Heron, Fig. 5.7. Dimensions: $3081 \times 2057 \times 69$			
Heron body	322	177	0.006
Feathers	0.0085	0.0020	500
Waves	0.18	0.083	12
Mountains	0.31	0.22	5

Table 5.2: Average execution times (in seconds) needed to compute an editing operation for one display frame during the creation of a variety of model details. Times are given for an implementation of RLE Sparse Level Sets [Houston et al., 2004] and our Spatial Hash method.

Model	**Std. Dev.**	**Max**	**Min**
Fig. 5.1 Initial	0.0084	0.0556	0.0171
Fig. 5.1 Final	0.0081	0.0533	0.0179
Fig. 5.5 Initial	0.0055	0.0391	0.0226
Fig. 5.5 Final	0.0093	0.0457	0.0160
Fig. 5.7 Initial	8.27E-05	0.0313	0.0312
Fig. 5.7 Final	0.0093	0.0326	0.0305

Table 5.3: Statistics for the VBO k-d trees. Given are the standard deviation, minimum and maximum sizes of the 32 VBOs used to display the example models. The average VBO size (percentage of vertices in a single VBO) is 1/32 (0.031).

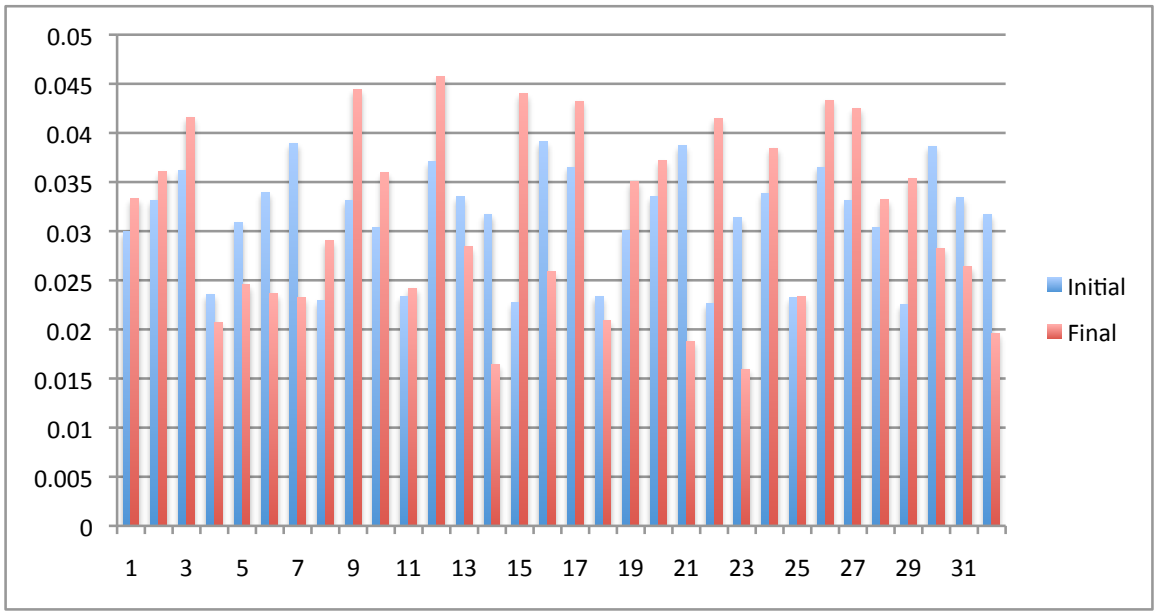


Figure 5.8: Percentage of vertices per VBO for the flower pot model shown in Figure 5.5. Blue bars represent the distribution for the initial "pot only" model and the red bars represent the vertex distribution for the final edited model.

run-length encoded data structure, mainly because random access/insertion is faster with the hash tables. Another major drawback of an RLE implementation is that as the surface changes it requires additional steps to keep the data structure condensed and simplified. As the model is modified, run-lengths need to be added, dropped and merged to keep the data structure compact and precise. Furthermore, we found that the RLE implementation used about 33% more memory than the spatial hash implementation in our studies, given the overhead of the run-length data and the pointers/indices needed to connect the run-lengths.

Figure 5.8 presents the distribution of display vertices (as a percentage) in the 32 VBOs used to display the flower pot model in Figure 5.5. The distribution is produced via the k-d tree subdivision described in Section 5.1.2. The average VBO size (percentage of vertices in a single VBO) is, of course, 1/32 (0.031), and the

Model	#Vertices
Fig. 5.1	3,749,988
Fig. 5.5	1,055,282
Fig. 5.7	6,291,456

Table 5.4: Number of vertices for each model.

	# VBOs										
	1	8		16		32		64		128	
Model	E	E	R	E	R	E	R	E	R	E	R
Fig. 5.1	.0656	.0083	5.86	.0036	7.13	.0022	7.87	.0012	8.74	.00063	9.71
Fig. 5.5	.0265	.0035	2.00	.0015	2.32	.00055	2.62	.00046	2.90	.00036	3.16
Fig. 5.7	.0994	.0121	7.96	.0065	9.96	.0034	12.1	.0016	14.2	0.0011	16.2

Table 5.5: Average frame times (in seconds) needed to remap, transfer graphics data and draw the VBOs after an editing operation (E). Times (in seconds) needed to rebuild (R) the VBO k-d tree. Times are given for rendering with 1, 8, 16, 32, 64 and 128 VBOs.

standard deviation of the initial model distribution is 0.0055 and the final model distribution is 0.0093, with the maximum and minimum sizes being 0.0457 and 0.0160. The distribution demonstrates that the fast, approximate technique used to assign vertices to VBOs creates a relatively even distribution, with most VBO sizes being within 33% of the mean. The imbalance of the VBOs does increase after the model has been modified, but as stated earlier, this imbalance does not significantly affect the time needed to display the model. We found similar results with the horse model. Since the heron model is effectively a height field, we utilize an X-Y only partitioning to produce a nearly uniform distribution of display points over the VBOs. The distribution statistics for the three models is listed in Table 5.3.

Having the display vertices subdivided and distributed amongst multiple VBOs in order to improve graphics performance raises the question of what is the optimal level of subdivision. We performed a simple editing operation, a point-click and pull

operation with a radius of 5 voxels that creates a 20 voxel protrusion, and gathered display time information for several levels of spatial subdivision (and therefore several total numbers of VBOs) to explore this issue. The editing operation was performed on the initial horse, pot and open level set models. The results of this study are presented in Table 5.5, and include average times (in seconds) needed to display the vertices after editing (E) for each frame using a number of VBOs (from 1 to 128) produced via k-d tree partitioning. The VBO data includes the time needed to remap, transfer graphics data and draw the VBO. Also included are the times (in seconds) required to repartition the display vertices of the model (R). While rendering times go down with increasing VBO number, the repartitioning times increase. Table 5.4 shows the number of vertices used to display each model.

These experiments led us to use 5 levels of subdivision with 32 VBOs when creating the example models. By comparing the editing operation times in Table 5.2 with the display times in the 32 VBOs column of Table 5.5 it can be seen that with this number of VBOs more time is needed to compute the editing operation than to display the resulting dynamic model. The one exception is the simple feather detailing operation used in Figure 5.7. Choosing to display the level set model with 32 VBOs gives graphics frame rates of 300 fps and better, and makes the editing functions and PDE processing the computational bottleneck during interactive modeling. Of course higher levels of subdivision produce greater repartitioning (R) times. Since repartitioning is required so infrequently we believe that achieving interactive display performance that is not limited by rendering times (for our models/application) is worth the few extra seconds that may be needed occasionally for VBO restructuring.

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 [Du and Qin, 2007] uses a maximum resolution of $65 \times 65 \times 65$ and an ap-

proximately 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 [Bærentzen and Christensen, 2002] using an octree representation for the level set model can edit volumes with an effective 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.

6. Detail Preserving Surface Editing for Multiresolution Level Set Models

Multiresolution modeling techniques have been developed for computer graphics and geometric modeling. They provide a powerful and expressive modeling paradigm which supports manipulations at varying levels of detail. An underlying level-of-detail representation enables the user to work at any desired resolution, hence providing speed-ups and modeling capabilities which are crucial for interactive editing applications. As defined by Zorin et al. [1997] and Kobbelt et al. [1998], a hierarchical structure is needed to perform multiresolution shape editing, where a model at level N of a hierarchy is produced by combining the model details stored at level N with the lower resolution model defined at level $N-1$. Given this structure, the user may edit the model at any level of the hierarchy, and the details defined at higher resolutions will automatically be added to and maintained on the modified model; thus high resolution details follow the movements of an edited low resolution model.

The editing operators described in Chapter 4 move the surface with an algorithmically generated speed field in directions normal to the surface. Such motion causes regions of high curvature to collapse and merge, and results in the smoothing of the surface in these areas. It is also well known that the computational methods used to advect a level set model smooth out the details of the interface because of numerical dissipation. Therefore it is necessary to develop techniques for maintaining geometric details during level set editing operations. There are methods developed to overcome/correct numerical errors through the use of higher order, hybrid and adaptive techniques. However, higher order methods [Harten et al., 1987, Liu et al., 1994] are computationally complex which makes them undesirable for interactive applications. Hybrid and adaptive methods [Losasso et al., 2004, Enright et al., 2005] are used in conjunction with a semi-Lagrangian scheme and octree subdivisions to

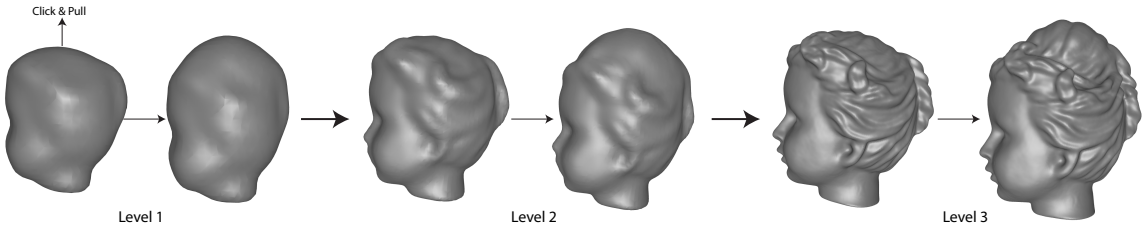


Figure 6.1: A multiresolution model is modified at Level 1. The modifications are incorporated into higher levels of the hierarchy.

provide solutions for capturing passively advected interfaces. These methods are also not applicable to our problem as explained later.

We have developed a hierarchical, multiresolution representation for level set models that allows for rapid decomposition and reconstruction of the complete full-resolution model. Editing in lower resolutions facilitates interactivity, as well as enabling a level-of-detail editing capability. The low-level modifications are later scaled up and incorporated into the higher resolution model as demonstrated in Figure 6.1. There are three key aspects of this process that lead to the loss of surface details. First, creating lower resolution models inherently produces a loss of detail. Second, the lower resolution surface is smoothed out during editing due to the motion in the normal direction and numerical dissipation. Third, incorporating the low-level manipulations into the higher resolution model overwrites the latter and the high resolution details are lost unless explicitly saved and restored.

A major focus of our research has been on how to link high resolution surface features to low resolution surface movements while maintaining detailed structures at all scales. We capture and store surface details at all resolutions in particle sets and later use these particles to add the surface details back to the models. The particle sets are dynamic and stay on the surface at all resolutions while the surfaces are edited. Compared to previous particle level set methods [Enright et al., 2005], our

technique does not depend on particles being advected with the surface. This allows us to use the particles to represent details without the need to evolve the interface at higher resolutions. The particle set also facilitates geometric texture transfers.

This chapter presents a hierarchical, multiresolution representation of level set models to facilitate multi-level surface manipulations, as well as algorithms for transferring low resolution surface movements to high resolution surface features while maintaining detailed structures at all scales.

6.1 Hierarchical Level Set Models

We have developed a hierarchical (H) data structure for representing multiresolution (M) level set models. We also developed methods for creating the H-M data structure from a high resolution level set model, and for processing the H-M data structure to create a high resolution level set model. The algorithms maintain the proper spatial relationship between high resolution details and low resolution shapes. Keeping high resolution details connected to the motion/deformation of an underlying low-resolution shape is the essential feature of multiresolution models. Given this hierarchical structure, the user may edit the model at any level of resolution, and the details defined at higher resolutions will automatically be added to and maintained on the modified model.

In our work a hierarchical level set model is defined by a successive set of sparse grids, where the spatial resolution of grid N is twice (in all three dimensions) the resolution of grid $N-1$. In other words for every grid point in grid $N-1$ there are eight grid points in grid N. The eight N grid points represent the same portion of 3-space represented by the one $N-1$ grid point. This is clearly the definition of an octree data structure. Given this hierarchical, multiresolution data structure which stores the surface at each level, it is straightforward to go through the octree

structure superimposed over the successive grids in order to produce the final, full, high resolution model that it represents.

A hierarchical, multiresolution level set model can be produced from a high resolution level set model by a successive set of smoothing and differencing operations. For example, a curvature-based smoothing deformation may be applied to the full resolution model to remove some details [Museth et al., 2002, Osher and Sethian, 1988, Tasdizen et al., 2003]. The smoothed model is used to extract the detail information that is stored in level N of the H-M data structure. The details are encoded as offset vectors from the smooth surface to the detailed surface and stored within a particle set that is sampled on the smooth surface. These details can later be used to add the high resolution details back onto the smooth model in order to reconstruct the original detailed model. The smoothed model now becomes the model associated with level $N-1$. The same smoothing and detail extraction is applied to the $N-1$ model to produce the grid point values for level $N-1$ of the H-M data structures and producing the $N-2$ model.

