Chapter 1
Isosurfaces and Level Sets

1.1 Introduction

1.1.1 Motivation
This chapter describes mechanisms for processing isosurfaces. The underlying philosophy is to use isosurfaces as a modeling technology that can serve as an alternative to parameterized models for a variety of important applications in visualization and computer graphics. Level-set methods [1] rely on partial differential equations (PDEs) to model deforming isosurfaces. These methods have applications in a wide range of fields such as visualization [2, 3], scientific computing [4], computer graphics [5, 6], image processing [7, 8] and computer vision [9]. Several nice overviews of the technology and applications are also available [10, 4].

This chapter reviews the mathematics and numerical techniques for describing the geometry of isosurfaces and manipulating their shapes in prescribed ways. Its starts with a basic introduction to the notation and fundamental concepts and then presents the geometry of isosurfaces. We then describe the method of level sets, i.e., moving isosurfaces, and present the mathematical and numerical methods they entail.

1.1.2 Representing Surfaces With Volumes
The level set method relies on an implicit representation of a surface, using a scalar function

$$\phi : U \xrightarrow{x, y, z} \mathbb{R},$$

(1.1)

where $U \subset \mathbb{R}^3$ is the domain of the volume (and the range of the surface model). Thus, a surface $S$ is

$$S = \{x | \phi(x) = k\}.$$  

(1.2)

The choice of $k$ is arbitrary, and $\phi$ is sometimes called the embedding. The surface $S$ is sometimes referred to as an isosurface of $\phi$. Notice that surfaces defined in this way divide $U$ into a clear inside and outside—such surfaces are always closed wherever they do not
Figure 1.1: A scalar field (show here in 2D) contains a nested family of isosurfaces, each with a difference grey scale value.

intersect the boundary of the domain. The embedding, $\phi$, is represented on a discrete grid, i.e. a volume.

The normal of an isosurface is given by the normalized gradient vector. Typically, we identify a surface normal with a point in the volume domain $\mathcal{D}$. That is

$$
n(x) = \frac{\nabla \phi(x)}{\left|\nabla \phi(x)\right|} \text{ where } x \in \mathcal{D}.
$$

The convention regarding the direction of this vector is arbitrary; the negative of the normalized gradient magnitude is also normal to the isosurface. The gradient vector points toward that side of the isosurface which has greater values (i.e. brighter). When rendering, the convention is to use outward pointing normals, and the sign of the gradient must be adjusted accordingly. However, for most applications any consistent choice of normal vector will suffice. On a discrete grid, one must also decide how to approximate the gradient vector (i.e., first partial derivatives). In many cases central differences will suffice. However, in the presence of noise, especially when volume rendering, it is sometimes helpful to compute first derivatives using some smoothing filter (e.g., convolution with a Gaussian [11, 12]). When using the normal vector to solve certain kinds of partial differential equations, it is sometimes necessary to approximate the gradient vector with discrete, one-sided differences, as discussed in successive sections.

Note that a single volume contains families nested isosurfaces, arranged like the layers of an onion, as in Fig. 1.1. We specify the normal to an isosurface as a function of the position within the volume. That is, $n(x)$ is the normal of the (single) isosurface that passes through the point $x$. The $k$ value associated with that isosurface is $\phi(x)$.

The curvature of the isosurface can be computed from the first- and second-order structure of the embedding, $\phi$. All of the isosurface shape information is contained field of normals given by $n(x)$. The $3 \times 3$ matrix of derivatives of this vector,

$$
N = \begin{bmatrix}
    n_x & n_y & n_z
\end{bmatrix}
$$

(1.4)
The projection of this derivative onto the tangent plane of the isosurface gives the shape matrix, $\beta$. Let $P$ denote normal projection operator, which is defined as

$$
P = n \otimes n = \frac{1}{|\nabla \phi|^2} \begin{pmatrix}
\phi_z^2 & \phi_z \phi_y & \phi_z \phi_x \\
\phi_y \phi_z & \phi_y^2 & \phi_y \phi_x \\
\phi_x \phi_z & \phi_x \phi_y & \phi_x^2
\end{pmatrix}.
$$

(1.5)

The tangential projection operator is $I - P$, and thus the shape matrix is

$$
\beta = NT = TD^2 \phi T,
$$

(1.6)

where $D^2 \phi$ is the Hessian of $\phi$. The shape matrix $\beta$ has 3, real, eigenvalues which are

$$
e_1 = k_1, e_2 = k_2, e_3 = 0.
$$

(1.7)

The corresponding eigenvectors are the principle directions (in the tangent plane) and the normal, respectively.

The *mean curvature* is the mean of the two principal curvatures, which is one half of the trace of $\beta$, which is equal to the trace of $N$: The total curvature, also called the deviation from flatness [13], $D_1$ is the root sum of squares of the two principal curvatures, which is the Euclidean norm of the matrix $\beta$.

