Volume 24 (2005), number 4 pp. 821–841 COMPUTER GRAPHICS *forum*

# Algorithms for Interactive Editing of Level Set Models

Ken Museth[1], David E. Breen[2], Ross T. Whitaker[3], Sean Mauch[4] and David Johnson[3]

[1]Linköping Institute of Technology
[2]Drexel University
[3]University of Utah
[4]California Institute of Technology

**Abstract**

*Level set models combine a low-level volumetric representation, the mathematics of deformable implicit surfaces and powerful, robust numerical techniques to produce a novel approach to shape design. While these models offer many benefits, their large-scale representation and numerical requirements create significant challenges when developing an interactive system. This paper describes the collection of techniques and algorithms (some new, some pre-existing) needed to overcome these challenges and to create an interactive editing system for this new type of geometric model. We summarize the algorithms for producing level set input models and, more importantly, for localizing/minimizing computation during the editing process. These algorithms include distance calculations, scan conversion, closest point determination, fast marching methods, bounding box creation, fast and incremental mesh extraction, numerical integration and narrow band techniques. Together these algorithms provide the capabilities required for interactive editing of level set models.*

**Keywords:** level set methods, implicit modelling, scan conversion, numerical methods, narrow band techniques, mesh extraction, interactive graphics

**ACM CCS:** I.3.5 Computer Graphics: *Geometric Algorithms, Languages and Systems*

## 1. Introduction

Level set models are a new type of geometric model for creating complex, closed objects. They combine a low-level volumetric representation, the mathematics of deformable implicit surfaces and powerful, robust numerical techniques to produce a novel approach to shape design. During an editing session, a user focuses on and conceptually interacts with the shape of a level set surface, while the level set methods 'under the hood' calculate the appropriate voxel values for a particular editing operation, completely hiding the volumetric representation of the surface from the user.

More specifically, level set models are defined as an iso-surface, i.e. a level set, of some implicit function $\phi$. The surface is deformed by solving a partial differential equation (PDE) on a regular sampling of $\phi$, i.e. a volume data set [1]. Thus, it should be emphasized that level set methods do not manipulate an explicit closed form representation of $\phi$, but only a sampling of it. Level set methods provide the techniques needed to change the voxel values of the volume in a way that moves the embedded iso-surface to meet a user-defined goal.

Defining a surface with a volume data set may seem unusual and inefficient, but level set models do offer numerous benefits in comparison to other types of geometric surface representations. They are guaranteed to define simple (non-self-intersecting) and closed surfaces. Thus level set editing operations will always produce a physically realizable (and therefore manufacturable) object. Level set models easily change topological genus, making them ideal for representing complex structures of unknown genus. They are free of the edge connectivity and mesh quality problems common in surface mesh models. Additionally, they provide the advantages of implicit models, e.g. supporting straightforward solid modeling operations and calculations, while offering a powerful surface modeling paradigm.



Submitted June 2004
Revised July 2005
Accepted August 2005

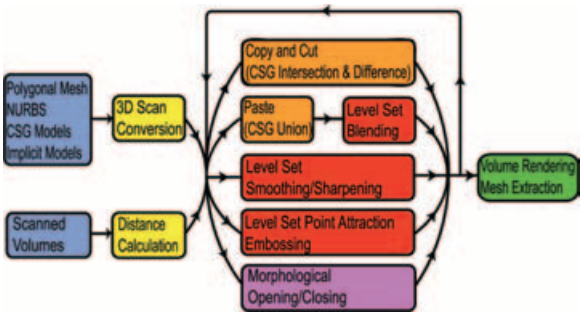


**Figure 1:** *Level set modeling system modules. The system consists of input models (blue), pre-processing (yellow), CSG operations (orange), local LS operators (red), global LS operators (purple) and rendering (green).*

### 1.1. Level Set Model Editing System

We have developed an interactive system for editing level set models. The modules and the data flow of the system is diagrammed in Figure 1. The blue modules contain the types of models that may be imported into the system. The yellow modules contain the algorithms for converting the input models into level set models. The orange, red and purple modules are the editing operations that can be performed on the models. The final (green) module renders the model for interactive viewing.

In a previous paper [2], Museth *et al.* described the mathematical details of the editing operators, some of which were based on concepts proposed in [3]. The cut-and-paste (orange) operators give the user the ability to copy, remove and merge level set models (using volumetric CSG operations) and automatically blend the intersection regions (1st red module). Our smoothing operator (2nd red module) allows a user to define a region of interest and smooths the enclosed surface to a user-defined curvature value. We have also developed a point-attraction operator. A regionally constrained portion of a level set surface may be attracted to a set of points to produce a surface embossing operator (3rd red operator). As noted by others, the opening and closing morphological (purple) operators [4] may be implemented efficiently in a level set framework [5,6]. We have also found them useful for global blending (closing) and smoothing (opening).

### 1.2. Challenges and Solutions

The volumetric representation and the mathematics of level set models create numerous challenges when developing an interactive level set editing system. Together they indicate the need for a massive amount of computation on large-scale data sets, in order to numerically solve the level set equation at each voxel in the volume. In this paper, we address these issues, focusing on the implementation details of our level set editing work and describing the collection of algorithms (some new, some pre-existing) needed to create an interactive level set model editing system.

The first challenge encountered when editing level set models is converting conventional surface representations into the volumetric format needed for processing with level set methods. Our goal has been to connect level set editing with other forms of geometric modeling. The user may utilize pre-existing modeling tools to create a variety of models. Developing a suite of model conversion tools allows those models to be imported into our system for additional modifications using editing operations unique to level set models. We therefore have implemented several 3D scan conversion algorithms. The essential computation for most of these algorithms involves calculating a closest point (and therefore the shortest distance) from a point to the model.

The second major challenge of interactive level set model editing is minimizing the amount of computation needed to perform the individual operations. The mathematics of level set models is defined globally, but in practice most level set operators only modify a small portion of the model. We therefore employ a variety of techniques to localize the level set computations in order to make the editing system interactive. Since we are only interested in one level set (the zero iso-surface) in the volume, narrow-band techniques [7–9] may be used to make the computation proportional to the surface area of the model. Additionally, the extensive use of bounding boxes further limits the region of computation on the surface. Some of our operators require a closest-point-in-set calculation. Here K–D trees [10,11] are utilized. Finally, interactive viewing is made possible by an incremental, optimized mesh extraction algorithm. Brought together, all of these techniques and data structures allow us to import a variety of models and interactively edit them with level set surface editing operators.

### 1.3. Related Work

Two areas of research are closely related to our level set surface editing work: volumetric sculpting and implicit modeling. Volumetric sculpting provides methods for directly manipulating the voxels of a volumetric model. Constructive Solid Geometry (CSG) Boolean operations [12,13] are commonly found in volume sculpting systems, providing a straightforward way to create complex solid objects by combining simpler primitives. One of the first volume sculpting systems is presented in [14]. Incremental improvements to the concept of volume sculpting soon followed. Wang and Kaufman [15] introduced tools for carving and sawing, [16] developed a haptic interface for sculpting, [17] introduced new sculpting tools and improved interactive rendering and [18] provides procedural methods for defining volumetric models. Physical behavior has been added to the underlying volumetric model in order to produce virtual clay [19]. McDonnell *et al.* [20] improved upon this work by representing the virtual clay with subdivision solids [21]. More

recently sculpting systems [22,23] have been based on octree representations [24,25], allowing for volumetric models with adaptive resolution.

There exists a large body of surface editing work based on implicit models [26]. This approach uses implicit surface representations of analytic primitives or skeletal offsets. The implicit modeling work most closely related to ours is found in [27]. They describe techniques for performing blending, warping and boolean operations on skeletal implicit surfaces. An interesting variation of implicit modeling is presented in [28], which uses a forest of trivariate functions [29] evaluated on an octree to create a multiresolution sculpting capability.

Level set methods have been successfully applied in computer graphics, computer vision and visualization [30–33], for example medical image segmentation [34–36], shape morphing [37,38], 3D reconstruction [8,39,40], volume sculpting [41], and the animation of liquids [42].

Our work stands apart from previous work in several ways. We have not developed volumetric modeling tools. Our editing system acts on surfaces that happen to have an underlying volumetric representation, but are based on the mathematics of deforming implicit surfaces. In our system, voxels are not directly modified by the user, instead voxel values are determined numerically by solving the level set equation, based on user input. Since level set models are not tied to any specific implicit basis functions, they easily represent complex models to within the resolution of the sampling. Our work is the first to utilize level set methods to perform a variety of interactive editing operations on complex geometric models.

It should also be noted that several of the algorithms described in this paper have been implemented in graphics hardware, e.g. solving level set equations [43,44], evaluating other types of differential equations [45–47], morphological operators [48,49], and voxelization [50,51]. This work predominantly focuses on coping with the issues that arise from mapping general algorithms onto hardware-specific GPUs with restrictive memory sizes, data types and instruction sets in order to shorten computation times.

## 2. Level Set Models

Level set models implicitly represent a deforming surface as an iso-surface (or level set),

$$\mathcal{S} = \{\boldsymbol{x} |\ \phi(\boldsymbol{x}) = k\}\,, \tag{1}$$

where $k \in \mathbb{R}$ is the iso-value, $\boldsymbol{x} \in \mathbb{R}^3$ is a point in space on the iso-surface and $\phi : \mathbb{R}^3 \to \mathbb{R}$ is an arbitrary scalar function. To allow for deformations of the level set surface, we assume that $S$ can change over time. Introducing time dependence into the right-hand side of Equation (1) produces two distinct types of level set surface representations. In the first, the iso-value $k$ can be considered time-dependent and the level set function $\phi$ is only implicitly time-dependent, leading to the *static level set formulation*,

$$\mathcal{S}(t) = \{\boldsymbol{x}(t) \mid \phi(\boldsymbol{x}(t)) = k(t)\}\,. \tag{2}$$

In the second, the iso-value $k$ is fixed in time and the level set function explicitly depends on time, leading to the *dynamic level set formulation*,

$$\mathcal{S}(t) = \{\boldsymbol{x}(t) \mid \phi(\boldsymbol{x}(t), t) = k\}\,. \tag{3}$$

These two level set formulations are *not* equivalent and offer very distinct advantages and disadvantages. A detailed discussion of level set methods is beyond the scope of this paper, but since both formulations play important roles in our work each is briefly described. For more details we refer the interested reader to [30,32].

### 2.1. Equation Formulations

The static formulation of Equation (2) describes the deforming surface as a family of level sets, $\mathcal{S}(t)$, of a static function $\phi(\boldsymbol{x})$, that evolves according to a time-dependent iso-function, $k(t)$. Without loss of generality, we can limit ourselves to the simple cases $k(t) = \pm t$ as this leads to intuitive interpretations of $\phi(\boldsymbol{x})$ as the forward or backward time of arrival of the level set surface at a point $\boldsymbol{x}$. The corresponding equation of motion for a point, $\boldsymbol{x}(t)$, on the surface is easily derived by differentiating both sides of $\phi(\boldsymbol{x}(t)) = \pm t$ with respect to time $t$, and applying the chain rule giving:

$$\nabla\phi(\boldsymbol{x}(t)) \cdot \frac{d\boldsymbol{x}(t)}{dt} = \pm 1. \tag{4}$$

Before interpreting this equation, it is first necessary to define the term *level set speed function*. Throughout this paper, we assume a *positive-inside/negative-outside* sign convention for $\phi$, i.e. normal vectors, $\boldsymbol{n}$, of any level set of $\phi$ *point outwards* and are simply given by $\boldsymbol{n} \equiv -\nabla\phi/|\nabla\phi|$. This allows us to define the following *speed function*:

$$\mathcal{F}(\boldsymbol{x}, \boldsymbol{n}, \phi, \ldots) \equiv \boldsymbol{n} \cdot \frac{d\boldsymbol{x}}{dt} = -\frac{\nabla\phi}{|\nabla\phi|} \cdot \frac{d\boldsymbol{x}}{dt} \tag{5}$$

which in general is a user-defined scalar function that can depend on any number of variables including $\boldsymbol{x}$, $\boldsymbol{n}$, $\phi$ and its derivatives evaluated at $\boldsymbol{x}$, as well as a variety of external data inputs. The geometric interpretation of this function is straightforward; since $d\boldsymbol{x}/dt$ denotes the velocity vectors of a point $\boldsymbol{x}$ on a level set surface, the speed function $\mathcal{F}$ defines the projection of this vector onto the local surface normal. In other words, $\mathcal{F}$ is a *signed* scalar function that defines the motion (i.e. speed) of the level set surface in the direction of the local normal $\boldsymbol{n}$ at a point $\boldsymbol{x}$ on a level set $\mathcal{S}$.