The challenge here is to automate this process, specifically controlling the smoothing process to remove a limited amount of detail at each iteration/level. Curvature-based smoothing can be used to remove high resolution details. However, since the amount of smoothing is directly proportional to the surface curvature, it is harder to define a uniform system that will generate consistent smoothing results for a wide variety of models. Downsampling and upsampling are two fundamental and widely used image operations, with applications in image display, compression, and progressive transmission. Downsampling is the reduction in spatial resolution, whereas upsampling increases the spatial resolution. Youssef [1999] examines several classes of filters for down/up-sampling, and finds that binomial filters produce the best results in terms of signal-to-noise ratio (SNR) between the original and the down-then-

upsampled images. Besides their superior SNR performance, binomial filters offer the added advantage of having rational coefficients with powers of two denominators that improve computational efficiency. Binomial filters form a compact rapid finite impulse response (FIR) approximation of the (discretized) Gaussian.

We use a volume filtering and downsampling method that creates consistent smoothing results with high efficiency. The original high resolution narrow-band volume is filtered using a three dimensional binomial filter with kernel size $3 \times 3 \times 3$. The resulting smoothed volume is then downsampled to reduce the spatial dimensions by two along each major axis. We then apply the binomial filter to the lower resolution volume and further downsample to create the hierarchical volumetric structure along with geometric details. The lower resolution volumes created via downsampling are renormalized in order to preserve them as signed distance fields using fast marching methods [Sethian, 1995, Tsitsiklis, 1995]. The downsampling stops upon reaching a predefined number of levels or when the resolution is too low to define the full extent of the narrow-band. Our framework can represent an object with a volume resolution as small as 8^3.

We have created a high resolution sampling of particles residing directly on the level set surface. These particles are placed on the smooth surface after binomial filtering and before downsampling during the hierarchy generation process. Each stores a scalar representing the signed distance to the pre-filtered detailed surface at the current resolution, as well as the surface normal of the smooth surface at that particle's location. The distance is calculated by finding the intersection of the surface normal and the original detailed surface. The signed distance field representing the detailed model facilitates finding the intersection point, as well as determining the inside/outside status of the particle with respect to the detailed model.

The particles are sampled uniformly on the surface with a rate of one particle

per surface crossing voxel. A voxel is called "surface crossing" if the distance value stored in that voxel has a sign (positive or negative) that is opposite of one or more of the distance values stored at the 26 voxels adjacent to it. Each particle carries an offset value and a direction vector that capture the difference between the filtered and unfiltered level set surfaces. The details are extracted by smoothing the level set surface, creating a sampling of particles on the smooth surface, and calculating the straight path that these particles would take in order to reach the detailed surface. This path is then stored within each particle as a combination of a unit vector and a floating point number. The vector represents the direction of the path, and is the surface normal at the particle's initial position on the smoothed surface. The floating point number is the signed Euclidean distance between the two surfaces in the path direction. The sign denotes the relative positioning of the particle with respect to the detailed surface, i.e. inside/outside status. This representation might seem redundant considering one can combine the unit vector and the scalar into a single vector; however, we later show that this representation facilitates the construction of the speed function required to evolve the level set surface when adding geometric details back into the model. The detail particles are also stored in spatial hash tables. Figure 6.2 illustrates how detail particles are created for a single level of the hierarchy.

After the multiresolution level set hierarchy is created, we can reconstruct the volume at level N through upsampling the volume at level $N-1$ and adding back the details stored at level N. The details are added via an iterative level set evolution that uses the offset values and direction vectors stored at the particles to form the evolution's speed function. The particles move with the interface and the speed function is calculated every iteration using offset values stored at the updated particles.

Section 6.3 describes in detail the creation of the particle set, as well as its use during the level set evolution that adds back the high resolution surface details. The

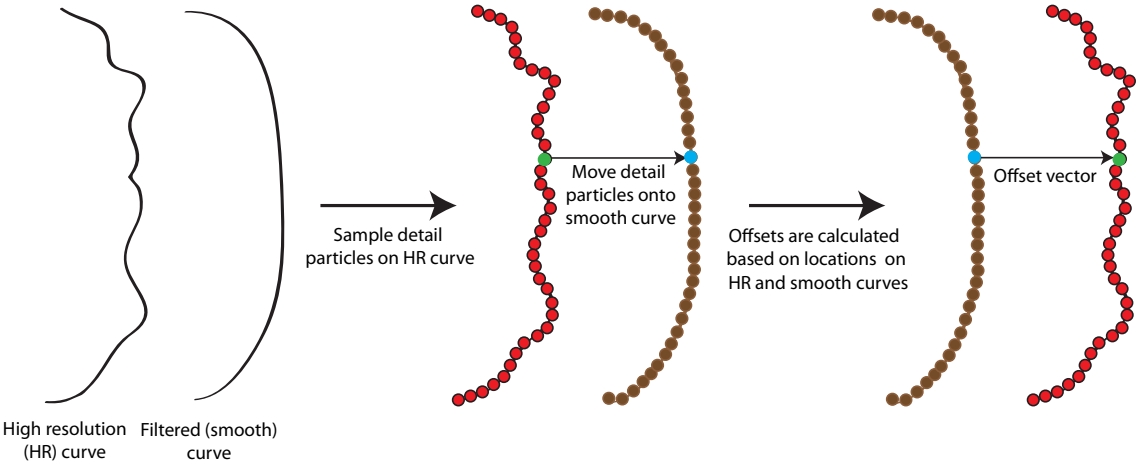


Figure 6.2: An illustration of the detail generation process.

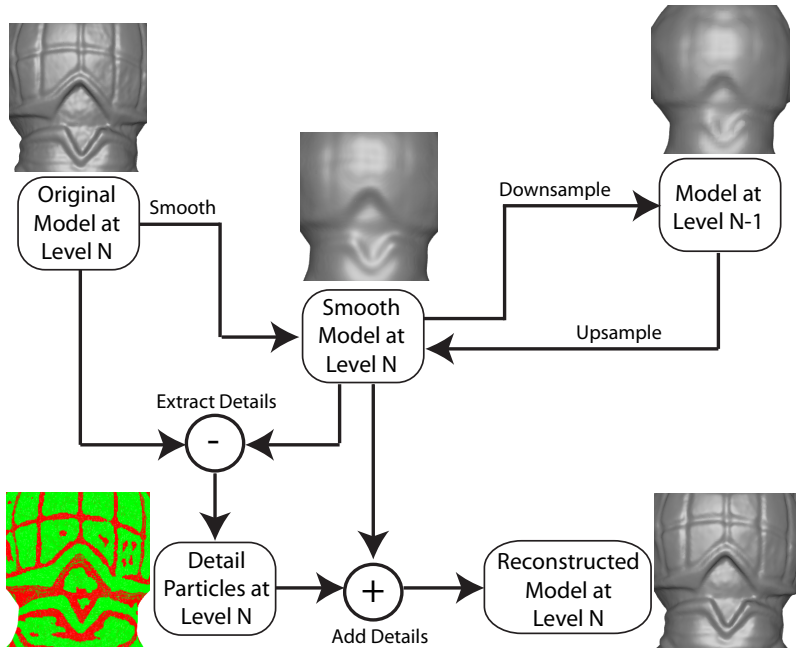


Figure 6.3: Flowchart of the hierarchical multiresolution level set modeling framework.

flowchart of the hierarchical model creation process is presented in Figure 6.3. A closeup of the armadillo model shows the surface at each step of the process. The original model at level N is smoothed to create an intermediate result that is used to extract details. The state of the surface particles prior to adding the details is shown on the lower left. The particles are color coded depending on the sign of the offset values. The smoothed model is then downsampled to create the lower resolution model at level $N-1$, which is saved as a part of the hierarchy and later can be upsampled to be used as the starting/base model in the detail adding process.

6.2 Multiresolution Surface Modeling with Level Sets

Given the evaluation/reconstruction algorithm previously defined in Section 6.1, the user is able to view and edit a model associated with level N of the H-M structure using our interactive editing tools. After the editing is complete the changes to the model are cascaded through the H-M structure in both directions of the hierarchy given that higher/lower-resolution models exist within the hierarchy. Figure 6.4 is a flowchart showing the steps taken throughout the hierarchy following an edit at the middle level.

We extract the narrow-band in a localized region around the modified surface and upsample this subvolume to one higher resolution, i.e. we double the spatial resolution in each dimension, using tri-linear interpolation. Since each modified voxel at level N is represented by 8 voxels at level $N+1$, the voxels within the ROI at level $N+1$ can be easily identified. All grid points within the ROI at level $N+1$ are then replaced with new values from the upsampled volume and the updated surface is seamlessly blended with the unmodified surface at the ROI boundary using curvature-based smoothing [Museth et al., 2002]. The upsampling increases the extend of the narrow-band, i.e. the radius of the narrow-band tube, by a factor of two. Fast marching

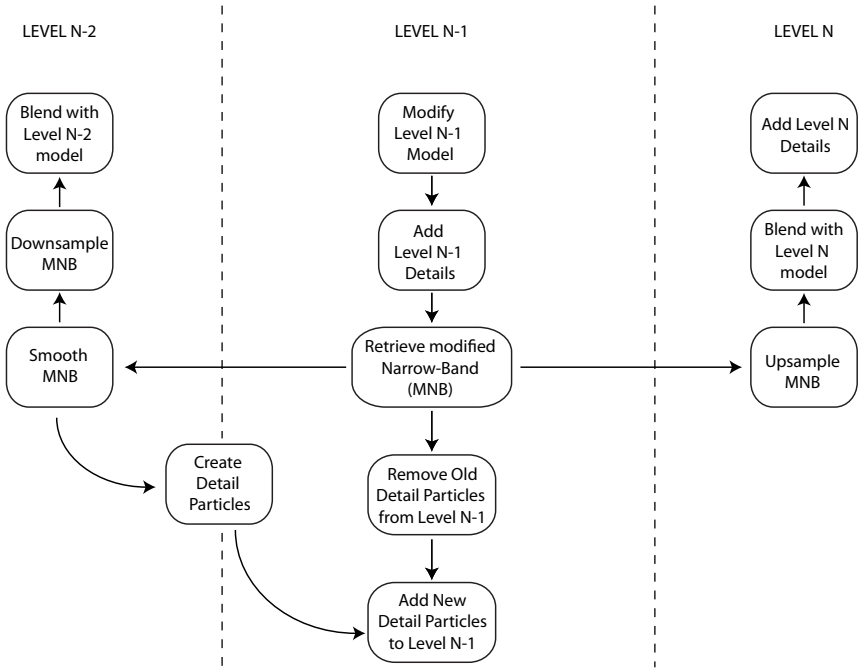


Figure 6.4: Multiresolution surface editing.

methods are used to re-initialize the narrow-band [Sethian, 1995]. Since the high resolution details are filtered out prior to downsampling, these details are also absent in the upsampled volume. We project the detail particles onto the level $N+1$ ROI and use level set methods described in Section 6.3 to reintroduce higher-resolution geometric details on the modified surface.

Going towards lower resolutions in the hierarchy, we recreate the H-M structure (grid points and particles) locally between levels 1 and $N-1$. This is achieved by the filtering and downsampling method explained in Section 6.1. The downsampling reduces the extent of the narrow-band, i.e. the radius of the narrow-band tube, by half. Fast marching methods are used to re-initialize the narrow-band. Every new sub-volume is then blended with the existing volume at one lower hierarchical level and so on until the lowest level is reached. If the higher level edit does not translate

to a sub-volume that can be represented with at least 8^3 voxels in lower resolution levels, the modifications are not incorporated into the lower levels of the model hierarchy. Section 6.5 demonstrates the level-of detail editing achieved through our multiresolution framework.

6.3 Detail Preserving Surface Editing

We capture and store surface details at all resolutions in particle sets and later use these particles to add the surface details back to the models. The particle sets are dynamic and stay on the surface at all resolutions while the surfaces are edited. Initially we implemented the particle level set method as explained in Enright et al. [2002a]. One drawback of this method is that it requires advecting the particles along with the surface at every step of the simulation. This process is slow at higher resolutions especially during substantial modifications to the surface, because large numbers of particles need to be moved several times to reach their destination on the deforming surface. We then developed a new method that can project these particles onto the final surface once the interactive editing is completed. Each method is described in Sections 6.3.1 and 6.3.2 and a comparison in terms of run times is provided in Section 6.6.

