Application of the Level Set Method to Hydrocephalus: Simulating the Motion of the Ventricles

by

Joseph J. West

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Applied Mathematics

Waterloo, Ontario, 2004

I hereby declare that I am the sole author of this thesis.
I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Joseph J. West

I authorize the University of Waterloo to reproduce this thesis by photocopying or other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Joseph J. West

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

This thesis investigates the application of the Level Set Method (LSM)—a numerical technique for propagating closed interfaces—to simulate the motion of hydrocephalic ventricles.

A brief introduction to hydrocephalus and related research is made, followed by a simplified problem involving curves and surfaces. The known theory of LSM is presented, then the method is applied to propagate curves and surfaces with constant and curvature-dependent speed. Then, a new non-constant speed approach is introduced which uses known deformation data.

Finally, a semi-automatic procedure based on active contours for segmenting the ventricles is implemented.

Acknowledgements

I would like to thank my supervisors Dr. Tenti and Dr. Sivalogathanan for their help, encouragement, and support since my first undergraduate research work term several years ago. Thanks also to Corina Drapaca, who I first worked with when learning the Level Set Method, and who has given much helpful advice since then; to Dr. Drake from the Hospital for Sick Children for his valuable collaboration with our research group; to Dr. Wan, for some very useful discussions; to Matt Scott for his interest and helpful insights; and to James Munroe, whos LyX expertise saved me many hours of typesetting work.

For financial support, I would like to thank the National Science and Engineering Research Council.

I would also like to thank the Applied Mathematics department as a whole, whose friendly and hard-working faculty has provided encouragement throughout my undergraduate and graduate career.

Finally, and most importantly, I want to thank my family and friends: my mom and dad, brother and sisters, my nephew and niece Nicholas and Larissa (who played with me when it was time to put work aside for awhile), my smartette Nina, and my grad school friends here at UW.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

At the University of Waterloo, research is being conducted on a condition called Hydrocephalus, in collaboration with the Hospital for Sick Children in Toronto. Hydrocephalus, also called "water on the brain," is a clinical condition which occurs when normal cerebrospinal fluid (CSF) circulation is impeded within the cranial cavity. As fluid accumulates, the intra-cranial pressure increases, producing a dilatation of the ventricular system, with the result that the brain tissue is compressed (see figure 1.1). We note in passing that hydrocephalus can develop without an increase of the intracranial pressure (normal-pressure hydrocephalus). This case, however, is not yet well understood, and we shall not consider it any further. In any case, untreated hydrocephalus can cause very serious neurological problems and even death.

The current treatment involves draining the excess fluid by inserting a shunt (siphon) into the ventricles. Within limits, the dilatation of the ventricles can be reversed by this shunting process, but the introduction of a foreign body into the system has inherent disadvantages, and the rate of shunt failure is unacceptably high in most of the reported series. In fact, Drake et al. [7] report recently that the failure rate is 50% after a two year period. One cause of shunt failure, for example, is the blockage of the catheter's tip with brain matter after the ventricular wall moves past the shunt (see Figure 1.2).

Our research has two aspects. The first, and most fundamental, is the modeling aspect. This involves finding an appropriate mathematical description of the relevant anatomy,

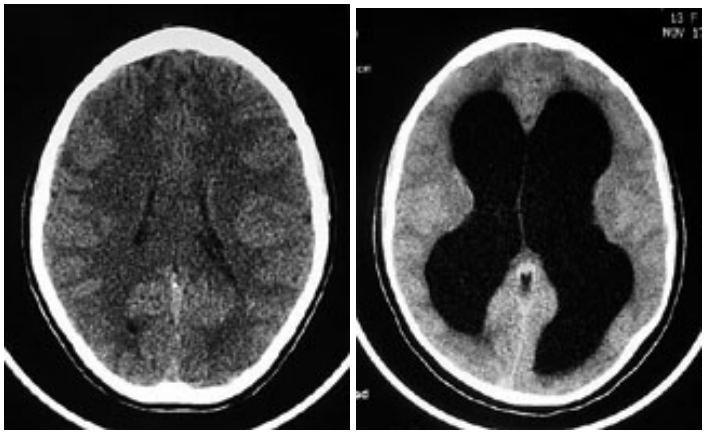

Figure 1.1: CT scan of a cross-section of a normal brain (left) and a hydrocephalic brain (right), showing the expansion of the ventricles (black cavities) and compression of brain tissue (gray material between the skull and cavities).

such as the brain and cerebrospinal fluid.

The second aspect involves computer modeling and image processing. Once the governing equations are derived, they can be applied to actual data (for example magnetic resonance (MR) or computed tomography (CT) scans of patients) to simulate the motion of the ventricular walls, with the help of imaging and numerical computation techniques.

The first aspect has proven to be a very challenging and long-term problem, and is not the subject of this paper. Instead, we will deal with the second aspect applied to a simplified model.

1.2 Literature Review

The first aspect of our research, that of modeling the physical properties of the brain, has two main approaches. The first is to treat the brain as a poroelastic material, that is, as a solid sponge with pores containing fluid. This is useful for studying the interaction between the brain and the CSF. The second approach is to treat the brain as a continuous material with both elastic properties (as solids have) and fluid properties. This approach, called the *viscoelastic* approach, is useful for modeling the brain deformations under loading, such as the loading caused by excess CSF pressure in the ventricles. This is not the topic of this paper, so we leave the interested reader with some references

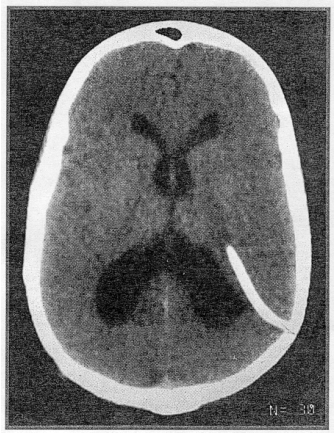

Figure 1.2: A blocked shunt [6].

dealing with the poroelastic approach ([27],[13],[9],[25],[34]) and the viscoelastic approach ([17],[35],[23],[1],[31],[11]). A good summary of all of these approaches is given in [18].

With regards to the second aspect, that of computer modeling, the Finite Element Method (FEM) is very popular. Several papers use FEM, along with one of the biomechanical models, to simulate the deformation of the brain("brain shift") during neurosurgery. This is important during stereotactic[1] neurosurgery, where important real-time information of the "brain shift" is not available (a dedicated MR scanner is too expensive, so only the pre-operative MR scan is available). See [26],[22], and [28]. In [26], a linear model is used (that is, the governing equations are linear), and predicted tissue displacements differ by 15% from the measured displacement. In [22], a hyper-viscoelastic model is used with 31% error. Another deformation model, found in [5], uses both statistics-based and combined statistics-based and physics-based models, with good results (statistics-based models use sample data, where physics-based models use governing equations).

Another paper combining a statistics-based and physics-based approach is [21], which presents a way to analyze shape deformation of structures (such as the ventricles) within the brain. This shape analysis is used to classify conditions/diseases such as schizophrenia, Alzheimer's disease, and hydrocephalus. The authors reported some success, with classification success rates of 60-80%. While not directly applicable to our problem, this paper reminds us not to exclude the possibility of statistics-based or combined models.

[1] 'Stereotactic' refers to the 3D positioning and movement of objects inside the brain.

While the above models have experienced some success, the quest continues for a realistic model, and for the correct parameters to be used in this model.

1.3 Purpose & Outline

The purpose of this paper is to present a simple deformation model of the ventricles using the level set method ([30],[24]). This method will also be used to automate the segmentation of the ventricles.

In the second chapter, the problem is simplified and viewed in light of the theory of evolution of curves and surfaces. A brief comparison of the level set method (LSM) with other methods is made.

The third chapter describes the LSM. Then, using numerical techniques for hyperbolic conservation laws, a numerical algorithm for the LSM is derived.

Using Matlab, the LSM is implemented and tested in chapter four using benchmark cases and then real examples. A new method is introduced and tested, followed by some three-dimensional simulations.

The fifth chapter deals with the extraction of the shape of the ventricles from the original MR scan. Some theoretical aspects of this is presented, followed by real examples.

We conclude with a discussion of the results and suggestions for future work in the last chapter.

Chapter 2

Propagation of Curves and Surfaces

2.1 The Simplified Problem

In the 2-dimensional situation, the ventricular walls usually form a simple, closed, non-convex curve. If we assume that, after treatment, the ventricles 'shrink' such that every point on the ventricular wall moves at the same constant speed in the direction of the inward normal, then we can view the problem mathematically as:

Simplified Problem: *Given a simple, closed, non-convex curve, propagate the curve such that each point moves with the same constant speed in the direction of the inward normal. As an extra challenge, generalize to three dimensions.*

2.2 Possible Approaches to the Simplified Problem

Noting that the curve will not, in general, be given analytically, and that an analytical solution is not required for our application, we seek a numerical solution.

The most obvious algorithm is to parametrize the curve with points (x_i, y_i) with arclength parameter s_i, and then use approximations to the derivatives (for example, central differences) to approximate the normal and curvature. With this information, one can propagate the points on the curve (see figure 2.1).

This is an example of a *Lagrangian* formulation, which means that one keeps track of the moving points on the curve, in contrast to an *Eulerian* formulation, where one uses a

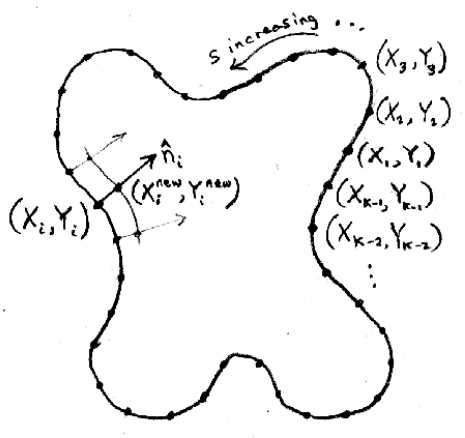

Figure 2.1: Discrete parametrization of a simple closed curve, showing inward normal $\hat{n}$ and the updating of a point along this normal.

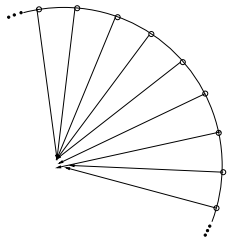

Figure 2.2: In a Lagrangian formulation, points can collide, causing numerical difficulties.

fixed set of points to describe the curve.

More sophisticated Lagrangian formulations exist, such as the use of splines, the front dynamics approach ([29]), and a perturbation approach ([8]). There are inherent numerical difficulties with Lagrangian approaches. For example, as the curve shrinks, the points move closer to each other and some eventually meet (figure 2.2). This causes numerical instabilities and must be remedied, typically by re-distributing the particles on the curve when necessary. Also, the Lagrangian formulation has difficulty dealing with sharp corners and with topological changes such as the breaking of the curve into two or more curves.

An elegant and robust method for our simplified problem lies in the level set method, which uses an Eulerian formulation. The LSM handles sharp corners and changes in topology automatically, and generalizes easily to three dimensions.

Chapter 3

The Level Set Method

3.1 Motivation

One can think of many situations in which one describes a certain system using a more general system. For example, the earth can be seen as a planet alone in the universe, in which case its trajectory would be quite puzzling. If seen as part of the solar system, however, things begin to make sense (with the help of Kepler and Newton, of course!). As a related example, Einstein's theory of general relativity uses four-dimensional space-time to describe our three-dimensional universe as it evolves in time. Curvature in space-time is interpreted as the force of gravity in our three-dimensional world. So something simple (like curvature) in a higher dimension can describe something complicated or mysterious (like gravity) in a lower dimension.

In our case, we will use the level set of a surface to describe a curve. One nice thing about this description is that the level set of a surface can do many strange things, such as break into several curves or merge disjoint curves together, all while the surface remains "well-behaved" (the function describing it remains single-valued and continuous). Figure 3.2 shows an example of a level set curve breaking off into two disjoint curves. Another advantage of using level sets is that one can easily generalize this method to describe a *surface* as the level set of a four-dimensional hypersurface. Given the difficulties of the Lagrangian methods and the nice properties mentioned above, the level set method is a good candidate for curve and surface evolution.

3.2 Description of the Method

Consider a simple, closed curve $\gamma(t)$ which depends on time. One can construct a higher-dimensional function, say $\phi(x, y, t)$, whose zero level set is the curve itself:

$$\gamma(t) = \{(x, y) | \phi(x, y, t) = 0\}.$$

Such a function is called a *level set function.* At each moment in time, ϕ represents a surface which implicitly defines the curve (to get the curve, one "slices" the surface at $\phi = 0$). For example, the function $z = \phi(x, y, t) = x^2 + y^2 - (t + 1)^2$ looks like a cone in xyz-space, and $\phi = 0$ defines a circle of radius $t + 1$ at time t (see figure 3.1).

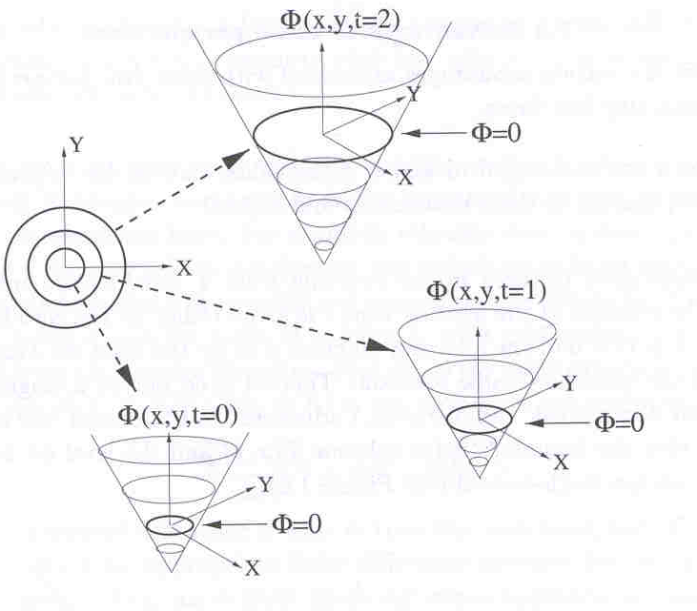

Figure 3.1: Visualization of the level sets $\phi = 0$ and the level set function $z = \phi(x, y, t)$.