Notice, these measures exist at every point in $U$, and at each point they describe the geometry of the particular isosurface that passes through that point. All of these quantities can be computed on a discrete volume using finite differences, as described in successive sections.

### 1.2 Deformable Surfaces

This section begins with mathematics for describing surface deformations on parametric models. The result is an evolution equation for a surface. Each of the terms in this evolution equation can be re-expressed in a way that is independent of the parameterization. Finally, the evolution equation for a parametric surface gives rise to an evolution equation (differential equation) on a volume, which encodes the shape of that surface as a level set.

#### 1.2.1 Surface Deformation

A regular surface $S \subset \mathbb{R}^3$ is a collection of points in 3D that can be be represented locally as a continuous function. In geometric modeling a surface is typically represented as a two-parameter object in a three-dimensional space, i.e., a surface is local a mapping $S$:

$$
S : V \times V \rightarrow \mathbb{R}^3
$$

(1.8)

where $V \times V \subset \mathbb{R}^2$. A deformable surface exhibits some motion over time. Thus $S = S(r, s, t)$, where $t \in \mathbb{R}^+$. We assume second-order-continuous, orientable surfaces; therefore at every
point on the surface (and in time) there is surface normal \( \mathbf{N} = \mathbf{N}(\mathbf{r}, \mathbf{s}, t) \). We use \( \mathcal{S}_t \) to refer to the entire set of points on the surface.

Local deformations of \( \mathbf{S} \) can be described by an evolution equation, i.e., a differential equation on \( \mathbf{S} \) that incorporates the position of the surface, local and global shape properties, and responses to other forcing functions. That is,

\[
\frac{\partial \mathbf{S}}{\partial t} = \mathbf{G} (\mathbf{S}, \mathbf{S}_r, \mathbf{S}_s, \mathbf{S}_{rr}, \mathbf{S}_{rs}, \mathbf{S}_{ss}, \ldots)
\]  

(1.9)

where the subscripts represent partial derivatives with respect to those parameters. The evolution of \( \mathbf{S} \) can be described by a sum of terms that depends on both the geometry of \( \mathbf{S} \) and the influence of other functions or data.

There are a variety of differential expressions that can be combined for different applications. For instance, the model could move in response to some directional “forcing” function [14, 15], \( \mathbf{F} : \mathbf{U} \mapsto \mathbb{R}^3 \). Alternatively, the surface could expand and contract with a spatially-varying speed, \( \mathbf{G} : \mathbb{R}^3 \mapsto \mathbb{R} \). The surface motion could also depend on the surface shape, e.g. curvature. There are myriad terms that depend on both the differential geometry of the surface and outside forces or functions to control the evolution of a surface.

The method of level sets, proposed by Osher and Sethian [16] and described extensively in [10, 4], provides the mathematical and numerical mechanisms for computing surface deformations as time-varying iso-values of \( \phi \) by solving a partial differential equation on the 3D grid. That is, the level-set formulation provides a set of numerical methods that describe how to manipulate the greyscale values in a volume, so that the isosurfaces of \( \phi \) move in a prescribed manner (shown in Figure 1.2).

We denote the movement of a point \( \mathbf{x} \) on the deforming surface as \( \mathbf{v} \), and we assume that this motion can be expressed in terms of the position of \( \mathbf{x} \in \mathbf{U} \) and the geometry of the surface at that point. In this case, there are generally two options for representing such surface movements implicitly:

Figure 1.2: Level-set models represent curves and surfaces implicitly using greyscale images: a) an ellipse is represented as the level set of an image, b) to change the shape we modify the greyscale values of the image.
Figure 1.3: In the case of the static level-set approach, Eq. (1.10), a single, static embedding, \( \phi \), captures the motion of a model in its various level values.

**Static:** A single, static \( \phi(x) \) contains a family of level sets corresponding to surfaces as different times \( t \) (as in Fig. 1.3). That is,

\[
\phi(x(t)) = k(t) \Rightarrow \nabla \phi(x) \cdot v = \frac{dk(t)}{dt}.
\]  

(1.10)

To solve this static method requires constructing a \( \phi \) that satisfies Eq. (1.10). This is a boundary value problem, which can be solved somewhat efficiently starting with a single surface using the fast marching method of Sethian [10]. This representation has some significant limitations, however, because (by definition) a surface cannot pass back over itself over time, i.e., motions must be strictly monotonic — inward or outward.

**Dynamic:** The approach is to use a one-parameter family of embeddings, i.e., \( \phi(x, t) \) changes over time, \( x \) remains on the \( k \) level set of \( \phi \) as it moves, and \( k \) remains constant. The behavior of \( \phi \) is obtained by setting the total derivative of \( \phi(x(t), t) = k \) to zero. Thus,

\[
\phi(x(t), t) = k \Rightarrow \frac{\partial \phi}{\partial t} = -\nabla \phi \cdot v.
\]  

(1.11)

This approach can accommodate models that move forward and backward and cross back over their own paths (over time). However, to solve this requires solving the initial value problem (using finite forward differences) on \( \phi(x, t) \) — a potentially large computational burden. The remainder of this discussion focuses on the dynamic case, because of its flexibility.