6.3.1 The Advection Method

Using the Advection Method, once created the detail particles are kept on the surface during all modifications made to the model. During modification, we can easily determine the new location of the detail particles using the signed-distance value of the level set function and the offset vector associated with the particles. The particles are moved along the offset vector, and an intersection at a zero crossing of the level set surface is calculated using the signed distance values. The particle is

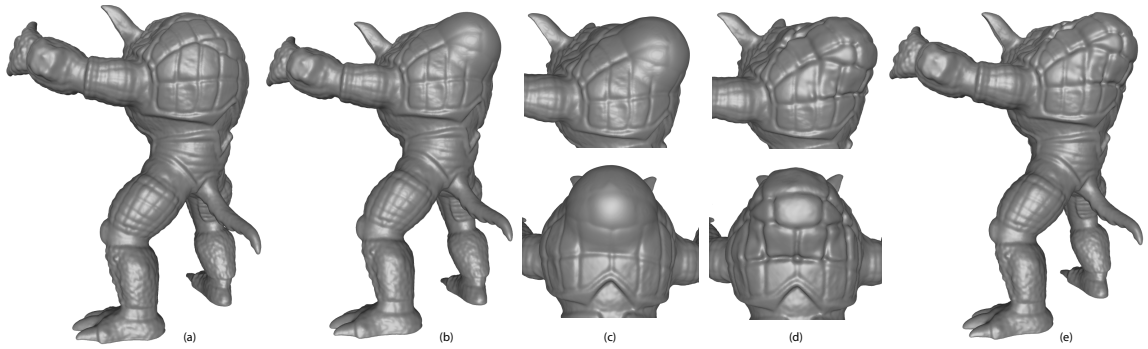


Figure 6.5: (a) The scan converted armadillo model. (b) The modifications to the model smooth out surface details on the back. (c) Two different views of the smooth surface. (d-e) Different views of the modified armadillo model after the surface details are added.

then moved to this intersection point on the surface. The CFL condition that restricts the surface to move less than a voxel every time step ensures that the particles will always reside within the narrow-band of the surface after each iteration of the level set evolution. Once the editing is complete, the direction of the offset vectors are updated with the surface normals of the edited surface. The next stage uses the particles to add the surface details back onto the edited model.

Figure 6.5 shows how details are added using this method after modifying the surface. The surface edit creates a bump on the turtle shell at the back of the model by pulling the surface within a symmetrical ROI around a single point. The surface is stretched and the turtle shell details are smoothed. The detail particles that were created prior to the edit follow the level set surface during the edit, and new particles are added using tri-linear interpolation. These detail particles are then used to add the shell details back onto the model using level set evolution and the speed function explained later in Section 6.3.3. The armadillo model is edited at 512^3 resolution at 1.9 fps (see Table 6.1).

6.3.2 The Spring Method

Even though the method of Section 6.3.1 can effectively keep track of the detail particles as the surface moves, it can only be used during the interactive editing of a single resolution/level of the hierarchical model. In our approach to multiresolution editing we modify the model on a particular resolution level and this modification is incorporated into higher and lower resolution representations within the hierarchy once the edit is completed. Since the models at higher resolutions are not edited incrementally but with a single block update, we cannot move the particles along with the surface at the higher levels of the hierarchy. We need a more direct method that remaps the detail particles onto a higher resolution model after it has been modified by an edit performed at a different level of the modeling hierarchy. The most important aspect of this process is keeping the relative positioning of the particles consistent during the projection. In order to keep the local neighborhoods of particles intact, we introduce a spring system that connects each particle to all of its 1-ring neighbors. The initial rest length of each spring is calculated as the Euclidean distance between particles, which initially places the system in a stable steady state. Once the connections are made, any dislocation of one or more of these particles from the steady state triggers a response from their neighbors which cascades through the entire set of particles until the system reaches a steady state once more.

Two concepts explained previously in Chapter 4 are pertinent for detail preservation. The first concept is the editing boundaries, which define the extent of local surface modifications. The model is modified only on the portion of the surface encircled by a geodesic curve drawn or automatically generated on the surface. Any surface voxel on or outside this boundary is considered frozen and does not change its value during interactive editing. This boundary curve provides us with the first set of constraints in our spring system. The particles located within a voxel intersected

by a boundary curve are called boundary particles. These particles are also considered frozen and are immobile during the entirety of the process that takes the spring system to a steady state.

The second concept is the use of 3D points and curves that are placed on the surface as handles in order to enable user interaction. Particles located within a voxel intersected by a handle curve or point are called handle particles. The handle points and curves, which are defined as a set of handle points, move with the surface during interactive editing. The final position of the handles can be used to move the handle particles onto the edited surface. These particles also stay fixed at their new location during the following step, which repositions particles to reach a new steady state with the spring system.

Particles that are positioned neither on the geodesic boundary nor on the handles can move under the influence of two energies, the energy created by the springs connected to them and the constraint keeping them on the level set surface. The first one is calculated from the equation for the potential energy of a spring. For each particle i located at x, the energy E_i is

$$E_i(x) = \sum_{s \in S_i} k \cdot (d_s - {d_s}^0)^2, \tag{6.1}$$

where S_i is the set of springs (s) that are attached to particle i. $k = 1$ for all springs, d_s is the spring's current length and ${d_s}^0$ is the initial rest length of the spring. The particles move in the direction of the spring energy gradient until they are evenly spread between the boundary and the handle particles, i.e. the system reaches a steady state. We also need these particles to stay on the level set surface. Therefore, following each spring-relaxation step the particles are projected onto the surface in the direction of the surface's distance field gradient.

These two processes are repeated one after another, 1 projection step followed by

20 steps of spring energy minimization, until a state is reached where all particles are in close proximity to the surface, and the spring energy gradient is close to zero. Two conditions, one for proximity to the surface and other for the steady state of the spring system, are tested between each successive sets of projection and energy minimization processes and the detail adding process terminates when both conditions are satisfied. The first condition is satisfied when the maximum closest distance to the surface for all particles is under 0.1 voxels, and the second condition is satisfied when the maximum magnitude for the gradient of Equation 6.1 at all particle positions is under 0.1 voxels. Once the stopping criteria is satisfied, a final projection step places all particles on the surface, and the direction of the offset vectors are updated with the surface normals at the final particle locations. The next stage uses the particles to add the surface details back onto the edited model.

6.3.3 The Speed Function

Once the low resolution edit is smoothly blended with the high resolution surface using curvature-based smoothing near the matching boundaries and the detail particles are placed on the surface, the algorithm moves to the next stage where a speed function is built using the offsets stored at the detail particles. The details are only added to the modified parts of the surface and only the detail particles that cover the modified areas contribute to the speed function. The following speed function is used to add details to a level set surface;

$$F(x) = D(x)\frac{\sum\limits_{p\in P(x)} \omega_p |O_p|}{\max(|O_p|)\sum\limits_{p\in P(x)} \omega_p} \tag{6.2}$$

$$D(x) = \begin{cases} -1 & \dfrac{\sum\limits_{p \in P(x)} \omega_p \vec{V_p}}{\left| \sum\limits_{p \in P(x)} \omega_p \vec{V_p} \right|} \cdot \vec{n}(x) < 0, \\ +1 & \text{otherwise} \end{cases} \tag{6.3}$$

$$\omega_p = e^{-\frac{G(l_p,x)^2}{2{\sigma_p}^2}} \tag{6.4}$$

where x is a point on the surface, $P(x)$ is the set of 8 nearest particles to x, O_p and $\vec{V_p}$ are the scalar (O) and the unit vector ($\vec{V}$) forming the offset associated with particle p, and $\max(|O_p|)$ is the maximum of the $|O|$ values stored at all of the particles within the ROI. $D(x)$ in Equation 6.3 determines the direction of the movement along the surface normal $\vec{n}(x)$ using the dot product of the normal and the weighted sum of the Vs stored at all contributing particles. The weight for each particle in $P(x)$ (ω_p) is calculated using Equation 6.4, based on the geodesic distance between x and the 3D location of each particle in this set (l_p). $G(x, y)$ is a function that returns the geodesic distance between two 3D points on the surface (x and y). The Gaussian function provides a smooth blending of all neighboring offsets. σ_p is set to one half of the geodesic distance to the farthest particle in set $P(x)$. This weight function has a positive but rapidly reducing value as the geodesic distance increases and allows the closest particles to contribute more to the speed function at point x. The speed function is incorporated into the level set PDE and is used to move the level set surface. Equation 6.2 uses the scalar offset values from a small neighborhood of particles to determine how much the level set function stored at each surface crossing voxel will change. The speed function $F(x)$ computes a floating point value between -1.0 and 1.0 that implicitly moves the level set surface along the surface normal direction at x, the location of the voxel.

6.3.4 Sampling

While constrained to the surface, the particles might clump together or stray away from each other as the surface shrinks or stretches. In order to address the latter, we keep a minimum sampling of one particle per surface crossing voxel by adding new particles as the surface stretches. A $3D$ point on the level set surface within a surface crossing voxel can be computed using the signed distance value stored at this voxel and the gradient calculated at this voxel's location. Together, the signed distance value and the normalized gradient creates a vector, which provides a point on the level set surface when added to the said voxel's location. A new particle is placed at this point and its offset is calculated by interpolating the offsets of the 6 nearest particles to the new location. We use tri-linear interpolation for calculating the offset from nearby particles.

We have also experimented with tri-cubic interpolation. Higher order interpolation methods generate better quality surfaces while lower order methods generate faster results. However, for the examples demonstrated in this paper, we did not observe a visible improvement in the edited surfaces when using tri-cubic interpolation and chose to use the lower order yet faster alternative in order to have better performance. The user has the option to choose the interpolation scheme for each editing session. Some particles may clump together in a single surface crossing voxel if the surface is shrinking. Removing or blending some of these clumping particles would result in a loss of information. Since the hash table can store multiple particles per (X, Y, Z) index, all detail particles associated with a single voxel can be maintained during surface editing.

6.3.5 Adding the Right Amount of Details

All particles keep track of how far they have moved during the detail addition process using a *path* variable. The *path* is set to zero for all particles initially and it is updated each time the interface and the particles move by the amount of displacement between the previous and new particle positions. The particle positions are updated by finding the intersection of the level set surface with one of the two vectors $(l_p + \vec{V_p})$ or $(l_p - \vec{V_p})$, where l_p is the current position of the particle and $\vec{V_p}$ is the direction component of the offset associated with this particle. Since the speed at a given point on the surface is based on information from a number of particles, it is possible that any point on the surface may move opposite to the offset direction associated with one or more of the closest particles at this point. The bidirectional computation of the intersection point ensures that the particles stay on the surface in such cases. The amount of displacement is subtracted from the path if the particle moves in $-\vec{V_p}$ direction and added otherwise.

The *path* variable is also used to determine when the detail-addition process should stop, producing the correct amount of details on the surface. The level set surface is moved with the speed function defined in Equation 6.2 until all particles have moved to within 0.1 voxels of the offset value (O) associated with each particle. Particles reaching these limits no longer contribute to the speed function, thus slowing down and/or stopping the surface within their influence. If all particles no longer contribute to the speed function the interface comes to a complete stop and no further details are added. After the process converges on a solution, the user may choose to boost the details manually by adding more steps to the level set evolution.

The tolerance of 0.1 voxels is used in order to avoid particle oscillation. This tolerance coupled with the bidirectional movement of the particles gradually moves the particles toward their destinations on the final model as they add details to the

surface. We observed that it is possible for particles to travel back and forth in close proximity to the surface (oscillate) due to the smoothing nature of the level set evolution. A small amount of added detail may be smoothed during an evolution. This happens rarely at very sharp corners and is a well-known property of level set surfaces. A binary flag coupled with an integer counter is used to keep track of particles crossing the surface at each consecutive iteration of the level set evolution. The binary flag is set each iteration with respect to the particle's direction, i.e. towards or opposite the offset vector, after each iteration. Each time there is a direction change the counter is increased, otherwise the counter is set to 0. If the counter reaches 2, we can conclude that the particle has moved back and forth once. This may very well happen during the natural course of the detail-adding process. For this reason, a particle is not considered oscillating until the counter reaches 5. Oscillating particles are considered to be at their destination when the automated criterion is evaluated for stopping the level set evolution.

A small set of particles contribute to the speed at every surface voxel. This may cause the surface to move slightly more than signified by a single particle. Such a case is easily detected and the surface is stopped in those areas with particles that have reached their final destinations. Since the surface does not move more that one voxel at each iteration, a particle's final location will always be within one voxel of the location specified by its direction vector and offset value.

6.4 Geometric Texture Transfers

Traditional texture mapping defines the surface color of an object with a 2D image. Geometric texture mapping is the 3-dimensional extension of traditional texture mapping, and it procedurally defines geometric surface features of a $3D$ model. With geometric texture transfer, surface characteristics of one 3D model are extracted and

applied onto another model to create a variety of geometric details. The techniques described so far for preserving surface detail can also be used to transfer geometric detail from one part of a level set surface to another or between two different level set surfaces.

Details can be extracted from an ROI on a source level set surface by the smoothing and differencing method that produces detail particles as described in Section 6.1, which are then transferred near a new location on a destination level set surface. A spring system is then used to place the particles on the destination surface. The speed function in Equation 6.2 can then be used to add these details onto the new surface.

The user can specify the details by clicking on a point on the source surface (P_s) and choosing a radius of influence (R_1). A flood fill algorithm goes over the surface and identifies all detail particles within a geodesic distance smaller than R_1 to P_s. The surface enclosed by the geodesic curve automatically generated at distance R_1 becomes the source region of influence (ROI). The next step is to transfer the detail particles to the destination ROI which is defined by the user in the same manner as the source ROI, using a point P_d and a radius R_2. The source and the destination ROIs are not necessarily the same size, i.e. $R_1 \neq R_2$. Since the spatial hash table of our level set data structure can store multiple particles per location, having particle spacing contract is easily handled when R_1 is greater than R_2. When R_1 is less than R_2, there are two traditional methods used to apply a texture over a larger area, stretching or tiling (repeating) the details to fill in the destination ROI.

