Proc. Natl. Acad. Sci. USA
Vol. 93, pp. 9389–9392, September 1996
Applied Mathematics

An $O(N \log N)$ algorithm for shape modeling

(Hamilton–Jacobi equation/Eikonal equation/shape recovery/medical image analysis/level sets)

R. MALLADI AND J. A. SETHIAN

Lawrence Berkeley National Laboratory and Department of Mathematics, University of California, Berkeley, CA 94720

Communicated by Alexandre J. Chorin, University of California, Berkeley, CA, June 3, 1996 (received for review January 24, 1996)

ABSTRACT **We present a shape-recovery technique in two dimensions and three dimensions with specific applications in modeling anatomical shapes from medical images. This algorithm models extremely corrugated structures like the brain, is topologically adaptable, and runs in $O(N \log N)$ time, where N is the total number of points in the domain. Our technique is based on a level set shape-recovery scheme recently introduced by the authors and the fast marching method for computing solutions to static Hamilton–Jacobi equations.**

In many medical applications such as cardiac boundary detection and tracking, tumor volume quantification, etc., accurately extracting shapes in two dimensions and three dimensions from medical images becomes an important task. These shapes are implicitly present in noisy images and the idea is to construct their boundary descriptions. Visualization and further processing like volume computation is then possible. In this paper, we present a fast shape modeling technique with specific applications in medical image analysis.

One of the challenges in shape recovery is to account for changes in topology as the shapes evolve. Malladi and colleagues (1, 2) have done this by modeling anatomical shapes as propagating fronts moving under a curvature dependent speed function (3). They adopted the level set formulation of interface motion due to Osher and Sethian (4). The central idea here is to represent a curve as the zero level set of a higher dimensional function; the motion of the curve is then embedded within the motion of the higher dimensional surface. As shown in ref. 4, this approach offers several advantages. First, although the higher dimensional function remains a function, its zero level set can change topology and form sharp corners. Second, a discrete grid can be used together with finite differences to devise a numerical scheme to approximate the solution. Third, intrinsic geometric quantities like normal and curvature of the curve can be easily extracted from the higher dimensional function. Finally, everything extends directly to moving surfaces in three dimensions.

To isolate shapes from images, an artificial speed term has been synthesized and applied to the front that causes it to stop near object boundaries; see refs. 1 and 2 for details. Subsequently, this work has been extended to three dimensions in ref. 5; see refs. 6–8 for related efforts. In this paper, we solve the shape modeling problem by using the fast marching methods devised recently by Sethian (9, 10) and extended to a wide class of Hamilton–Jacobi equations in ref. 11. The marching method is a numerical technique for solving the Eikonal equation and results from combining upwind schemes for viscosity solutions of Hamilton–Jacobi equations, narrow-band level set methods, and a *min*-heap data structure.

THE FAST MARCHING METHOD

We now briefly discuss the marching algorithm introduced in ref. 9. Let Γ be the initial position of a hypersurface and let F be its speed in the normal direction. In the level set perspective, one views Γ as the zero level set of a higher dimensional function $\psi(x, y, z)$. Then, by chain rule, an evolution equation for the moving hypersurface can be produced (4), namely

$$\psi_t + F(x, y, z)|\nabla\psi| = 0. \quad [1]$$

Consider the special case of a surface moving with speed $F(x, y, z) > 0$. Now, let $T(x, y, z)$ be the time at which the surface crosses a given point (x, y, z). The function $T(x, y, z)$ then satisfies the equation

$$|\nabla T|F = 1; \quad [2]$$

this simply says that the gradient of arrival time is inversely proportional to the speed of the surface. Broadly speaking, there are two ways of approximating the position of the moving surface: (*i*) iteration toward the solution by numerically approximating the derivatives in Eq. **1** or (*ii*) explicit construction of the solution function $T(x, y, z)$ from Eq. **2**. Our marching algorithm relies on the latter approach.