Using the definition of a speed function in Equation (5), Equation (4) may be simplified to

$$|\nabla\phi|\mathcal{F} = \pm 1 \tag{6}$$

which only depends implicitly on time, and therefore

describes a simple boundary value problem. Equation (6) is known as the *fundamental stationary level set equation* and can be efficiently solved using fast marching methods (FMMs) [52,53], which will be described in detail in Section 3.1.6 We point out that the *Eikonal equation*

$$|\nabla\phi| = 1 \tag{7}$$

which produces a *signed distance field* to an initial surface $\mathcal{S}_0$, can be considered a special case of the stationary level set equation (Equation (6)) with a unit speed function and the boundary condition $\{\boldsymbol{x} \in \mathcal{S} \mid \phi(\boldsymbol{x}) = 0\}$. Therefore, FMMs may be used to efficiently compute the signed distance field to an arbitrary (closed and orientable) surface.

The stationary level set representation has a significant limitation, however. It follows directly from Equation (6) that the speed functions, $\mathcal{F}$, has to be strictly positive or negative depending on the sign of the right-hand side. Consequently, surface deformations are limited to strict monotonic motions— always inwards or outwards, similar to the layers of an onion. This limitation stems from the fact that $\phi(\boldsymbol{x})$ by definition has to be single-valued (time-of-arrival), i.e. the level set surface resulting from the stationary formulation cannot self-intersect over time. The inherit limitation of the static formulation can be overcome by adding an explicit time-dependence to $\phi$, which leads to the dynamic level set Equation (3).

Following the same steps as the stationary case, the dynamic equation of motion may be derived by differentiating the right-hand side of Equation (3) with respect to time and applying the speed function definition (Equation (5)), giving

$$\frac{\partial\phi}{\partial t} = -\nabla\phi \cdot \frac{d\boldsymbol{x}}{dt} \tag{8a}$$

$$= |\nabla\phi|\, \mathcal{F}(\boldsymbol{x}, \boldsymbol{n}, \phi, \ldots). \tag{8b}$$

These so-called Hamilton-Jacobi equations are often referred to as 'the level set equations' in the literature, even though they are strictly speaking the dynamic counterpart to the stationary level set Equation (6). The RHS of Equation (8a) and (8b) is collectively called Hamiltonians of the level set equation. Note that in contrast to the stationary form, the dynamic surface representation of Equation (8b) does not limit the sign of the speed function, $\mathcal{F}$, and therefore allows for arbitrary surface deformations. The speed function is usually based on a set of geometric measures of the implicit level set surface and data inputs. The challenge when working with level set methods is determining how to combine these components to produce a local motion that creates a desired global or regional surface structure. Museth *et al.* [2] define several such speed functions that may be used to edit geometric objects.

### 2.2. Geometric Properties

The speed function, $\mathcal{F}$, introduced in the previous section typically depends on different geometric properties of the level set surface. These properties can conveniently be expressed as zero, first or second order derivatives of $\phi$. Examples include the shortest distance from an arbitrary point to the surface, the local surface normal and different curvature measures. Assuming $\phi$ is properly normalized, i.e. satisfies Equation (7), the distance is simply the numerical value of $\phi$, and as indicated above the normal vector is just a normalized gradient of $\phi$. The latter is easily proved by noting that all directional derivatives in the tangent plane of the level set function by definition vanish, i.e.

$$\frac{d\phi}{d\boldsymbol{T}} \equiv \boldsymbol{T} \cdot \nabla\phi = 0 \tag{9}$$

where $\boldsymbol{T}$ is an arbitrary unit vector in the tangent plane of the level set surface.

There are many different curvature measures for surfaces, but as we will see (at the end of this section) geometric flow based on the mean curvature is very useful since it can be proven to minimize surface area, i.e. equivalent to smoothing. From the definition of the mean curvature in differential geometry [54], we have

$$K \equiv (K_1 + K_2)/2 \equiv (\mathrm{Div}_{\boldsymbol{e}_1}[\boldsymbol{n}] + \mathrm{Div}_{\boldsymbol{e}_2}[\boldsymbol{n}])/2 \tag{10a}$$

$$= (\boldsymbol{e}_1(\boldsymbol{e}_1 \cdot \nabla) \cdot \boldsymbol{n} + \boldsymbol{e}_2(\boldsymbol{e}_2 \cdot \nabla) \cdot \boldsymbol{n})/2 \tag{10b}$$

where $\{K_1, K_2\}$ are the principle curvatures, and $\mathrm{Div}_{\boldsymbol{e}_1}[\boldsymbol{n}]$ denotes the divergence of the normal vector $\boldsymbol{n}$ in the principle direction $\boldsymbol{e}_1$. Next, resolving the gradient operator in the orthonormal frame of the principle directions $\{\boldsymbol{e}_1, \boldsymbol{e}_2\}$ in the tangent plane and the normal vector $\boldsymbol{n}$ gives

$$\nabla = \boldsymbol{e}_1(\boldsymbol{e}_1 \cdot \nabla) + \boldsymbol{e}_2(\boldsymbol{e}_2 \cdot \nabla) + \boldsymbol{n}(\boldsymbol{n} \cdot \nabla). \tag{11}$$

Equation (10b) simplifies to

$$K = (\nabla \cdot \boldsymbol{n} - \boldsymbol{n}(\boldsymbol{n} \cdot \nabla) \cdot \boldsymbol{n})/2 \tag{12a}$$

$$= \frac{1}{2}\nabla \cdot \boldsymbol{n} = \frac{1}{2}\nabla \cdot \frac{\nabla\phi}{|\nabla\phi|} \tag{12b}$$

where we have also made use of the fact that $\boldsymbol{n}(\boldsymbol{n} \cdot \nabla) \cdot \boldsymbol{n} = 0$ which follows from the relation

$$\sum_j \boldsymbol{n}_j \sum_i \boldsymbol{n}_i \nabla_i \boldsymbol{n}_j = \sum_{ij} \boldsymbol{n}_i \frac{1}{2}\nabla_i \left[\boldsymbol{n}_j^2\right] = 0 \tag{13}$$

since the normal vector is always normalized to one. Note that the factor one-half in Equation (12b) is often (but incorrectly) ignored in the level set literature. In Section 3.2.2, we will discuss two different numerical techniques to compute the mean curvature, as well as other types of surface curvature, directly from Equation (12b).

Finally, we note that global properties such as volume, $V$, and area, $A$, of level set surfaces, $\phi$, can easily be computed as

$$V = \int_{\Omega} H(\phi(\boldsymbol{x}))\, d\boldsymbol{x} \tag{14a}$$

$$A = \int_{\Omega} \delta(\phi(\boldsymbol{x}))|\nabla\phi(\boldsymbol{x})|\, d\boldsymbol{x} \tag{14b}$$

where $\Omega$ denotes the domain of definition, $H(x)$ is a Heaviside function and $\delta(x) \equiv dH(x)/dx$ is a Dirac's delta function. For numerical implementations it is convenient to use the following 'smeared-out' and continuous approximations

$$H(\phi) \sim \begin{cases} 0 & \text{if} \quad \phi < -\epsilon \\ \frac{1}{2} + \frac{\phi}{2\epsilon} + \frac{1}{2\pi}\sin\left(\frac{\pi\phi}{\epsilon}\right) & \text{if} \quad |\phi| \le \epsilon \\ 1 & \text{if} \quad \phi > \epsilon \end{cases} \tag{15a}$$

$$\delta(\phi) \sim \begin{cases} 0 & \text{if} \quad |\phi| > \epsilon \\ \frac{1}{2\epsilon} + \frac{1}{2\epsilon}\cos\left(\frac{\pi\phi}{\epsilon}\right) & \text{if} \quad |\phi| \le \epsilon. \end{cases} \tag{15b}$$

From calculus of variation, we have the following fundamental Euler–Lagrange equation that minimizes a functional $\int_{\Omega} f(\boldsymbol{x}, \phi, \nabla\phi)d\boldsymbol{x}$

$$\left(\frac{\partial\phi}{\partial t} - \nabla\cdot\left[\frac{\partial}{\partial\phi_x}, \frac{\partial}{\partial\phi_y}, \frac{\partial}{\partial\phi_z}\right]\right) f = 0. \tag{16}$$

Along the same lines as [55], we can then apply Equation (16) to Equation (14), replace $\delta(\phi)$ with $|\nabla\phi|$ and equate the resulting Euler–Lagrange equation with $\frac{\partial\phi}{\partial t}$ to get the gradient decent expressions. This leads to the following fundamental level set equations

$$\frac{\partial\phi}{\partial t} = |\nabla\phi| \tag{17a}$$

$$\frac{\partial\phi}{\partial t} = \nabla\cdot\frac{\nabla\phi}{|\nabla\phi|}|\nabla\phi|. \tag{17b}$$

Equation (17a), which corresponds to minimization of volume, is a dynamic level set equation with a unit speed function, i.e. erosion of the level set surface. Equation (17b) states that surface area is minimized by mean curvature flow, i.e. $\mathcal{F} = K$. The last observation is especially important since our blending operators correspond to localized mean curvature flow, i.e. surface smoothing by area minimization.

## 3. Types of Computation

The main algorithms employed by our level set modeling system may be placed in three categories: distance computations; level set evolutions and efficient mesh extractions.

### 3.1. Distance Computations

A level set model is represented by a distance volume, a volume data set where each voxel stores the shortest distance to the surface of the object being represented by the volume. The inside–outside status of the point is defined by its sign, positive for inside and negative for outside. Since we are only interested in one level set (iso-surface) embedded in the volume, distance information is only maintained around one level set (usually of iso-value zero). Depending on the accuracy of the spatial discretization schemes, this 'narrow band' is typically only a few voxels wide. For the results presented in this paper, a width of five voxels was sufficient (two voxels on each side of the zero level set) (see Figure 2).

**Figure 2:** *A slice through a narrow-band distance volume.*

Before an object can be edited in our system, it must first be converted into a narrow-band distance volume. Currently, we are able to convert polygonal, NURBS, implicit and CSG models, as well as general volumetric models into the appropriate volumetric format. The fundamental operation performed in the conversion process is the calculation of the shortest distance from an arbitrary point to the geometric model being scan converted. Since the calculation is performed repeatedly, efficient computation is essential to minimizing the time needed for conversion.

#### 3.1.1. Narrow Bands and Re-Normalization

All of our level set editing operators assume that our models are represented as 'narrow-band' distance volumes. Unfortunately, our operators do not necessarily produce this representation, signed distance in a narrow band and constant values outside of the band.† The level set equation (Equation (8)) contains no explicit constraints that maintain $\phi$ as a signed distance function as time evolves. This is, however, a serious problem since stability of the finite difference schemes is only guaranteed if $|\nabla\phi|$ is (approximately) one at all times. We address this problem with a technique known as *velocity extension* [55]. Here, the speed function off of the interface is defined as the speed function at the closest-point-transform (CPT) on the surface

$$\mathcal{F}_{\text{ext}}(\boldsymbol{x}) \equiv \mathcal{F}(\text{CPT}[\boldsymbol{x}]) = \mathcal{F}(\boldsymbol{x} - \phi(\boldsymbol{x})\boldsymbol{\nabla}\phi(\boldsymbol{x})). \tag{18}$$

† They do properly produce the correct zero crossings in the resulting volumes.

We can now show that the corresponding level set propagation, $\frac{\partial \phi}{\partial t} = \mathcal{F}_{\text{ext}}|\nabla\phi|$, is in fact norm-conserving

$$\frac{\partial}{\partial t}|\nabla\phi|^2 = 2\nabla\phi \cdot \nabla\frac{\partial}{t}\phi \tag{19a}$$

$$= 2\nabla\phi \cdot [\nabla\mathcal{F}_{\text{ext}}\,|\nabla\phi| + \mathcal{F}_{\text{ext}}\nabla\,|\nabla\phi|] \tag{19b}$$

$$= 2\nabla\phi \cdot \nabla\mathcal{F}(\boldsymbol{x} - \phi\nabla\phi) \tag{19c}$$

$$= 2\nabla\phi \cdot \nabla\mathcal{F}[1 - |\nabla\phi|^2 + \phi\nabla\nabla\phi] \tag{19d}$$

$$= 2\nabla\phi \cdot \nabla\mathcal{F}\phi\nabla\nabla\phi \tag{19e}$$

$$= \nabla\mathcal{F}\phi\nabla|\nabla\phi|^2 = 0 \tag{19f}$$

where we have assumed that $\phi$ is initialized as an Euclidean distance function (i.e. $|\nabla\phi| = 1$) in Equations (19b), (19d) and (19f). This is used in our interactive (but approximate) level set solver to avoid costly explicit re-normalization at each iteration.