The spring system repositions the particles evenly between the editing handles and the boundary regardless of the change in scale. However, it converges on a solution more rapidly if the initial placement of particles is within the narrow-band around the destination surface. We can initially move all source particles closer to the destination ROI by using the translation vector $T = l_d - l_s$, where l_s and l_d are the 3D locations of

the points clicked on the source(P_s) and destination(P_d) surfaces respectively. Taking the cross product of the surface normals at P_s and P_d provides us with an axis of rotation ($\vec{V_r}$) and the dot product of these normals can produce the positive angle between them (θ). We rotate all particles by θ around $\vec{V_r}$. Note that the aim is not to move them exactly on the destination surface but to get the particles close to the destination ROI in order to prepare them for the next stage which moves them onto the surface.

Boundary particles are not projected with the method described above. Boundary particles are moved to the destination surface using two different algorithms depending on the technique chosen by the user, stretching or tiling the details. The particle at P_s moves to P_d initially and stays fixed for the remainder of this stage for both techniques.

In order to stretch the details, the boundary curve on the source surface (B_s) needs to be stretched in order to match the boundary curve on the destination surface (B_d). After matching the source and destination boundaries, the spring system can be used to stretch the details over the surface. Both curves are defined as Catmull-Rom splines that are drawn on the level set surface. A point by point matching between the source and the destination boundary is used to move the points forming the source boundary into positions lying on the destination boundary. We create two tangent planes to the level set surface, one at P_s and the other at P_d. The source boundary curve (B_s) is projected onto the tangent plane created at P_s and the destination boundary curve (B_d) is projected onto the tangent plane created at P_d. The points of the projected source and destination curves keep references back to the points from which they originated on B_s and B_d respectively. All points forming the projected B_s are then translated and rotated onto the tangent plane at P_d.

A series of rays connecting P_d to each point on the now projected and transformed

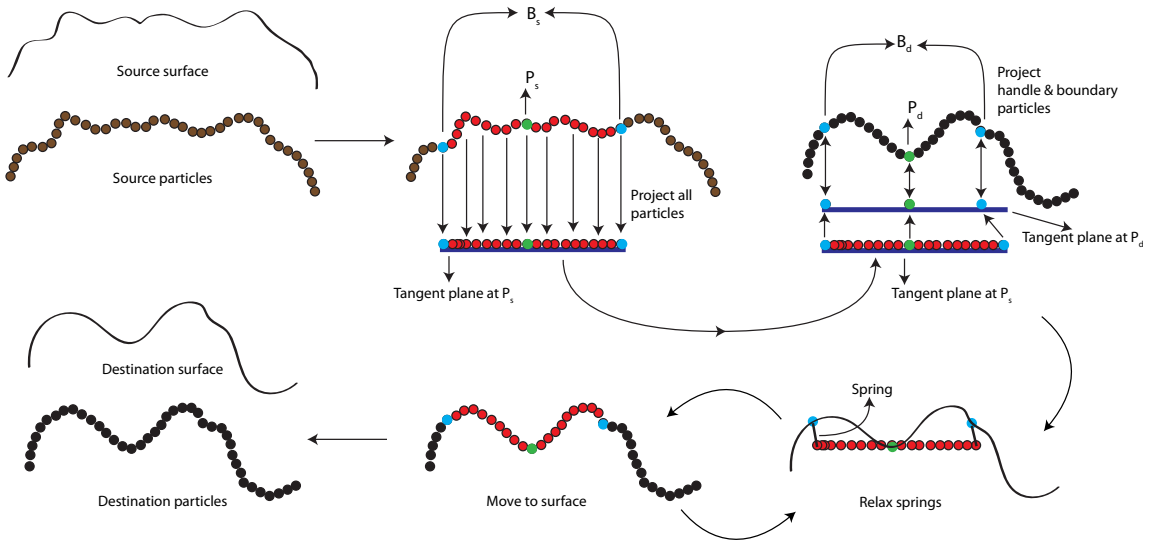


Figure 6.6: A 2D illustration of geometric texture mapping via detail particles. (The tangent planes are drawn below their actual locations).

B_s curve is intersected with the projected B_d. This intersection point lies between two projected points on the tangent plane, and each carry a reference to a point on the original B_d. Using the position of the intersection point between the two projected points and the two points on B_d, a position on the destination curve is interpolated. Using the backwards reference stored at the point on the projected source boundary, the point on B_s is moved to this new position lying on B_d. All particles that are within a voxel of the point on B_s can now be moved to a new position around B_d. The relative positioning of these particles and points on B_s is kept the same. With this, all points on B_s move so that they are on B_d, and all particles that are within a voxel around B_s move so that they are positioned around B_d. These particles also stay fixed during the next step that uses the spring system to reposition the rest of the particles within the ROI. Once the handle and the boundary particles are fixed in their places, the spring system moves all other particles within the ROI distributing them over the surface. New particles are added to create a uniform sampling using

techniques explained earlier in Section 6.3.4. Figure 6.6 illustrates the geometric texture mapping process in 2D. Note that the tangent planes at P_s and P_d are drawn below their actual coordinates to present a clearer image of the projection process. The particles labeled P_s and P_d (drawn in green) are the same particles drawn in the middle on the tangent planes (also drawn in green).

Another solution tiles/repeats the details as many times as necessary to fill in the destination ROI. In this solution a set of center points (C_p) $d = 2 * R_1 - \epsilon$ away from each other are chosen on the destination surface. Initially, P_d is added to C_p and its location is marked along with all surface voxels within a geodesic distance d to P_d. The closest unmarked surface voxel in the destination ROI is then added to C_p and all unmarked surface voxels within d around this voxel are marked. This algorithm keeps adding points to C_p until all surface voxels within the destination ROI are marked. The next step transfers the detail particles from the source surface to each of the sub-ROIs on the destination surface defined by the points in C_p and the radius R_1. The geometric details are incorporated into the sub-ROIs using the methods described in Sections 6.3.2 and 6.3.3 . The offsets of overlapping sub-ROIs are blended using weighted averaging within ϵ distance of sub-boundary curves defined by R_1 to create smooth transitions. Figure 6.11 shows an example of the details being both stretched and tiled over a surface.

6.5 Results

We have used our multiresolution surface modeling framework to create and modify the models shown in Figures 6.1, 6.7, 6.8, 6.9 and 6.10. The frog, girl and lion models were originally scan converted at 512^3 voxels resolution. The disc model is created from an implicit equation and sampled on a 256^3 volume.

The frog model (Figure 6.7) represented with a 256^3 volume dataset is modified

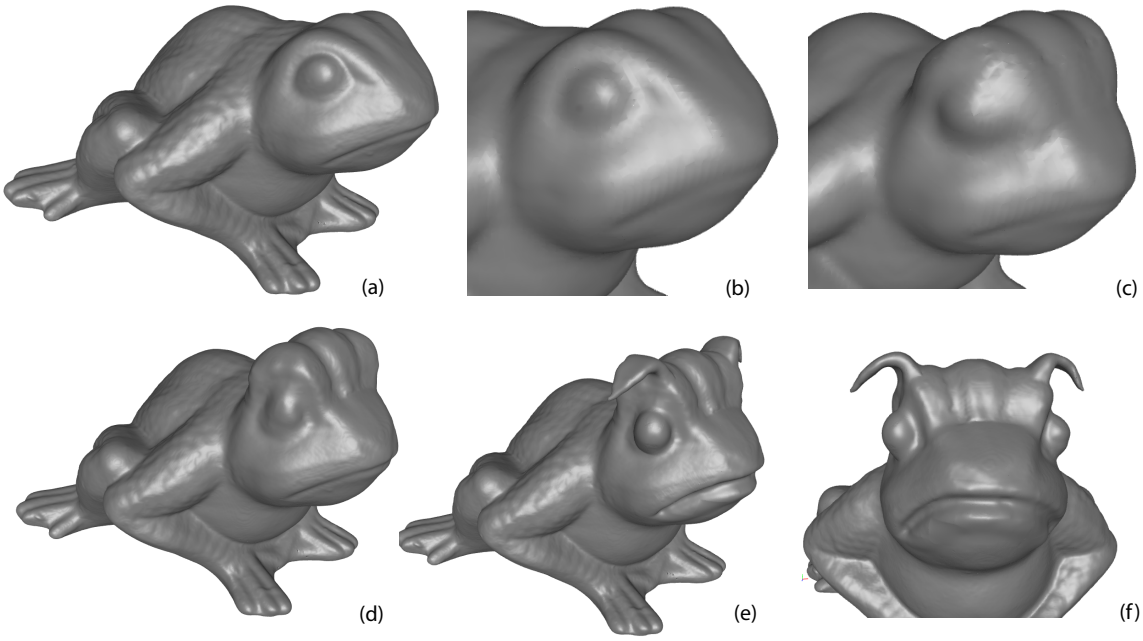


Figure 6.7: (a) Original scan converted model. (b) Low-resolution(LR) model after filtering and downsampling. (c) A low-resolution edit modifies the general shape of the head. (d) The modified high resolution model with details. (e-f) An example of further editing the model at the higher resolution.

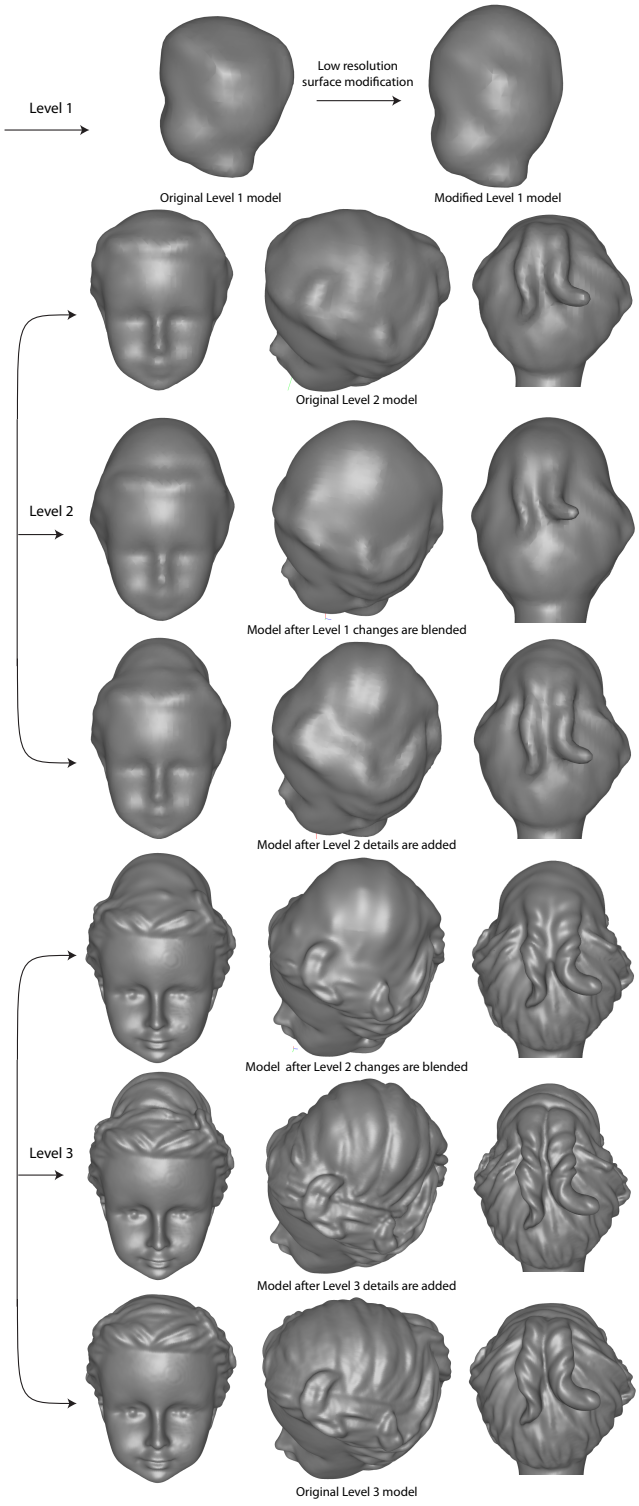


Figure 6.8: The model is modified at Level 1. The modified part of the surface is upsampled and blended in with the Level 2 model. Level 2 details are added and the detailed Level 2 surface is upsampled and blended with the Level 3 model. Finally, Level 3 details are added to create the modified high resolution surface. The original models at all levels are also included in the figure.

by elongating its head. Although it is localized, this operation is still slower on a higher resolution model. Modifying the models at lower resolutions facilitates not only level-of-detail editing but also interactivity. Once the coarse adjustments are done, the higher resolution (512^3) details are added in at a higher level of the modeling hierarchy as shown in Figure 6.7(d) and (e).

Figure 6.8 shows several stages of a multiresolution edit applied to the girl model shown also in Figure 6.1. The model is edited at the first level of the hierarchy and the changes are upsampled and blended with the second and third levels. Details are added to the model at both its second and third levels. Three different stages of each detail-addition process are provided along with the original unmodified versions of each level for better comparison.

The lion head is edited at a resolution of 256^3 to remove the capital on top of its head. The modified part is then upsampled and locally blended into the 512^3 model. Figure 6.9d shows the smooth area on top of the head. The details are extracted from the back of the same model as shown in Figure 6.9e. The detail particles are color-coded, green representing positive and red representing the negative offsets. A positive offset means that the surface will move in the normal direction during detail addition and negative offsets move into the surface. The final lion model with hair details on top of the head is shown in the final frame.