All surface movements depend on position and geometry, and the level-set geometry is expressed in terms of the differential structure of \( \phi \). Therefore the dynamic formulation from
Eq. (1.11) gives a general form of the partial differential equation on $\phi$:

$$
\frac{\partial \phi}{\partial t} = -\nabla \phi \cdot \mathbf{v} = -\nabla \phi \cdot \mathbf{F}(\mathbf{x}, D\phi, D^2\phi, \ldots),
$$

(1.12)

where $D^n\phi$ is the set of order-$n$ derivatives of $\phi$ evaluated at $\mathbf{x}$. Because this relationship applies to every level-set of $\phi$, i.e. all values of $k$, this equation can be applied to all of $\phi$, and therefore the movements of all the level-set surfaces embedded in $\phi$ can be calculated from Equation 1.12.

The level-set representation has a number of practical and theoretical advantages over conventional surface models, especially in the context of deformation and segmentation. First, level-set models are topologically flexible, they can easily represent complicated surface shapes that can, in turn, form holes, split to form multiple objects, or merge with other objects to form a single structure. This topological flexibility can be visualized (for curves) by imagining the model as the boundary of a lake whose shoreline has a fixed altitude (as in Fig. 1.4). In order to change the shape of the shoreline, one must modify the height of the surrounding terrain (which is $\phi$). As the topography of the terrain changes—new hills for or hills join or split—the topology of the shoreline can change. That is, the model can form new holes, pieces can split off, or separate pieces can join.

Level-set models can also incorporate many (millions) of degrees of freedom, and therefore they can accommodate complex shapes. Indeed, the shapes formed by the level sets of $\phi$ are restricted only by the resolution of the sampling. Thus, there is no need to reparameterize the model as it undergoes significant deformations.

Such level-set methods are well documented in the literature [16, 17] for applications such as computational physics [18], image processing [19, 20], computer vision [21, 8], medical image analysis [7, 8], and 3D reconstruction [22, 23]. For instance, in computational physics level-set methods are a powerful tool for modeling moving interfaces between different materials (see Osher and Fedkiw [18] for a nice overview of recent results). Examples are water-air and water-oil boundaries. In such cases, level-set methods can be used to compute deformations that minimize surface area while preserving volumes for materials that split and merge in arbitrary ways. The method can be extended to multiple, non-overlapping objects.
Level-set methods have also been shown to be effective in extracting surface structures from biological and medical data. For instance Malladi et al.[8] propose a method in which the level-sets form an expanding or contracting contour which tends to “cling” to interesting features in 2D angiograms. At the same time the contour is also influenced by its own curvature, and therefore remains smooth. Whitaker et al.[7, 24] have shown that level sets can be used to simulate conventional deformable surface models, and demonstrated this by extracting skin and tumors from thick-sliced (e.g. clinical) MR data, and by reconstructing a fetal face from 3D ultrasound. A variety of authors [25, 26, 20, 27] have presented variations on the method and presented results for 2D and 3D data. Sethian [10] gives several examples of level-set curves and surface for segmenting CT and MR data.

1.3 Numerical Methods

By taking the strategy of embedding surface models in volumes, we have converted equations that describe the movement of surface points to nonlinear, partial differential equations defined on a volume, which is generally a rectilinear grid. The expression \( \phi^n_{i,j,k} \) refers to the \( n \)th time step at position \( i, j, k \), which has an associated value in the 3D domain of the continuous volume \( \phi(x_i,y_j,z_k) \).

The discretization of these equations raises two important issues. First is the availability of accurate, stable numerical schemes for solving these equations. Second is the problem of computational complexity and the fact that we have converted a surface problem to a volume problem, increasing the dimensionality of the domain over which the evolution equations must be solved.

Various level-set terms can be combined, based on the needs of the application, to create a partial differential equation on \( \phi(x,t) \). The solutions to these equations are computed using finite differences. Along the time axis solutions are obtained using finite forward differences, beginning with an initial model (i.e., volume) and stepping sequentially through a series of discrete times steps (which are denoted as superscripts on \( \phi_{i,j,k} \)). Thus the update equation is:

\[
\phi^{n+1}_{i,j,k} = \phi^n_{i,j,k} + \Delta t \Delta \phi^n_{i,j,k},
\]

The term \( \Delta \phi^n_{i,j,k} \) is a discrete approximation to \( \partial \phi / \partial t \), which is a sum of terms that depend on derivatives of \( \phi \). These terms must, in turn, be approximated using finite differences on the volume grid.