However, the CSG operations used extensively in our editing system are also known not to produce true distance values for all circumstances [22,56]. We must therefore re-set the volumetric representation of our models after each editing operation in order to ensure that $\phi$ is approximately equal to the shortest distance to the zero level set in the narrow band.

Such re-normalization after each editing operation can be implemented in a number of ways. One option is to directly solve the Eikonal equation, $|\nabla\phi| = 1$ using algorithms such as the FMM [52,53] or the Fast Sweeping Method of [57]. The former has a computational complexity of $\mathcal{O}(N \log N)$, where $N$ denotes the number of voxels in the narrow band, whereas the latter scales as $\mathcal{O}(N)$. Alternatively, one can solve the following time-dependent Hamilton–Jacobi equation until it reaches a steady state.

$$\frac{\partial \phi}{\partial t} = S(\phi)(1 - |\nabla\phi|) \tag{20}$$

where $S(\phi)$ returns the sign of $\phi$ [9]. This technique also has a computational complexity of $\mathcal{O}(N)$. Although linear complexity is optimal, these direct approaches are still iterative and thus too slow in the context of our interactive editing framework. Instead we use a faster but approximate solution where points on the zero level set (iso-surface) of the embedded surface are found by linearly interpolating the voxel values along grid edges that span the zero crossings. These 'zero-crossing' edges have end-points (voxels) whose associated $\phi$ values have opposite signs. The first step in rebuilding $\phi$ in the narrow band after an editing operation consists of creating the list of 'active' voxels, those adjacent to a zero crossing. Euclidean distance values are computed for these active voxels and an approximate distance metric is then used for the remaining voxels. If $\boldsymbol{x}_1$ denotes an off-surface point and $\boldsymbol{x}_0$ the corresponding closest point on the level set, $\text{CPT}[\boldsymbol{x}_1] \equiv \boldsymbol{x}_0$ then the Taylor expansion around $\boldsymbol{x}_0$ reads as

$$\phi(\boldsymbol{x}_1) = \phi(\boldsymbol{x}_0) + h\left.\frac{d\phi}{d\boldsymbol{n}}\right|_{\boldsymbol{x}_0} + \frac{h^2}{2}\left.\frac{d^2\phi}{d\boldsymbol{n}^2}\right|_{\boldsymbol{x}_0} + \cdots \tag{21a}$$

$$= \phi(\boldsymbol{x}_0) + h\,(\boldsymbol{n}\cdot\nabla\phi)|_{\boldsymbol{x}_0} + \mathcal{O}(h^2) \tag{21b}$$

$$\sim \phi(\boldsymbol{x}_0) + h\,|\nabla\phi|_{\boldsymbol{x}_0} = h\,|\nabla\phi|_{\boldsymbol{x}_0} \tag{21c}$$

where $\frac{d^k\phi}{d\boldsymbol{n}^k}|_{\boldsymbol{x}_0}$ denotes the $k$th order directional derivative of $\phi$ evaluated at $\boldsymbol{x}_0$. Hence for the active voxels we can approximate

$$\phi_{\text{new}}(\boldsymbol{x}) = \phi_{\text{old}}(\boldsymbol{x})/|\nabla\phi_{\text{old}}(\boldsymbol{x})|, \tag{22}$$

which is clearly most accurate near the zero level set.

The $\phi$ values of the next $N$ layers of voxels that form a narrow band on either side of the active list voxels are approximated by a simple city block distance metric. First, all of the voxels that are adjacent to the active list voxels are found. They are assigned a $\phi$ value that is one plus the smallest $\phi$ value of their 6-connected neighbors in the active list. Next, all of the voxels that are adjacent to the first layer, but not in the active list, are identified and their $\phi$ values are set to be one plus the smallest value of their 6-connected neighbors. This process continues until a narrow-band $\ell$ voxels thick has been created.

### 3.1.2. Polygonal Mesh Models

This section describes an algorithm for calculating a distance volume from a 3D closed, orientable polygonal mesh composed of triangular faces, edges, vertices and normals pointing outwards. The algorithm computes the closest point on and shortest signed distance to the mesh by solving the Eikonal equation by the method of characteristics. The method of characteristics is implemented[‡] efficiently with the aid of computational geometry and polyhedron scan conversion producing an algorithm with computational complexity that is linear in the number of faces, edges, vertices and voxels [58,59].

Let $\boldsymbol{\xi}$ be the closest point on a manifold to the point $\mathbf{x}$. The distance to the manifold is $|\mathbf{x} - \boldsymbol{\xi}|$ . $\mathbf{x}$ and $\boldsymbol{\xi}$ are the endpoints of the line segment that is a characteristic of the solution of the Eikonal equation. If the manifold is smooth, then the line connecting $\mathbf{x}$ to $\boldsymbol{\xi}$ is orthogonal to the manifold. If the manifold is not smooth at $\boldsymbol{\xi}$, then the line lies 'between' the normals of the smooth parts of the manifold surrounding $\boldsymbol{\xi}$.

Based on this observation, a 3D Voronoi diagram is built for the faces, edges and vertices of the mesh, with each Voronoi cell defined by a polyhedron. Scan conversion is then utilized to determine which voxels of the distance volume lie in each

‡http://www.acm.caltech.edu/∼seanm/projects/cpt/cpt.html

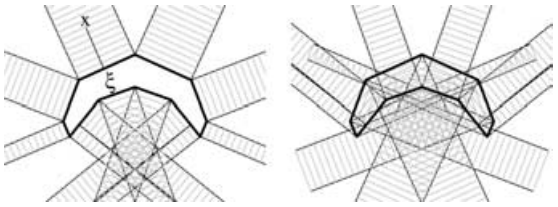

**Figure 3:** *Strips containing points with negative (left) and positive (right) distance to edges.*

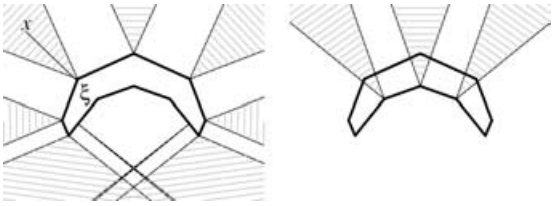

**Figure 4:** *Wedges containing points with negative (left) and positive (right) distance to vertices.*

Voronoi cell. By definition the face, edge or vertex associated with the Voronoi cell is the closest element on the mesh to the voxels in the cell. The closest point/shortest distance to the element is then calculated for each voxel.

Suppose that the closest point $\boldsymbol{\xi}$ to a grid point $\mathbf{x}$ lies on a triangular face. The vector from $\boldsymbol{\xi}$ to $\mathbf{x}$ is orthogonal to the face. Thus, the closest points to a given face must lie within a triangular prism defined by the edges and normal vector of the face. Faces produce prisms of both positive and negative distance depending on their relationship to the face's normal vector. The sign of the distance value in the prism in the direction of the normal (outside the mesh) is negative and is positive opposite the normal (inside the mesh). A 2D example is presented in Figure 3. In two dimensions, the Voronoi cells are defined as strips with negative and positive distance.

Consider a grid point $\mathbf{x}$ whose closest point $\boldsymbol{\xi}$ is on an edge. Each edge in the mesh is shared by two faces. The closest points to an edge must lie in a wedge defined by the edge and the normals of the two adjacent faces. We define only one Voronoi cell for each edge in the direction where the angle between the faces is greater than $\pi$. Finally, consider a grid point $\mathbf{x}$ whose closest point $\boldsymbol{\xi}$ is on a vertex. Each vertex in the mesh is shared by three or more faces. The closest points to a vertex must lie in a faceted cone defined by the normals to the adjacent faces. Similar to the edge Voronoi cells, we only define one polyhedron for each vertex. The cone will point outwards and contain negative distance if the surface is convex at the vertex. The cone will point inwards and contain positive distance if the surface is concave at the vertex. Figure 4 may be thought of as a 2D cross-section of an edge or a vertex Voronoi cell and demonstrates the conditions

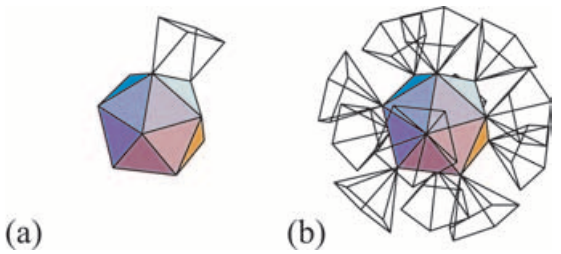


**Figure 5:** *(a) The polyhedron for a single edge. (b) The polyhedra for the vertices.*

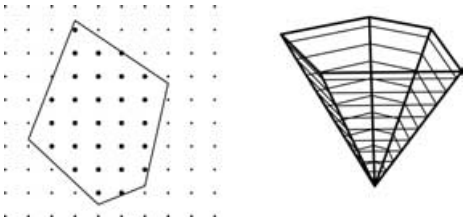

**Figure 6:** *Scan conversion of a polygon in 2D. Slicing a polyhedron to form polygons.*

for defining one positive or negative polyhedron. Figure 5a shows a Voronoi cell (polyhedron) for a single edge. Figure 5b shows all of the vertex polyhedra of an icosahedron.

Once the Voronoi diagram is constructed, the polyhedra associated with each cell is scan converted in order to associate the closest face, edge or vertex with each voxel for the shortest distance calculation. Each polyhedron is intersected with the planes that coincide with the grid rows to form polygons. This reduces the problem to polygon scan conversion (see Figure 6). For each grid row that intersects the resulting polygon, we find the left and right intersection points and mark each grid point in between as being inside the polygon. The polyhedra that define the Voronoi cells must be enlarged slightly to make sure that grid points are not missed due to finite precision arithmetic. Therefore, some grid points may be scan converted more than once. In this case, the smaller distance, i.e. the closer point, is chosen to produce the correct weak solution to the Eikonal equation. Thus, we use a set of 'generalized' Voronoi cells that are fast to construct but do overlap—unlike true Voronoi cells. This leads to a scan conversion algorithm with an optimal linear computational complexity in the size of the polygonal mesh.

### 3.1.3. Superellipsoids

Superellipsoids are used as modeling primitives and region-of-influence (ROI) primitives for some of our operators. In both cases, a scan-converted representation is needed. The

parametric equation for a superellipsoid is

$$\mathbf{S}(\eta, \omega) = \begin{bmatrix} a_1 \cos^{\epsilon 1}(\eta)\cos^{\epsilon 2}(\omega) \\ a_2 \cos^{\epsilon 1}(\eta)\sin^{\epsilon 2}(\omega) \\ a_3 \sin^{\epsilon 1}(\eta) \end{bmatrix} \tag{23}$$

where $\eta \in [-\pi/2, \pi/2]$ and $\omega \in [-\pi, \pi]$ are the longitudinal and latitudinal parameters of the surface, $a_1$, $a_2$, $a_3$ are the scaling factors in the $X$, $Y$ and $Z$ directions, and $\epsilon 1$ and $\epsilon 2$ define the shape in the longitudinal and latitudinal directions [60].

The distance to a point on the surface of a superellipsoid defined at $[\eta, \omega]$ from an arbitrary point $\mathbf{P}$ is

$$d(\eta, \omega) = ||\mathbf{S}(\eta, \omega) - \mathbf{P}||. \tag{24}$$

Squaring and expanding Equation (24) give

$$\begin{aligned} \hat{d}(\eta, \omega) = &\left(a_1 \cos^{\epsilon 1}(\eta)\cos^{\epsilon 2}(\omega) - P_x\right)^2 \\ &+ \left(a_2 \cos^{\epsilon 1}(\eta)\sin^{\epsilon 2}(\omega) - P_y\right)^2 \\ &+ \left(a_3 \sin^{\epsilon 1}(\eta) - P_z\right)^2. \end{aligned} \tag{25}$$

The closest point to the superellipsoid from an arbitrary point $\mathbf{P}$ can then be calculated by determining the values of $[\eta, \omega]$ which minimize Equation (25). In general, Equation (25) is minimized with a gradient descent technique utilizing variable step-sizes. The values of $[\eta, \omega]$ may then be plugged into Equation (23) to give the closest point on the surface of the superellipsoid, which in turn may be used to calculate the shortest distance.