One advantage of using level sets is that we do not need to assume the source and destination surfaces are similar in topology. The disc model in Figure 6.10 is created from the implicit equation for a superellipsoid in Equation 6.5 with $a = b = 128$, $c = 50$ and $n_1 = n_2 = 1.0$. The texture is also created implicitly from tiling a set of superellipses defined is Equation 6.6 with $a = b = 10$ and $n = 4.0$, side by side on a 2D grid in a repeating pattern.

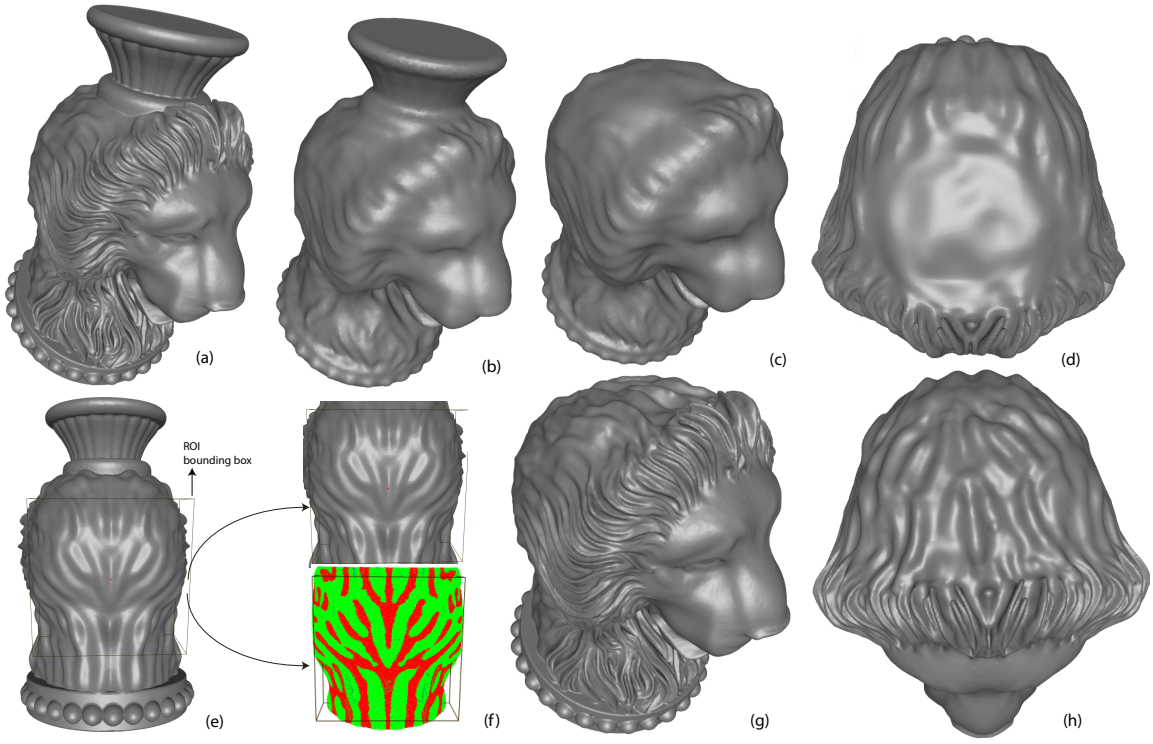


Figure 6.9: (a) Original scan converted model. (b) Low-resolution(LR) model after filtering and downsampling. (c) An edit on the LR model removes the top part and smoothes the head. (d) The LR modifications are upsampled and blended into the high resolution(HR) model. (e) Back view of the original scan converted model. The details are extracted from within the highlighted ROI. (f) Top: The detailed surface Bottom: The detail particles (g-h) The details that are extracted from the back of the original HR model are added on top of the edited model.

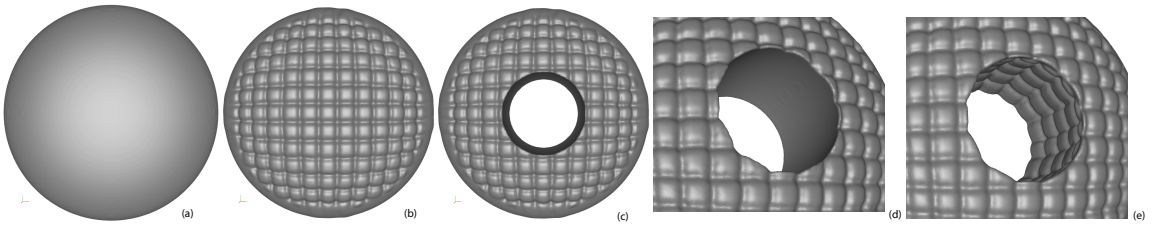


Figure 6.10: (a)The original genus-0 disc model. (b) The surface is modified via geometric texture mapping using a checkerboard pattern. (c) A hole is cut in the center, creating a genus-1 model. (d) A close-up of the smooth surface before details are added. (e) A close-up of the final model after the details are added on the modified surface.

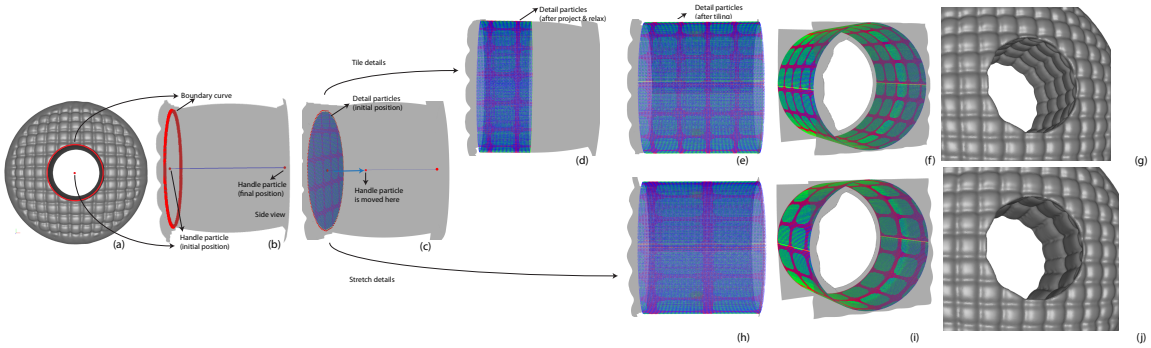


Figure 6.11: (a) A hole is cut at at the center of the disc model. The boundary particles are drawn in red. (b) A closer view of the center hole (model rotated 90° to provide a side view.) (c) Initial position of the detail particles. (d) The particles are moved partway through the hole by projection and relaxation of springs to avoid stretching. (e) The details are repeated over the rest of the surface. (f) Alternate view of the detail particles shown in (e). (g) Final surface with repeated details. (h) The details are stretched over the surface. (i) Alternate view of the detail particles shown in (h). (j) Final surface with stretched details.

$$\phi(x,y,z) = \left(\left(\frac{x}{a}\right)^{2/n_2} + \left(\frac{y}{b}\right)^{2/n_2}\right)^{n_2/n_1} + \left(\frac{z}{c}\right)^{2/n_1} - 1.0 \tag{6.5}$$

$$\phi(x,y) = \left(\frac{x}{a}\right)^{n} + \left(\frac{y}{b}\right)^{n} - 1.0 \tag{6.6}$$

A set of particles is then sampled over this 2D grid, one particle per grid point, each taking on the value of the implicit function sampled on the associated grid point as an offset. The particles all have the same direction vector, which is also the normal for the 2D plane on which they are sampled, $(0.0, 0.0, 1.0)$ for this example. The next step projects and moves the particles over the disc surface and adds these details on the model as explained in Section 6.3. A hole is cut in the center using volumetric CSG [Museth et al., 2002]. The detail particles within the ROI are stored prior to the edit and projected over the newly created smooth surface in the center following

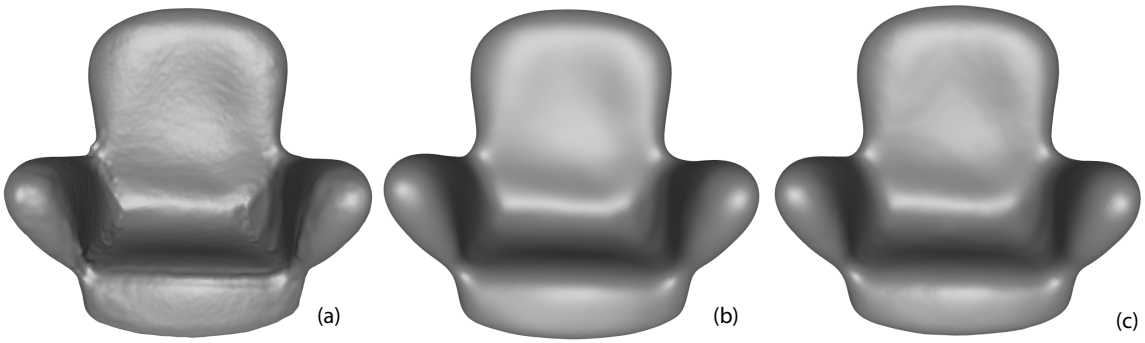


Figure 6.12: (a) The original model is scan converted from a triangle mesh, producing a noisy level set model. (b) Model after 100 steps of curvature-based smoothing. (c) Model after a single application of the binomial filter.

the CSG operation. Since the final edited ROI is significantly larger than the initial ROI, the details only cover a small part of the final surface unless stretched or tiled. We show the resulting surface from each method in Figure 6.11.

6.6 Discussion

A hierarchical, multiresolution level set model can be produced from a high resolution level set model by a successive set of smoothing and differencing operations. Curvature-based smoothing can be used to remove high resolution details. However, since the amount of smoothing is directly proportional to the surface curvature, it is difficult to define a uniform system that will generate consistent smoothing results for a wide variety of models. Additionally, numerical methods used to solve the level set equation require small time steps in order to create stable solutions. This prevents more than a small amount of detail to be smoothed every time step and requires the level set evolution to be run several iterations before removing surface details between levels of the hierarchy. Instead, we use a volume filtering method as explained in Section 6.1 that creates consistent smoothing results with high efficiency.

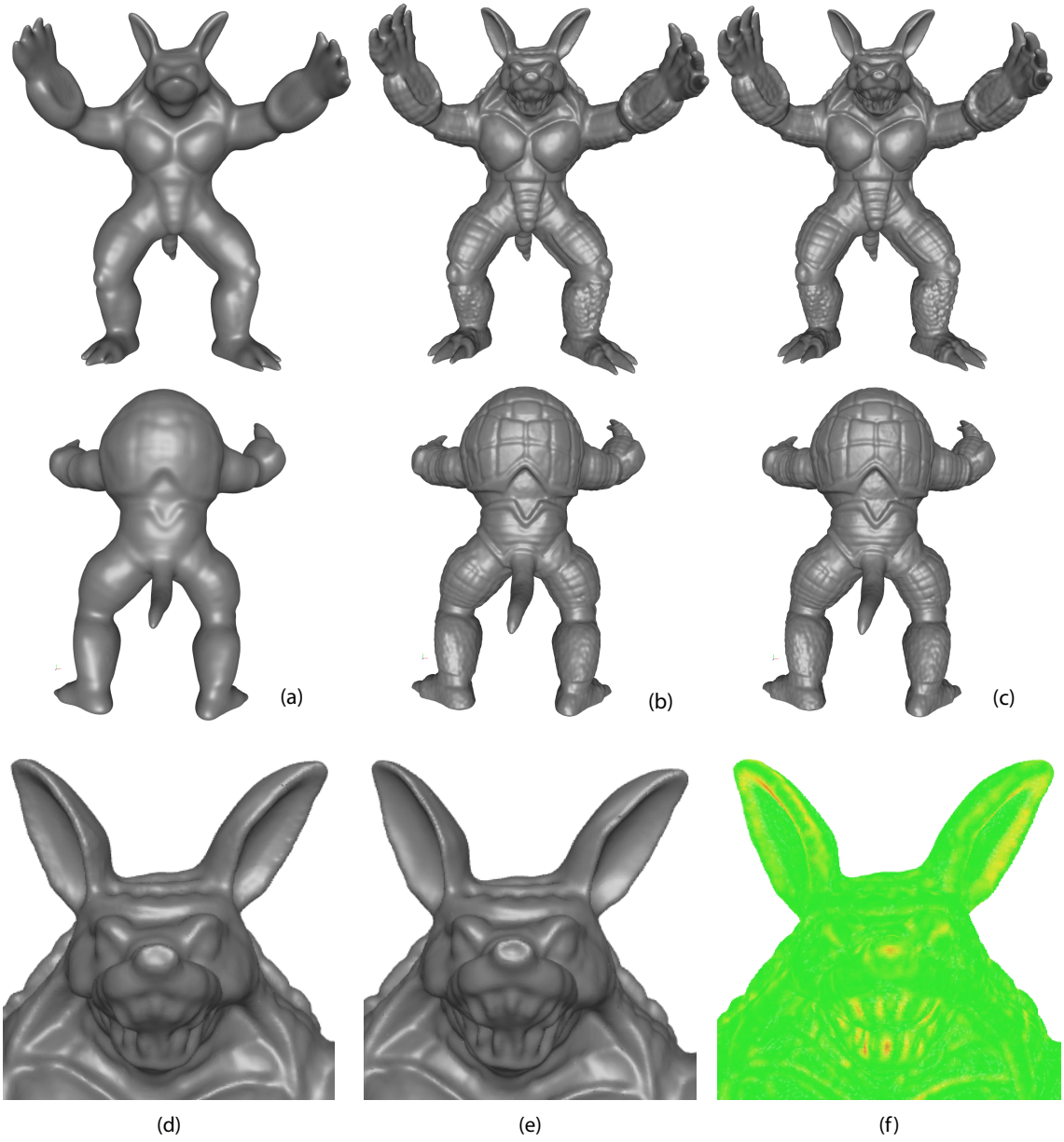


Figure 6.13: (a) Filtered model at level N (b) Reconstructed model at level N, created by adding level N details to the filtered volume shown in (a) . (c) Original high resolution model at level N. (d) A closer view of the head belonging to the reconstructed model. (e) A closer view of the head belonging to the original model. (f) Distance between the reconstructed and the original model is color coded, red representing the maximum and green representing the minimum distances.