1.3.1 Up-wind Schemes

The terms that drive the deformation of the surface fall into two basic categories: the first-order terms and the second-order terms (items 3 through 5). The first-order terms describe a moving wave front with a space-varying velocity (expression 1) or speed (expression 2). Equations of this form cannot be solved with a simple finite forward difference scheme. Such schemes tend to overshoot, and they are unstable. To address this issue Osher and Sethian [1] have proposed an up-wind scheme. The up-wind method relies on a one-sided
Figure 1.5: The up-wind numerical scheme uses one-sided derivatives to prevent overshooting and the creation of new level sets.

derivative that looks in the up-wind direction of the moving wave front, and thereby avoids the over-shooting associated with finite forward differences.

To describe the numerical implementation, we use the notation for finite difference operators introduced in Chapter ???. Second-order terms are computed using the tightest-fitting central difference operators. For example,

\[
\begin{align*}
\delta_{xx} \phi_{i,j,k} & = \frac{(\phi_{i+1,j,k} + \phi_{i-1,j,k} - 2\phi_{i,j,k})}{h^2}, \\
\delta_{zz} \phi_{i,j,k} & = \frac{(\phi_{i,j,k+1} + \phi_{i,j,k-1} - 2\phi_{i,j,k})}{h^2}, \quad \text{and} \\
\delta_{xy} \phi_{i,j,k} & = \delta_x \delta_y \phi_{i,j,k}
\end{align*}
\]  

(1.14) (1.15) (1.16)

The discrete approximation to the first-order terms \( \delta \phi \) are computed using the up-wind proposed by Osher and Sethian [16]. This strategy avoids overshooting by approximating the gradient of \( \phi \) using a one-sided differences in the direction that is up-wind of the moving level-set thereby ensuring that no new contours are created in the process of updating \( \phi_{i,j,k}^{n} \) (as depicted in Figure 1.5). The scheme is separable along each axis (i.e., \( x, y, \) and \( z \)).

Consider the vector-valued advection term \( F(x) \). If we use superscripts to denote the vector components, i.e.,

\[ F(x, y, z) = (F^{(x)}(x, y, z), F^{(y)}(x, y, z), F^{(z)}(x, y, z)), \]

(1.17)

the up-wind calculation for a grid point \( \phi_{i,j,k}^{n} \) is

\[
F(x_i, y_j, z_k) \cdot \nabla \phi(x_i, y_j, z_k, t) \approx \sum_{q \in \{x, y, z\}} F^{(q)}(x_i, y_j, z_k) \left\{ \begin{array}{ll}
\delta_q^+ \phi_{i,j,k}^{n} & F^{(q)}(x_i, y_j, z_k) > 0 \\
\delta_q^- \phi_{i,j,k}^{n} & F^{(q)}(x_i, y_j, z_k) < 0
\end{array} \right.
\]

(1.18)

The time steps are limited—the fastest moving wave front can move only one grid unit per iteration. That is

\[
\Delta t_F \leq \frac{1}{\sum_{q \in \{x, y, z\}} \sup_{i,j,k \in X} \{ |\nabla F^{(q)}(x_i, y_j, z_k)| \}}.
\]

(1.19)
For speed terms of the form $G(\mathbf{x})\mathbf{n}(\mathbf{x})$ the direction of the moving surface depends on the normal, and therefore the same up-wind strategy is applied in a slightly different form.

$$G(x_i, y_j, z_k) |\nabla \phi(x_i, y_j, z_k)| \approx \sum_{q \in \{x,y,z\}} G(x_i, y_i, z_i) \begin{cases} \max^2(\delta^+_q \phi_{i,j,k}^n, 0) + \min^2(\delta^-_q \phi_{i,j,k}^n, 0) & G(x_i, y_i, z_i) > 0 \\ \min^2(\delta^+_q \phi_{i,j,k}^n, 0) + \max^2(\delta^-_q \phi_{i,j,k}^n, 0) & G(q)(x_i, y_i, z_i) < 0 \end{cases}$$

(1.20)

The time steps are, again, limited by the fastest moving wave front:

$$\Delta t_G \leq \frac{\sup_{i,j,k \in X} \{ |\nabla G(x_i, y_j, z_k)| \}}{3}$$

(1.21)

To compute approximation the update to the second-order terms, one typically uses central differences. Thus, the mean curvature is approximated as:

$$H_{i,j,k}^n = \frac{1}{2} \left( \left( \delta_x \phi_{i,j,k}^n \right)^2 + \left( \delta_y \phi_{i,j,k}^n \right)^2 + \left( \delta_z \phi_{i,j,k}^n \right)^2 \right) - \frac{1}{2} \left[ \left( \delta_x \phi_{i,j,k}^n \right)^2 + \left( \delta_y \phi_{i,j,k}^n \right)^2 + \left( \delta_z \phi_{i,j,k}^n \right)^2 \right] \delta_{xx} \phi_{i,j,k}^n$$

(1.22)

Such curvature terms can also be computing by calculating the normals at staggered grid locations and then taking finite differences of the normals, as described in [28]. In some cases this is advantageous—but the details are beyond the scope of this paper.