The idea is to take the initial curve, $\gamma(0)$, and initialize the level set function according to

$$\phi(x, y, 0) = \pm d(x, y) \tag{3.1}$$

where $d(x, y)$ is the distance from (x, y) to the closest point on the curve $\gamma(0)$, and the $+$ sign ($-$ sign) is chosen if (x, y) is outside (inside) of the curve. Then $\phi(x, y, 0) = 0$ will

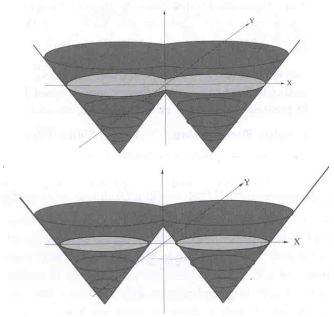

Figure 3.2: An example of a front (in light gray) breaking into two fronts as the level set function $z = \phi(x, y, t)$ (in dark gray) advances in time.

define $\gamma(0)$. Then, to find $\gamma(t)$, one solves the partial differential equation

$$\frac{\partial \phi}{\partial t} + F\,|\nabla \phi| = 0$$

where $F = F(x, y)$ is the speed of the curve, to find $\phi(x, y, t)$, and consequently the evolving curve $\gamma(t)$. This method can also be used to evolve a surface $\Sigma(t)$ defined by $\phi(x, y, z, t) = 0$, which satisfies the same partial differential equation with initial condition $\phi(x, y, z, 0) = \pm d(x, y, z)$.

Of course, we will deal with all of these quantities numerically, so an appropriate discretization will be used, and numerical algorithms will be derived.

3.3 Theory and Numerical Scheme

We will now derive the governing equation for the evolution of our level set function, and then devise a numerical algorithm. The theory presented in this section is a compression of chapter 5 and 6 of Sethian ([30]). Let's consider the 2-dimensional case of an evolving curve.

Consider some point $(x(t), y(t))$ on $\gamma(t)$. Then

$$\phi(x(t), y(t), t) = 0,$$

and taking the derivative with respect to t, the chain rule yields

$$\frac{\partial \phi}{\partial t} + \nabla \phi \bullet (x'(t), y'(t)) = 0.$$

Now, the velocity of any point is in the direction of the outward normal $\hat{n} = \nabla\phi / |\nabla\phi|$. So, we have

$$(x'(t), y'(t)) = F \frac{\nabla \phi}{|\nabla \phi|}$$

where the (signed) speed is given by $F = -1$ in the case of constant speed in the direction of the inward normal, but in general $F = F(x, y)$. The last two equations imply that

$$\frac{\partial \phi}{\partial t} + F |\nabla \phi| = 0 \tag{3.2}$$

which is the governing partial differential equation for $\phi(x, y, t)$. A similar derivation, with the same resulting equation, is used for the three-dimensional level set function $\phi(x, y, z, t)$.

We'd like to find a numerical algorithm to solve (3.2) for certain forms of F. For simplicity, consider the one-dimensional version

$$\frac{\partial \phi}{\partial t} + F \left| \frac{\partial \phi}{\partial x} \right| = 0$$

where $\phi = \phi(x, t)$ and $F = F(x)$. Define the *Hamiltonian* to be $H(u) = F\sqrt{u^2} = F |u|$ so that the above equation becomes

$$\phi_t + H(\phi_x) = 0,$$

where subscripts indicate partial differentiation. Letting $u = \phi_x$ and differentiating with respect to x, we get the *hyperbolic conservation law*

$$u_t + [H(u)]_x = 0. \tag{3.3}$$

We will develop numerical schemes to solve this equation. First, the variables must be discretized, and some notation is needed.

Discretization and Notation

When using a numerical technique, one has finite resources and finite computational time, so it is necessary to discretize the domain into a finite set of points. Let the domain be a bounded rectangular subset of $\mathbb{R}^2$, say $[0, P] \times [0, Q]$, which contains the curve.

Dividing our domain into a grid as illustrated in figure 3.3, let Δx and Δy be the mesh spacing, and define the grid points

$$x_i = i\Delta x,\ y_j = j\Delta y$$

(where $i = 0, 1, 2, ..., P/\Delta x$ and $j = 0, 1, 2, ..., Q/\Delta y$) and let Δt be the time step, so the discrete values of time are

$$t_n = n\Delta t$$

where $n = 0, 1, 2, ...N$. For convenience, let ϕ_{ij}^n denote the value of $\phi(x_i, y_j, t_n)$:

$$\phi_{ij}^n \equiv \phi(x_i, y_j, t_n).$$

(In one dimension $\phi_i^n \equiv \phi(x_i, t_n)$ and in three dimensions $\phi_{ijk}^n \equiv \phi(x_i, y_j, z_k, t_n)$ where $z_k = k\Delta z$ where $k = 0, 1, 2, ...R/\Delta z$ and the domain is $[0, P] \times [0, Q] \times [0, R]$.)

3.3.1 Hyperbolic Conservation Laws

1) The Linear Wave Equation

Our 1D governing equation $u_t + [H(u)]_x = 0$ has some resemblance to the linear wave equation

$$u_t + u_x = 0 \tag{3.4}$$

where we again consider the one-dimensional case $u = u(x, t)$ for simplicity. A solution is $u(x, t) = f(x - t)$, which means that this solution is constant along the lines $x - t = constant$ (see figure 3.4). These lines are called *characteristics* of the solution, and, since they move to the right as t increases, we say that "information flows to the right" in this case.

We note in passing that hyperbolic waves are just one class of solutions of the wave

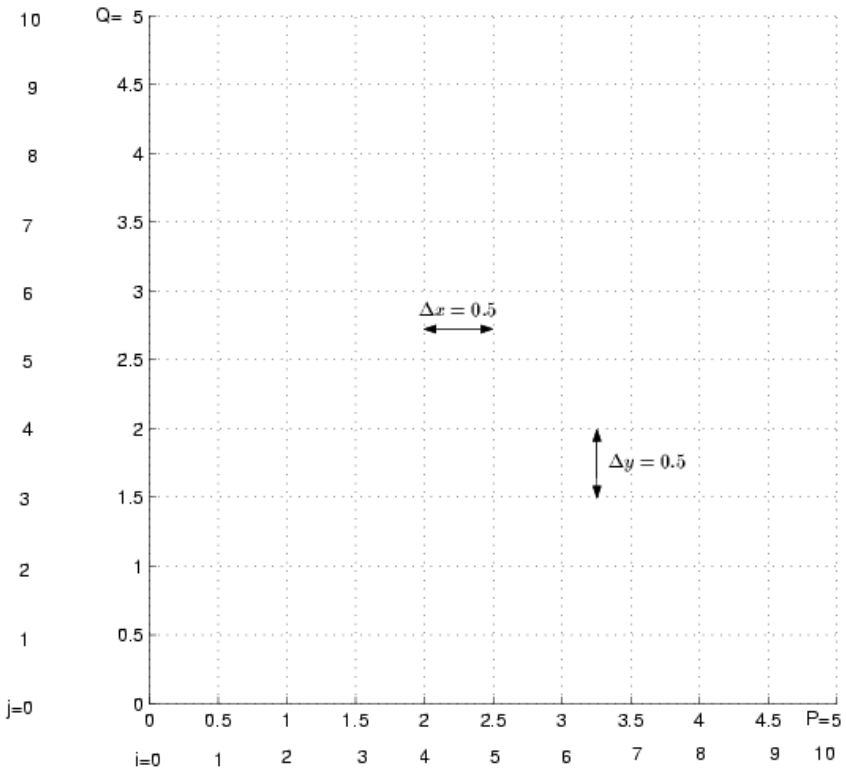

Figure 3.3: Example of a computational grid in two dimensions.

equation; another class is given by functions of the form $u = a\cos(kx - \omega t)$, where k is the wave number and $c = \frac{\omega}{k}$ is the propagation speed (above we had $c = 1$). These waves are called dispersive.

To approximate the (3.4) numerically, we can use a Taylor expansion

$$u(x, t + \Delta t) = u(x, t) + u_t(x, t)\Delta t + O(\Delta t^2)$$

to obtain a *forward difference approximation*

$$u_t(x, t) = \frac{u(x, t + \Delta t) - u(x, t)}{\Delta t} + O(\Delta t).$$

Similarly, the Taylor expansion

$$u(x, t - \Delta t) = u(x, t) - u_t(x, t)\Delta t + O(\Delta t^2)$$

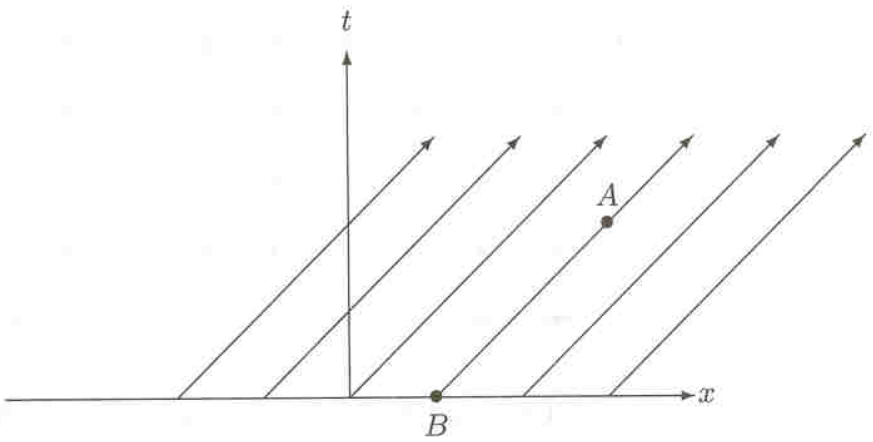

Figure 3.4: A solution of $u_t + u_x = 0$ is constant along the lines $t = x + constant$.

yields the *backward difference approximation*

$$u_t(x,t) = \frac{u(x,t) - u(x,t-\Delta t)}{\Delta t} + O(\Delta t).$$

By including one more term in each of the above Taylor expansions and subtracting, one can obtain the *central difference approximation*

$$u_t(x,t) = \frac{u(x,t+\Delta t) - u(x,t-\Delta t)}{2\Delta t} + O(\Delta t^2),$$

which is second-order accurate, while the forward and backward differences are first-order accurate. It is easy to interpret these approximations geometrically: they are just the slope of a secant line, which, in the limit as $\Delta t \to 0$, becomes the tangent line, giving the correct slope. In fact, as $\Delta t \to 0$, these approximations become exact, by the limit definition of the derivative. It is interesting to note that the leading term in the central difference approximation can be obtained by taking the average of the leading terms in the other two approximations.

Using discretized notation $u_i^n \equiv u(x_i, n\Delta t)$, we can write these derivative approximations using forward, backward, and central derivative operators

$$D_t^+ u_i^n \equiv \frac{u_i^{n+1} - u_i^n}{\Delta t},$$

$$D_t^- u_i^n \equiv \frac{u_i^n - u_i^{n-1}}{\Delta t},$$

$$D_t^0 u_i^n \equiv \frac{u_i^{n+1} - u_i^{n-1}}{2\Delta t},$$

respectively. Similarly, spatial derivatives can be approximated by

$$D_x^+ u_i^n \equiv \frac{u_{i+1}^n - u_i^n}{\Delta x},$$

$$D_x^- u_i^n \equiv \frac{u_i^n - u_{i-1}^n}{\Delta x},$$

and

$$D_x^0 u_i^n \equiv \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x}.$$

To approximate the time derivative in (3.4), we use a forward difference operator to carry the solution forward in time by one step. Thus, the wave equation is approximated by

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = -D_x^? u_i^n,$$

or

$$u_i^{n+1} = u_i^n - \Delta t \cdot D_x^? u_i^n. \tag{3.5}$$

(note that $D_x^?$ means either D_x^+, D_x^-, or D_x^0) The above equation (3.5) is a numerical scheme for updating u_i^n to the next time step. The question is: which derivative approximation should we use for the x-derivative? One might think that it is always best to use the central difference approximation, since it is higher-order and hence more accurate. However, second-order approximations such as this one can lead to unwanted oscillations in the solution (see, for example, page 43 in [30]). In this case, recall that the information flows from left to right (figure 3.4), so it makes sense to use a backward difference operator:

$$u_i^{n+1} = u_i^n - \Delta t \cdot D^{-x} u_i^n.$$

This is referred to as an *upwind scheme.* If information is thought of as wind, we want to look upwind for information about where the wind will go, and send that information downwind. The basic idea is to "go with the flow."

Before moving on, note that by taking more terms in the Taylor series, higher-order

approximations can be developed for the derivatives. However, despite increased accuracy, there are often drawbacks involved (see [24] and [30] for details) and the extra accuracy is simply not yet necessary.

2) The non-linear wave equation

Consider a wave equation with non-constant speed:

$$u_t + a(x)u_x = 0. \tag{3.6}$$

If we write the update scheme (3.5) as

$$u_i^{n+1} = u_i^n - \Delta t \left[\max(0, a_i)D^{-x}u_i^n + \min(0, a_i)D^{+x}u_i^n\right]$$

then, if the speed $a_i = a(x_i)$ is positive at x_i, we select the backward operator as before; if a_i is negative, information flows from right to left so we select the forward operator. This scheme selects the correct direction of "upwinding".

Now, look at the fully non-linear wave equation

$$u_t + uu_x = 0. \tag{3.7}$$

Depending on the initial conditions, the characteristics of the solution are not parallel and either converge or diverge, as in figure 3.5 and 3.6, respectively. If the characteristics collide, it is unclear how to continue the solution ahead in time so that it remains single-valued (recall that each characteristic represents a different value of u in general). But standard partial differential equations (PDE) theory explains how the characteristics form a *shock*[1] (this can be seen in figure 3.5) over which the solution is discontinuous.

Conversely, if the solutions diverge, a gap needs to be filled and it is unclear how to do so.

Two ways to fill in the gap in figure 3.6 are by using i) a rarefaction shock and ii) a rarefaction fan (see figure 3.7). Which way, if any, is correct? The answer lies in the *entropy condition* which requires that information can be destroyed but not created, so

[1] Physical examples of shocks include the steepening of a wave before it "rolls over" and the sonic boom of a supersonic aircraft as it breaks the sound barrier.

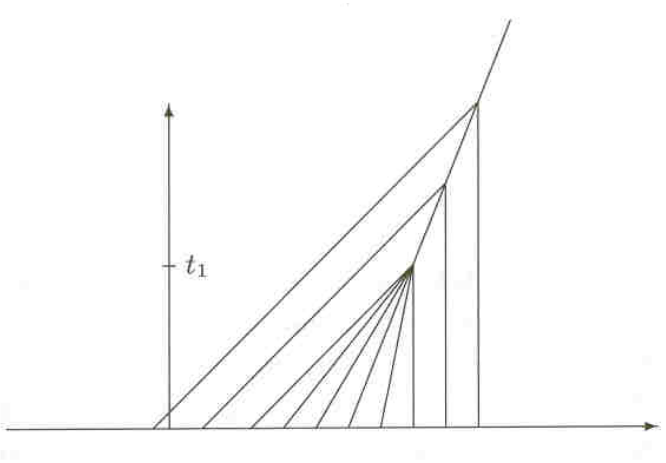

Figure 3.5: Characteristics colliding to form a shock.