Finding the values of $\eta$ and $\omega$ at the closest point with a gradient descent technique involves calculating the gradient of Equation (25),

$$\nabla \hat{d} = [\partial \hat{d}/\partial \eta, \partial \hat{d}/\partial \omega]. \tag{26}$$

Unfortunately, superellipsoids have a tangent vector singularity near $[\eta, \omega]$ values that are multiples of $\pi/2$. To overcome this problem, we re-parameterize $\mathbf{S}$ by arc length [54]. Once our steepest descent (on $\hat{d}$) is redefined so that it is steepest with respect to the normalized parameters $(\alpha, \beta)$ we can use the gradient of the re-parameterized $\hat{d}$,

$$\nabla d' = [\partial \hat{d}/\partial \alpha, \partial \hat{d}/\partial \beta], \tag{27}$$

to find the closest point with greater stability. For more details, see [56].

The general formulation of Equation (27) significantly simplifies for values of $\eta$ and $\omega$ near multiples of $\pi/2$. Instead of deriving and implementing these simplifications for all regions of the superellipsoid, the calculation is only performed in the first octant ($0 \le \eta \le \pi/2$, $0 \le \omega \le \pi/2$). Since a superellipsoid is 8-way symmetric, point $\mathbf{P}$ may be reflected into the first octant, the minimization performed and the solution point reflected back into $\mathbf{P}$'s original octant.

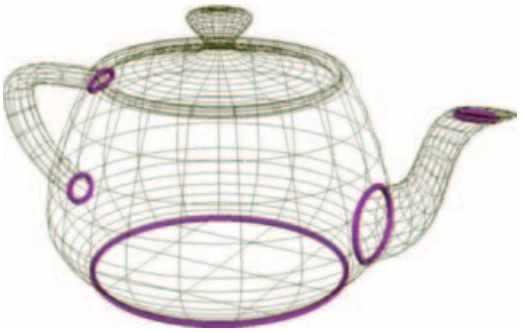

**Figure 7:** *A trimmed NURBS teapot model. The trimming curves remove portions of each surface's domain and maintain topological connectivity between adjacent surfaces.*

It should be noted that for certain values of $\epsilon 1$ and $\epsilon 2$ the normals of a superellipsoid become discontinuous, producing special degenerate primitives that must be dealt with separately. The most common cases are the cuboid ($\epsilon 1 = \epsilon 2 = 0$), and the cylinder ($\epsilon 1 = 0, \epsilon 2 = 1$). The shortest distance to these primitives may be determined by calculating the shortest to each individual face (6 for the cuboid, 3 for the cylinder), and choosing the smallest value.

A faster, but less accurate, alternative for scan-converting any implicit primitive involves utilizing the approximation from Section 3.1.1 at the voxels adjacent to the primitive's surface. Given these voxel values, the distance values at the remaining voxels may be calculated with an FMM [52,53]. (see Section 3.1.6). Also, once shortest distance can be calculated for any closed primitive, distance to a CSG model consisting of combinations of the primitive may also be computed [56].

### 3.1.4. Trimmed NURBS models

A trimmed NURBS model has portions of its domain, and thus portions of the surface, trimmed away [61,62]. The trimming data structure is commonly a piecewise linear curve in the parameter space and a companion piecewise curve in the space of the surface. A set of trimmed surfaces may be joined together into a solid model with topological connectivity maintained by the trimming curves (Figure 7). Our approach to converting a trimmed NURBS model consists of three stages: 1) compute the minimum distance to the Euclidean trimming curves, 2) compute local distance minima to the NURBS surface patches, discarding solutions that lie outside the trimmed domain and 3) perform an inside/outside test on the resulting closest point.

**3.1.4.1. Distance to Trimming Curves** Models typically contain thousands of trimming segments and computing the closest point on these segments is very similar to finding the minimum distance to polygonal models, except the primitives

are line segments instead of triangles. We have modified the publically available PQP package[§], which computes swept sphere volume hierarchies around triangulated models, to use line segments. This reduces the query time for the distance to trimming loops from $\mathcal{O}(N)$ closer to $\mathcal{O}(\log N)$.

**3.1.4.2. Local Distance to NURBS Surfaces** Similar to a superquadric, the distance between a point and parametric surface is described by

$$\mathbf{D}^2(u, v) = ||\mathbf{S}(u, v) - \mathbf{P}||^2. \tag{28}$$

Minimizing Equation (28) corresponds to finding the parameter values of the local closest point on that surface and can be done by finding the simultaneous roots of the partial derivatives of $\mathbf{D}^2$ $(u, v)$,

$$(\mathbf{S}(u, v) - \mathbf{P}) \cdot \mathbf{S}_u = 0 \tag{29}$$

$$(\mathbf{S}(u, v) - \mathbf{P}) \cdot \mathbf{S}_v = 0. \tag{30}$$

We search for local minima in distance until the closest local minimum inside the trimmed domain is found. Multidimensional Newton's method can quickly find a local minimum when given a reasonable starting point. However, Newton's method does not always converge. For robustness, we use Newton's method at multiple starting locations around a potential local minimum. As a preprocess, each original polynomial span of the model's original surfaces are refined into new sub-surfaces. These refined sub-surfaces provide multiple starting locations for each potential minimum.

The algorithm initializes Newton's method by projecting the query point onto the control polygon of the tested surface. This projection is used to compute a surface point using *nodal mapping*. Nodal mapping associates a parameter value (the node) to each control point of that surface piece, and then linearly interpolates between node values using the projection onto the control mesh. Evaluating the surface at this interpolated value produces a first order approximation to the closest point on the surface and a reasonable starting value for Newton's method to improve.

The closest point returned by Newton's method may be outside of the trimmed domain. Again, a modified PQP algorithm for planar line segments is used to find the closest point on the parametric trimming segments of that surface. Valid parametric solutions are to the right of the closest trimming line segment. Average normals are stored at the vertices of the trimming curve in order to correctly perform the inside/outside domain test.

The closest valid point on the surface is compared with the distance to the spatial representation of the trimming loops, and the closest is used as the closest point on the model. If the closest point is on a surface patch, the inside/outside status of the query is determined by dotting the vector from the query point to the closest point with the surface normal at the closest point. The sign of the dot product gives the sign of the distance. If the closest point is on a trimming loop, we use a classic ray shooting algorithm and count the number of model crossings to determine whether the query point is inside or outside.

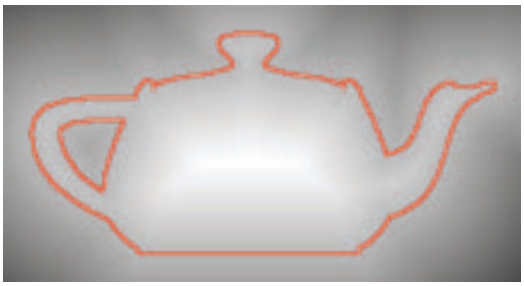

**Figure 8:** *A slice through a 232 × 156 × 124 distance volume of the Utah teapot. The zero level set is highlighted in red.*

**3.1.4.3. Acceleration Techniques** NURBS surfaces have a local convex hull property when the homogeneous coordinate of the control points have positive values. These convex hulls provide a natural means of computing a lower bound on the minimum distance from the query point to the contained portion of surface. We use the GJK package[¶] as a robust implementation of Gilbert's algorithm [63,64] to efficiently compute a lower bound on distance. Refined surfaces with a lower distance bound larger than the current minimum distance cannot contribute a closer point, so they can be ignored.

We note that the minimum distance from one query to the next cannot vary more than the distance between the two query points. We therefore initialize the minimum distance for a new query as the last minimum distance plus the space between query points. This helps to quickly remove surfaces to be tested using the convex hull technique. Finally, the ray intersection inside/outside test can be accelerated by noting that if the last query point was outside, then the new query cannot be inside if the last minimum distance was larger than distance between query points, and vice versa when the last query was inside. This efficiently removes the need to test the sign of many queries. Figure 8 presents a slice from a signed distance volume produced from a trimmed NURBS models containing 20 patches and four trimming curves.

### 3.1.5. Point sets

Some level set editing operators need to determine the closest member in a point set to another arbitrary point. This capability is used during level set blending (when calculating the distance to an intersection 'curve', see Figure 15) and embossing (moving a level set surface toward a point set; see

[§] http://www.cs.unc.edu/∼geom/SSV/

[¶] URL: http://web.comlab.ox.ac.uk/oucl/work/stephen.cameron/distances/

**Figure 9:** *An embossed level set teapot model.*

Figure 9). We utilize the ANN library of Mount and Arya.[||] The library calculates closest point queries of a point set in $\mathcal{O}(\log N)$ time by first storing the point set in a hierarchical data structure that partitions the space around the point set into non-overlapping cells. Given an input point, the hierarchical structure is traversed and candidate cells are identified and sorted [11]. A priority search technique is then utilized to find the closest point (within some tolerance $\epsilon$) in the list of candidate cells [65]. When the points are uniformly distributed, we have found that storing the point set in a K-D tree [10] provides the best performance. For clustered points, storing the point set in the *balanced box decomposition (BBD) tree* described in [11] produces the fastest result.

#### 3.1.6. Fast Marching Method

We utilize an FMM to generate distance volumes when given distance values only at voxels immediately adjacent to the zero level set. This can occur when scan-converting implicit primitives, and generating distance volumes from a level set segmentation [35]. The FMM is also used to calculate the distance values needed for our morphological operators.

The solution of the Eikonal Equation (7) with the boundary condition $\phi|_S = 0$ (a zero level set) is the distance from the manifold $S$. The characteristics of the solution are straight lines which are orthogonal to $S$. We call the direction in which the characteristics propagate the *downwind* direction. More than one characteristic may reach a given point. In this case, the solution is multi-valued. One can obtain a single-valued weak solution by choosing the smallest of the multi-valued solutions at each point. This is a weak solution because $\phi$ is continuous, but not everywhere differentiable. The equation may be efficiently and directly solved by ordering the grid points of the volume, so that information is always propagated in the direction of increasing distance. This is FMM [53]. It achieves a computational complexity of $\mathcal{O}(N \log N)$.

[||] URL: http://www.cs.umd.edu/~mount/ANN

The FMM is similar to Dijkstra's algorithm [66,67] for computing the single-source shortest paths in a weighted, directed graph. In solving this problem, each vertex is assigned a distance, which is the sum of the edge weights along the minimum-weight path from the source vertex. As Dijkstra's algorithm progresses, the status of each vertex is either *known*, *labeled* or *unknown*. Initially, the source vertex in the graph has *known* status and zero distance. All other vertices have *unknown* status and infinite distance. The source vertex labels each of its adjacent neighbors. A *known* vertex labels an adjacent vertex by setting its status to *labeled* if it is *unknown* and setting its distance to be the minimum of its current distance and the sum of the *known* vertices' weight and the connecting edge weight. It can be shown that the labeled vertex with minimum distance has the correct value. Thus the status of this vertex is set to *known*, and it labels its neighbors. This process is repeated until no labeled vertices remain. At this point, all the vertices that are reachable from the source have the correct shortest path distance. The performance of Dijkstra's algorithm depends on quickly determining the labeled vertex with minimum distance. One can efficiently implement the algorithm by storing the labeled vertices in a binary heap. Then the minimum labeled vertex can be determined in $\mathcal{O}(\log n)$ time where $n$ is the number of labeled vertices.

Sethian's FMM differs from Dijkstra's algorithm in that a finite difference scheme (see Equation (36)) is used to label the adjacent neighbors when a grid point becomes known. If there are $N$ grid points, the labeling operations have a computational cost of $\mathcal{O}(N)$. Since there may be at most $N$ labeled grid points, maintaining the binary heap and choosing the minimum labeled vertices make the total complexity $\mathcal{O}(N \log N)$.

### 3.2. Solving the Level Set Equation

Several editing operators modify geometric objects, represented by volume data sets (a 3D grid), by evolving the level set PDE (Equation (8)). As was first noted by Osher and Sethian [1] this PDE can be solved effectively using finite difference (FD) schemes originally developed for Hamilton–Jacobi type equations. The strategy involves discretizing Equation (8) on a regular 3D spatial grid and an adaptive 1D temporal grid. The use of such grids raises a number of numerical and computational issues that are important to the stability, accuracy and efficiency of the implementation. The central issues are the proper choice of a time integration scheme, the spatial discretization of the Hamiltonian (e.g. $\mathcal{F}|\nabla\phi|$) and finally the development of an appropriate narrow band algorithm for localizing computation in the spatial dimensions.