For the following discussion, we limit ourselves to a two-dimensional problem. Using the correct "entropy" satisfying (3, 4) approximation to the gradient, we are therefore looking for a solution in the domain to the following equation:

$$[\max(D^{-x}_{i,j}T,0)^2 + \min(D^{+x}_{i,j}T,0)^2 + \max(D^{-y}_{i,j}T,0)^2 + \min(D^{+y}_{i,j}T,0)^2]^{1/2} = 1/F_{i,j}, \quad [3]$$

where D^- and D^+ are backward and forward difference operators, respectively. Since the above equation is in essence a quadratic equation for the value at each grid point, we can iterate until convergence by solving the equation at each grid point and selecting the largest possible value as the solution. This is in accordance with obtaining the correct viscosity solution.

The Algorithm. The key to constructing the fast algorithm is the observation that the upwind difference structure of Eq. **3** means that information propagates from smaller values of T to larger values. Hence, our algorithm rests on building a solution to Eq. **3** outward from the smallest T value. Motivated by the methods in refs. 2 and 12, we confine the "building zone" to a narrow band around the front. The idea is to sweep the front ahead in an upwind fashion by considering a set of points in the narrow band around the current front, and to march this narrow band forward. We explain this algorithmically:

To illustrate, we imagine that we want to solve the Eikonal equation on an M by M grid on the unit box $[0, 1] \times [0, 1]$ with right-hand-side $F_{i,j} > 0$; furthermore, we are given an initial set $T = 0$ along the top of the box.

1. Initialize
   - (*i*) Alive points: Let *Alive* be the set of all grid points at which the value of T is fixed. In our example, *Alive* $= \{(i, j) : i \in \{1, \ldots, M\}, j = M\}$.
   - (*ii*) Narrow band points: Let *NarrowBand* be the set of all grid points in the narrow band. For our example,

The publication costs of this article were defrayed in part by page charge payment. This article must therefore be hereby marked "*advertisement*" in accordance with 18 U.S.C. §1734 solely to indicate this fact.

NarrowBand = $\{(i,j) : i \in \{1, \ldots, M\}, j = M-1\}$, also set $T_{i,M-1} = dy/F_{i,j}$.

(*iii*) Far away points: Let *FarAway* be the set of all the rest of the grid points $\{(i,j) : j < M-1\}$, set $T_{i,j} = \infty$ for all points in *FarAway*.

2. Marching forwards

(*i*) Begin loop: Let $(i_{\min}, j_{\min})$ be the point in *NarrowBand* with the smallest value for T.

(*ii*) Add the point $(i_{\min}, j_{\min})$ to *Alive*; remove it from *NarrowBand*.

(*iii*) Tag as neighbors any points $(i_{\min}-1, j_{\min})$, $(i_{\min}+1, j_{\min})$, $(i_{\min}, j_{\min}-1)$, $(i_{\min}, j_{\min}+1)$ that are not *Alive*; if the neighbor is in *FarAway*, remove it from that set and add it to the *NarrowBand* set.

(*iv*) Recompute the values of T at all neighbors according to Eq. **3**, solving the quadratic equation given by our scheme.

(*v*) Return to top of loop.

For more general initial conditions and for a proof that the above algorithm produces a viable solution, see ref. 11.

The *Min*-Heap Data Structure. An efficient version of the above technique relies on a fast way of locating the grid point in the narrow band with the smallest value for T. We use a variation on a heap algorithm [see Segdewick (13)] with back pointers to store the T values.

Specifically, we use a *min*-heap data structure. In an abstract sense, a *min*-heap is a "complete binary tree" with a property that the value at any given node is less than or equal to the values at its children. In practice, it is more efficient to represent a heap sequentially as an array by storing a node at location k and its children at locations $2k$ and $2k+1$. From this definition, the parent of a given node at k is located at $\lfloor k/2 \rfloor$. Therefore, the root that contains the smallest element is stored at location $k = 1$ in the array. Finding the parent or children of a given element are simple array accesses that take $O(1)$ time.

We store the values of T together with the indices that give their location in the grid structure. Our marching algorithm works by first looking for the smallest element in the *NarrowBand*; this FindSmallest operation involves deleting the root and one sweep of DownHeap to ensure that the remaining elements satisfy the heap property. The algorithm proceeds by tagging the neighboring points that are not *Alive*. The *FarAway* neighbors are added to the heap using an Insert operation and values at the remaining points are updated using Eq. **3**. Insert works by increasing the heap size by one and trickling the new element upward to its correct location using an UpHeap operation. Lastly, to ensure that the updated T values do not violate the heap property, we need to perform a UpHeap operation starting at that location and proceeding up the tree.