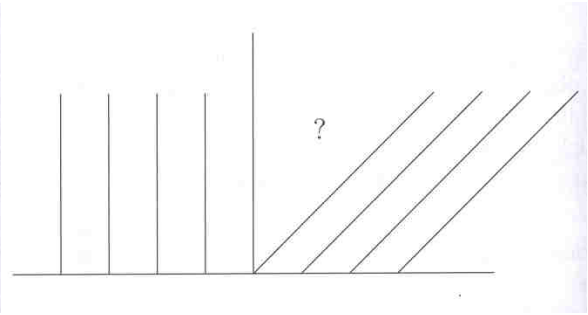

Figure 3.6: Characteristics diverging, leaving a gap in the solution.

that each point in the (x,t) domain can be "traced back" along a characteristic to the initial line $t = 0$. This implies that characteristics can flow into shocks (as in figure 3.5) but not out of shocks (as in figure 3.7 left). A solution satisfying the entropy condition is called an entropy solution.

Now, consider the vanishing-viscosity solution, which is obtained by adding a viscous term to the right hand side of the wave equation to get

$$u_t + uu_x = \varepsilon u_{xx}, \tag{3.8}$$

where $\varepsilon > 0$ can be thought of as a small parameter. The viscous term εu_{xx} "smoothes out" the solution, removing any sharp corners before they form (basically, as a sharp corner is about to form, the curvature increases rapidly, making εu_{xx} significant; otherwise εu_{xx} is

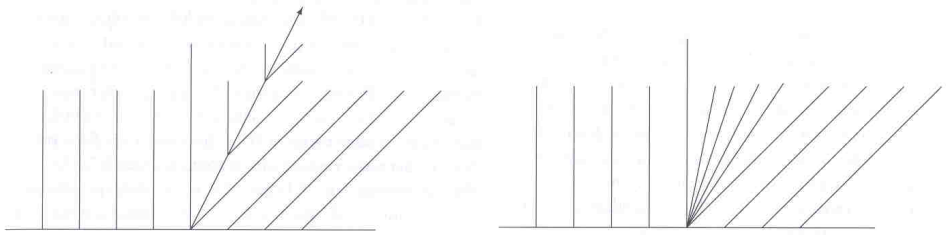

Figure 3.7: A rarefaction shock (left) and a rarefaction fan (right).

small). The solution to (3.8) stays smooth and is unique for all time (see [19],[3]). The vanishing viscosity solution is the limit of this unique solution as $\varepsilon \to 0$. As proven in [16], the limiting viscous solution and the entropy solution are equivalent.

In our example above, then, we would dismiss the rarefaction shock and use the rarefaction fan as the correct solution.

To help visualize the correct entropy solution in two dimensions, recall figure 2.2, which shows a sharp corner forming as our simple closed curve shrinks in the normal direction. This sharp corner represents a shock in the solution to our governing PDE $\phi_t + (-1)\,|\nabla\phi| = 0$. If we imagine the curve to be the edge of a forest fire, where everything *outside* of the curve is on fire, then as the fire advances (and the curve shrinks), any point that gets burnt stays burnt. That is, once a point is outside of the curve, it stays outside. This is the entropy condition as applies to the case when characteristics collide. When characteristics diverge, we would expect a sharp corner to grow and become rounded. For example, consider a square forest fire growing in all directions. If the flames propagate with the same speed in all directions (assuming no wind, of course), one expects the corners to become rounded as the fire grows outward. It would be strange indeed if the forest fire remained a square as it expanded.

Assuming that the entropy theory above generalizes to two and three dimensions (see [4]), the entropy solution is equivalent to limit, as $\varepsilon \to 0$, of the solution to the viscous PDE

$$\phi_t + (-1)\,|\nabla\phi| = \varepsilon\kappa\,|\nabla\phi|$$

or $\phi_t + (-1 - \varepsilon\kappa)\,|\nabla\phi| = 0$, where κ is the curvature term containing second derivatives. Therefore, the viscous PDE represents a curvature-dependent speed $F = -1 - \varepsilon\kappa$ in

the governing PDE (3.2), and it makes sense that the smooth and unique solution as $\varepsilon \to 0$ should match the correct solution in the constant speed case $F = -1$. Our goal is to construct numerical schemes to approximate this "entropy solution." Only the one-dimensional case will be dealt with in detail.

Weak solutions

If we integrate the hyperbolic conservation law

$$u_t + [G(u)]_x = 0 \tag{3.9}$$

from a to b, then we obtain

$$\begin{aligned} 0 &= \int_a^b u_t \, dx + \int_a^b G(u(x,t))_x \, dx \\ &= \frac{d}{dt} \int_a^b u(x,t) \, dx + G(u(b,t)) - G(u(a,t)) \end{aligned}$$

by Leibniz' integral formula and the fundamental theorem of calculus. We thus have

$$\frac{d}{dt} \int_a^b u \, dx = G(u(a,t)) - G(u(b,t)) \tag{3.10}$$

which is the integral form of (3.9). Because everything has been integrated, the restrictions on the solutions to (3.10) are less severe than on solutions to (3.9). The solutions to (3.10) are called *weak solutions*, which are useful because they allow for sharp, non-differentiable solutions (which is the case when characteristics collide and sharp corners form). However, weak solutions are not neccessarily unique, so it is neccessary to enforce the entropy condition to pick out the correct solution.

From the above equation, we can interpret $G(u)$ to be a *flux function:* the change in $\int_a^b u \, dx$ is equal to the flux, or net change, of $G(u)$ over $[a, b]$.

Computing $u(x,t)$ Numerically

One way to numerically compute u is called the *method of artificial viscosity*, and involves approximating the solution to the viscous equation $u_t + [G(u)]_x = \varepsilon u_{xx}$ for "small" ε, using the appropriate derivative approximations for upwinding. While this method is used in many settings, it turns out that it often involves too much diffusion, as the sharp corners are "smoothed out."

A second way is as follows. Consider a discrete version of (3.10) with $a = x_{i-\frac{1}{2}}$ and $b = x_{i+\frac{1}{2}}$:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t}\Delta x = G(u^n_{i-\frac{1}{2}}) - G(u^n_{i+\frac{1}{2}})$$

where the difference on the right-hand side is centred about u_i^n and spans one grid cell (here $\Delta x = h$ is the width of one grid cell). Of course, we don't explicitly know the value of $G(u^n_{i\pm\frac{1}{2}})$, which we denote $G_{i\pm\frac{1}{2}}$; we will try to interpolate the values of G at neighboring grid points, $G_{i\pm1}$ and G_i, to get an approximation. We use a "numerical flux function" introduced in the following definition taken from [30].

Definition: A scheme is in *conservation form* if there exists a numerical flux function $g(u,v)$ such that $g(u_{i-1}, u_i)$ and $g(u_i, u_{i+1})$ approximate values for $G_{i-\frac{1}{2}}$ and $G_{i+\frac{1}{2}}$ respectively, such that

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = -\frac{G(u^n_{i+\frac{1}{2}}) - G(u^n_{i-\frac{1}{2}})}{\Delta x}.$$

To ensure that the scheme satisfies the entropy condition, it must be in conservation form and must satisfy one more restriction. If the scheme is of the form

$$u_i^{n+1} = W(u^n_{i-1}, u^n_i, u^n_{i+1})$$

for some function W, then we say that the scheme is *monotone* if W is a non-decreasing function of all of its arguments. The key fact, proven in [33] (see also [20]), is the statement

> A conservative, monotone scheme produces a solution that satisfies the entropy condition.

So, with a scheme in conservation form, all we need to do is check that it is monotone. One simple scheme, called the *Lax-Friedrichs scheme,* is given by

$$u_i^{n+1} = \frac{1}{2}\left[u_{i-1}^n + u_{i+1}^n\right] - \frac{\Delta t}{2}\frac{G_{i+1} - G_{i-1}}{\Delta x}.$$

By inspection, the scheme is monotone if $\frac{\Delta t}{\Delta x}\frac{dG}{du} \leq 1$. We can put it into conservation form via the numerical flux function

$$g_{LF}(u_1, u_2) = \frac{1}{2}\left[G(u_2) + G(u_1)\right] - \frac{\Delta x}{2\Delta t}(u_2 - u_1)$$

Notice that the Lax-Friedrichs scheme does not require us to know many properties of G. However, if we know that G is convex $\left(\frac{d^2G}{du^2} > 0\right)$ then one can limit smearing even more.

Consider the *Engquist-Osher* flux function given by

$$g_{EO}(u_1, u_2) = G(u_1) + \int_{u_1}^{u_2} \min\left(\frac{dG}{du}, 0\right) du, \tag{3.11}$$

and assume that $\frac{d^2G}{du^2} > 0$. Equation (3.9) gives

$$u_t + \frac{dG}{du}u_x = 0$$

which indicates that $\frac{dG}{du} =: a(u)$ represents the "speed" of propagation of the characteristics (this follows from standard PDE theory) in the xt-plane.

The form of the scheme (3.11) depends on the sign of the speed between u_1 and u_2:

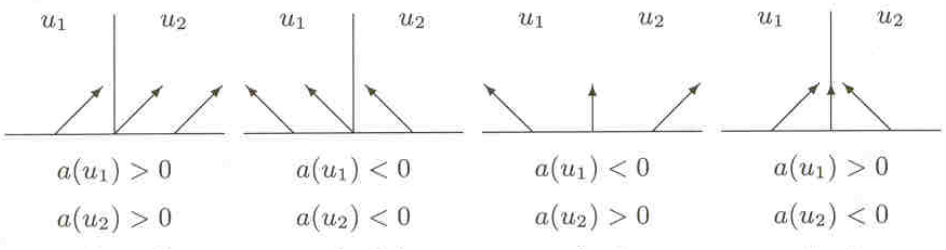

Figure 3.8: The four cases of neighboring propagating characteristics.

If $a(u_1) > 0$ and $a(u_2) > 0$, then $\frac{dG}{du} > 0$ (note $u \in [u_1, u_2]$) and $g_{EO}(u_1, u_2) = G(u_1)$ and so the wave moves to the right using an upwind scheme.

If $a(u_1) < 0$ and $a(u_2) < 0$, then $\frac{dG}{du} < 0$ and $g_{EO}(u_1, u_2) = G(u_1) + G(u_2) - G(u_1) = G(u_2)$ and so the wave moves to the left using an upwind scheme.

If $a(u_1) < 0$ and $a(u_2) > 0$, then there is a rarefaction zone between u_1 and u_2 and, since $\frac{dG}{du} > 0$ for $u > a^{-1}(0)$, we have

$$\begin{aligned} g_{EO}(u_1, u_2) &= G(u_1) + \int_{u_1}^{a^{-1}(0)} \frac{dG}{du}\, du \\ &= G(u_1) + G(a^{-1}(0)) - G(u_1) \\ &= G(a^{-1}(0)) \end{aligned}$$

so the scheme uses information from where the speed of the characteristics is zero.

If $a(u_1) > 0$ and $a(u_2) < 0$, then a shock forms and

$$\begin{aligned} g_{EO}(u_1, u_2) &= G(u_1) + \int_{a^{-1}(0)}^{u_2} \frac{dG}{du}\, du \\ &= G(u_1) + G(u_2) - G(a^{-1}(0)). \end{aligned}$$

According to [30], the first three cases give the exact solution and the fourth case introduces "a little diffusion." Depending on the speed of the shock, either $G(u_1)$ or $G(u_2)$ should be selected. For example, if the speed of the shock is positive, then $G(u_1)$ should be selected (for upwinding) and the quantity $G(u_2) - G(a^{-1}(0))$ represents diffusion in the solution. Sethian indicates that the converging characteristics "sharpen things up" again, so that the diffusion is less severe than in other schemes.

For the non-linear wave equation

$$u_t + \left[u^2\right]_x = 0 \tag{3.12}$$

the Engquist-Osher flux takes the form

$$\begin{aligned} g_{EO}(u_1, u_2) &= u_1^2 + \int_{u_1}^{u_2} \min(u, 0)\, du \\ &= \max(u_1, 0)^2 + \min(u_2, 0)^2, \end{aligned} \tag{3.13}$$

which follows from the fact that $G(u) = u^2$, $a(u) = 2u$, and $a^{-1}(0) = 0$. The above result satisfies all four cases, and is easy to implement numerically. Now, we seek to modify the above numerical scheme so that it approximates solutions to the governing equation

$$\phi_t + F\,|\nabla\phi| = 0 \tag{3.14}$$

in one or more dimensions.

3.3.2 Schemes for solving the governing equation

One Dimensional Case

Recall that the governing equation in one dimension is

$$\phi_t + H(\phi_x) = 0,$$

where $H(u) = F\,|u|$, and is equivalent to the hyperbolic conservation law

$$u_t + [H(u)]_x = 0,$$

with $u = \phi_x$. Note that the Hamiltonian $H(u)$ plays the same role as $G(u)$ in the previous section.

One can approximate the solution to the first equation by writing

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = -H(u_i^n).$$

Using the above theory, we can approximate $H(u_i^n)$ with the numerical flux function:

$$H(u_i^n) \approx g(u_{i-\frac{1}{2}}^n, u_{i+\frac{1}{2}}^n).$$

Knowing that $u = \phi_x$, we can approximate $u^n_{i-\frac{1}{2}}$ and $u^n_{i+\frac{1}{2}}$ with difference schemes. It is appropriate to use a backward and forward scheme, respectively. Thus, we have the scheme

$$\phi_i^{n+1} = \phi_i^n - \Delta t\, g\left(D_x^- \phi_i^n, D_x^+ \phi_i^n\right) \tag{3.15}$$

where g is a numerical flux function. The simplest thing to do is to modify the Engquist-Osher flux (3.13), which was an approximation to $G(u) = u^2$, to approximate our Hamiltonian $H(u) = F\sqrt{u^2}$:

$$g(u_1, u_2) = F\sqrt{\max(u_1, 0)^2 + \min(u_2, 0)^2}.$$

This can be done provided that $F > 0$ and is convex. If $F < 0$, the situation is reversed (imagine the characteristics flowing in the deceasing t direction in the xt-plane). To accommodate both cases, our scheme can be written as

$$\begin{aligned} g(u_1, u_2) &= \max(F, 0)\left(\max(u_1, 0)^2 + \min(u_2, 0)^2\right)^{\frac{1}{2}} \\ &\quad + \min(F, 0)\left(\max(u_2, 0)^2 + \min(u_1, 0)^2\right)^{\frac{1}{2}} \end{aligned} \tag{3.16}$$

where F is the speed at the appropriate grid point. To be clear, the flux function can be written as

$$\begin{aligned} g(u^n_{i-\frac{1}{2}}, u^n_{i+\frac{1}{2}}) &= \max(F_i, 0)\left(\max(u^n_{i-\frac{1}{2}}, 0)^2 + \min(u^n_{i+\frac{1}{2}}, 0)^2\right)^{\frac{1}{2}} \\ &\quad + \min(F_i, 0)\left(\max(u^n_{i+\frac{1}{2}}, 0)^2 + \min(u^n_{i-\frac{1}{2}}, 0)^2\right)^{\frac{1}{2}} \end{aligned} \tag{3.17}$$

where $F_i = F(x_i)$ is the speed at grid point i. Thus for a convex speed $F(x)$, (3.15) together with (3.17) fully define the scheme for updating ϕ at each time step.