There exists a large number of implicit and explicit integration schemes that can be used to propagate Equation (8) forward in time [68]. The implicit schemes have the advantage of being unconditionally stable with respect to the

time discretization, but typically at the cost of large truncation errors. They also require massive matrix manipulations which make them hard to implement and more importantly increase the computation time per time step. This is in strong contrast to explicit methods like forward Euler or the more accurate TVD Runge-Kutta scheme [69] that are relatively simple to set up and program. Unfortunately, explicit schemes often have stability constraints on their time discretization given a certain space discretization. One exception to this rule is the semi-Lagrangian integration scheme that can be considered an unconditionally stable explicit scheme. However, the semi-Lagrangian scheme is developed specifically for transport (i.e. advection) equations and as such it is unclear how to generalize it to diffusion problems like mean curvature flow used extensively in our level set framework.

It is our experience that for the level set problems considered in this paper, the stability constraints associated with a simple explicit integration scheme like the 'forward Euler method'

$$u^{m+1}_{i,j,k} = u^m_{i,j,k} + \Delta t \, \Delta u^m_{i,j,k} \tag{31}$$

offer a good balance of speed, fast update times and simplicity. In this equation, $u^m$ denotes the approximation of $\phi(\boldsymbol{x}, t)$ at the $m$th discrete time step, $\Delta t$ is a time-increment that is chosen to ensure stability and $\Delta u^m_{i,j,k}$ is the discrete approximation to $\partial\phi/\partial t$ evaluated at grid point $\boldsymbol{x}_{i,j,k}$ and time-step $t_n$. We shall assume, without a loss in generality, that the grid spacing is unity. The initial conditions $u^0$ are established by the scan conversion algorithms discussed in the previous sections and the boundary conditions produce zero derivatives toward the outside of the grid (Neumann type).

The next step expresses the time-increment, $\Delta u^m{}_{i,j,k}$ of Equation (31), in terms of the fundamental level set Equation (8b)

$$\Delta u^m_{i,j,k} = \mathcal{F}_{i,j,k} \left| \nabla u^m_{i,j,k} \right| \tag{32a}$$

$$\approx \mathcal{F}_{i,j,k} \sqrt{\sum_{w \in x,y,z} \left( \delta_w u^m_{i,j,k} \right)^2} \tag{32b}$$

where $\delta_w \, u^m_{i,j,k}$ approximates $\partial u^m_{i,j,k}/\partial w$, i.e. the discretization of the partial derivative of $u$ with respect to the spatial coordinate $w \in x, y, z$. The final step expresses these spatial derivatives as well as the speed function, $\mathcal{F}_{i,j,k}$, in terms of finite differences (FD) on the spatial 3D grid. Many different FD schemes with varying stencil and truncation error exist. For accurate level set simulations, the ENO [70] or WENO [71] schemes are very popular, but we are more concerned with computational efficiently in which case the following simple FD schemes are preferred:

$$\frac{\partial u^m_{i,j,k}}{\partial w} = \delta^+_w u^m_{i,j,k} + \mathcal{O}(\Delta w) \tag{33a}$$

$$= \delta^-_w u^m_{i,j,k} + \mathcal{O}(\Delta w) \tag{33b}$$

$$= \delta^\pm_w u^m_{i,j,k} + \mathcal{O}(\Delta w^2) \tag{33c}$$

where short-hand notations are defined for the following FD expressions:

$$\delta^+_x u^m_{i,j,k} = \frac{u^m_{i+1,j,k} - u^m_{i,j,k}}{\Delta x} \tag{34a}$$

$$\delta^-_x u^m_{i,j,k} = \frac{u^m_{i,j,k} - u^m_{i-1,j,k}}{\Delta x} \tag{34b}$$

$$\delta^\pm_x u^m_{i,j,k} = \frac{u^m_{i+1,j,k} - u^m_{i-1,j,k}}{2\Delta x}. \tag{34c}$$

It is very important to note that the explicit choice of the FD scheme used to discretize the Hamiltonian, $\mathcal{F}|\nabla\phi|$, is highly dependent on the actual functional expression of $\mathcal{F}$. This is a consequence of the fact that the corresponding solutions to the level set PDE with different speed-functions can exhibit very different mathematical behavior. This is formulated more precisely by the CFL condition for the two important classes of level set PDEs, namely hyperbolic and parabolic.

### 3.2.1. Upwind Schemes for Hyperbolic Advection

Two versions of the fundamental level set equation, Equation (8), are

$$\frac{\partial\phi}{\partial t} = V \cdot \nabla\phi = a|\nabla\phi| \tag{35}$$

which correspond to advection (i.e. transport) of a level set surface by a vector field, $V$, or by a scalar, $a$, in the normal direction of the surface. Advection examples are the embossing operator described in [2,72] or constant normal flow when performing surface dilation or erosion. In the case of embossing the level set surface is *advected* in a flow field generated by attraction forces to other geometry like a surface or a set of points. Such advection problems are also common in computational fluid dynamics, and the corresponding *hyperbolic* PDEs have the mathematical property of propagating information in certain *characteristic* directions. The explicit finite difference scheme used for solving the corresponding hyperbolic level set equations should be consistent with the information flow direction. Indeed, this is nothing more than requiring the numerical scheme to obey the underlying 'physics' of the level set surface deformation.

The Courant–Friedrichs–Lewy (CFL) stability condition [73] states that the domain of dependence of the discretized FD problem has to include the domain of dependence of the differential equation in the limit as the length of the FD steps goes to zero. Loosely speaking, this means the stencil used for the FD approximation of the spatial derivatives in Equation (32a) should only include sample points (or more correctly information) from the domain of dependence of the differential equation, i.e. from the side of the zero-crossing opposite to the direction in which it moves— or simply

up-wind to the level set surface. This amounts to using an *up-wind scheme* that employs anisotropic FD like the *single-sided* derivatives in Equations (34a) and (34b). The partial derivatives in the term $|\nabla\phi|$ of Equation (32b) are computed using only those derivatives that are up-wind relative to the movement of the level set. In our initial work, we used the upwind scheme described in [8], but we now use the more accurate Godunov's method [73] which can be expressed in the following compact form:

$$\left(\delta_w u^m\right)^2 = \mathrm{Max}\left(S(\mathcal{F})\delta_w^+ u^m,\ -S(\mathcal{F})\delta_w^- u^m,\ 0\right)^2 \quad (36)$$

where the grid indices $(i, j, k)$ have been omitted for simplicity and $S(\mathcal{F})$ denotes the sign of $\mathcal{F}$. Note that Equation (36) assumes $\mathcal{F}|\nabla\phi|$ to be a convex function.

Another consequence of the CFL condition is that for the numerical FD scheme to be stable, the corresponding numerical wave has to propagate at least as fast as the level set surface. Since the maximum surface motion is defined by the speed-function $\mathcal{F}_{i,j,k}$ and the FD scheme (by definition) propagates the numerical information exactly one grid cell (defined by $\{\Delta x, \Delta y, \Delta z\}$) per time iteration, an upper bound is effectively imposed on the numerical time steps, $\Delta t$ in Equation (31). This can be expressed in a conservative time step restriction

$$\Delta t < \frac{\mathrm{Min}(\Delta x, \Delta y, \Delta z)}{\sup_{i,j,k\in S} |\mathcal{F}_{i,j,k}|} \quad (37)$$

which can be derived by Von Neumann stability analysis [74] assuming $\mathcal{F}$ can be approximated as a linear function in $\phi$.

### 3.2.2. Central-Difference for Parabolic Diffusion

Another fundamental level set equation is the geometric heat equation

$$\frac{\partial\phi}{\partial t} = \alpha K|\nabla\phi| \approx \alpha\nabla^2\phi, \quad (38)$$

where $\alpha$ is a scaling parameter and $K$ is mean curvature, Equation (10b). As discussed in Section 2.2, mean curvature flow corresponds to minimization of surface area, Equation (17b). In our level set framework [2,72], such curvature based flow is used extensively in the blending and smoothing/sharpening operators. If the level set function is normalized to a signed distance function, i.e. $|\nabla\phi| = 1$, the geometric heat equation simplifies to the regular (thermo-dynamic) heat equation as indicated in Equation (38). Thus, the physical interpretation of Equation (38) is diffusion, and the corresponding *parabolic* PDE has a mathematical behavior that is very different from the hyperbolic transport equations in Equation (35). In contrast to the latter, Equation (38) does not propagate information in any particular direction. More specifically, the *parabolic* PDE has no real characteristics associated with it and hence the corresponding solution at a particular time and position depends (in principle) on the previous global solutions. Consequently, parabolic PDEs have infinite domain of dependence and one needs to use ordinary central finite difference schemes to discretize the spatial derivatives. So, for the first-order partial derivatives in Equation (32b), we simply use Equation (34c). Discretization of the mean curvature will be the topic of the next section.

Since parabolic PDEs have infinite mathematical domain of dependence, the propagation speed is also infinite and the CFL stability condition described in the previous section does not apply — or more correctly is not sufficient. Instead, one has to perform a Von Neumann stability analysis [74] on the FD scheme described above. This is an error analysis in Fourier space which leads to the following stability constraint on the time steps,

$$\Delta t < \left(\frac{2\alpha}{\Delta x^2} + \frac{2\alpha}{\Delta y^2} + \frac{2\alpha}{\Delta z^2}\right)^{-1} = \frac{\Delta x^2}{6\alpha}. \quad (39)$$

Hence, we conclude that when discretizing the parabolic Equation (38), on a uniform grid, $\Delta t$ scales as $\mathcal{O}(\Delta x^2)$ which is significantly more stringent than the hyperbolic Equation (37) where $\Delta t$ scales as $\mathcal{O}(\Delta x)$. This is a consequence of the fact that the CFL condition is a necessary, but not always sufficient stability, condition for a numerical FD scheme.

### 3.2.3. Computing Mean Curvature

According to Equation (12b), the mean curvature, appearing in Equation (38), can be expressed as

$$\begin{aligned} K = \frac{1}{2}\nabla\cdot\frac{\nabla\phi}{|\nabla\phi|} = {} & \frac{\phi_x^2(\phi_{yy}+\phi_{zz}) - 2\phi_y\phi_z\phi_{yz}}{2\left(\phi_x^2+\phi_y^2+\phi_z^2\right)^{3/2}} \\ & + \frac{\phi_y^2(\phi_{xx}+\phi_{zz}) - 2\phi_x\phi_z\phi_{xz}}{2\left(\phi_x^2+\phi_y^2+\phi_z^2\right)^{3/2}} \\ & + \frac{\phi_z^2(\phi_{xx}+\phi_{yy}) - 2\phi_x\phi_y\phi_{xy}}{2\left(\phi_x^2+\phi_y^2+\phi_z^2\right)^{3/2}} \end{aligned} \quad (40)$$

using the short-hand notation $\phi_{xy} \equiv \partial^2\phi/\partial x\partial y$. For the discretization, we can use the following second-order central difference schemes

$$\frac{\partial^2 u^m_{i,j,k}}{\partial x^2} = \frac{u^m_{i+1,j,k} - 2u^m_{i,j,k} + u^m_{i-1,j,k}}{\Delta x^2} + \mathcal{O}(\Delta x^2) \quad (41a)$$

$$\begin{aligned} \frac{\partial^2 u^m_{i,j,k}}{\partial x\partial y} = {} & \frac{u^m_{i+1,j+1,k} - u^m_{i+1,j-1,k}}{4\Delta x\Delta y} \\ & + \frac{u^m_{i-1,j-1,k} - u^m_{i-1,j+1,k}}{4\Delta x\Delta y} + \mathcal{O}(\Delta x^2, \Delta y^2). \end{aligned} \quad (41b)$$

We found this direct discretization of mean curvature by central FD to occasionally produced instabilities and small oscillations. As an alternative, we developed a different FD