The time steps are limited, for stability, to

$$\Delta t_H \leq \frac{1}{6}$$

(1.23)

When combining terms, the maximum time steps for each terms is scaled by one over the weighting coefficient for that term.

### 1.3.2 Narrow-Band Methods

If one is interested in only a single level set, the formulation described previously is not efficient. This is because solutions are usually computed over the entire domain of $\phi$. The solutions, $\phi(x, y, z, t)$ describe the evolution of an embedded family of contours. While this dense family of solutions might be advantageous for certain applications, there are other applications that require only a single surface model. In such applications the calculation of solutions over a dense field is an unnecessary computational burden, and the presence of contour families can be a nuisance because further processing might be required to extract the level set that is of interest.

Fortunately, the evolution of a single level set, $\phi(x, t) = k$, is not affected by the choice of embedding. The evolution of the level sets is such that they evolve independently (to within the error introduced by the discrete grid). Furthermore, the evolution of $\phi$ is important only
in the vicinity of that level set. Thus, one should perform calculations for the evolution of $\phi$ only in a neighborhood of the surface $S = \{x | \phi(x) = k\}$. In the discrete setting, there is a particular subset of grid points whose values control a particular level set (see Figure 1.6). Of course, as the surface moves, that subset of grid points must change to account for its new position.

Adalsteinson and Sethian [29] propose a narrow-band approach which follows this line of reasoning. The narrow-band technique constructs an embedding of the evolving curve or surface via a signed distance transform. The distance transform is truncated, i.e., computed over a finite width of only $m$ points that lie within a specified distance to the level set. The remaining points are set to constant values to indicate that they do not lie within the narrow band, or tube as they call it. The evolution of the surface (they demonstrate it for curves in the plane) is computed by calculating the evolution of $\phi$ only on the set of grid points that are within a fixed distance to the initial level set, i.e. within the narrow band. When the evolving level set approaches the edge of the band (see Figure 1.7), they calculate a new distance transform and a new embedding, and they repeat the process. This algorithm relies on the fact that the embedding is not a critical aspect of the evolution of the level set. That is, the embedding can be transformed or recomputed at any point in time, so long as such a transformation does not change the position of the $k$th level set, and the evolution will be unaffected by this change in the embedding.

Despite the improvements in computation time, the narrow-band approach is not optimal for several reasons. First it requires a band of significant width ($m = 12$ in the examples of [29]) where one would like to have a band that is only as wide as necessary to calculate the derivatives of $\phi$ near the level set (e.g. $m = 2$). The wider band is necessary because the narrow-band algorithm trades off two competing computational costs. One is the cost of stopping the evolution and computing the position of the curve and distance transform (to sub-cell accuracy) and determining the domain of the band. The other is the cost of computing the evolution process over the entire band. The narrow-band method also requires additional techniques, such as smoothing, to maintain the stability at the boundaries of the

Figure 1.6: A level curve of a 2D scalar field passes through a finite set of cells. Only those grid points nearest to the level curve are relevant to the evolution of that curve.
band, where some grid points are undergoing the evolution and nearby neighbors are static.

1.3.3 The Sparse-Field Method

The basic premise of the narrow band algorithm is that computing the distance transform is so costly that it cannot be done at every iteration of the evolution process. The sparse-field method [23] uses an approximation to the distance transform that makes it feasible to recompute the neighborhood of the level-set model at each time step. Computation of the evolution equation is computed on a band of grid points that is only one point wide. The embedding is extended from the active points to a neighborhood around those points that is precisely the width needed at each time. This extension is done via a fast distance transform approximation.

This approach has several advantages. First, the algorithm does precisely the number of calculations needed to compute the next position of the level curve. It does not require explicitly recalculating the positions of level sets and their distance transforms. Because the number of points being computed is so small, it is feasible to use a linked-list to keep track of them. Thus, at each iteration the algorithm visits only those points adjacent to the k-level curve. For large 3D data sets, the very process of incrementing a counter and checking the status of all of the grid points is prohibitive.

The sparse-field algorithm is analogous to a locomotive engine that lays down tracks before it and picks them up from behind. In this way the number of computations increases with the surface area of the model rather than the resolution of the embedding. Also, the sparse-field approach identifies a single level set with a specific set of points whose values control the position of that level set. This allows one to compute external forces to an accuracy that is better than the grid spacing of the model, resulting in a modeling system that is more accurate for various kinds of “model fitting” applications.
The sparse-field algorithm takes advantage of the fact that a $k$-level surface, $S$, of a discrete image $\phi_{i,j,k}$ (of any dimension) has a set of cells through which it passes, as shown in Figure 1.6. The set of grid points adjacent to the level set is called the active set, and the individual elements of this set are called active points. As a first-order approximation, the distance of the level set from the center of any active point is proportional to the value of $\phi$ divided the gradient magnitude at that point. Because all of the derivatives (up to second order) in this approach are computed using nearest neighbor differences, only the active points and their neighbors are relevant to the evolution of the level-set at any particular time in the evolution process. The strategy is to compute the evolution given by equation 1.12 on the active set and then update neighborhood around the active set using a fast distance transform. Because active points must be adjacent to the level-set model, their positions lie within a fixed distance to the model. Therefore the values of $\phi$ for locations in the active set must lie within a certain range. When active-point values move out of this active range they are no longer adjacent to the model. They must be removed from the set and other grid points, whose values are moving into the active range, must be added to take their place. The precise ordering and execution of these operations is important to the operation of the algorithm.