The DownHeap and UpHeap operations (in the worst case) carry an element all the way from root to bottom or vice versa. Therefore, this takes $O(\log M)$ time, assuming there are M elements in the heap. It is important to note that the heap that is a complete binary tree is always guaranteed to

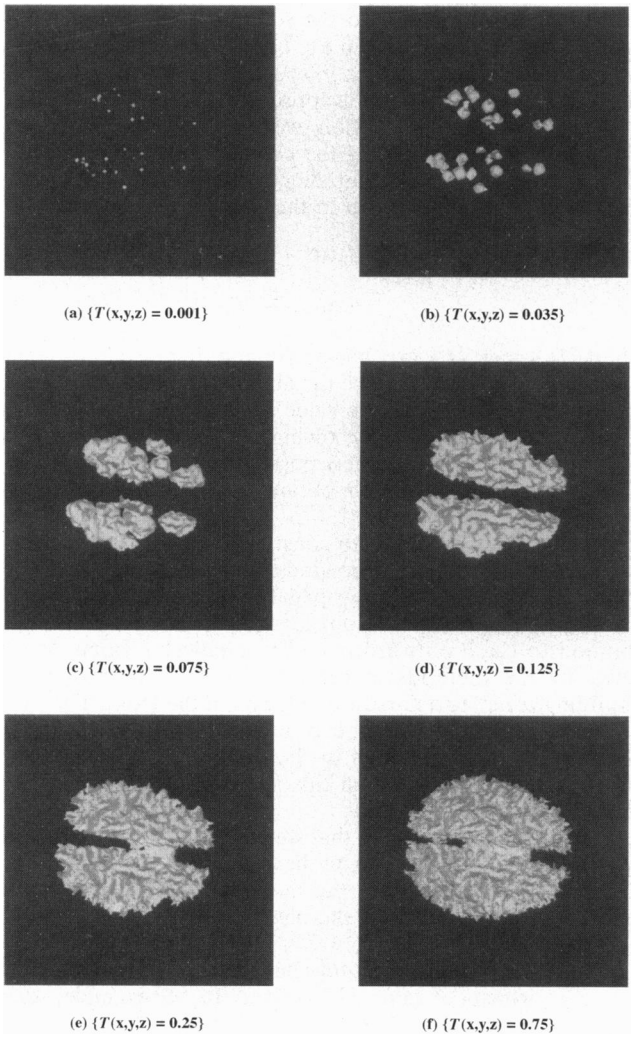

FIG. 1. Evolutionary sequence showing the brain reconstruction.