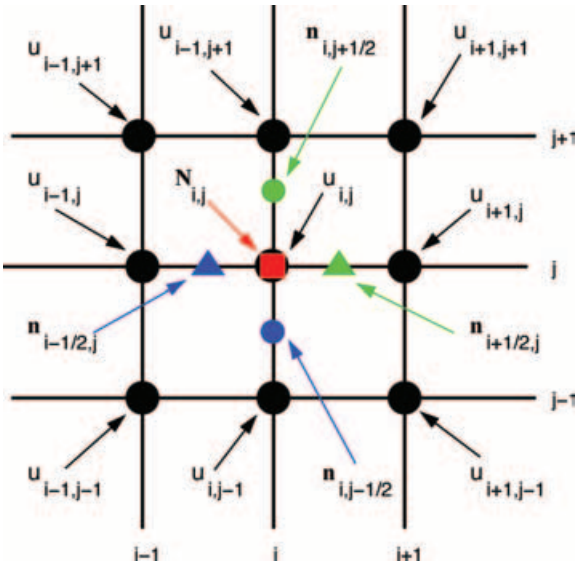


**Figure 10:** *The normal derivative matrix* **N**, *defined in Equation (42), is computed by using the central finite differences of staggered (i.e. not grid-centered) normals. For instance, the normal vector centered at the green triangle is approximated using Equations (44a–c).*

scheme based on staggered normals that proved more stable and also had the added benefit of easily allowing for the computation of other types of curvature. The principle curvatures and principle directions are the eigenvalues and eigenvectors of the *shape matrix* [54]. For an implicit surface, the shape matrix is the derivative of the normalized gradient (surface normals) projected onto the tangent plane of the surface. If we let the normals be $\mathbf{n} = \nabla\phi/|\nabla\phi|$, the derivative of this is the $3 \times 3$ matrix

$$\mathbf{N} \equiv \nabla \otimes \boldsymbol{n} = \left[\frac{\partial \boldsymbol{n}}{\partial x} \quad \frac{\partial \mathbf{n}}{\partial y} \quad \frac{\partial \mathbf{n}}{\partial z}\right]^T \tag{42}$$

where $\otimes$ is the exterior (tensor) product. The projection of this derivative matrix onto the tangent plane gives the shape matrix $\mathbf{B} = \mathbf{N}(I - \boldsymbol{n} \otimes \boldsymbol{n})$. The eigenvalues of the matrix $\mathbf{B}$ are $k_1$, $k_2$ and zero, and the eigenvectors are the principle directions and the normal, respectively. Because the third eigenvalue is zero, we can express mean curvature as $\mathrm{Tr}[\Delta]/2$ where $\boldsymbol{\Delta}$ denotes the diagonalization of the shape matrix, $\mathbf{B}$. However, since the trace of a matrix is invariant under orthogonal transformations we also have

$$K = \frac{1}{2}\mathrm{Tr}[\mathbf{B}] = \frac{1}{2}\mathrm{Tr}\left[\nabla \otimes \boldsymbol{n}(I - \boldsymbol{n} \otimes \boldsymbol{n})\right]. \tag{43}$$

We directly compute Eq. (43) by the method of *differences of normals* [75,76] in lieu of central differences. This approach computes normalized gradients at staggered grid points and takes the difference of these staggered normals to get centrally located approximations to **N**. (see Figure 10). The shape matrix **B** is computed with gradient estimates based on central differences. The resulting curvatures are treated as speed functions (motion in the normal direction), and the associated gradient magnitude is computed using central-difference (i.e. Equation (34c)). For instance, the normal vector centered at the green triangle in Figure 10 is approximated using the following first-order difference expressions

$$\frac{\partial u^m_{i+\frac{1}{2},j,k}}{\partial x} = \delta_x^+ u^m_{i,j,k} + \mathcal{O}(\Delta x) \tag{44a}$$

$$\frac{\partial u^m_{i+\frac{1}{2},j,k}}{\partial y} = \frac{1}{2}\left(\delta_y u^m_{i+1,j,k} + \delta_y u^m_{i,j,k}\right) + \mathcal{O}(\Delta y) \tag{44b}$$

$$\frac{\partial u^m_{i+\frac{1}{2},j,k}}{\partial z} = \frac{1}{2}\left(\delta_z u^m_{i+1,j,k} + \delta_z u^m_{i,j,k}\right) + \mathcal{O}(\Delta z) \tag{44c}$$

which involve the six nearest neighbors (18 in 3D).

### 3.2.4. Sparse-Field Narrow Band Method

In the original level set formulation of [1], the fundamental PDE, Eq. (8), is solved in the full embedding space of the surface leading to a computational complexity of $\mathcal{O}(n^3)$, where $n$ is the side length of a bounding volume. This sub-optimal computational complexity was improved with the introduction of a *narrow band* method in [7] which restricted (most) computations to a thin band of active voxels immediately surrounding the interface, thus reducing the time complexity to $\mathcal{O}(n^2)$. This work was later followed up with more advanced narrow band methods like the accurate scheme of [9] and the fast but approximate sparse fields method of [8]. These narrow band methods all rely on the assumption that only one level set solution (typically the zero level set) is of interest, which is exactly the case in our editing framework. Since our focus is on interactivity rather then accuracy, we found the latter sparse fields method to be most useful. Below we briefly outline the main ideas behind this narrow band method and refer the reader to [8] for more details and experimental validations. We will present the sparse field algorithm in 2D and simply note that the generalization to 3D (or higher) is trivial.

Figure 11 illustrates the discrete sampling of a level set, $u_{i,j}$, for a closed (black) curve on a uniform 2D grid. If we assume the grid size to be one, we can define layers, $L_n$, of grid points embedding the curve

$$L_n = \left\{(i,j) \,\middle|\, u_{i,j} \in \left[n - \frac{1}{2}, n + \frac{1}{2}\right]\right\} \tag{45}$$

where we have made use of the fact that $u_{i,j}$ is (initially) defined as a signed Euclidean distance function to the curve. Convenient data structures for these layers are linked lists. This is illustrated in Figure 11 for the zero crossing layer, $L_0$, and the first inner layer, $L_{-1}$, shaded, respectively, green and red. It then follows that the set of $2N + 1$ layers, $L_n$, $n = 0$, $\pm 1$, $\pm 2, \ldots, \pm N$ effectively defines a narrow band of grid

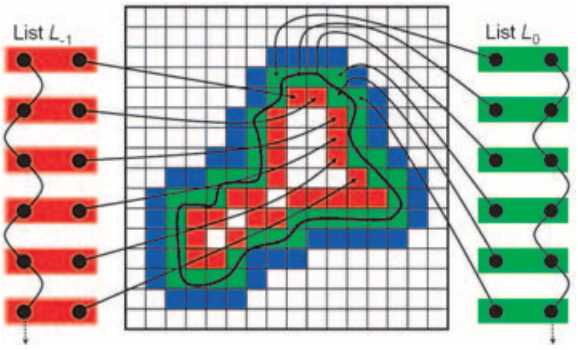


**Figure 11:** *Linked-list data structures provide efficient access to those grid points with values and status that must be updated.*

points with values $[-N - \frac{1}{2}, N + \frac{1}{2}]$ embedding the curve. The number of layers in the narrow band should be determined by the footprint of the finite difference stencils used to calculate derivatives. Since our editing framework extensively uses mean curvature, it follows from the discussion in Section 3.2.3 that at least five layers are necessary (2 inside layers, 2 outside layers and the zero crossing layer).

The sparse field method approximately solves the level set equation, Equation (8), only in the narrow band of $2N + 1$ layers in a self-consistent way, i.e. Equation (45) should remain valid after each iteration of the time integration. In fact, to improve speed and preserve normalization (i.e. $|\nabla\phi| = 1$), the sparse field method only explicitly solves Equation (8) in $L_0$ using velocity extension, Equation (18), and then uses simple 'city-block' distances to update $\phi$ in the remaining layers. To improve accuracy, it further employs the first-order accurate distance approximation in Equation (22). (See discussions related to Equations 19 and 21). For the temporal and spatial discretizations, we use Equations (36) and (37) with advection (hyperbolic speed functions) whereas Equations (34c) and (39) are used for diffusion (e.g. parabolic mean curvature flow). As grid points in a layer $L_n$ pass out of the range $[n - \frac{1}{2}, n + \frac{1}{2}]$, they are removed and other neighboring grid points are added. The overall structure of the sparse field method is outlined in Algorithm 1.

```
foreach (i, j) ∈ L_0 do
    compute F_{i,j} using Eq. (18) and Eq. (22);
end
compute Δt using Eq. (37) or Eq. (39);
foreach (i, j) ∈ L_0 do
    compute |∇u^m_{i,j}| using Eq. (34c) or Eq. (36);
    u^{m+1}_{i,j} = u^m_{i,j} + Δt F_{i,j} |∇u^m_{i,j}|;
    if u^{m+1}_{i,j} ∉ [-1/2, 1/2] then add to list S_0;
end
foreach L_n, n = ±1, ±2, ..., ±N do
    foreach (i, j) ∈ L_n do
        find closest neighbor (i', j') ∈ L_{n∓1};
        u^{m+1}_{i,j} = u^{m+1}_{i',j'} ± 1;
        if _{i,j} ∉ [n - 1/2, n + 1/2] then add to S_n;
    end
end
foreach S_n, n = ±1, ±2, ..., ±N do
    foreach (i, j) ∈ S_n do
        remove (i, j) from L_n;
        if |n| < N then
            add (i, j) to L_{n-1} or L_{n+1};
        else if |u^{m+1}_{i,j}| < N - 1/2 then
            add (i, j) to L_{N-1} or L_{-N+1};
            add to L_n all neighbors ∉ narrow band;
        end
    end
end
```

***Algorithm 1:*** *Pseudo-code for the sparse-fields method of* [8]. *As illustrated in Figure 11, $L_n$ denotes linked lists of grid points in the nth layer of the narrow band and $S_n$ are simply auxiliary lists of grid points that change layer. An implementation of this algorithm is available in the VISPACK library at http://www.cs.utah.edu/~whitaker/vispack.*

### 3.2.5. Level Set Subvolumes

One of the most effective techniques for increasing interactivity in our level set editing system involves restricting computations to a subregion of the volume data set. This is feasible because many of the editing operators by their very nature are local. The selection of the proper subvolume during the editing process is implemented with grid-aligned bounding boxes. Having the bounding boxes axis-aligned makes them straightforward to compute and manipulate, and having them grid-aligned guarantees that intersections directly correspond to valid subvolumes. The bounding box position and size are based on the geometric primitive, e.g. superellipsoid, triangle mesh or point set, utilized by a particular operator.

Employing bounding boxes within the local level set editing operators (blending, smoothing, sharpening and embossing) significantly lessens the computation time during the editing process. These operators are defined by speed functions ($\mathcal{F}()$) that specify the speed of the deformation on the surface. For the smoothing, sharpening and embossing operators, the user specifies the portion of the model to be edited by positioning a region-of-influence (ROI) primitive. The speed function is defined to be zero outside of the ROI primitive. During a blending operation, a set of intersection voxels (those containing both surfaces being blended) are identified and blending only occurs within a user-specified distance of these voxels. The speed function is zero beyond this distance. In both cases, no level set computation is needed in the outer regions. Given the ROI primitive and the distance information from the set of intersection voxels, a grid/

**Figure 12:** *Volume renderings (left & center) of a winged, two-headed dragon created by merging pieces from a griffin and dragon model. A physical model (right) manufactured from the level set model.*

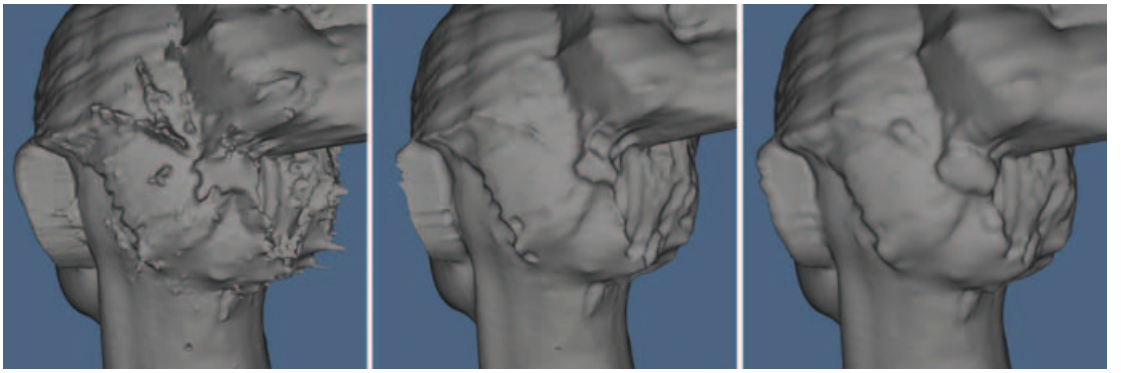

**Figure 13:** *Applying a morphological opening to a laser scan reconstruction of a human head (left). The opening performs global smoothing by removing protruding structures smaller than a user-defined value d. First, an offset surface a distance d inwards (erosion) is created (center). Then the signed distance is computed to this d level set using the Fast Marching Method [53] and next it is used to define an offset surface distance d outwards (dilation) to produce the smoothed result (right).*

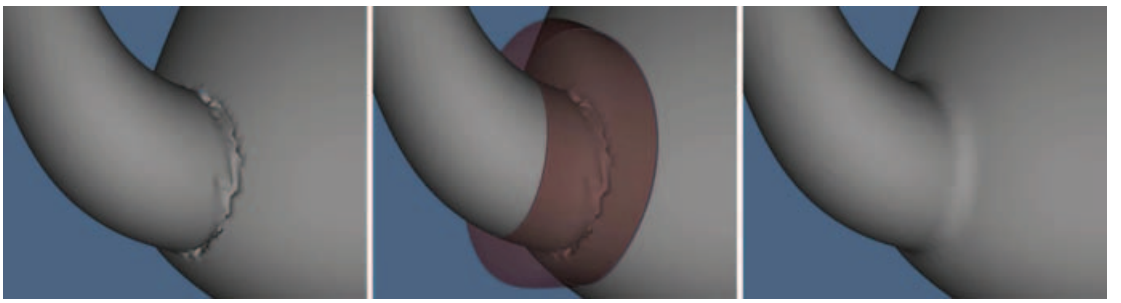

**Figure 14:** *Scan conversion errors near the teapot spout (left). These errors were produced by an early pre-debugged version of our software. Placing a (red) superellipsoid around the errors (middle). The errors are smoothed away with a level set smoothing operator (right).*

axis-aligned bounding box that contains only those regions where the speed function is non-zero can be defined. A subvolume is 'carved' out from the complete model by performing a CSG intersection operation with the signed distance field associated with the bounding box and the model's volume. The resulting subvolume is then passed to the level set solver, and inserted back into the model's volume after processing.