Higher dimensional cases

It is straightforward to generalize the scheme to higher dimensions. In two dimensions, the governing equation is

$$\phi_t + F\sqrt{\phi_x^2 + \phi_y^2} = 0$$

and so the Hamiltonian $H(u,v) = F\sqrt{u^2+v^2}$ is symmetric in all of its arguments. This being so, it is common practice to simply replicate the scheme in each space variable, as done in [30]:

$$\phi_{ij}^{n+1} = \phi_{ij}^n - \Delta t\, g\left(D_x^-\phi, D_x^+\phi, D_y^-\phi, D_y^+\phi\right) \tag{3.18}$$

with

$$\begin{aligned} g(u^n_{i-\frac{1}{2}}, u^n_{i+\frac{1}{2}}, v^n_{i-\frac{1}{2}}, v^n_{i+\frac{1}{2}}) \;&= \max(F_{ij},0)\;\Big\{\max(u^n_{i-\frac{1}{2}},0)^2 + \min(u^n_{i+\frac{1}{2}},0)^2 \\ &\qquad + \max(v^n_{i-\frac{1}{2}},0)^2 + \min(v^n_{i+\frac{1}{2}},0)^2\Big\}^{\frac{1}{2}} \\ &\quad + \min(F_{ij},0)\;\Big\{\max(u^n_{i+\frac{1}{2}},0)^2 + \min(u^n_{i-\frac{1}{2}},0)^2 \\ &\qquad + \max(v^n_{i+\frac{1}{2}},0)^2 + \min(v^n_{i-\frac{1}{2}},0)^2\Big\}^{\frac{1}{2}} \end{aligned} \tag{3.19}$$

where $F_{ij} = F(x_i, y_j)$, and the arguments in (3.18) correspond to the arguments in (3.19); that is, $u^n_{i-\frac{1}{2}} = D_x^-\phi$, $u^n_{i+\frac{1}{2}} = D_x^+\phi$, $v^n_{i-\frac{1}{2}} = D_y^-\phi$, and $v^n_{i+\frac{1}{2}} = D_y^+\phi$. This scheme also requires that F be convex[2], that is $\frac{\partial^2 F}{\partial x \partial y} > 0$. In three dimensions, one uses an analogous scheme:

$$\phi_{ijk}^{n+1} = \phi_{ijk}^n - \Delta t\, g\left(D_x^-\phi, D_x^+\phi, D_y^-\phi, D_y^+\phi, D_z^-\phi, D_z^+\phi\right) \tag{3.20}$$

with flux

$$\begin{aligned} g(u^n_{i-\frac{1}{2}}, u^n_{i+\frac{1}{2}}, v^n_{i-\frac{1}{2}}, v^n_{i+\frac{1}{2}}, w^n_{i-\frac{1}{2}}, w^n_{i+\frac{1}{2}}) \;&= \max(F_{ijk},0)\;\Big\{\max(u^n_{i-\frac{1}{2}},0)^2 + \min(u^n_{i+\frac{1}{2}},0)^2 \\ &\qquad + \max(v^n_{i-\frac{1}{2}},0)^2 + \min(v^n_{i+\frac{1}{2}},0)^2 \\ &\qquad + \max(w^n_{i-\frac{1}{2}},0)^2 + \min(w^n_{i+\frac{1}{2}},0)^2\Big\}^{\frac{1}{2}} \\ &\quad + \min(F_{ijk},0)\;\Big\{\max(u^n_{i+\frac{1}{2}},0)^2 + \min(u^n_{i-\frac{1}{2}},0)^2 \\ &\qquad + \max(v^n_{i+\frac{1}{2}},0)^2 + \min(v^n_{i-\frac{1}{2}},0)^2 \\ &\qquad + \max(w^n_{i+\frac{1}{2}},0)^2 + \min(w^n_{i-\frac{1}{2}},0)^2\Big\}^{\frac{1}{2}}. \end{aligned} \tag{3.21}$$

[2]In N dimensions, a function $F(x_1, ..., x_N)$ is convex if $F(\lambda p + (1-\lambda)q) \le \lambda F(p) + (1-\lambda)F(q)$ for all $0 \le \lambda \le 1$, $p, q \in \mathbb{R}^N$. Equivalently, F is convex if $\frac{\partial^2 F}{\partial x_i \partial x_j} \ge 0$ for all $i, j = 1, 2, ..., N$.

Curvature Dependent Speed

Suppose that F takes the form $F = F_0 - \varepsilon\kappa$ where F_0 is constant, $\varepsilon > 0$ is constant, and $\kappa = \kappa(x, y)$ is the curvature. It may be useful to use this form of a curvature dependent speed since it makes the governing equation equivalent to the viscosity equation (which yields the entropy solution) $\phi_t + F_0 |\nabla\phi| = \varepsilon\kappa |\nabla\phi|$, which is the same as the governing equation for constant speed motion, except now there is a diffusive term on the right hand side. To see this, consider the case where $|\nabla\phi| = 1$, which implies that the right-hand side becomes $\varepsilon\nabla^2\phi$ (using the definition of κ below) which is a parabolic term. For example, this term appears in the heat equation $\phi_t = \varepsilon\nabla^2\phi$ where ε is the diffusion coefficient. The curvature κ can be approximated using a central difference approximation which uses information from all directions. Diffusion caused by this approximation is not a major problem, since the curvature term is already diffusive.

In two dimensions, curvature is given by

$$\begin{aligned} \kappa &= \nabla \bullet \frac{\nabla\phi}{|\nabla\phi|} \\ &= \frac{\phi_{xx}\phi_y^2 - 2\phi_y\phi_x\phi_{xy} + \phi_{yy}\phi_x^2}{\left(\phi_x^2 + \phi_y^2\right)^{3/2}}. \end{aligned}$$

Numerically, one can approximate this quantity κ_{ij} at grid point (i, j) by using the above formula with the following approximations:

$$\begin{aligned} [\phi_{ij}]_x &\approx D_x^0\phi_{ij} = \frac{\phi_{i+1,j} - \phi_{i-1,j}}{2\Delta x} \\ [\phi_{ij}]_y &\approx D_y^0\phi_{ij} = \frac{\phi_{i,j+1} - \phi_{i,j-1}}{2\Delta y} \\ [\phi_{ij}]_{xx} &\approx D_x^+ D_x^-\phi_{ij} = \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} \\ [\phi_{ij}]_{yy} &\approx D_y^+ D_y^-\phi_{ij} = \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} \\ [\phi_{ij}]_{xy} &\approx D_x^0 D_y^0\phi_{ij} = \frac{\phi_{i+1,j+1} - \phi_{i+1,j-1} - \phi_{i-1,j+1} + \phi_{i-1,j-1}}{4\Delta x\Delta y}. \end{aligned}$$

In three dimensions, one can use Gaussian curvature or mean curvature (among other

types of curvature). The formula for mean curvature is simpler and is given by

$$\kappa = \frac{\left\{ \begin{array}{c} \left(\phi_{yy} + \phi_{zz}\right)\phi_x^2 + \left(\phi_{xx} + \phi_{zz}\right)\phi_y^2 + \left(\phi_{xx} + \phi_{yy}\right)\phi_z^2 \\ -2\phi_x\phi_y\phi_{xy} - 2\phi_x\phi_z\phi_{xz} - 2\phi_y\phi_z\phi_{yz} \end{array} \right\}}{\left(\phi_x^2 + \phi_y^2 + \phi_z^2\right)^{3/2}}.$$

Hence, the two-dimensional scheme for curvature dependent speed is

$$\phi_{ij}^{n+1} = \phi_{ij}^n - \Delta t\, g\left(D_x^-\phi, D_x^+\phi, D_y^-\phi, D_y^+\phi\right) + \Delta t\, \varepsilon\kappa_{ij}\left|\nabla\phi\right|$$

(where the last term is approximated by central differences) and the three-dimensional scheme is

$$\phi_{ijk}^{n+1} = \phi_{ijk}^n - \Delta t\, g\left(D_x^-\phi, D_x^+\phi, D_y^-\phi, D_y^+\phi, D_z^-\phi, D_z^+\phi\right) + \Delta t\, \varepsilon\kappa_{ijk}\left|\nabla\phi\right|$$

where the notation $\phi_{ijk}^n = \phi(x_i, y_j, z_k, t_n)$ is used, and the difference approximations are generalized to 3D in a straightforward manner. For example,

$$\left[\phi_{ijk}\right]_{xz} \approx D_x^0 D_z^0 \phi_{ijk} = \frac{\phi_{i+1,j,k+1} - \phi_{i+1,j,k-1} - \phi_{i-1,j,k+1} + \phi_{i-1,j,k-1}}{4\Delta x \Delta z}$$

Boundary Conditions

Since the computational domain is finite, boundary conditions are required at the edges. The simple approach taken in [30] is to use a layer of ghost cells on each edge of the computational domain, whose values are direct copies of their neighbor in the computational domain. See the code in the appendix for details.

Signed Distance Functions

The following discussion applies to 2 and 3 space dimensions and the term "curve" is used to describe a curve in the 2 dimensional case or a surface in the 3 dimensional case.

The equation (3.2) has a particularly simple solution if ϕ is a signed distance function

$$\phi(\vec{x}, t) = \pm d(\vec{x}, \gamma(t))$$

where $d(\vec{x}, \gamma(t))$ is the distance from $\vec{x}$ to the nearest point on the evolving curve $\gamma(t)$, the sign is chosen as in (3.1), and $F = F_0$ in (3.2) is a constant speed function. This being the case, it follows that $|\nabla\phi| = 1$ for all t and the governing equation (3.2) reduces to

$$\phi_t + F_0 = 0$$

whose solution is simply

$$\phi = -F_0 t + \phi^0$$

where $\phi^0 = \phi(\vec{x}, 0)$ is the initial LS function at $t = 0$. In fact, if ϕ^0 is a signed distance function, and the above formula is used to calculate ϕ at any time t, then ϕ will remain a signed distance function for all t because the above update formula simply adds a constant value to ϕ, which is equivalent to selecting the level set $\phi = F_0 t$ as the curve of interest.

To see that $|\nabla\phi| = 1$ when ϕ is a signed distance function, consider the following heuristic argument. Let $\vec{a}$ be a point on the level curve $\phi = 0$, assumed for now to be smooth (say $\phi \in C^2$). Observe that the path of minimum distance from a point near $\vec{a}$ to the level curve lies along a line normal to the level curve, and hence parallel to $\nabla\phi(\vec{a})$. The rate of change of ϕ with respect to distance along this normal line is the rate of change of distance with respect to distance, and is therefore equal to 1. Since the directional derivative in the normal direction is $\nabla\phi \bullet \hat{n} = \nabla\phi \bullet \frac{\nabla\phi}{|\nabla\phi|} = |\nabla\phi|$, it follows that $|\nabla\phi| = 1$ at every point $\vec{a}$ on the level curve $\phi = 0$. At nonzero level curves $\phi = C$, the same argument applied to the function $\phi(\vec{x}, t) - C = \pm d(\vec{x}, t) - C$ gives $|\nabla(\phi - C)| = 1$ which implies that $|\nabla\phi| = 1$ on every point on the level curve $\phi = C$. So $|\nabla\phi| = 1$ everywhere that the level curves are locally smooth. Note that this accounts for almost all points in the domain, as sharp corners are generally very sparse.

The CFL Condition

For stability reasons, it is required that no level set crosses more than one grid cell during a single time step. This is intuitively clear since we only use information as far as one cell away to update the value of each cell. Thus, we can impose the restriction

$$\max_{\Omega} F\Delta t \leq h$$

where $h = \min(\Delta x, \Delta y, \Delta z)$ is the smallest grid spacing and Ω is the *entire* domain. This condition comes from the *Courant-Friedreichs-Lewy (CFL) condition*, which states that the numerical domain of dependence should contain the mathematical domain of dependence. Thus, since our numerical domain of dependence at each point includes grid cells at a distance of one cell away from the point, it follows that information should be moving no more quickly than one grid cell per timestep. The reason why we restrict the motion of *all* level sets (over the entire domain), and not just $\phi = 0$, is that instabilities can "spread" from other level sets to affect the zero level set (which defines the moving interface). There is an example of this in figure 4.6 in the next chapter. It is important to note that in general the CFL condition is a neccessary but not sufficient condition for numerical stability.

Efficiency

The programs used in the next two chapters are not very time-consuming to run. On a 300 MHz processor with 128 Mb of available RAM, the 2-dimensional programs had a run time of 1-15 minutes, the three dimensional program for surface propagation completes within 30 minutes, and the 3-dimensional program for ventricle segmentation (see Chapter 5) ran in about one hour. These times depend on the mesh size, of course. Normally, a 100×100 grid is used in 2 dimensions and a $100 \times 100 \times 40$ grid is used in 3 dimensions. As the mesh is refined, the run time increases quickly (and so does the accuracy). In fact, the run time for each time step is $O(N^n)$, where n is the dimension and N is the number of cells in each dimension. Furthermore, the time step Δt is proportional to mesh spacing h (because of the CFL condition), and so $\Delta t \to 0$ as $h \to 0$, so more time steps are needed as the grid is refined, increasing the computational cost even more. If approximately N time steps are needed, then, the complexity becomes $O(N^n N) = O(N^{n+1})$.

The programs in the appendix have been "vectorized" for efficiency, which means that computations on an array (such as the 3D array representing ϕ_{ijk}) are performed using special parallel computing Matlab routines and not iterated loops. Further optimizations can be made, such as "narrow banding" in which one only keeps track of grid points in a narrow band near $\phi = 0$ and not the entire domain. See [30] for details on this and other optimization strategies.