Figure 6.12 shows a comparison of two surfaces obtained through curvature-based smoothing and binomial filtering applied to a level set model of an armchair that is scan converted from a noisy mesh model. In an attempt to remove the noise, we initially used curvature-based smoothing. The model after 100 iterations of the level set evolution is shown in Figure 6.12(b). Figure 6.12(c) is created via a single application of the $3 \times 3 \times 3$ binomial filter. The amount of smoothing obtained via 100 steps of curvature-based smoothing is comparable to a single application of binomial filtering. Even though both methods can be utilized to create a hierarchical model, the binomial filter removes surface artifacts, and consequently geometric details, more efficiently and consistently than curvature-based smoothing.

We also evaluated the accuracy of the detail preserving techniques described in Section 6.3. Figure 6.13 shows an example of applying filtering and reconstruction to the armadillo model. The high resolution surface details are removed using the binomial filter and added back using the methods explained in Sections 6.3.2 through 6.3.5. We can measure the reconstruction error by calculating the Euclidean distance between the original and the reconstructed surfaces at the reconstructed surface's volume grid points. The distance at a grid point is computed as the difference between the distance field value of the original and the reconstructed surfaces at that point. The minimum, maximum and average distances for the armadillo reconstruction are 4.09×10^{-8}, 1.48 and 0.07 voxels respectively and color coded in Figure 6.13(f), green representing minimum and red representing the maximum distances. 99% of the reconstructed surface was less than 0.5 voxels away from the original surface and only 0.025% of the reconstructed surface was more than 1 voxel away from the original surface. Figure 6.14 shows the reconstruction error in terms of percentage of overall voxels (y-axis) with the margin of error shown along the x-axis. The chart is only drawn up to 0.5 voxels on the x-axis. Less than 1% of the surface voxels have an error

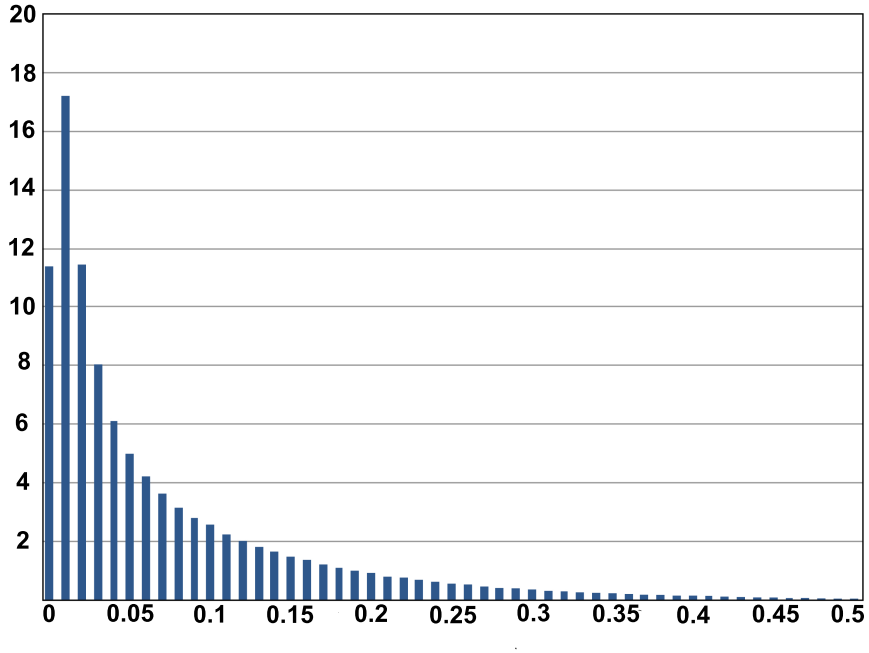


Figure 6.14: The reconstruction error is measured as the distance in voxels between the reconstructed and the original surface voxels. The error is shown up to 0.5 voxels. Less than 1% of the surface voxels have an error of more than 0.5 voxels.

of more than 0.5 voxels.

Multiresolution techniques enable interactive frame rates during large modifications made to high resolution models, as well as level-of-detail surface manipulations. The armadillo model shown in Figure 6.5 is edited twice, once using only the high resolution model (Figure 6.5(a)) and the advection method and once using the multiresolution editing system and the spring method. Table 6.1 shows run times for each step of both approaches on a 3.2 GHz Intel Xeon CPU.

In the first approach, the model was edited at 512^3, advecting the particles along with the surface. The model can be edited at this resolution, however, at a very low frame rate (1.9 fps). In order to provide a comparison, Table 6.1 also shows the frame rate of editing this model without advecting the detail particles with the surface (5

Method	Steps	Run times	Total overhead
Advection Method	Editing 512^3 model	5 fps	12 sec
	Editing & Advecting particles	1.9 fps	
	Adding Details	12 sec	
Spring Method	Editing 256^3 model	37 fps	51.1 sec
	Upsampling	6.8 sec	
	Projecting particles	31.6 sec	
	Resampling particles	0.7 sec	
	Adding Details	12 sec	

Table 6.1: A comparison of the advection and the spring methods.

fps).

In the second approach, the model is edited at 256^3 resolution. The particles are not advected with the surface, but are projected after the edit is done. The ROI is then upsampled and blended in with the 512^3 model. The details at 512^3 are projected onto the modified (smooth) surface and finally added back to the create the detailed surface. Even though the extra calculations to upsample the narrow-band and project the particles onto the smooth surface using the spring method add 40 seconds to the overall time to create the final result, the actual user interaction of the editing can be done approximately 20 times faster using the multiresolution framework. The final step that adds the details using the speed function explained in Section 6.3.3 is the same for both approaches and takes 12 seconds for this particular example.

7. Conclusions

There are several applications in computer graphics and medical science that use volumetric models. Usually, these models are converted into explicit surface representations before they can be utilized by such applications. The conversion and reconstruction algorithms are cumbersome, and the raw data may have a significant amount of noise/errors, requiring user manipulation and clean up prior to further processing and analysis. Furthermore, drastic deformations of complex explicit models produce a series of problems such as cracks and rough patches where surfaces meet. They require re-meshing in the areas that are thinned or expanded too much. The topological errors caused by self intersections are nontrivial to correct. To the best of our knowledge there is currently a paucity of adequate volumetric editing tools capable of high resolution and high level surface manipulations. To address these shortcomings we have developed techniques and algorithms for interactive freeform editing of large-scale, multiresolution level set models. We believe that creating techniques for directly editing volumetric, implicit models is the most logical and advantageous approach to modifying these types of models, rather than relying on conversion techniques and explicit surface editing capabilities.

Level set models combine a low-level volumetric representation, the mathematics of deformable implicit surfaces, and robust numerical techniques to produce a powerful approach to geometric modeling. They are guaranteed to define simple (non-self-intersecting) and closed surfaces, and they easily change topological genus.These models provide the advantages of volumetric models, e.g. supporting straightforward solid modeling operations and calculations, while simultaneously offering a surface modeling paradigm. The benefits offered by these features provide the motivation for utilizing level set models to process and manipulate volumetric, implicit surfaces,

and make them unique for applications utilizing complex surfaces with dynamically changing topology, such as "amorphous" characters interacting with other solid or soft objects, cracking or exploding surfaces, fluid and smoke simulations, as well as representing surfaces acquired from medical scan data. Even though they have found some use in major movie productions and some medical applications such as volume segmentation, level set models are not highly utilized in either the special effects industry or medical science. The space complexity of the volumetric representation and the time complexity of the algorithms needed for modifying implicit surfaces prevent level set models from being utilized in interactive modeling systems. A current state-of-the-art system should support models containing one billion voxels and provide 25-30 frames-per-second (fps) evaluation rates at these resolutions. Even the most advanced data structures and algorithms developed for level set models are not able to meet both of these requirements at the same time. Our work closes the gap between level set methods and interactive modeling applications by providing new techniques and algorithms that make it possible to incorporate these models in state-of-the-art modeling frameworks.

We have created a comprehensive set of computational tools to interactively modify large scale level set surfaces within a multiresolution framework. These tools provide benefits for a number of fields including geometric modeling and CAD, and advance the field of level set modeling in general. The advances presented in this thesis will find immediate use in two disparate application areas, special effects/animation and medical/biological imaging. In special effects, our techniques can facilitate controlling and shaping morphing sequences, which are used to perform animations of amorphous characters. These techniques can also be utilized in medical imaging for guiding and directing the automated volume segmentation process.

A set of interactive, free-form editing operators for direct manipulation of level set

models that supports the creation and removal of surface detail have been devised. The mathematics, i.e. level set speed functions, and algorithms needed to implement numerous level set modeling capabilities have been developed. The first component of these capabilities allows the user to identify the Region-of-Influence (ROI) on the surface to be modified, and specify geometric handles, i.e. a point or a curve within the ROI, that are used to control the free-form surface edits. The second component incorporates this user input into the level set PDE, which is then used to evolve the surface to create desired surface modifications. The editing operators include pulling the level set surface by a handle with the surface changes occurring symmetrically around the handle or within the ROI, surface offsetting and carving, deformations towards a profile curve and localized smoothing. Additional sketch-based level set editing operators have been developed within the system.

The editing operators are implemented with specialized speed functions, which are incorporated into the level set partial differential equation (PDE). The PDE is then evolved to produce the desired model modification. 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 and provide interactive performance. A variety of level set models are presented to demonstrate the effectiveness of the editing operators.

We also described an approach for localized editing of Catmull-Rom (C-R) splines. We prefer to use C-R splines for their ability to interpolate every control point, which is important for accurately translating user input into a mathematical representation in our sketch-based surface editing system. Localized editing gives the user more control over the scale of editing to be performed, and the range of influence of a single editing operation. An active window is placed around the selected control

point to limit the modifications to a sub-region of the curve. The user can change the size of the window and the control point resolution within the window any time during editing. The offset of the control points within the active window can be described through a set of schemes that interpolates the displacement of a selected control point. We discussed two alternative ideas, interpolating the displacement directly, and interpolating the displacement vector with the normal to the curve at the boundaries of the active window. These schemes provide a versatile, expressive and powerful localized curve editing capability for Catmull-Rom splines.

We have described data structures that enable interactive editing of large-scale level set surface models. The new approach utilizes spatial hashing to represent a narrow-band of voxels around the level set interface, as well as a k-d tree to hold the model's display points that lie on the surface itself. This sparse representation of voxels and surface points lets us create and modify high resolution level set models with modest memory requirements, while supporting fast data access/modifications and interactive graphics updates. The data structures also support out-of-the-box editing, i.e. no bounding box limits the surface editing region. Through a number of experiments we have shown that the data structures have the properties necessary to meet our performance requirements. They allow us to interactively edit (at frame rates typically over 25 fps) high resolution level set models (with voxel counts equivalent to a 1500^3 volume dataset). The spatial hash function of Teschner et al. [2003] has been shown to satisfactorily distribute surface locations in the hash table, thus minimizing collisions and maximizing access/modification times. Storing display points in a k-d tree supports the localization of graphics processing, which minimizes the amount of data that needs to be transferred from the application to the GPU during editing of small regions of a high resolution model. Together, these data structures provide a new capability for the interactive modification of large-scale level set models.

Level set models can lose details during surface modifications in under-resolved regions, as well as because interface movements in the normal direction inherently smooth out high-frequency surface structures. We have developed a framework that identifies surface details prior to editing and introduces them back afterwards. Additionally, we have developed techniques for creating hierarchical level set models that allow a user to manipulate/edit a level set surface at different geometric scales and levels of detail. Combining these two features provides a detail-preserving level set editing capability that may be used for multi-resolution modeling and texture transfer.

The contributions of this work include:

1. Techniques and algorithms that implement novel level set modeling capabilities
2. Data structures for representing large scale level set models while providing fast random data access to facilitate interactive editing
3. Detail preserving techniques for level set surface editing
4. Multiresolution techniques for level-of-detail editing of level set surfaces
5. Geometric texture transfer using level set surfaces
6. Schemes that provide a versatile, expressive and powerful localized curve editing capability for Catmull-Rom splines

This research will enable interactive control and guidance of dynamic level set processes such as volume segmentation and morphing. Future work may also create techniques to facilitate mid-sequence manipulations in level set simulations such as fluid flow. An important feature of modifying such simulations is the need to maintain the continuity of the models between simulation time steps.