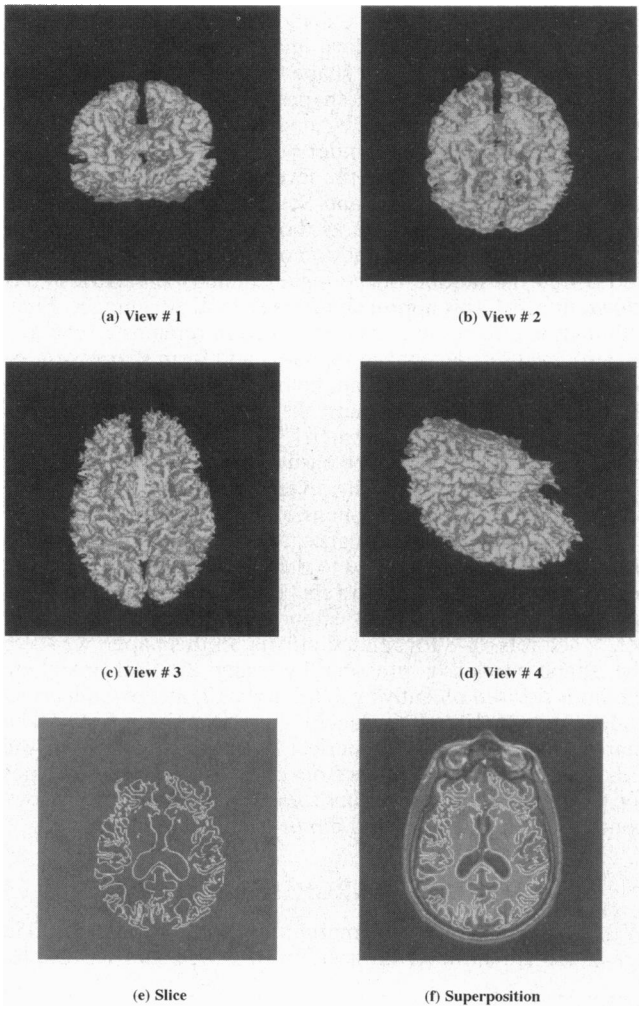

FIG. 2. Cortical structure rendered from different perspectives. (*e* and *f*) A slice of the surface on the corresponding image.

remain balanced. This leaves us with the operation of searching for the *NarrowBand* neighbors of the smallest element in the heap. We make this $O(1)$ in time by maintaining back pointers from the grid to the heap array. Without the back pointers, the above search takes $O(M)$ in the worst case.

THREE-DIMENSIONAL SHAPE MODELING

Given a noisy image function $I(x,y,z)$, the objective in shape modeling is to extract mathematical descriptions of certain anatomical shapes contained in it. This can be done by applying a special image-based speed function $k_I > 0$, which controls the outward propagation of an initial surface (an interior point in this case) such that the interface stops in the vicinity of shape boundaries. Mathematically, this procedure corresponds to solving a static Hamilton–Jacobi equation (see Eq. **1**) that, when recast in the arrival time framework, is

$$|\nabla T| = \frac{1}{k_I}. \qquad [4]$$

The speed function defined as

$$F(x, y, z) = k_I(x, y, z) = e^{-\alpha|\nabla G_\sigma * I(x,y,z)|}, \qquad \alpha > 0, \quad [5]$$

has values very close to zero near high image gradients, i.e., possible edges. False gradients due to noise can be avoided by applying a Gaussian smoothing filter or more sophisticated edge-preserving smoothing schemes (see refs. 5 and 14–16). With this model, the surface does not stop at the shape boundary unless the speed values are very close to zero. The definition in Eq. **5** ensures that the speed F goes to zero rapidly and minimizes the "over shoot" effect. To be further accurate, we now follow the ideas in refs. 1, 2, and 5 and outline how additional constraints can be imposed on the surface motion.

First, note that the shape model is represented implicitly as the zero level set of a function $\psi(x,y,z)$ defined in the image domain. Let the surface move under a simple speed law $F = 1 - \varepsilon H$, $\varepsilon > 0$, where H is the mean curvature given by the following expression:

$$H = \frac{\psi_{xx}(\psi_y^2 + \psi_z^2) + \psi_{yy}(\psi_x^2 + \psi_z^2) + \psi_{zz}(\psi_x^2 + \psi_y^2) - 2\psi_{xy}\psi_x\psi_y - 2\psi_{yz}\psi_y\psi_z - 2\psi_{zx}\psi_z\psi_x}{(\psi_x^2 + \psi_y^2 + \psi_z^2)^{3/2}}. \qquad [6]$$

The driving force that molds the initial surface into desired anatomical shapes comes from two image-based terms. The first term is similar to Eq. **5** and the second term attracts the surface towards the object boundaries; the latter term has a stabilizing effect especially when there is a large variation in the image gradient value. Specifically, the equation of motion is

$$\psi_t + k_I(1 - \varepsilon H)|\nabla\psi| - \beta\nabla P\cdot\nabla\psi = 0. \qquad [7]$$

where,

$$k_I = \frac{1}{1 + |\nabla G_\sigma * I(x,y,z)|}. \qquad [8]$$

The second term $\nabla P\cdot\nabla\psi$ denotes the projection of an (attractive) force vector on the surface normal. This force which is realized as the gradient of a potential field

$$P(x, y, z) = -|\nabla G_\sigma * I(x, y, z)|, \qquad [9]$$

attracts the surface to the edges in the image; coefficient β controls the strength of this attraction.