Chapter 4

Simulating Ventricular Motion

4.1 Front Evolution in Two Dimensions

4.1.1 Algorithm

In two dimensions, we use the following algorithm to propagate the curve, assuming knowledge of the initial value of the LS function $\phi(x, y, 0)$ (which can be obtained by manual tracing or semi-automatically using the method of the next chapter):

```
plot initial contour φ_ij = 0 using a contour plotter
for t = 0 to T with increment Δt
  for i, j
    if CFL condition is satisfied
      update φ_ij according to (3.18)
    else
      update φ_ij according to (3.18) recursively, with smaller timesteps
    end if
  end for
  plot contour φ_ij = 0 at specified points in time (e.g. t = 0, 2, 4, 6, 8, 10)
end for
```

For more details, see the Matlab code in the appendix.

4.1.2 Simple Benchmark Tests

To start with a simple example, we propagate a circle inward, with speed $F = -1$, and outward, with speed $F = +1$. Recalling the discussion of signed-distance functions, one sees that if the circle is initialized as a signed distance function, then the numerical techniques of the last chapter are not neccessary. However, to test the techniques we will use it anyway. Figure 4.1 shows the results. Indeed, since the circle moves by 20 units of

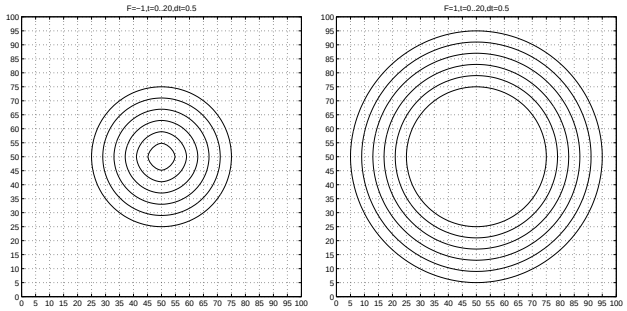

Figure 4.1: Shrinking and expanding a circle.

distance at a speed of 1 over 20 time units, the code passes the first simple test.

Now, to test the method's ability to handle shocks and rarefactions, we shrink and expand a square in a similar fashion (see figure 4.2). Indeed again, we see that the corners

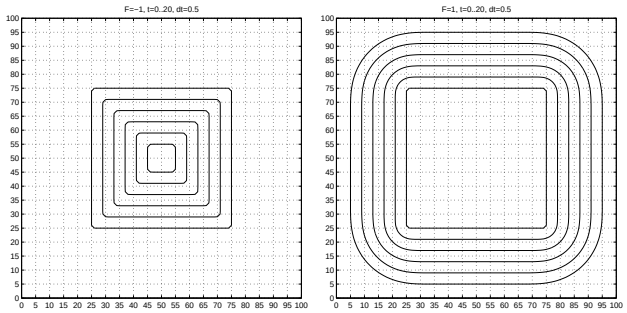

Figure 4.2: Shrinking and expanding a square.

stay reasonably sharp (as sharp as the finite grid and contour plotter allow them to) as the curve shrinks (this is a result of a shock) and that the corners become rounded as it expands (a result of a rarefaction fan). As a further demonstration, we shrink a square

under curvature-depended speed $F = -1 - \varepsilon\kappa$, where $\varepsilon = 5$, and we see that the corners become smooth as expected (figure 4.3).

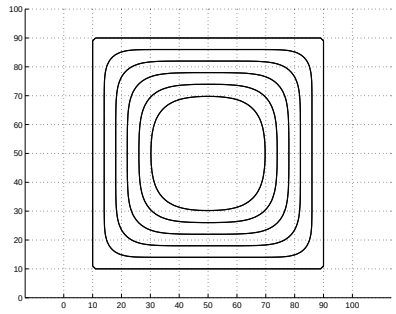

Figure 4.3: Shrinking a square under curvature dependent speed.

Figure 4.4 shows an example of topology change as a square-intersect-circle shape evolves under $F = -1$.

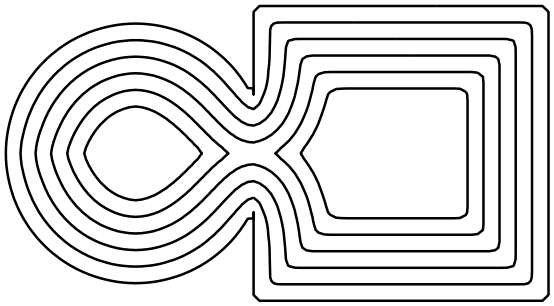

Figure 4.4: Illustration of topology change as the curve breaks into two.

Another test involves shrinking and then expanding the circle and square in sequence. In the case of the circle, one expects to recover the original shape after shrinking and expanding the same distance. In the case of a square, the shrinking results in loss of information (due to the corners), and so we expect a difference in the initial and final shape. As the square expands again, its corners will become rounded due to the rarefaction fans in the solution. If one expands first and then shrinks the square, then no information is lost and the original shape is recovered, with a little smoothing due to numerical errors. See figure 4.5.

Finally, to demonstrate instability arising in the solution, we choose a time step and speed which violate the CFL condition. Figure 4.6 shows snapshots of various level sets

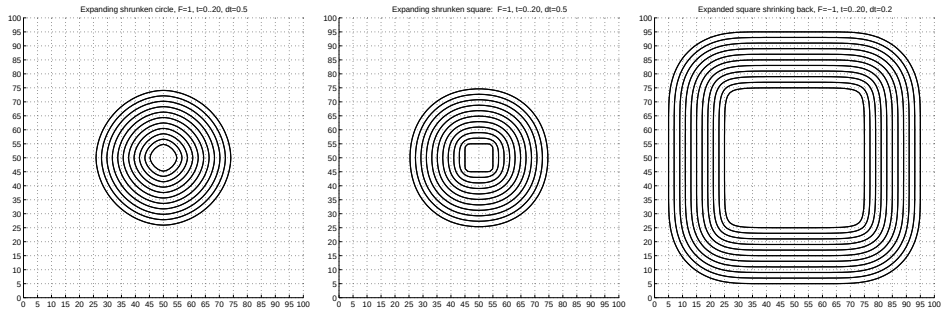

Figure 4.5: Expanding a shrunken circle (left) and square (centre) at constant speed for the same amount of time; shrinking an expanded square (right).

of the LS function as time goes by. Note that the instability starts away from the zero level set, but spreads quickly.

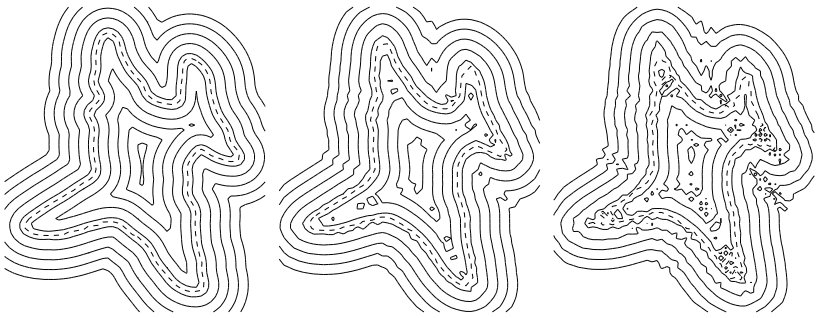

Figure 4.6: An example of a spreading instability at different points in time. The dashed contour represents the zero level set, and the other contours are nonzero level sets.

These simple tests do not give a rigorous estimate of the accuracy of the numerical method, but merely illustrate that it is quite plausible that the method accurately tracks the motion of the front. More rigorous tests and discussion are found in [24] and [30], and they are not pursued here.

4.1.3 Real examples

Pre-shunt and post-shunt 2-dimensional CT scans for eight patients are used. The ventricles are segmented using the technique of the next chapter. Figure 4.7 shows the CT scans of one patient with contours segmenting the ventricles. The goal is to predict the post-shunt curve from the pre-shunt curve. Using $F = -1$ we can shrink the pre-shunt

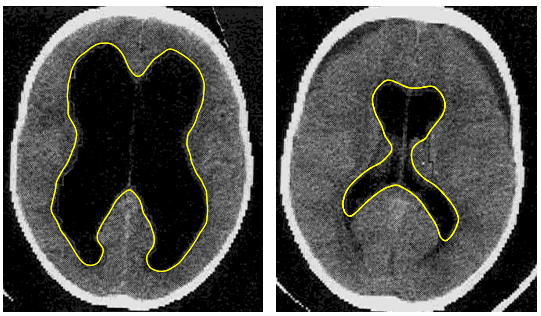

Figure 4.7: Pre-shunt and post-shunt images, showing the segmented ventricles. A typical time period between these scans is 1 year.

curve. The problem is, we don't know *when* to stop. Alternatively, using $F = -F_0$ for fixed time T, we don't know what the value of F_0 should be.

One way to temporarily avoid this problem is to assume knowledge of the final area of the ventricles, and use $F = -1$ until the areas match. The results are shown in figure 4.8. Another way is to use $F = -F_0$ for fixed time T and estimate F_0 by taking the average distance between pre and post curves and dividing by T.

The above methods use knowledge about the final curve to predict the final curve, and are therefore unsatisfactory. However, one could use knowledge of a sample (one or more) of the final curves in order to predict the other final curves. Figure 4.9 shows the results when we use knowledge of the final area of the first four patients to evolve the curves for the final four patients. It is unclear whether ε is an important parameter here, since the curves with $\varepsilon = 0$ are very similar to those with $\varepsilon = 5$. It has been included to give some generality to the speed function, and to keep the curve smooth. Even the value $\varepsilon = 5$ was chosen by trial and error, as smaller values do not make significant overall effects, and larger values make the calculation unstable (or require prohibitively small timesteps).

Time-dependent Speeds

Also, it is worth noting that time-dependent speeds can be used. Although the theory in the previous chapter assumes a constant-in-time speed function $F(x, y)$, in practice one can use a piecewise-constant speed which is constant over each timestep. An example is shown in figure 4.10. Since the time-dependence does not significantly affect the shape of the final contour, except when ε is very large, we do not pursue the time-dependent

speed here. However, when time-series data becomes available, then piecewise-constant speeds could be used to make predictions from one time-point to another.

4.2 Surface evolution in three dimensions

4.2.1 Algorithm

Again assuming knowledge of $\phi(x, y, z, t = 0)$, which can be obtained manually by tracing each slice of a 3D MR scan or by the method of the next chapter, we use the following algorithm:

```
draw initial isosurface φ_ijk = 0 using an isosurface plotter
for t = 0 to T with increment Δt
  for i, j, k
    if CFL condition is satisfied
      update φ_ijk according to (3.20)
    else
      update φ_ijk according to (3.20) recursively, with smaller timesteps
    end if
  end for
draw isosurface φ_ijk = 0 at specified points in time (e.g. t = 0, 2, 4, 6, 8, 10)
end for
```

Again, see the Matlab code in the appendix.

4.2.2 Benchmark examples: a cube and a dumbell

First, we shrink and expand a cube in the same manner as we tested the square in section 4.1.2. Results are shown in figures 4.11 and 4.12.

The second example involves a "dumbell": two spheres connected with a cylinder. We shrink the dumbell under $F = -1 - \varepsilon\kappa$ for $\varepsilon = 0$ and 3 as shown in figure 4.13. When $\varepsilon = 0$, expanding the final shrunken surface from figure 4.13 for the same amount of time under speed $F = +1$ yields the surface in figure 4.14. Notice that the permanent loss of the handle indicates an effect of entropy on the evolving surface.

4.2.3 A Real Example

At the time of this writing, only one 3D MR scan of a hydrocephalic brain was available to us. Evolving the ventricles under $F = -1 - \varepsilon\kappa$ for $\varepsilon = 0$ and 3 is shown in figure 4.15 and 4.16. Since we know the length units on this data, we can interpret T as the number of months and let F be the speed of the ventricular wall in millimeters per month, where the mesh spacing Δx, Δy, and Δz are in millimeters.

4.3 Determining the Speed Field F & the Shrinking Transformation

4.3.1 Motivation

Using knowledge of the post-shunt LS function, we now present a way to find a speed field F which will result in the deformation of the pre-shunt curve to the post-shunt curve.

Consider the two dimensional case. Let $\phi_1(x, y) = \phi(x, y, t = t_1)$ be the pre-shunt LS function, and $\phi_2(x, y) = \phi(x, y, t = t_2)$ be the post-shunt LS function. The idea is to solve the governing equation (3.2) for F, to get

$$F = \frac{-\phi_t}{|\nabla\phi|}$$

and then assume that $\phi(x, y, t)$ linearly interpolates ϕ_1 and ϕ_2

$$\phi(x, y, t) = \frac{t_2 - t}{t_2 - t_1}\phi_1 + \frac{t - t_1}{t_2 - t_1}\phi_2 \tag{4.1}$$

so that

$$\phi_t = \frac{\phi_2 - \phi_1}{t_2 - t_1}$$

which is a constant in time. One could then run the algorithm for curve propagation with speed

$$F(x, y, t) = \frac{\phi(x, y, t_1) - \phi(x, y, t_2)}{(t_2 - t_1)\,|\nabla\phi(x, y, t)|}. \tag{4.2}$$

An example of this is shown in figure 4.17.

Note however that such an elaborate process is not neccessary, since (4.1) gives an expression for the LS function at any time t.

4.3.2 The Shrinking Transformation

In this section, this above idea is presented more simply and extended to use pre-shunt LS functions to predict unknown post-shunt LS functions.

Consider a set of pre-shunt LS functions

$$\phi^{Apre}, \phi^{Bpre}, ..., \phi^{Hpre}$$

defining the ventricles for each patient before shunting. Suppose that we know the post-shunt LS function ϕ^{Apost} for the first patient, say. Then one can let

$$\Delta\phi^A = \phi^{Apost} - \phi^{Apre}$$

so that

$$\phi^{Apost} = \phi^{Apre} + \Delta\phi^A,$$

which appears to be a formula that gives ϕ^{Apost} in terms of ϕ^{Apre} (of course, it is in terms of ϕ^{Apost} as well!). The key idea is to try to predict the post-shunt LS functions for other patients using the known $\Delta\phi^A$. For example, since

$$\phi^{Bpost} = \phi^{Bpre} + \Delta\phi^B$$

where $\Delta\phi^B = \phi^{Bpost} - \phi^{Bpre}$ is unknown (say ϕ^{Bpost} is unknown), we could try replacing $\Delta\phi^B$ with $\Delta\phi^A$:

$$\tilde{\phi}^{Bpost} = \phi^{Bpre} + \Delta\phi^A$$

Hopefully $\tilde{\phi}^{Bpost}$ will approximate ϕ^{Bpost} in some sense. Before trying this out, some notation and further considerations are useful.