8. Future Work

The surface modifications described in this thesis are different from the deformations made to analytical implicit surfaces using a skeleton structure. These types of deformations, e.g. bending, twisting and tapering, can also be performed using level set models. A skeleton may also be used to animate such models. Future work could develop techniques to create efficient skeleton structures that deform and animate level set models.

The detail preserving scheme presented in this work represents high resolution surface details as height fields. The type of surface detail a height field can represent is limited. However, given the type of motion the level set interface undergoes during surface editing as explained in this thesis, i.e. using motion in the normal direction, details that are more complicated than a height field cannot be added. If more complicated details can be extracted from the higher resolution model, the advection equation should be used to add these details back onto the level set surface.

Multithreading may be utilized to more efficiently implement the techniques and algorithms described in this dissertation. However, it is not trivial to implement a distributed application using dynamically changing data. If the surface edit happens in the boundary of several adjacent regions on the surface, i.e. a topology change occurs, some or all the regions have to change the underlying data that is influenced or influence other regions. This would create a deadlock situation caused by the cyclic dependencies among individual neighboring forked processes. This is in fact quite common in distributed algorithms especially when they share/modify a common data structure, and can be resolved by using a parallel model such as *exclusive read exclusive write* (EREW). To prevent such cyclic dependencies, a master processor would need to maintain the data and coordinate the individual processes. This model

adds to the overhead and the protocol is not trivial to implement. Instead, future work may include creating GPU-friendly algorithms in order to achieve better modeling performance.

We realize that these techniques should be incorporated into current state-of-the-art 3D modeling systems such as Maya or Houdini to better benefit the special effects and animation communities. A possible way that they could be included into a commercial system is by including these techniques into OpenVDB[Miller et al., 2012], an open source C++ library comprised of a hierarchical data structure and a suite of tools for the efficient manipulation of sparse and time-varying volumetric data.

Bibliography

D. Adalsteinsson and J. A. Sethian. A fast level set method for propagating interfaces. *Journal of Computational Physics*, 118(2):269–277, 1995.

A. Alexe, V. Gaildrat, and L. Barthe. Interactive modeling from sketches using spherical implicit functions. In *Proc. AFRIGRAPH '04*, pages 25–34, 2004.

V. Andersen, M. Desbrun, J. A. Bærentzen, and H. Aanæs. Height and tilt geometric texture. In *Proc. International Symposium on Advances in Visual Computing: Part I*, pages 656–667, 2009.

L.-E. Andersson and N. F. Stewart. *Introduction to the Mathematics of Subdivision Surfaces*. SIAM, 2010.

A. Angelidis and M.-P. Cani. Adaptive implicit modeling using subdivision curves and surfaces as skeletons. In *Proc. ACM Symposium on Solid Modeling and Applications*, pages 45–52, 2002.

A. Angelidis, P. Jepp, and M.-P. Cani. Implicit modeling with skeleton curves: Controlled blending in contact situations. In *Proc. International Conference on Shape Modeling and Applications*, pages 137–144, 2002.

A. Angelidis, G. Wyvill, and M.-P. Cani. Sweepers: Swept user-defined tools for modeling by deformation. In *Proc. International Conference on Shape Modeling and Applications*, pages 63–73, 2004.

A. Angelidis, M.-P. Cani, G. Wyvill, and S. King. Swirling-sweepers: constant-volume modeling. *Graphical Models*, 68(4):324–332, 2006.

H. Arata, Y. Takai, N. Takai, and T. Yamamoto. Free-form shape modeling by 3D cellular automata. In *Proc. International Conference on Shape Modeling and Applications*, pages 242–247, 1999.

B. Araujo and J. Jorge. Blobmaker: Free-form modelling with variational implicit surfaces. In *Proc. Encontro Portugues de Computacao Graca*, pages 17–26, 2003.

J. Bærentzen and N. Christensen. Volume sculpting using the level-set method. In *Proc. International Conference on Shape Modeling and Applications*, pages 175–182, 2002.

A. Barr. Global and local deformations of solid primitives. In *Proc. ACM SIGGRAPH*, pages 21–30, 1984.

A. Barr. Superquadrics and angle-preserving transformations. *IEEE Computer Graphics and Applications*, 1(1):11–23, 1981.

P. Bhat, S. Ingram, and G. Turk. Geometric texture synthesis by example. In *Proc. Eurographics/ACM ACM SIGGRAPH Symposium on Geometry Processing*, pages 41–44, 2004.

R. Blanch, E. Ferley, M.-P. Cani, and J.-D. Gascuel. Non-realistic haptic feedback for virtual sculpture. Technical Report RR-5090, INRIA, 2004.

J. Bloomenthal. An implicit surface polygonizer. In *Graphics Gems IV*, pages 324–349. Academic Press, 1994.

J. Bloomenthal and B. Wyvill. Interactive techniques for implicit modeling. *ACM SIGGRAPH Computer Graphics*, 24(2):109–116, 1990.

J. Bloomenthal and B. Wyvill, editors. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc., 1997.

M. Botsch, M. Spernat, and L. Kobbelt. Phong splatting. In *Proc. Symposium on Point-Based Graphics*, pages 25–32, 2004.

M. Botsch, A. Hornung, M. Zwicker, and L. Kobbelt. High-quality surface splatting on today's GPUs. In *Proceedings Eurographics/IEEE VGTC Symposium on Point-Based Graphics*, pages 17–141, 2005.

D. E. Breen and R. T. Whitaker. A level set approach for the metamorphosis of solid models. *IEEE Transactions on Visualization and Computer Graphics*, 7(2): 173–192, 2001.

D. E. Breen, S. Mauch, R. T. Whitaker, and J. Mao. 3D metamorphosis between different types of geometric models. *Computer Graphics Forum*, 20(3):36–48, 2001.

D. E. Breen, R. T. Whitaker, K. Museth, and L. Zhukov. Level set segmentation of biological volume datasets. In *Handbook of Medical Image Analysis, Volume I: Segmentation Part A*, pages 415–478. Kluwer, 2005.

A. Brodersen, K. Museth, S. Porumbescu, and B. Budge. Geometric texturing using level sets. *IEEE Transactions on Visualization and Computer Graphics*, 14(2): 277–288, 2008.

M.-P. Cani and M. Desbrun. Animation of deformable models using implicit surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 3(1):39–50, 1997.

J. E. Cates, A. E. Lefohn, and R. T. Whitaker. Gist: An interactive, GPU-based level set segmentation tool for 3D medical images. *Journal on Medical Image Analysis*, 8(3):217–231, 2004.

E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10(6):350–355, 1978.

E. Catmull and R. Rom. A class of local interpolating splines. In *Proc. Computer Aided Geometric Design*, pages 317–326, 1974.

W. Chen, L. Ren, M. Zwicker, and H. Pfister. Hardware-accelerated adaptive EWA volume splatting. In *Proc. Conference on Visualization*, pages 67–74, 2004.

J. J. Cherlin, F. Samavati, M. C. Sousa, and J. A. Jorge. Sketch-based modeling with few strokes. In *Proc. Spring Conference on Computer Graphics*, pages 137–145, 2005.

J. H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, 1976.

L. Coconu and H.-C. Hege. Hardware-oriented point-based rendering of complex scenes. In *Proc. Eurographics Workshop on Rendering*, pages 43–52, 2002.

E. Cohen, R. F. Riesenfeld, and G. Elber. *Geometric Modeling with Splines*. A. K. Peters, 2001.

M. Desbrun and M.-P. Cani. Animating soft substances with implicit surfaces. In *Proc. ACM SIGGRAPH*, pages 287–290, 1995.

M. Desbrun and M.-P. Cani. Active implicit surface for animation. In *Proc. Graphics Interface*, pages 143–150, 1998.

D. Doo and M. Sabin. Behavior of recursive division surfaces near extraordinary points. *Computer-Aided Design*, 10(6):356–360, 1978.

M. Droske, B. Meyer, M. Rumpf, and C. Schaller. An adaptive level set method for medical image segmentation. In *Proc. International Conference on Information Processing in Medical Imaging*, pages 416–422, 2001.

H. Du. Interactive shape design using volumetric implicit PDEs. In *Proc. ACM Symposium on Solid Modeling and Applications*, pages 235–246, 2003.

H. Du and H. Qin. A shape design system using volumetric implicit PDEs. *Computer Aided Design*, 36(11):1101–1116, 2004.

H. Du and H. Qin. Dynamic PDE-based surface design using geometric and physical constraints. *Graphical Models*, 67(1):43–71, 2005.

H. Du and H. Qin. Free-form geometric modeling by integrating parametric and implicit PDEs. *IEEE Transactions on Visualization and Computer Graphics*, 13 (3):549–561, 2007.

M. Eitz and G. Lixu. Hierarchical spatial hashing for real-time collision detection. In *Proc. International Conference on Shape Modeling and Applications*, pages 61–70, 2007.

G. Elber. Multiresolution curve editing with linear constraints. In *Proc. Symposium on Solid Modeling and Applications*, pages 109–119, 2001.

G. Elber. Geometric texture modeling. *IEEE Computer Graphics and Applications.*, 25(4):66–76, 2005.

G. Elber and C. Gotsman. Multiresolution control for nonuniform B-spline curve editing. In *Proc. Pacific Graphics Conference on Computer Graphics and Applications*, pages 267–278, 1995.

D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell. A hybrid particle level set method for improved interface capturing. *Journal of Computational Physics*, 183:83–116, 2002a.

D. Enright, S. Marschner, and R. Fedkiw. Animation and rendering of complex water surfaces. *ACM Transactions on Graphics*, 21(3):736–744, 2002b.

D. Enright, F. Losasso, and R. Fedkiw. A fast and accurate semi-lagrangian particle level set method. *Computers and Structures*, 83:479–490, 2005.

G. Farin. *Curves and Surfaces for CAGD: A Practical Guide*. Morgan-Kaufmann, 5th edition, 2002.

E. Ferley, M.-P. Cani, and J.-D. Gascuel. Practical volumetric sculpting. *The Visual Computer*, 16(8):469–480, 2000.

E. Ferley, M.-P. Cani, and J.-D. Gascuel. Resolution adaptive volume sculpting. *Graphical Models*, 63(6):459–478, 2001.

A. Finkelstein and D. Salesin. Multiresolution curves. In *Proc. ACM SIGGRAPH*, pages 261–268, 1994.

S. F. Frisken and R. N. Perry. Designing with distance fields. In *ACM SIGGRAPH Course #2 Notes*, pages 60–66, 2006.

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 *Proc. ACM SIGGRAPH*, pages 249–254, 2000.

T. A. Galyean and J. F. Hughes. Sculpting: An interactive volumetric modeling technique. In *Proc. ACM SIGGRAPH*, pages 267–274, 1991.

M. Gross and H. Pfister. *Point-Based Graphics*. Morgan Kaufmann, San Francisco, 2007.

X. Gu, S. J. Gortler, and H. Hoppe. Geometry images. *ACM Transactions on Graphics*, 21(3):355–361, 2002.

I. Guskov, W. Sweldens, and P. Schröder. Multiresolution signal processing for meshes. In *Proc. ACM SIGGRAPH*, pages 325–334, 1999.

I. Guskov, K. Vidimče, W. Sweldens, and P. Schröder. Normal meshes. In *Proc. ACM SIGGRAPH*, pages 95–102, 2000.

I. Guskov, A. Khodakovsky, P. Schröder, and W. Sweldens. Hybrid meshes: Multiresolution using regular and irregular refinement. In *Proc. ACM Symposium on Computational Geometry*, pages 264–272, 2002.

A. Harten, B. Engquist, S. Osher, and S. R. Chakravarthy. Uniformly high order accurate essentially non-oscillatory schemes, III. *Journal of Computational Physics*, 71(2):231–303, 1987.

E. J. Hastings, J. Mesit, and R. K. Guha. Optimization of large-scale, real-time simulations by spatial hashing. In *Proc. Summer Computer Simulation Conference*, pages 9 – 17, 2005.

S. Hornus, A. Angelidis, and M.-P. Cani. Implicit modelling using subdivision curves. *The Visual Computer*, 19(2-3):94–104, 2003.

B. Houston, M. Wiebe, and C. Batty. RLE sparse level sets. In *ACM SIGGRAPH Technical Sketches*, page 137, 2004.

B. Houston, M. Nielsen, C. Batty, O. Nilsson, and K. Museth. Hierarchical RLE level

set: A compact and versatile deformable surface representation. *ACM Transactions on Graphics*, 25(1):151–175, 2006.

J. Hua and H. Qin. Scalar-field guided adaptive shape deformation and animation. *The Visual Computer*, 20(1):47–66, 2004.

T. Igarashi and J. F. Hughes. Smooth meshes for sketch-based freeform modeling. In *Proc. ACM Symposium on Interactive 3D Graphics*, pages 139–142, 2003.

T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: A sketching interface for 3-D freeform design. In *Proc. ACM SIGGRAPH*, pages 409–416, 1999.

M. W. Jones, J. A. Bærentzen, and M. Sramek. 3D distance fields: A survey of techniques and applications. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):581–599, 2006.

O. Karpenko, J. Hughes, and R. Raskar. Free-form sketching with variational implicit surfaces. *Computer Graphics Forum*, 21(3):585–594, 2002.

A. Kaufman, D. Cohen, and R. Yagel. Volume graphics. *Computer*, 26(7):51–64, 1993.

A. Khodakovsky, P. Schröder, and W. Sweldens. Progressive geometry compression. In *Proc. ACM SIGGRAPH*, pages 271–278, 2000.