In this work, we adopt the following two stage approach when necessary. We first construct the arrival time function using our marching algorithm. If a more accurate reconstruction is desired, we treat the final $T(x, y, z)$ function as an initial condition to our full model. In other words, we solve Eq. **7** for a few time steps using explicit finite differencing with $\psi(x, y, z; t = 0) = T(x, y, z)$. This too can be done very efficiently in the narrow band framework. Finally, the above initial condition is a valid one since the surface of interest is a particular level set of the final time function T.

RESULTS

In this section, we consider the problem of reconstructing the entire cortical structure of the human brain from a dense MRI data set. The data is given as intensity values on a $256 \times 256 \times 124$ grid. We start by defining "seed" points in the domain. The value of T at these points is set to zero and the initial heap is constructed from their neighbors. The fast marching algorithm in three dimensions is then used to march ahead to fill the grid with time values according to Eq. **2**. We visualize various stages of our reconstruction by rendering particular level surfaces of the final time function $T(x, y, z)$. These stages are shown in Fig. 1. The corrugated structure shown in Fig. 1*f* is our final shape. In Fig. 2 *a–d*, we render the same $\{T(x, y, z) = 0.75\}$ surface from different perspectives. Finally, in Fig. 2*e*, we slice the surface parallel to the *xy* plane and superimpose the resulting contours on the corresponding image (see Fig. 2*f*). The timings for various stages of calculation on a Sun SPARC 1000 machine are shown in Table 1.

Table 1. Timing for marching to $T = 0.75$; Sun SPARC 1000

Grid size	Time to load speed file	Time to propagate surface	Total time
$256 \times 256 \times 124$	8.61 sec	74.92 sec	83.53 sec

This work was supported in part by the Applied Mathematical Sciences Subprogram of the Office of Energy Research, U.S. Department of Energy under Contract DE-AC03-76SD00098 and by the National Science Foundation Advanced Research Projects Agency under Grant DMS-8919074.

1. Malladi, R., Sethian, J. A. & Vemuri, B. C. (1994) *Lect. Notes Comput. Sci.* **800,** 3–13.
2. Malladi, R., Sethian, J. A. & Vemuri, B. C. (1995) *IEEE Trans. Pattern Analysis Machine Intell.* **17,** 158–175.
3. Sethian, J. A. (1985) *Commun. Math. Phys.* **101,** 487–499.
4. Osher, S. & Sethian, J. A. (1988) *J. Comput. Phys.* **79,** 12–49.
5. Malladi, R. & Sethian, J. A. (1996) in *Proceedings of the International Conference on Mathematics and Visualization*, ed. Polthier, K. (Springer, Berlin), in press.

6. Caselles, V., Catté, F., Coll, T. & Dibos, F. (1993) *Numer. Math.* **66,** 1–32.
7. Caselles, V., Kimmel, R. & Sapiro, G. (1995) *Proceedings of the International Conference on Computer Vision* (MIT Press, Cambridge, MA), pp. 694–699.
8. Kichenassamy, S., Kumar, A., Olver, P., Tannenbaum, A. & Yezzi, A. (1995) *Proceedings of the International Conference on Computer Vision* (MIT Press, Cambridge, MA), pp. 810–815.
9. Sethian, J. A. (1996) *Proc. Natl. Acad. Sci. USA* **93,** 1591–1595.
10. Sethian, J. A. (1995) *Acta Numer.*, in press.
11. Adalsteinsson, D., Kimmel, R., Malladi, R. & Sethian, J. A. (1996) *SIAM J. Numer. Anal.*, in press.
12. Adalsteinsson, D. & Sethian, J. A. (1995) *J. Comp. Phys.* **118,** 269–277.
13. Sedgewick, R. (1988) *Algorithms* (Addison–Wesley, Reading, MA).
14. Malladi, R. & Sethian, J. A. (1995) *Proc. Natl. Acad. Sci. USA* **92,** 7046–7050.
15. Malladi, R. & Sethian, J. A. (1996) *Graphics Models Image Processing* **58,** 127–141.
16. Malladi, R. & Sethian, J. A. (1996) *IEEE Trans. Image Processing*, in press.