Definition: Define the *shrinking transformation* $T_{A_i} : \mathbb{R} \to \mathbb{R}$ by

$$T_{A_i}\phi = \phi + \Delta\phi^{A_i}$$

where $\Delta\phi^{A_i} = \phi^{A_i post} - \phi^{A_i pre}$.

For example, $T_A\phi^{Bpre} = \phi^{Bpre} + \Delta\phi^A$.

Roughly speaking, the shrinking transformation T_{A_i} deforms its argument in the "same manner" as $\phi^{A_i pre}$ deforms to $\phi^{A_i post}$. Exactly how it does this is difficult to understand. Despite its simple, easy-to-calculate form, T_{A_i} is a very complicated, nonlinear transformation. It acts on the entire domain, and not just on the curves $\phi = 0$. Some geometric considerations can shed light on this transformation.

Understanding the shrinking transformation

Consider the 2D case for simplicity, and consider two circles with the same centre (x_0, y_0) as shown in figure 4.18. Let $\phi_1(x, y)$ be a LS function for the outer circle, and $\phi_2(x, y)$ be a LS function for the inner circle. Assuming that these are signed-distance functions and plotting $z = \phi(x, y_0)$ in the xz-plane as shown in figure 4.18, we can see some geometrical connections. First of all, since ϕ_1 and ϕ_2 are signed-distance functions, $|\nabla\phi_1| = |\nabla\phi_2| = 1$ everywhere (except at the of the circle). Furthermore, at $y = y_0$, $\frac{\partial\phi}{\partial y} = 0$ since the normal vector $\hat{n} = \frac{\nabla\phi}{|\nabla\phi|} = \nabla\phi$ has zero y-component. Thus $\left|\frac{\partial\phi}{\partial x}\right| = 1$ at $y = y_0$, so the slope of the lines in figure 4.18 is 1. Let A be the point shown on the inner curve and B be the point shown on the outer curve. Then $|\phi_2 - \phi_1|\,|_{at\ A\ or\ B} = AC = DB = AB$ as drawn on the diagram. Therefore, $|\phi_2 - \phi_1|$ evaluated at either A or B gives the distance from A to B. More generally, $|\phi_2 - \phi_1|$ at any point on either circle gives the distance between that point and the closest point on the other circle.

Generalizing further, if we consider two arbitrary curves or surfaces given by signed distance functions $\phi_1 = 0$ and $\phi_2 = 0$, then $\phi_2 - \phi_1$ at a point $\vec{a}$ on $\phi_1 = 0$ gives the signed distance between $\vec{a}$ and the nearest point on $\phi_2 = 0$, since $(\phi_2 - \phi_1)(\vec{a}) = \phi_2(\vec{a}) - \phi_1(\vec{a}) = \phi_2(\vec{a})$. Similarly, $\phi_2 - \phi_1$ at a point $\vec{b}$ on $\phi_2 = 0$ gives the negative of the signed distance between $\vec{b}$ and the nearest point on $\phi_1 = 0$. On different level sets, such as $\phi_1 = C$ for some C, $\phi_2 - \phi_1$ at a point $\vec{a}$ on $\phi_1 = C$ gives the signed distance to the closest point on the level set $\phi_2 = C$, if the level set exists. This can be seen by writing $\phi_2 - \phi_1 = (\phi_2 - C) - (\phi_1 - C)$ and using the above argument. Similarly, $\phi_2 - \phi_1$ at any point $\vec{b}$ on $\phi_2 = C$ gives the negative of the signed distance from $\vec{b}$ to the nearest point on $\phi_1 = C$, if the level set exists.

Since the nonzero level sets of a function are complicated in general, so too will be the transformation T_{A_i}. To further help visualize the transformation, which is represented as a scalar field, we plot an intensity image, showing various values of the additive term $\Delta\phi^{A_i}$ at the grid points. An example of a circle-to-square shrinking transformation is shown in figure 4.19, and a real example is shown in figure 4.20.

Applying the shrinking transformation

Assuming knowledge of $\Delta\phi^A$, we apply the transformation to $\phi^{Bpre}, \phi^{Cpre}, ..., \phi^{Hpre}$ in the hopes of approximating $\phi^{Bpost}, \phi^{Cpost}, ..., \phi^{Hpost}$. Results are shown in figure 4.21. Note that the images are of different sizes, so some resizing was done. See the code in the appendix for details.

One can also assume knowledge of $\Delta\phi^A, \Delta\phi^B, \Delta\phi^C$ and $\Delta\phi^D$, for example, to predict $\phi^{Epost}, \phi^{Fpost}, \phi^{Gpost}$, and ϕ^{Hpost}, by using the average of the known $\Delta\phi$'s in the transformation. An example of this is shown in figure 4.22. Conceivably, one could use a weighted average of known $\Delta\phi$'s in the transformation, where the weights are chosen according to the similarity of the pre-operative LS functions. For example, if the pre-shunt ventricles of patient D are very similar to those of patient A, then the transformation applied to ϕ^{Dpre} would use a heavier weight on $\Delta\phi^A$ than on the other three. However, the notion of 'similar' must be defined, and some way of comparing the images in a common coordinate system must be used. This lies in the area of image registration, and a proper treatment of this is beyond the scope of this work.

However, there is at least one simple way to 'match' two images, and it is worth a brief look.

Definition: Define the *converting transformation* by

$$T_{A_iA_j}\phi = \phi + \Delta\phi^{A_iA_j}$$

where $\Delta\phi^{A_iA_j} = \phi^{A_jpre} - \phi^{A_ipre}$.

The converting transformation is similar to the shrinking transformation, except that it maps one pre-shunt LS function to another pre-shunt LS function, as opposed to mapping the pre-shunt function to the post-shunt function.

The idea is to apply the converting transformation as well as the shrinking transformation to each LS function. For example, assuming knowledge of $\Delta\phi^A$ (and ϕ^{Apre}, ϕ^{Apost}) we can try to approximate ϕ^{Bpost} as

$$\begin{aligned}\tilde{\phi}^{Bpost} &= T_{AB}T_A\phi^{Bpre} \\ &= \phi^{Bpre} + \Delta\phi^A + (\phi^{Bpre} - \phi^{Apre}).\end{aligned}$$

Very loosely speaking, the term in brackets maps the other terms "from ϕ^A space into ϕ^B space," in the same way that ϕ^{Apre} is mapped to ϕ^{Bpre}. Results are shown in figure 4.22 and 4.23. Again, more knowledge of statistics and image registration would be needed to assess the value of this approach, and if proven valuable, it could perhaps be made more rigorous.

Generalizations

The above methods can be generalized to accommodate time-series data where the image of the brain is known at several times between the pre-shunt and post-shunt times. In this case the transformations would be piecewise-constant in time, or one could even interpolate to make the transformation fully continuous in time. More ideas are discussed at the end of chapter 6. Finally, the above methods generalize straighforwardly to three dimensions.

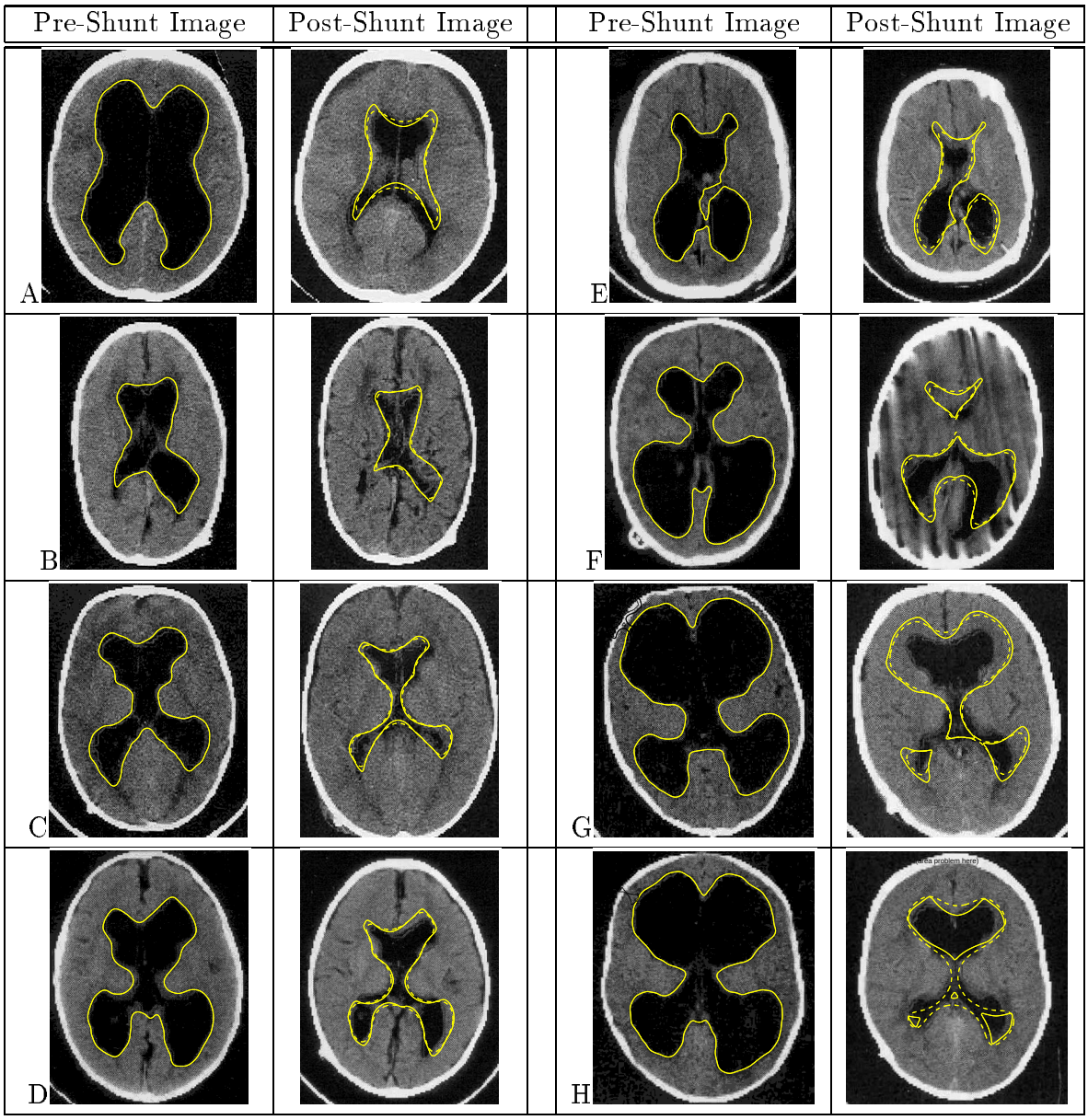

Figure 4.8: Assuming knowledge of final area, the pre-shunt curve is propagated under constant speed (solid lines) and curvature-dependent speed with $\varepsilon = 5$ (dashed line). The brain images are from the database of the hydrocephalus group at the Hospital for Sick Children.

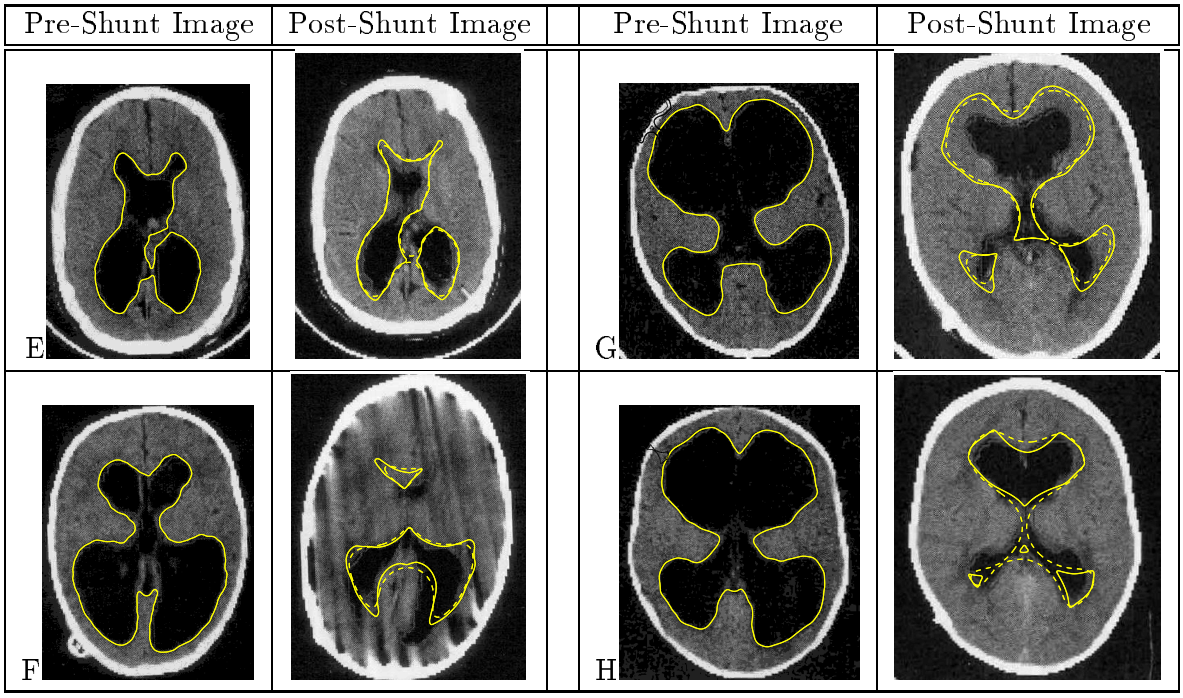

Figure 4.9: Using knowledge of the final area of the post-shunt ventricles for patients A-D, the same procedure used in figure 4.8 is carried out for patients E-H.

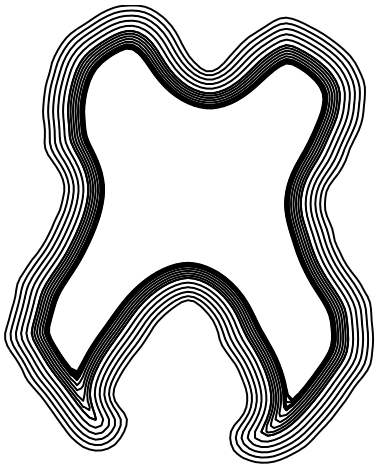

Figure 4.10: Using a time-dependent speed function $F(t)$ affects the spacing of the contours over time, but not the shape (assuming F does not depend on x or y). The initial contour is the outermost contour shown, and in this example it slows down after a certain time.