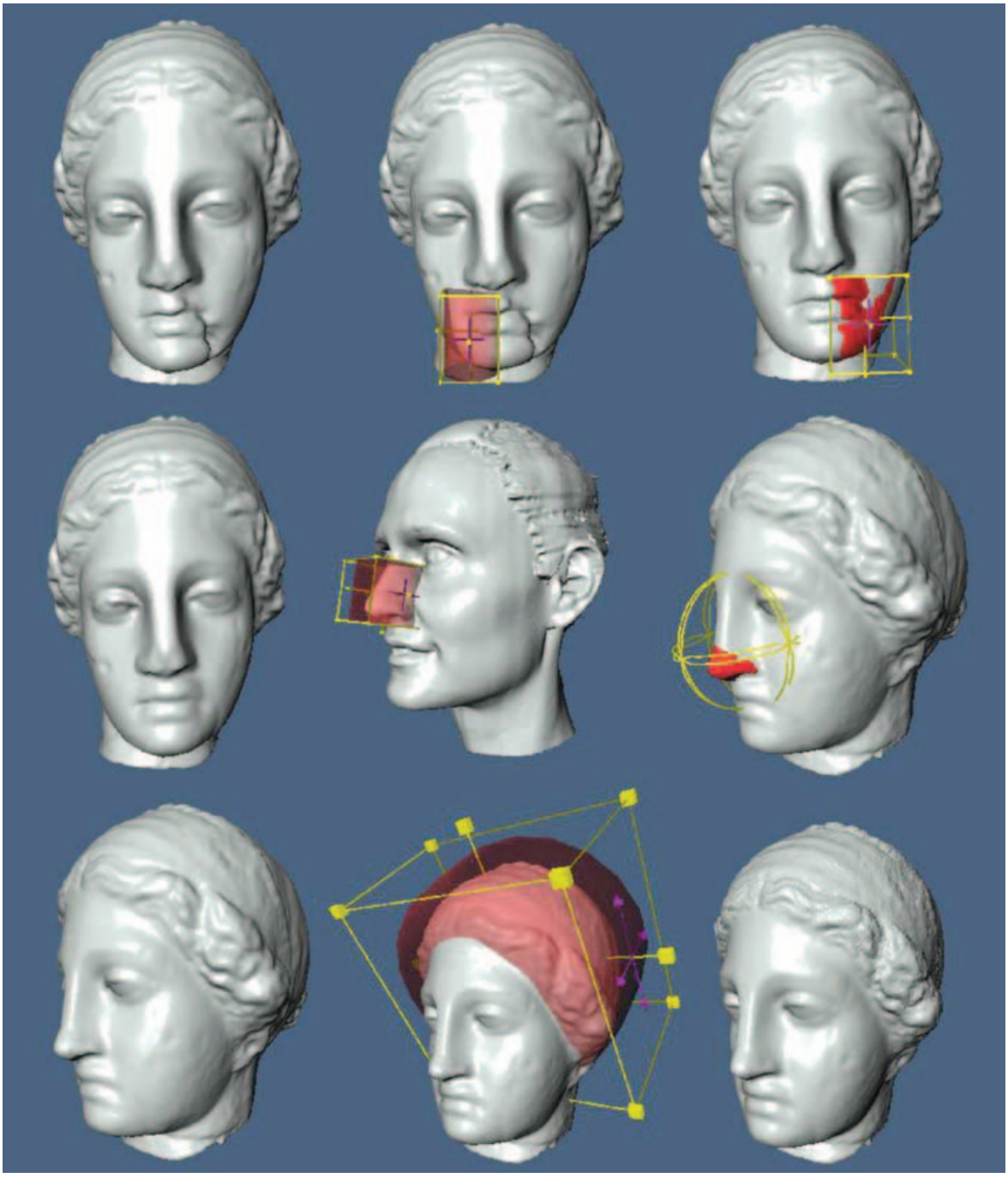

**Figure 15:** *Repairing a Greek bust. The right cheek is first copied, mirrored, pasted and blended back onto the left side of the bust. Next a nose is copied from a human head model, scaled and blended onto the broken nose of the Greek bust. Finally the hair of the bust is chiseled by a localized sharpening operation.*

### 3.3. Efficient Mesh Extraction

As indicated by the green box in Figure 1, level set surfaces may be rendered either directly by means of ray casting or indirectly by a simple two-step procedure (a polygonal mesh is extracted from the volume data set and displayed on graphics hardware). We have successfully tested both (see Figures 12 and 15) and found the latter to perform and scale better with

the size of our volumes. Implementing a few straightforward mesh extraction procedures makes the overhead of the indirect rendering approach insignificant. Conventional graphics hardware is then capable of providing interactive frame-rates for all of the models presented in this paper.

#### 3.3.1. Fast Marching Cubes

Much work has been presented over the years on improving the quality of the triangle meshes extracted from volume data sets, the fundamental data structure of level set models [77–79]. However, these improvements come at a cost, and sacrifice speed for improved mesh structure. Fortunately, the simplicity of the original Marching Cubes (MC) algorithm [80] allows us to easily optimize mesh extraction in the level set editing system.

The first optimization relies on the fact that level set models are represented by a signed distance field. This allows us to easily leap-frog through the volume as opposed to marching through the entire volume. An effective implementation of this idea increments the innermost loop in the triple-nested for-loop of the MC algorithm by the distance of the current voxel value (i.e. floor $|u_{i,j,k}|$). While more sophisticated space-pruning schemes can certainly be designed, we found this straightforward step balances the potential complexity of leap-frogging and the relatively fast triangulation table look-up of the MC algorithm.

Another variation of the MC algorithm that works effectively with our level set models utilizes the sparse-field representation presented in Section 3.2.4. Since the sparse-field method implements a narrow-banded distance field with a linked list of active voxels, we know at each step which voxels contain the level set of interest. The list is traversed and only those voxels needed to generate the MC mesh are processed.

#### 3.3.2. Incremental Mesh Extraction

Even though the procedures described so far significantly improve the original MC algorithm, they still do not make our indirect rendering approach truly interactive. Fortunately, there are other algorithms that can be employed to achieve the goal of interactive rendering of the deforming level set surfaces. Mesh extraction can be significantly accelerated by incrementally updating the mesh only in regions where the level set surface changes.

We start by making the following observations about the bounding boxes introduced in Section 3.2.5. First, the definition of the speed functions that utilize bounding boxes guarantees that the mesh outside of the bounding boxes is unchanged after a local editing operation. Second, the bounding boxes are by definition grid-aligned and all vertices of an MC mesh lie, by construction, on grid edges. These observations lead to the following incremental mesh extraction algorithm.

**Table 1:** *Distribution of algorithms used in each module in our interactive level set model editing system.*

| Modules \ Algorithms | Distance Calculations | Scan Conversion | Closest Point in Set | Fast Marching Methods | Bounding Boxes | Numerical Integration | Narrow Band Methods |
|---|---|---|---|---|---|---|---|
| Input Model Generation | X | X | | X | | | X |
| CSG Operations | | | | | | | X |
| LS Blending | | | X | | X | X | X |
| LS Smoothing/Sharpening | X | | | | X | X | X |
| LS Embossing | X | | X | | X | X | X |
| Morphological Operations | | | | X | | | X |
| Mesh Extraction | | | | | X | | X |

Given a complete global mesh, we first trim away all triangles with vertices inside a bounding box. Next, for each subsequent iteration of the level set calculation, new triangles are only extracted from the sub-volume defined by the bounding box. The resulting new triangles are then incrementally added to the trimmed mesh, which by construction properly connect without the need for additional triangle clipping.

Given the collection of these procedures, the mesh of the deforming level set surface may be interactively displayed while the level set equation is being iteratively solved, allowing the user to view the evolving surface and terminate processing once a desired result is achieved.

## 4. System Modules

Table 1 identifies the specific algorithms utilized in each of the modules in our interactive level set model editing system. Since a wide variety of geometric models may be imported into our system, many algorithms are needed to perform the necessary conversions, including shortest distance calculations (Sections 3.1.3, 3.1.4), scan conversion (Section 3.1.2) and the Fast Marching Method (Section 3.1.6). All of the level set deformation operators (blending, smoothing, sharpening and embossing) use bounding boxes (Section 3.2.5), numerical integration (Section 3.2) and the sparse-field techniques (Section 3.2.4). The blending and embossing operators use K–D trees (Section 3.1.5) to quickly find closest points. The smoothing, sharpening and embossing operators utilize shortest distance calculations (Section 3.1.3) for localizing computation. The morphological operators employ the Fast Marching Method (Section 3.1.6) to calculate the needed distance information. Our mesh extraction algorithm also extensively utilizes bounding boxes and the active list of the level set solver to implement an incremental version of the Marching Cubes algorithm [80]. All of the modules use some

kind of narrow band calculation to either limit computation to only those voxels near the level set of interest (Section 3.2.4), or to re-establish proper distance information in the narrow band after performing its operation (Section 3.1.1).

## 5. Results

We have produced numerous models with our level set editing system. The teapot (NURBS surface), dragon (scanned volume), human head and bust (polygonal surfaces) and eyelet on the winged dragon's back (superquadric) in Figures 9, 12, 13, 15 demonstrate that we are able to import several types of models into our system. The CSG operators with blending were utilized to produce the winged, double-headed dragon and repaired bust in Figures 12 and 15. The images of the dragon are volume rendered and were interactively produced by VTK's Volview program utilizing TeraRecon's VolumePro 1000 volume rendering hardware. The smoothing operator is used to fix problems in a model produced by an early, unfinished version of the NURBS scan conversion code in Figure 14. The embossing operator produced the result in Figure 9. The results of our morphological operators [4] are presented in Figure 13. It should be noted that the images in Figure 15 are screen shots from an interactive editing session with our system, running on a Linux PC with an AMD Athlon 1.7 GHz processors. All of the following timing information is produced on this computer.

A level set editing session, as illustrated in Figure 15, begins by first importing a level set model into our system. The process of generating an initial level set model, e.g. with scan conversion, is not incorporated into the system. It is considered a separate preprocessing step. Once a model** is brought into the system, it and the tools to modify it may be interactively (at $\sim$30Hz) manipulated and viewed. Once a level set editing operation (e.g. blending, smoothing, embossing and opening) is invoked, an iterative computational process modifies the model. After each iteration, the current state of the model is displayed, allowing the user to stop the operation, once a desired result is produced. We have found that most operations need approximately 10 iterations to produce a satisfactory result. Each iteration takes approximately 1/2 to 1 second on an AMD Athlon 1.7 GHz – this includes level set evolution, mesh extraction and display. Therefore most level set operations take 5 – 10 seconds to complete. The CSG operations are not iterative and require less than 1 second of computation time. These computation times provide an environment that allows a user to quickly specify an operation, and then wait just a few seconds for it to complete. Our system includes an undo facility, giving the user the ability to rapidly try numerous editing operations until the best result is found.