The values of the points in the active set can be updated using the up-wind scheme for first-order terms and central differences for the mean-curvature flow, as described in the previous sections. In order to maintain stability, one must update the neighborhoods of active grid points in a way that allows grid points to enter and leave the active set without those changes in status affecting their values. Grid points should be removed from the active set when they are no longer the nearest grid point to the zero crossing. Because the embedding $\phi$ is a discrete approximation to the distance transform of the model, the distance of a particular grid point, $x_m = (i, j, k)$, to the level set is given by the value of $\phi$ at that grid point. If the distance between grid points is defined to be unity, then we should remove a point from the active set when the value of $\phi$ at that point no longer lies in the interval $[-\frac{1}{2}, \frac{1}{2}]$ (see Figure 1.8). If the neighbors of that point maintain their distance of 1, then those neighbors will move into the active range just $x_m$ is ready to be removed.

There are two operations that are significant to the evolution of the active set. First, the values of $\phi$ at active points change from one iteration to the next. Second, as the values of active points pass out of the active range they are removed from the active set and other, neighboring grid points are added to the active set to take their place. In [23] the author gives some formal definitions of active sets and the operations that affect them, which show that active sets will always form a boundary between positive and negative regions in the image, even as control of the level set passes from one set off active points to another.

Because grid points that are near the active set are kept at a fixed value difference from the active points, active points serve to control the behavior of non-active grid points to which they are adjacent. The neighborhoods of the active set are defined in layers, $L_{i+1}, \ldots L_{i+N}$ and $L_{i-1}, \ldots L_{i-N}$, where the $i$ indicates the distance (city block distance) from the nearest active grid point, and negative numbers are used for the outside layers. For notational convenience the active set is denoted $L_0$. 

12
Figure 1.8: The status of grid points and their values at two different points in time show that as the zero crossing moves, activity is passed one grid point to another.

The number of layers should coincide with the size of the footprint or neighborhood used to calculate derivatives. In this way, the inside and outside grid points undergo no changes in their values that affect or distort the evolution of the zero set. Most of the level-set work relies on surface normals and curvature, which require only second-order derivatives of \( \phi \). Second-order derivatives are calculated using a \( 3 \times 3 \times 3 \) kernel (city-block distance 2 to the corners). Therefore only five layers are necessary (2 inside layers, 2 outside layers, and the active set). These layers are denoted \( L_1, L_2, L_{-1}, L_{-2}, \) and \( L_0 \).

The active set has grid point values in the range \( [-\frac{1}{2}, \frac{1}{2}] \). The values of the grid points in each neighborhood layer are kept 1 unit from the next layer closest to the active set (as in Figure 1.8). Thus the values of layer \( L_i \) fall in the interval \( [i - \frac{1}{2}, i + \frac{1}{2}] \). For \( 2N + 1 \) layers, the values of the grid points that are totally inside and outside are \( N + \frac{1}{2} \) and \( -N - \frac{1}{2} \), respectively. The procedure for updating the image and the active set based on surface movements is as follows:

1. For each active grid point, \( x_m = (i, j, k) \), do the following:

   (a) Calculate the local geometry of the level set.

   (b) Compute the net change of \( \phi_{x_m} \), based on the internal and external forces, using some stable (e.g., up-wind) numerical scheme where necessary.

2. For each active grid point \( x_j \) add the change to the grid point value and decide if the new value \( \phi_{x_m}^{n+1} \) falls outside the \( [-\frac{1}{2}, \frac{1}{2}] \) interval. If so, put \( x_m \) on lists of grid points that are changing status, called the status list; \( S_1 \) or \( S_{-1} \), for \( \phi_{x_m}^{n+1} > 1 \) or \( \phi_{x_m}^{n+1} < -1 \), respectively.

3. Visit the grid points in the layers \( L_i \) in the order \( i = \pm 1, \ldots \pm N \), and update the grid point values based on the values (by adding or subtracting one unit) of the next inner layer, \( L_{i+1} \). If more than one \( L_{i+1} \) neighbor exists then use the neighbor that indicates a level curve closest to that grid point, i.e., use the maximum for the outside layers.
and minimum for the inside layers. If a grid point in layer \( L_i \) has no \( L_{i+1} \) neighbors, then it gets demoted to \( L_{i-1} \), the next level away from the active set.