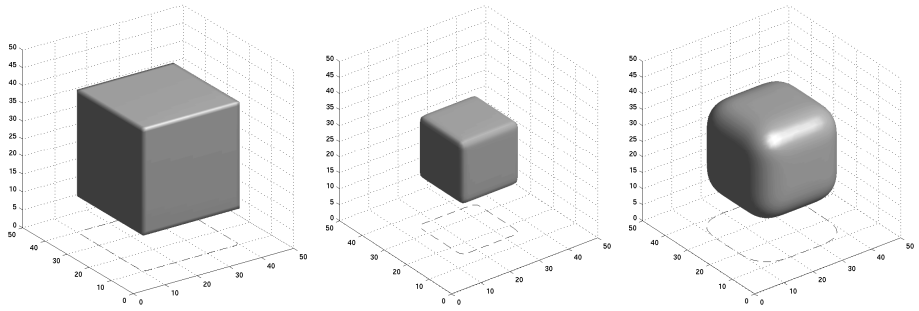

Figure 4.11: Shrinking then expanding a cube.

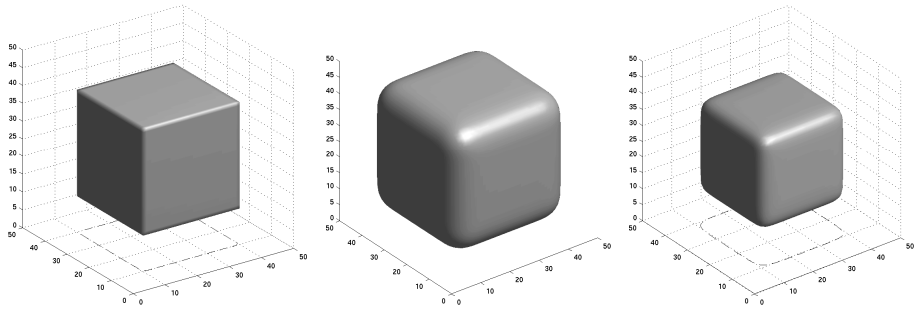

Figure 4.12: Expanding then shrinking a cube. The axes count the grid cells.

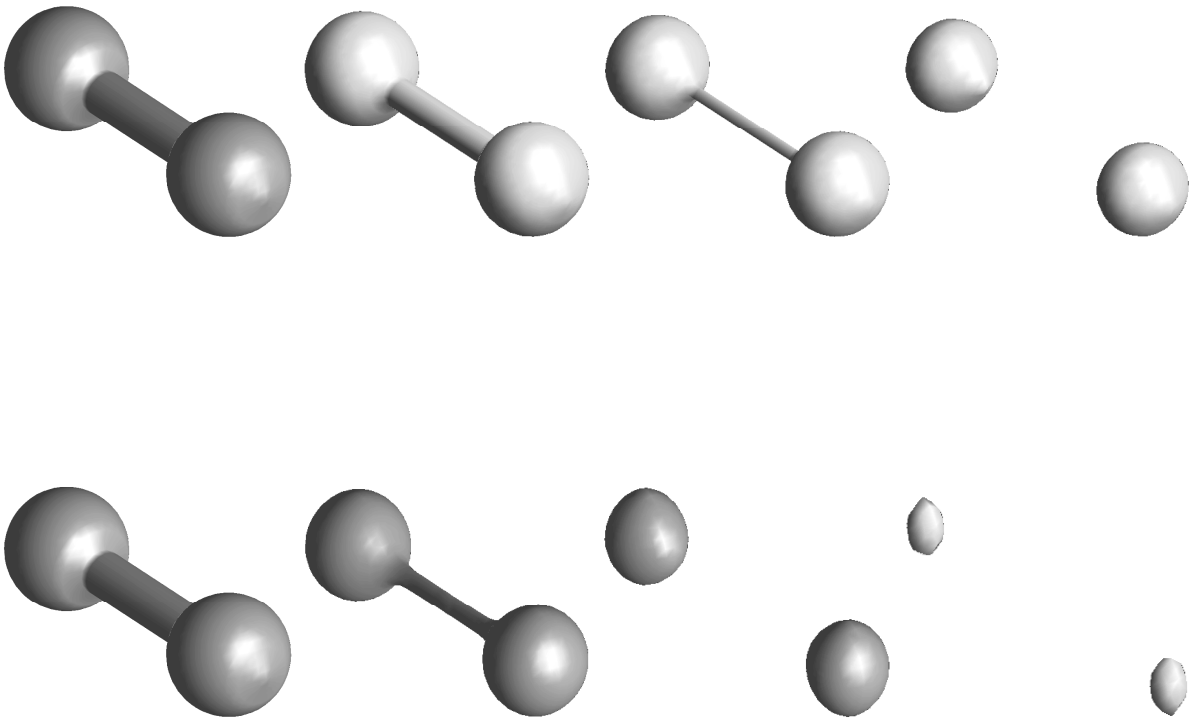

Figure 4.13: Shrinking a dumbell with $\varepsilon = 0$ (top) and $\varepsilon = 3$ (bottom). (A 30x30x80 grid was used, with speed $F_0 = -1$ for 0.9 time units between each surface plot.)

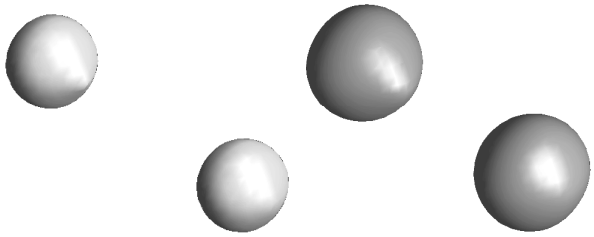

Figure 4.14: Expanding the shrunken dumbell from top half of figure 4.13.

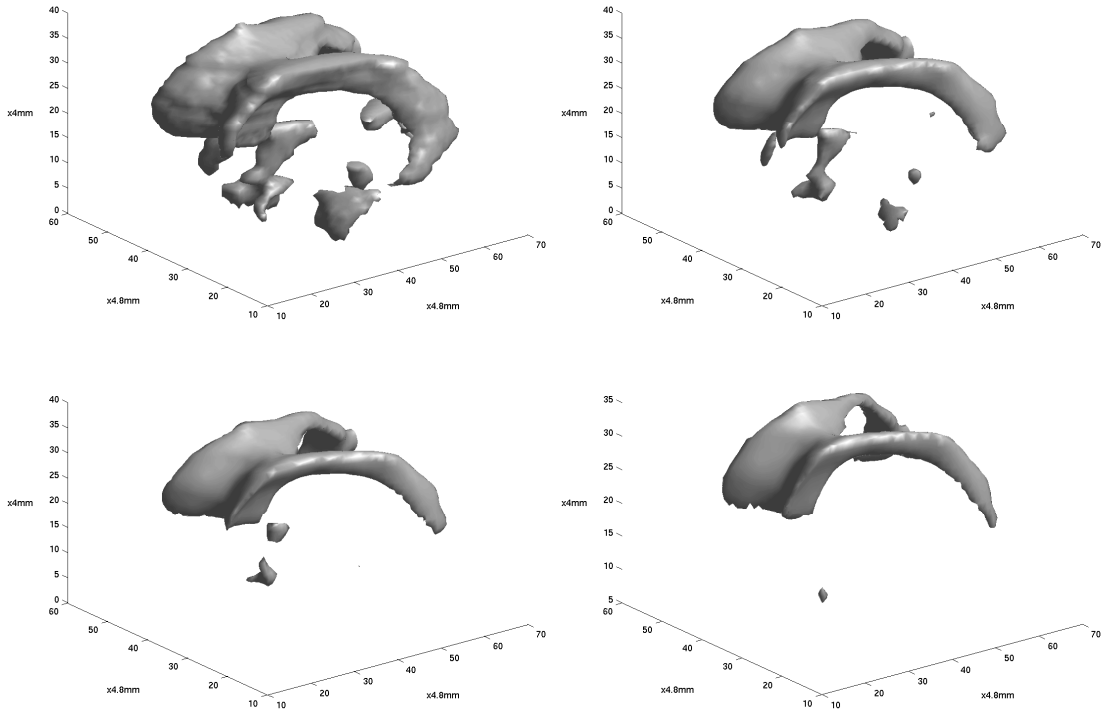

Figure 4.15: Shrinking the ventricles in 3D under constant speed $F = -1$ mm/month for 2, 4, and 6 months (spatial dimensions are in millimeters). The time sequence goes from left to right, top to bottom.

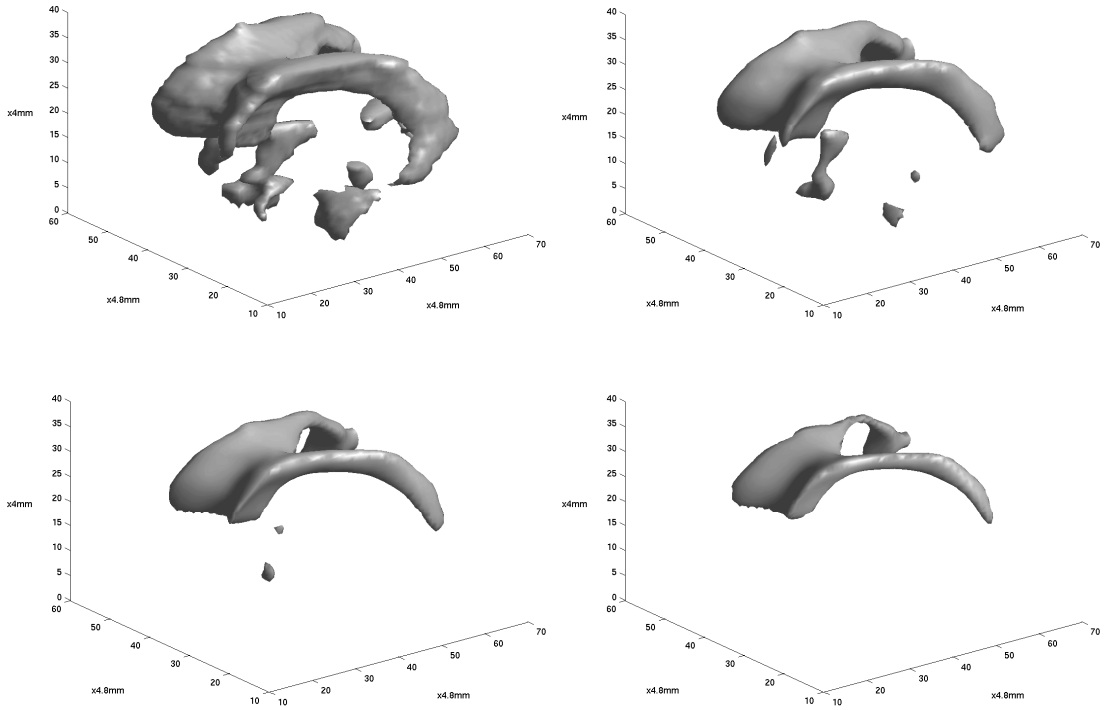

Figure 4.16: Shrinking the ventricles in 3D under curvature-dependent speed $F = -1-3\kappa$.

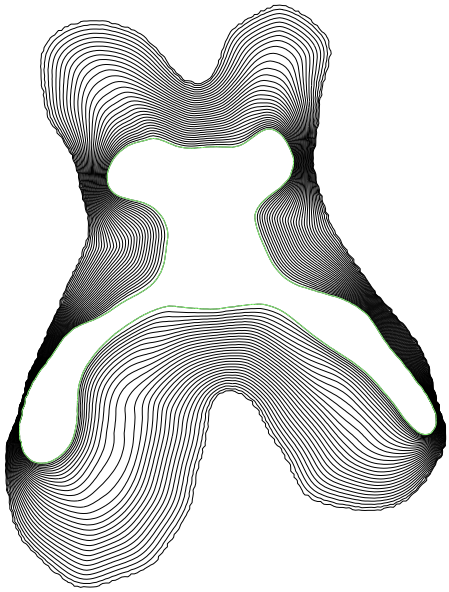

Figure 4.17: Deforming one curve to another by choosing F as in (4.2)

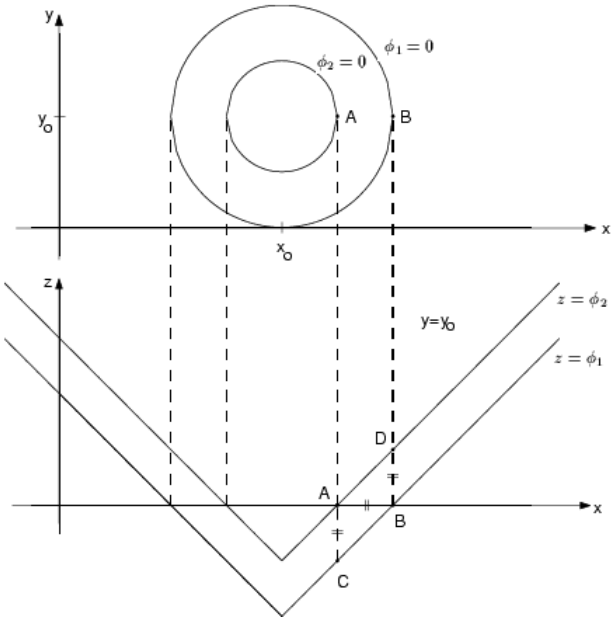

Figure 4.18: Two circles in the xy-plane, and their signed-distance level set functions in the xz-plane at $y = y_0$.

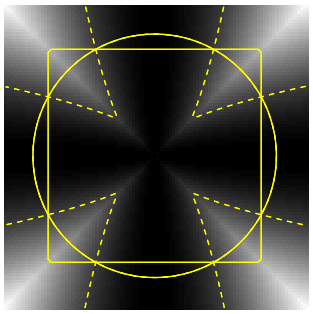

Figure 4.19: Visualization of the shrinking transformation mapping a circle to a square. Lighter areas represent expansion and darker areas represent contraction. Dashed lines represent curves of zero change.