** The models in this paper are represented by volume data sets with a resolution of approximately $256^3$.

## 6. Conclusions

This paper has described the collection of techniques and algorithms (some new, some pre-existing) needed to create an interactive editing system for level set models. It has summarized the algorithms for producing level set input models and, more importantly, for localizing/minimizing computation during the editing process. These algorithms include distance calculations, scan conversion, closest point determination, fast marching methods, bounding box creation, incremental and fast mesh extraction, numerical integration and narrow band techniques. Together, these algorithms provide the capabilities required for the interactive editing of level set models.

In the near future, we plan to implement our editing framework with the advanced and compact level set data structure of Nielsen and Museth [81]. This will allow us for the first time to edit high-resolution level set surfaces.

## Acknowledgments

We would like to thank Alan Barr and the other members of the Caltech Computer Graphics Group for their assistance and support. Additional thanks to Katrine Museth and Santiago Lombeyda for assistance with the figures and Jason Wood for developing useful visualization tools. The teapot model and the manufactured dragon figurine were provided by the University of Utah's Geometric Design and Computation Group. The Greek bust and human head models were provided by Cyberware, Inc. The dragon and griffin models were provided by the Stanford Computer Graphics Laboratory and Caltech's Multires Modeling Group. This work was financially supported by National Science Foundation grants ASC-8920219, ACI-9982273, ACI-0083287 and ACI-0089915, and subcontract B341492 under DOE contract W-7405-ENG-48 as well as the Swedish Research Council (Grant# 617-2004-5017). Finally, we would like to thank the anonymous reviewers of this paper.

## References

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

2. R. Whitaker and D. Breen. Level-set models for the deformation of solid objects. In *The Third International Workshop on Implicit Surfaces*, Eurographics, pp. 19–35, 1998.

3. R. Courant, K. O. Friedrichs and H. Lewy. Uber die partiellen differenzengleichungen der mathematisches physik. *Math. Ann.,* 100, pp. 32–74, 1928.

4. J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, New York, 1982.

5. G. Sapiro and A. Tannenbaum. Affine invariant scale space. *International Journal of Computer Vision,* 11, pp. 25–44, 1993.

6. P. Maragos. Differential morphology and image processing. *IEEE Trans. on Image Processing* 5(6):922–937, June 1996.

7. D. Adalsteinsson and J. Sethian. A fast level set method for propagating interfaces. *Journal of Computational Physics*, pp. 269–277, 1995.

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

9. 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, pp. 410–438, 1999.

10. M. de Berg, M. van Kreveld, M. Overmars and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, Berlin, 1997.

11. S. Arya, D. Mount, N. Netanyahu, R. Silverman and A. Wu. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM,* 45, pp. 891–923, 1998.

12. C. Hoffmann. *Geometric and Solid Modeling*. Morgan Kaufmann, 1989.

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

14. T. Galyean and J. Hughes. Sculpting: An interactive volumetric modeling technique. In *Proc. SIGGRAPH '91*, pp. 267–274, July 1991.

15. S. Wang and A. Kaufman. Volume sculpting. In *1995 Symposium on Interactive 3D Graphics*, pp. 151–156, 1995.

16. R. Avila and L. Sobierajski. A haptic interaction method for volume visualization. In *Proc. IEEE Visualization '96*, pp. 197–204, 1996.

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

18. B. Cutler, J. Dorsey, L. McMillan, M. Müller and R. Jagnow. A procedural approach to authoring solid models. *ACM Trans. on Graphics (Proc. SIGGRAPH),* 21(3):302–311, July 2002.

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

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

21. R. MacCracken and K. Roy. Free-form deformations with lattices of arbitrary topology. In *Proc. SIGGRAPH '96*, pp. 181–188, 1996.

22. R. Perry and S. Frisken. Kizamu: A system for sculpting digital characters. In *Proc. SIGGRAPH,* 2001, pp. 47–56, August 2001.

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

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

25. S. Frisken, R. Perry, A. Rockwood and T. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proc. SIGGRAPH 2000,* pp. 249–254, 2000.

26. J. Bloomenthal *et al.* (Eds.). *Introduction to Implicit Surfaces*. Morgan Kaufmann, 1997.

27. 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, June 1999.

28. A. Raviv and G. Elber. Three-dimensional freeform sculpting via zero sets of scalar trivariate functions. *Computer-Aided Design,* 32, pp. 513–526, August 2000.

29. M. Casale and E. Stanton. An overview of analytic solid modeling. *IEEE Computer Graphics and Applications,* 5(2):45–56, February 1985.

30. J. Sethian. *Level Set Methods and Fast Marching Methods*, second ed. Cambridge University Press, Cambridge, UK, 1999.

31. G. Sapiro. *Geometric Partial Differential Equations and Image Analysis*. Cambridge University Press, Cambridge, UK, 2001.

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

33. S. Osher and N. Paragios (Eds.). *Geometric Level Set Methods in Imaging, Vision and Graphics*. Springer, New York, 2003.

34. R. Malladi, J. Sethian and B. Vemuri. Shape modeling with front propagation. *IEEE Trans. PAMI,* 17(2):158–175, 1995.

35. R. Whitaker, D. Breen, K. Museth and N. Soni. Segmentation of biological datasets using a level-set framework. In *Volume Graphics 2001*, Chen M., Kaufman A., (Eds.). Springer, Vienna, 2001, pp. 249–263.

36. D. Breen, R. Whitaker, K. Museth and L. Zhukov. Level set segmentation of biological volume datasets. In *Handbook of Medical Image Analysis, Volume I: Segmentation Part A*, Suri J., (Ed.). Kluwer, New York, 2005, ch. 8, pp. 415–478.

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

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

39. K. Museth, D. Breen, L. Zhukov, R. Whitaker. Level set segmentation from multiple non-uniform volume datasets. In *Proc. IEEE Visualization 2002,* pp. 179–186, October 2002.

40. H.-K. Zhao, S. Osher and R. Fedkiw. Fast surface reconstruction using the level set method. In *Proc. 1st IEEE Workshop on Variational and Level Set Methods,* pp. 194–202, 2001.

41. J. Baerentzen and N. Christensen. Volume sculpting using the level-set method. In *Proc. Shape Modeling International Conference,* pp. 175–182, 2002.

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

43. M. Rumpf and R. Strzodka. Level set segmentation in graphics hardware. In *Proc. ICIP '01,* vol. 3, pp. 1103–1106, 2001.

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

45. M. Rumpf and R. Strzodka. Nonlinear diffusion in graphics hardware. In *Proc. EG/IEEE TCVG Symposium on Visualization,* pp. 75–84, 2001.

46. Z. Bolz, I. Farmer, E. Grinspun and P. Schröder. Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. *ACM Trans. on Graphics (Proc. SIGGRAPH),* 22(3):917–924, July 2003.

47. A. Sherbondy, M. Houston and S. Napel. Fast volume segmentation with simultaneous visualization using programmable graphics hardware. In *Proc. IEEE Visualization 2003,* pp. 171–176, 2003.

48. M. Hopf, T. Ertl. Accelerating morphological analysis with graphics hardware. In *Proc. Workshop on Vision, Modeling and Visualization,* pp. 337–345, 2000.

49. R. Yang and G. Welch. Fast image segmentation and smoothing using commodity graphics hardware. *Journal of Graphics Tools,* 7(4):91–100, 2002.

50. S. Fang and H. Chen. Hardware accelerated voxelization. *Computers & Graphics,* 24(3):433–442, June 2000.

51. C. Sigg, R. Peikert and M. Gross. Signed distance transform using graphics hardware. In *Proc. IEEE Visualization 2003,* pp. 83–90, 2003.

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

53. J. Sethian. A fast marching level set method for monotonically advancing fronts. In *Proceedings of the National Academy of Science*, vol. 93 of 4, pp. 1591–1595, 1996.

54. M. Do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, Englewood Cliffs, NJ, 1976.

55. T. F. Chan and L. A. Vese. *Image segmentation using level sets and the piecewise-constant Mumford-Shah model*. Tech. rep., UCLA CAM Report 00-14, 2000.

56. D. Breen, S. Mauch and R. Whitaker. 3D scan conversion of CSG models into distance, closest-point and colour volumes. In *Volume Graphics*, Chen M., Kaufman A., Yagel R., (Eds.). Springer, London, pp. 135–158, 2000.

57. H. Zhao. Fast sweeping method for Eikonal equations. *Mathematics of Computation,* 74, pp. 603–627, 2004.

58. S. Mauch. *A Fast Algorithm for Computing the Closest Point and Distance Transform*. Tech. Rep. 077, California Institute of Technology, ASCI, 2000.

59. S. Mauch. *Efficient Algorithms for Solving Static Hamilton–Jacobi Equations*. PhD thesis, California Institute of Technology, Pasadena, California, 2003.

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

61. M. Shantz and S.-L. Chang. Rendering trimmed NURBS with adaptive forward differencing. In *Proc. SIGGRAPH '88,* pp. 189–198, 1988.

62. L. Piegl and W. Tiller. Geometry-based triangulation of trimmed NURBS surfaces. *Computer-Aided Design,* 30, pp. 11–18, 1998.

63. E. Gilbert, D. Johnson and S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Trans. on Robotics and Automation,* 4(2):193–203, April 1988.

64. S. Cameron. Enhancing GJK: Computing minimum penetration distances between convex polyhedra. In *Proc. Int. Conf. On Robotics and Automation,* pp. 3112–3117, 1997.

65. S. Arya and D. Mount. Algorithms for fast vector quantization. In *Proc. IEEE Data Compression Conference,* pp. 381–390, 1993.

66. E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik,* 1, pp. 333–352, 1959.

67. T. Cormen, C. Leiserson, R. Rivest and C. Stein. *Introduction to Algorithms, Second Edition*. MIT Press, Cambridge, MA, 2001.

68. R. Burden and J. Faires. *Numerical Analysis, Seventh Edition*. Brooks/Cole Publishing, Pacific Grove, CA, 2001.

69. C. Shu and S. Osher. Efficient implementation of essentially non-oscillatory shock capturing schemes. *Journal of Computional Physics,* 77, pp. 439–471, 1988.

70. S. Osher and C. Shu. High-order essentially nonoscillatory schemes for hamilton-jacobi equations. *SIAM Journal on Numerical Analysis,* 28, pp. 907–922, 1991.

71. X. Liu, S. Osher and T. Chan. Weighted essentially nonoscillatory schemes. *Journal of Computational Physics,* 115, pp. 200–212, 1994.

72. K. Museth, D. Breen and R. Whitaker. Editing geometric models. In *Geometric Level Set Methods in Imaging, Vision and Graphics,* Osher S., Paragios N., (Eds.) Springer, New York, 2003, ch. 23, pp. 441–460.

73. E. Rouy and A. Tourin. A viscosity solutions approach to shape-from-shading. *SIAM J. Num. Anal.,* 29, pp. 867–884, 1992.

74. J. Strikwerda. *Finite Difference Schemes and Partial Differential Equations*. Wadsworth Publishing Co., Belmont, CA, 1989.

75. L. Rudin, S. Osher and C. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D,* 60, pp. 259–268, 1992.

76. R. Whitaker and X. Xue. Variable-conductance, level-set curvature for image denoising. In *Proc. IEEE International Conference on Image Processing,* pp. 142–145, October 2001.

77. Z. Wood, M. Desbrun, P. Schröder and D. Breen. Semi-regular mesh extraction from volumes. In *Proc. IEEE Visualization 2000,* pp. 275–282, 2000.

78. M. Gavriliu, J. Carranza, D. Breen and A. Barr. Fast extraction of adaptive multiresolution meshes with guaranteed properties from volumetric data. In *Proc. IEEE Visualization 2001,* pp. 295–302, 2001.

79. L. P. Kobbelt, M. Botsch, U. Schwanecke and H.-P. Seidel. Feature sensitive surface extraction from volume data. In *Proc. SIGGRAPH,* pp. 57–66, 2001.

80. W. Lorensen and H. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *Proc. SIGGRAPH,* pp. 163–169, July 1987.

81. M. B. Nielsen and K. Museth. Dynamic Tubular Grid: An efficient data structure and algorithms for high resolution level sets. *Linköping Electronic Articles in Computer and Information Science,* 9(001: Accepted for publication in *Journal of Scientific Computing*, 2004.