L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. Interactive multi-resolution modeling on arbitrary meshes. In *Proc. ACM SIGGRAPH*, pages 105–114, 1998.

Y.-K. Lai, S.-M. Hu, D. X. Gu, and R. R. Martin. Geometric texture synthesis and transfer via geometry images. In *Proc. ACM Symposium on Solid and Physical Modeling*, pages 15–26, 2005.

D. Laney, M. Bertram, M. Duchaineau, and N. Max. Multiresolution distance volumes for progressive surface compression. In *Proc. International Symposium on 3D Data Processing Visualization and Transmission*, pages 470–480, 2002.

Y. N. Law, H. K. Lee, and A. M. Yip. A multiresolution stochastic level set method for mumford-shah image segmentation. *IEEE Transactions on Image Processing*, 17(12):2289–2300, 2008.

J. Lawrence and T. Funkhouser. A painting interface for interactive surface deformations. *Graphical Models*, 66(6):418–438, 2004.

S. Lefebvre and H. Hoppe. Perfect spatial hashing. *ACM Transactions on Graphics*, 25(3):579–588, 2006.

A. E. Lefohn, J. M. Kniss, C. D. Hansen, and R. T. Whitaker. Interactive deformation and visualization of level set surfaces using graphics hardware. In *Proc. IEEE Visualization*, pages 75–82, 2003.

A. E. Lefohn, J. M. Kniss, C. D. Hansen, and R. T. Whitaker. A streaming narrow-band algorithm: Interactive computation and visualization of level sets. *IEEE Transactions on Visualization and Computer Graphics*, 10:422–433, 2004.

M. C. Leu and W. Zhang. Virtual sculpting with surface smoothing based on level set method. *CIRP Annals - Manufacturing Technology*, 57:167–170, 2008.

X. Li, L. Gu, S. Zhang, J. Zhang, G. Zheng, P. Huang, and J. Xu. Hierarchical spatial hashing-based collision detection and hybrid collision response in a haptic surgery simulator. *International Journal of Medical Robotics and Computer Assisted Surgery*, 4(1):77–86, 2008.

X.-D. Liu, S. Osher, and T. Chan. Weighted essentially non-oscillatory schemes. *Journal of Computational Physics*, 115(1):200–212, 1994.

C. Loop. Smooth subdivision surfaces based on triangles. M.S. in Mathematics thesis, University of Utah, 1987.

W. E. Lorensen and H. E. Cline. Marching Cubes: A high resolution 3D surface construction algorithm. In *Proc. ACM SIGGRAPH*, pages 163–169, 1987.

F. Losasso, F. Gibou, and R. Fedkiw. Simulating water and smoke with an octree data structure. *ACM Transactions on Graphics*, 23(3):457–462, 2004.

K. McDonnell, H. Qin, and R. Wlodarczyk. Virtual clay: A real-time sculpting system with haptic toolkits. In *Proc. Symposium on Interactive 3D Graphics*, pages 179–190, 2001.

K. T. McDonnell and H. Qin. PB-FFD: A point-based technique for free-form deformation. *Journal of Graphics Tools*, 12(3):25–41, 2007.

D. Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129–147, 1982.

B. Miller, K. Museth, D. Penney, and N. B. Zafar. Cloud modeling and rendering for Puss In Boots, ACM SIGGRAPH Talk, 2012.

C. Min. Local level set method in high dimension and codimension. *Journal of Computational Physics*, 200(1):368–382, 2004.

Y. Mori and T. Igarashi. Plushie: an interactive design system for plush toys. *ACM Transactions on Graphics*, 26(3):45, 2007.

P. Mullen, A. McKenzie, Y. Tong, and M. Desbrun. A variational approach to Eulerian geometry processing. *ACM Transactions on Graphics*, 26(3):66, 2007.

K. Museth, D. E. Breen, R. T. Whitaker, and A. Barr. Level set surface editing operators. *ACM Transactions on Graphics*, 21(3):330–338, 2002.

K. Museth, D. E. Breen, R. T. Whitaker, S. Mauch, and D. Johnson. Algorithms for interactive editing of level set models. *Computer Graphics Forum*, 24(4):821–841, 2005.

A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa. Fibermesh: designing freeform surfaces with 3D curves. *ACM Transactions on Graphics*, 26(3):41, 2007.

M. Nielsen and K. Museth. Dynamic tubular grid: An efficient data structure and algorithms for high resolution level sets. *Journal of Scientific Computing*, 26(3): 261–299, 2006.

M. Nielsen, O. Nilsson, A. Söderström, and K. Museth. Out-of-core and compressed level set simulations. *ACM Transactions on Graphics*, 26(4), 2007.

J. Nievergelt and P. Widmayer. Spatial data structures: Concepts and design choices. In *Algorithmic Foundations of Geographic Information Systems*, volume 1340 of *Lecture Notes in Computer Science*, pages 153–197. Springer, Berlin, 1997.

S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer, Berlin, 2002.

S. Osher and J. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79:12–49, 1988.

S. Owada, F. Nielsen, K. Nakazawa, and T. Igarashi. A sketching interface for modeling the internal structures of 3D shapes. In *Proc. International Symposium on Smart Graphics*, pages 49–57, 2003.

D. Peng, B. Merriman, S. Osher, H.-K. Zhao, and M. Kang. A PDE-based fast local level set method. *Journal of Computational Physics*, 155:410–438, 1999.

K. L. Perng, W. T. Wang, M. Flanagan, and M. Ouhyoung. A real-time 3D virtual sculpting tool based on modified marching cubes. *International Conference on Artificial Reality and Tele-Existence*, 11:64–72, 2001.

R. N. Perry and S. F. Frisken. Kizamu: A system for sculpting digital characters. In *Proc. ACM SIGGRAPH*, pages 47–56, 2001.

J. Peters and U. Reif. *Subdivision Surfaces*. Springer, 2008.

S. D. Porumbescu, B. Budge, L. Feng, and K. I. Joy. Shell maps. *ACM Transactions on Graphics*, 24(3):626–633, 2005.

L. Ren, H. Pfister, and M. Zwicker. Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. *Computer Graphics Forum*, 21(3):461–470, 2002.

C. Reynolds. Big fast crowds on PS3. In *Proc. ACM SIGGRAPH Symposium on Videogames*, pages 113–121, 2006.

M. Rumpf and R. Strzodka. Level set segmentation in graphics hardware. In *Proc. IEEE International Conference on Image Processing*, pages 1103–1106, 2001.

S. Rusinkiewicz and M. Levoy. Qsplat: a multiresolution point rendering system for large meshes. In *Proc. ACM SIGGRAPH*, pages 343–352, 2000.

M. Sainz and R. Pajarola. Point-based rendering techniques. *Computers & Graphics*, 28(6):869–879, 2004.

H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.

A. Sarti and S. Tubaro. Multiresolution implicit object modeling. In *Proc. Vision Modeling and Visualization Conference*, pages 93–100, 2001.

R. Schmidt and K. Singh. Sketch-based procedural surface modeling and compositing using surface trees. *Computer Graphics Forum*, 27(2):321–330, 2008.

R. Schmidt and B. Wyvill. Generalized sweep templates for implicit modeling. In *Proc. International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, pages 187–196, 2005.

R. Schmidt, B. Wyvill, and E. Galin. Interactive implicit modeling with hierarchical spatial caching. In *Proc. International Conference on Shape Modeling and Applications*, pages 104–113, 2005a.

R. Schmidt, B. Wyvill, M. Sousa, and J. A. Jorge. Shapeshop: Sketch-based solid modeling with blobtrees. In *Proc. Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 53–62, 2005b.

P. Schröder. Subdivision as a fundamental building block of digital geometry processing algorithms. *Journal of Computational and Applied Mathematics*, 149(1): 207–219, 2002.

C. A. Schroeder, D. E. Breen, C. D. Cera, and W. C. Regli. Stochastic microgeometry for displacement mapping. In *Proc. International Conference on Shape Modeling and Applications*, pages 166–175, 2005.

T. Sederberg and S. Parry. Free-form deformation of solid geometric models. In *Proc. ACM SIGGRAPH*, pages 151–160, 1986.

J. A. Sethian. A fast marching level set method for monotonically advancing fronts. In *Proc. National Academy of Sciences*, pages 1591–1595, 1995.

J. A. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, 2nd edition, 1999.

K. Shoemake. Animating rotation with quaternion curves. In *Proc. ACM SIGGRAPH*, pages 245–254, 1985.

C.-W. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock-capturing schemes. *Journal of Computational Physics*, 77(2):439–471, 1988.

K. Singh and E. Fiume. Wires: a geometric deformation technique. In *Proc. ACM SIGGRAPH*, pages 405–414, 1998.

G. G. Slabaugh and R. W. Schafer. Multi-resolution space carving using level set methods. In *Proc. IEEE International Conference on Image Processing*, pages 545–548, 2002.

J. Stam. Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values. In *Proc. ACM SIGGRAPH*, pages 395–404, 1998.

M. Stamminger and G. Drettakis. Interactive sampling and rendering for complex and procedural geometry. In *Proc. Eurographics Workshop on Rendering Techniques*, pages 151–162, 2001.

J. Strain. Tree methods for moving interfaces. *Journal of Computational Physics*, 151(2):616–648, 1999.

M. Sugihara, E. de Groot, B. Wyvill, and R. Schmidt. A sketch-based method to control deformation in a skeletal implicit surface modeler. In *Proc. Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 65–72, 2008.

C. L. Tai, H. Zhang, and J. C.-K. Fong. Prototype modeling from sketched silhouettes based on convolution surfaces. *Computer Graphics Forum*, 23(1):71–83, 2004.

T. Tasdizen, R. T. Whitaker, P. Burchard, and S. Osher. Geometric surface processing via normal maps. *ACM Transactions on Graphics*, 22(4):1012–1033, 2003.

M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, and M. Gross. Optimized spatial hashing for collision detection of deformable objects. In *Proc. Vision, Modeling and Visualization*, pages 47–54, 2003.

J. N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control*, 40(9):1528–1538, 1995.

G. Turk and J. O'Brien. Modeling with implicit surfaces that interpolate. *ACM Transactions on Graphics*, 21(4):855–873, 2002.

L. Velho, J. Gomes, and L. H. de Figueiredo. *Implicit Objects in Computer Graphics*. Springer, 2002.

W. von Funck, H. Theisel, and H.-P. Seidel. Vector field based shape deformations. *ACM Transactions on Graphics*, 25(3):1118–1125, 2006.

W. von Funck, H. Theisel, and H.-P. Seidel. Explicit control of vector field based shape deformations. In *Proc. Pacific Conference on Computer Graphics and Applications*, pages 291–300, 2007.

S. W. Wang and A. E. Kaufman. Volume-sampled 3D modeling. *IEEE Computer Graphics and Applications*, 14(5):26–32, 1994.

S. W. Wang and A. E. Kaufman. Volume sculpting. In *Proc. Symposium on Interactive 3D Graphics*, pages 151–156, 1995.

J. Warren and H. Weimer. *Subdivision Methods for Geometric Design: A Constructive Approach*. Morgan Kaufmann, 2001.

R. T. Whitaker. VISPACK. Technical Report UUCS 08-0011, School of Computing, University of Utah, 2008.

R. T. Whitaker. A level-set approach to 3D reconstruction from range data. *International Journal of Computer Vision*, 29(3):203–231, 1998.

M. Wiebe and B. Houston. The tar monster: Creating a character with fluid simulation. In *ACM SIGGRAPH Technical Sketches*, 2004.

B. Wyvill and G. Wyvill. Field functions for implicit surfaces. *The Visual Computer*, 5(1&2):75–82, 1989.

B. Wyvill, C. McPheeters, and G. Wyvill. Animating soft objects. *The Visual Computer*, 2(4):235–242, 1986a.

B. Wyvill, E. Galin, and A. Guy. Extending the CSG tree. Warping, blending and boolean operations in an implicit surface modeling system. *Computer Graphics Forum*, 18(2):149–158, 1999.

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

T. Yoo. *Insight into Images: Principles and Practice for Segmentation, Registration, and Image Analysis*. AK Peters, 2004.

A. Youssef. Image downsampling and upsampling methods. In *Proc. International Conference on Imaging, Science, Systems, and Technology*, pages 132–138, 1999.

R. C. Zeleznik, K. P. Herndon, and J. F. Hughes. Sketch: an interface for sketching 3D scenes. In *Proc. ACM SIGGRAPH*, pages 163–170, 1996.

J. J. Zhang and Y. Lihua. Surface representation using second, fourth and mixed order partial differential equations. In *Proc. International Conference on Shape Modeling and Applications*, pages 250–256, 2001.

K. Zhou, X. Huang, X. Wang, Y. Tong, M. Desbrun, B. Guo, and H.-Y. Shum. Mesh quilting for geometric texture synthesis. In *Proc. ACM SIGGRAPH*, pages 690–697, 2006.

J. Zimmermann, A. Nealen, and M. Alexa. Silsketch: automated sketch-based editing of surface meshes. In *Proc. Eurographics Workshop on Sketch-based Interfaces and Modeling*, pages 23–30, 2007.

D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. In *Proc. ACM SIGGRAPH*, pages 259–268, 1997.

M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Surface splatting. In *Proc. ACM SIGGRAPH*, pages 371–378, 2001.