4. For each status list \( S_{\pm 1}, S_{\pm 2}, \ldots, S_{\pm N} \) do the following:

   (a) For each element \( x_j \) on the status list \( S_i \), remove \( x_j \) from the list \( L_{i+1} \), and add it to the \( L_i \) list, or, in the case of \( i = \pm (N + 1) \), remove it from all lists.

   (b) Add all \( L_{i+1} \) neighbors to the \( S_{i \pm 1} \) list.

This algorithm can be implemented efficiently using linked-list data structures combined with arrays to store the values of the grid points and their states as shown in Figure 1.9. This requires only those grid points whose values are changing, the active points and their neighbors, to be visited at each time step. The computation time grows as \( m^{n-1} \), where \( m \) is the number of grid points along one dimension of \( \phi \) (sometimes called the resolution of the discrete sampling). Computation time for dense-field approach increases as \( m^n \). The \( m^{n-1} \) growth in computation time for the sparse-field models is consistent with conventional (parameterized) models, for which computation times increase with the resolution of the surface, rather than the volume in which it resides.

Another important aspect of the performance of the sparse-field algorithm is the larger time steps that are possible. The time steps are limited by the speed of the “fastest” moving level curve, i.e., the maximum of the force function. Because the sparse-field method calculates the movement of level sets over a subset of the image, time steps are bounded from below by those of the dense-field case, i.e.,

\[
\sup_{x \in A \subset X} (g(x)) \leq \sup_{x \in X} (g(x)),
\]

(1.24)
where $g(x)$ is the space varying speed function and $\mathcal{A}$ is the active set.

Results from previous work [23] have demonstrated several important aspects of the sparse-field algorithm. First, the manipulations of the active set and surrounding layers allow the active set to “track” the deformable surface as it moves. The active set always divides the inside and outside of the objects it describes (i.e., it stays closed). Empirical results show significant increases in performance relative to both the computation of full domain and the narrow-band method, as proposed in the literature. Empirical results also show that the sparse-field method is about as accurate as both the full, discrete solution, and the narrow-band method. Finally, because the method positions level sets to sub-voxel accuracy it avoids aliasing problems and is more accurate than these other methods when it comes to fitting level-set models to zero sets of scalar or vector-valued functions. This sub-voxel accuracy is important aspect of the implementation, and will significantly impact the quality of the results for the applications that follow.

### 1.4 Applications

There a variety of applications of level sets to problems in image processing, vision, graphics, computational physics, and visualization. The following sections present a pair of applications from the literature that elucidate some of the technical aspects of level sets.

#### 1.4.1 Segmentation

This section describes a mechanism by which level-set models can be used to segment volumes. It starts with a description of the speed function and the associated parameters and then shows results on MRI datasets.

Level-set models, like the snakes technology that preceded them, can lock onto edges using second-derivative information from the input data. In this way, they can be used to find Canny type edges [30] or Marr-Hildreth [31] edges in a region of interest (specified by the initialization), while also introducing a smoothness term. If we let the model move uphill on the gradient magnitude of the input, this gives:

$$
\phi_t = \alpha |\nabla I| + \beta I \nabla \phi |H, 
$$

(1.25)

where $I$ is the input data. Even with the appropriate level of smoothing for $\nabla I$ this strategy requires the initial model to be near edges in the input image in order for the second derivatives of those edges to pull the surface model in the correct direction. This can be alleviated if we add another speed term that that encourages the model to flow into regions with greyscale values that lie within a specified intensity range. This gives

$$
\phi_t = \alpha |\nabla \phi |H + \beta |\nabla I| \cdot \nabla \phi + \gamma |\nabla \phi| (\epsilon - |I - T|),
$$

(1.26)

where $T$ controls the brightness of the region to be segmented and $\epsilon$ controls the range of greyscale values around $T$ that could be considered inside the object. Thus when the model
lies on a voxel with a greyscale level between \( T - \epsilon \) and \( T + \epsilon \), the model expands and otherwise it contracts. This speed term is gradual, and thus the effects of the \( D \) diminish as the model approaches the boundaries of regions whose greyscale levels lie within the \( T \pm \epsilon \) range. This second speed term is a simple approximation to a one-dimensional statistical classifier, which assumes a single density (with noise) for the regions of interest. However, it allows very poor initializations to deform a long way in order to fill regions of interest.

With this combined edge- and region-based scheme, a user must specify four free parameters, \( T, \epsilon, \alpha, \) and \( \gamma \), as well as an initialization.

If a user were to initialize a model in a volume and use only the region-based term, i.e. \( \alpha = \beta = 0 \), the results would be the same as a simple flood fill over the region bounded by the upper and lower thresholds. However, the inclusion of the curvature and edge terms alleviate the critical *leaking* problem that arises when using flood filling as a segmentation technique. The leaking effect is particularly acute in 3D segmentations and is easily demonstrated on a brain tumor data set, as shown in Fig. 1.11. Figure 1.12 shows the results of using this strategy to segment the cortex, white matter, and a brain tumor from an MRI dataset.

### 1.4.2 Surface Processing

The proliferation of high-resolution volume and surface measurement technologies has generated an interested in producing a set of tools for processing surfaces which mimic the wide range of tools available for processing images or volumes. One strategy for generalizing image processing algorithms to surfaces is via the *surface normal vectors*. Thus a smooth surface is one that has smoothly varying normals. Penalty functions on the surface normals typically give rise to fourth-order partial differential equations (PDE). Tasdizen et al. proposed processing level sets in this way using a two step approach: (i) operating on the normal map of a surface while letting the surface shape to lag the normals, and (ii) manipulating the surface to fit the processed normals. Iterating this two-step process, we can efficiently implement
Figure 1.11: Showing one slice of a MRI volume: (a) The spherical initialization. (b) A model expands to fill the tumor but leaks through gaps and expands into other anatomy. (c) The same scenario with a degree of curvature prevents unwanted leaking. The level set isosurface is shown in yellow.

Figure 1.12: a) An MRI image. b) Slice of a 3D segmentation of the cortex. c) Surface rendering of cortex segmentation and d) tumor segmentation.
fourth-order flows by solving a set of coupled second-order PDEs.

For general fourth-order surface flows, both of these steps involve solving second-order PDEs. The first second-order PDE is used for minimizing a penalty function on the normals. The other second-order PDE minimizes the discrepancy between the modified normals and the surface; in other words, it refits the surface to the normals. Figure 1.13 shows this three step process graphically in 2D—shapes give rise to normal maps, which, when filtered give rise to new normal maps, which finally give rise to new shapes.

The first step is to formulate the total curvature of a surface from its normal map. Using this, we can derive the variational PDEs on the normal map that minimize functions of total curvature. When using implicit representations, one must account for the fact that derivatives of functions defined on the surface are computed by projecting their 3D derivatives onto the surface tangent plane. The shape matrix for an implicit surface is the gradient of the 3D normal field projected onto the tangent plane [32]: $\nabla N P_T$ where $P_T$ is the projection operator.

The Euclidean norm of the shape matrix is the sum of squared principal curvatures, i.e. total curvature,

$$\kappa^2 = \| (\nabla N) P_T \|^2.$$  \hspace{1cm} (1.27)

We can use (1.27) to define an energy of the normal map

$$G_N = \int_U G(\| (\nabla N) P_T \|^2) \, dx.$$  \hspace{1cm} (1.28)

The first variation of this energy with respect to the normals is a second order PDE. It is crucial to observe that, even though the projection operator $P_T$ is a function of $\phi$, it is independent of $N$ because we fix $\phi$ as we process $N$.

By choosing appropriate energy functions $G$, we can obtain isotropic and anisotropic diffusion processes on the normals among others. However, the final goal is to process the surface, which requires deforming $\phi$. Therefore, the next step is to relate the deformation of the level sets of $\phi$ to the evolution of $N$. Suppose that we are given the normal map $N$
\[ \phi^{n+1} \]

Iterate for \( p \) to catch \( n \)

\[ dD \quad dN \]

Iterate to process \( n \)

\[ N^n \]
successful in dealing with this problem in a wide range of images. A generalization of Perona and Malik diffusion to surfaces is achieved by

\[ G(\kappa^2) = 2\mu^2 \left( 1 - e^{-\frac{\kappa^2}{2\mu^2}} \right), \]

where \( \mu \) is a positive, free parameter that controls the level of feature preservation. Figure 1.15(a) illustrates an example of the skin surface, which was extracted, via isosurfacing, from an MRI data set. Notice that the roughness of the skin is noise, an artifact of the measurement process. Isotropic diffusion, shown in Figure 1.15(b), is marginally effective for denoising the head surface. Notice that the sharp edges around the eyes, nose, lips and ears are lost in this process. The differences between anisotropic diffusion and isotropic diffusion can clearly be observed in Figure 1.15(c). The two processes produce similar smoothing results in those areas of the surface which are somewhat featureless, such as the forehead and the cheeks. However, these results show significant differences exist around the lips and the eyes. The creases in these areas, which have been eliminated by isotropic diffusion, are preserved by the anisotropic process.

1.5 Summary

Volumes provide a powerful tool for modeling deformable surfaces, especially when dealing with measured data. With measured data the shape, topology, and complexity of the surface are dictated by the application rather than the user. Implicit deformable surfaces, implemented as level sets, provide a natural mechanism for processing such data in a manner that relieves the user of having to decide on an underlying parameterization. This technology easily handles the many degrees of freedom that important to capturing the fine detail of measured data. Furthermore the level set approach provides a powerful mechanism for
constructing geometric flows, which results in output which depends only on the shape of input (and the resolution) and does not produce artifacts which are tied to an arbitrary, intermediate parameterization.
Bibliography