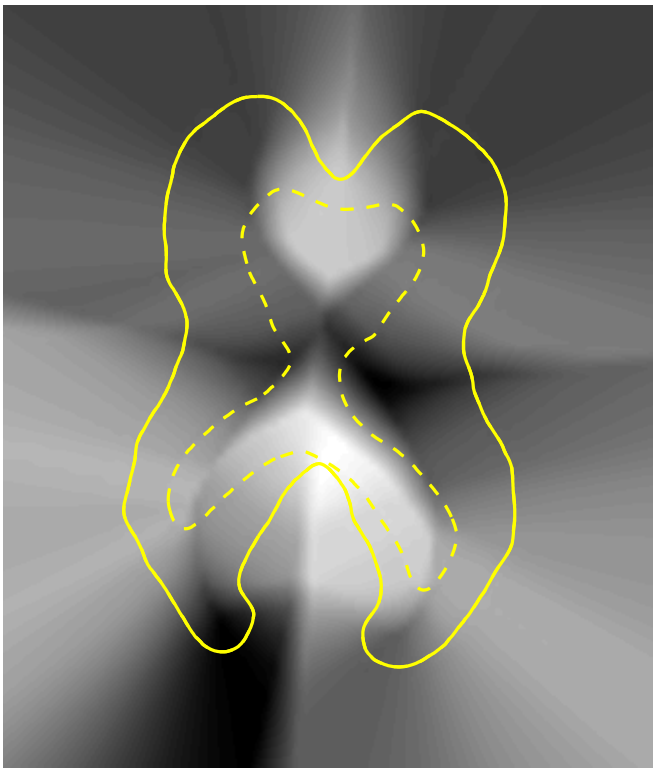

Figure 4.20: Visualization of the shrinking transformation for a real example. Darker areas represent large differences in the level set functions, and consequently areas of significant change.

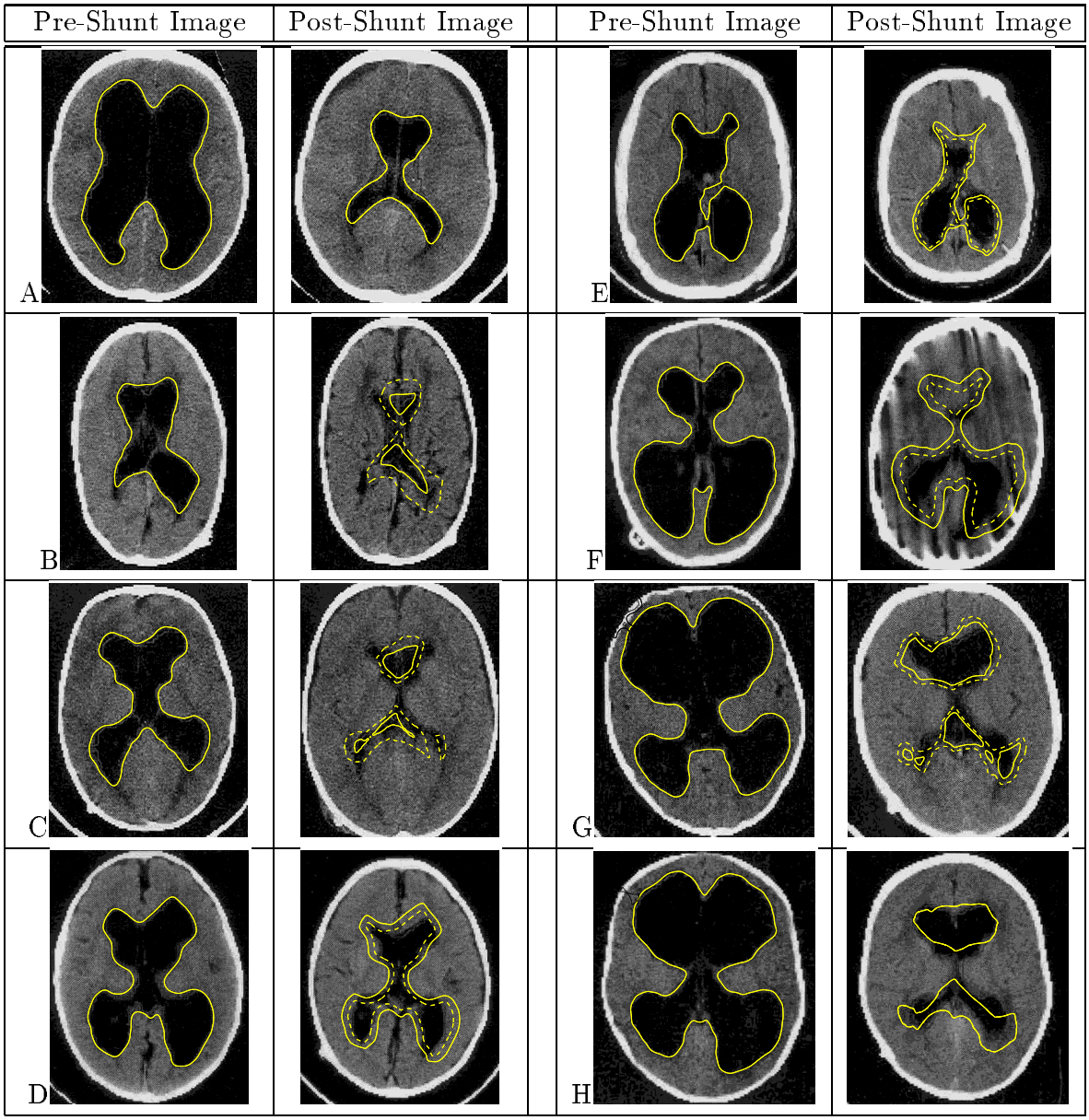

Figure 4.21: Using knowledge of the post-shunt curve for patient A, the shrinking transformation is applied to the other patients. Dashed lines represent a different level set of the solid curve whose area roughly matches that of the desired curve.

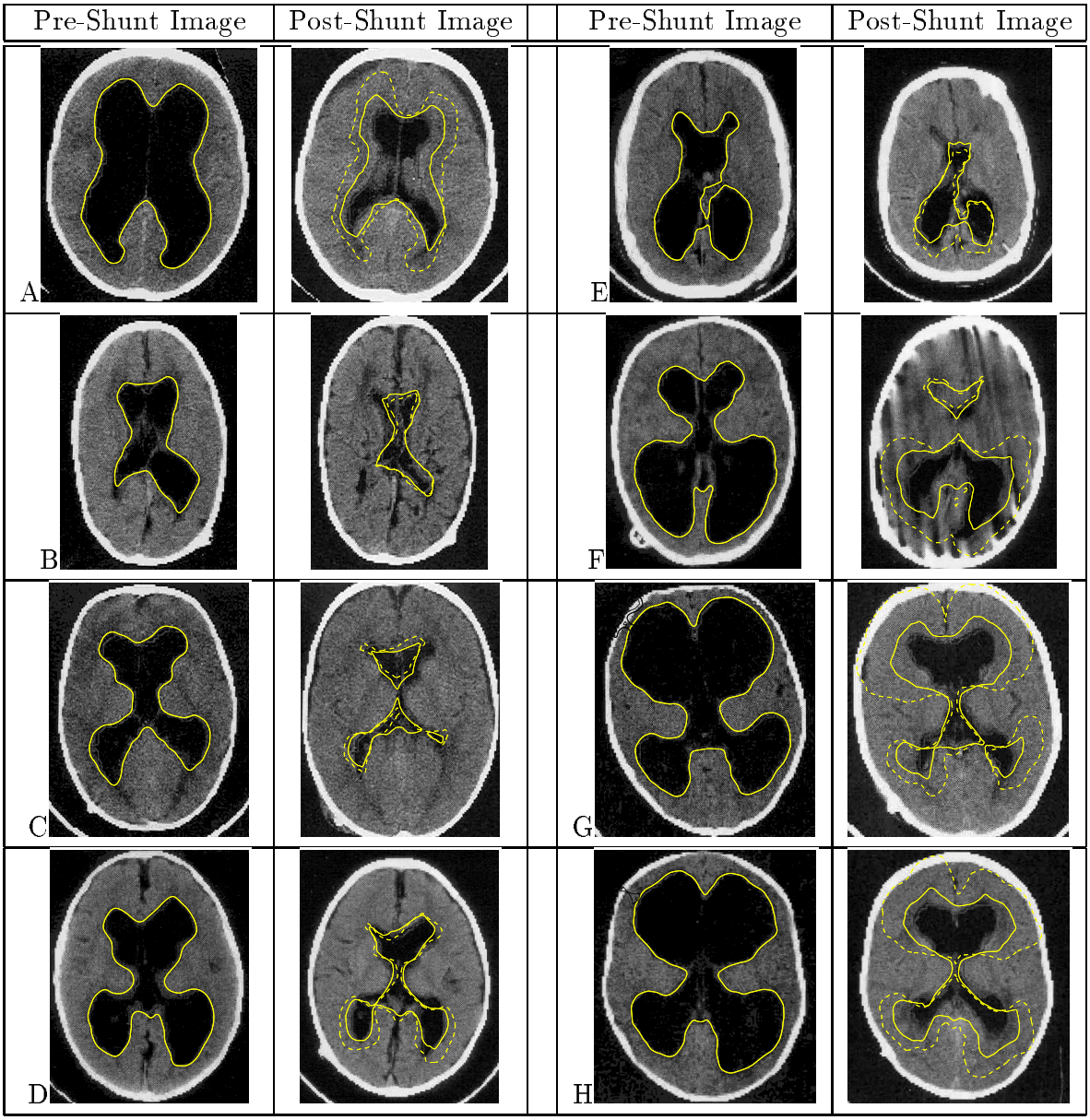

Figure 4.22: Averaged shrinking transformation, using knowledge of post-shunt curves for patient A to D (solid curves); using the converting transformation in an attempt to improve upon these results (dashed curves).

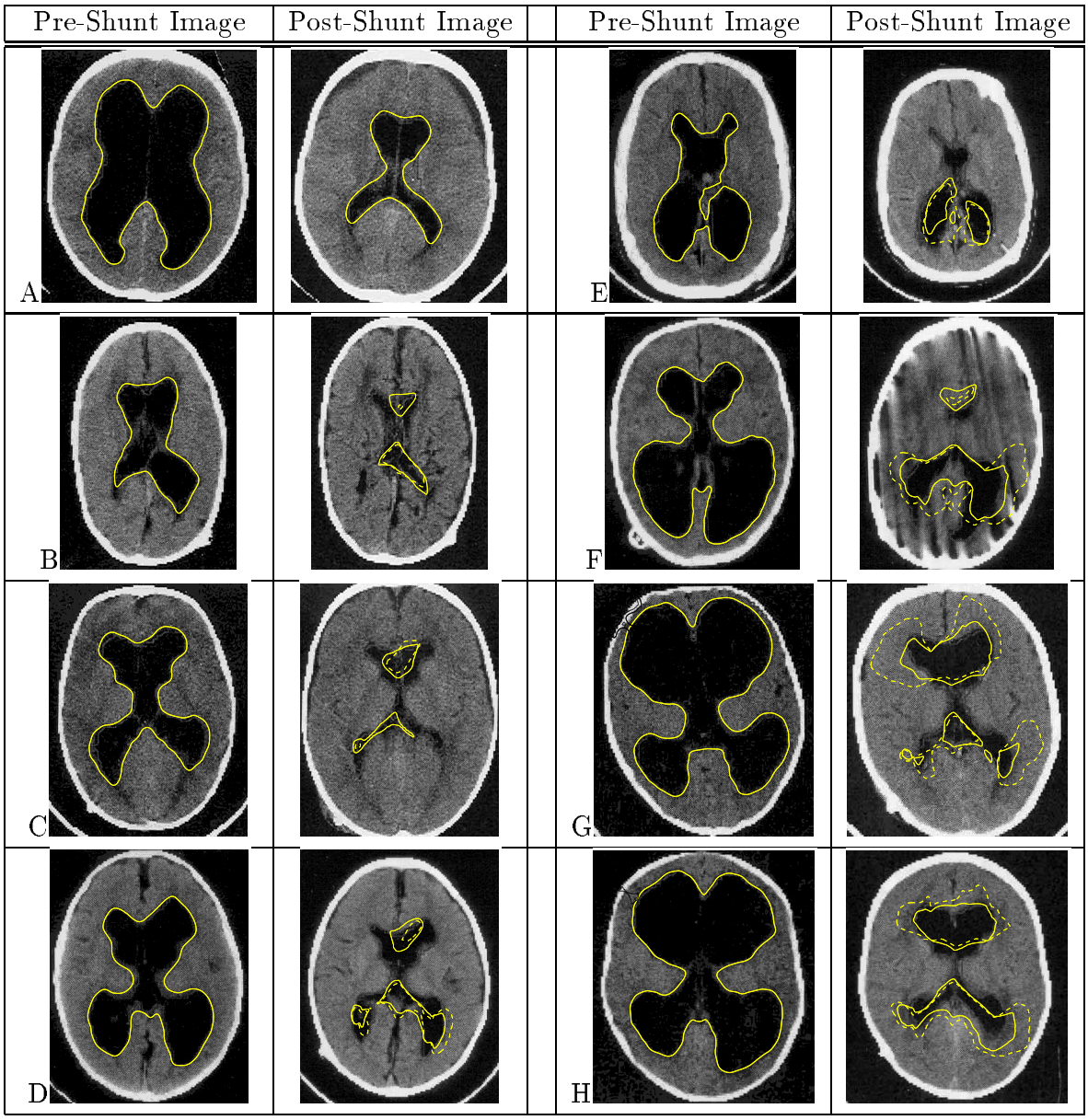

Figure 4.23: Using the converting transformation in an attempt to improve upon the results in figure 4.21.

Chapter 5

Segmentation of the Ventricles

It is tedious and time-consuming to manually trace the ventricles in order to get the initial curve and hence the initial LS function. This is especially true in the 3-dimensional case, where there are usually 30 or more cross-sections to trace. This chapter presents a semi-automatic procedure based on the active contour/surface model in [15] to segment, or extract, the shape of the ventricles.

5.1 Theory of Active Contours/Surfaces

The main idea is as follows. Taking the 2-dimensional case for simplicity, look at the intensity values of the pixels in an MRI (which has been blurred for smoothness). Figure 5.1 shows contours of an MRI with intensities normalized between 0 (black) and 1 (white). Note that the ventricles lie on a contour with an intensity value of about 0.3 in this case. A simple solution is to simply extract this particular contour, while ignoring the other contours of intensity value 0.3, then build an LS function from the points on the contour. Instead of this, we use a method from the literature, referred to as *active contours*, which generalizes easily to 3 dimensions.

The idea is to define a circle which lies within, intersects, or encloses the ventricles, while not intersecting other contours where the image intensity I is the same as the image intensity E at the ventricular walls. See figure 5.2. Then, one can propagate the circle in such a way that points inside the ventricles move outward and points outside of the ventricles move inward, so that the evolving curve converges to the desired boundary.

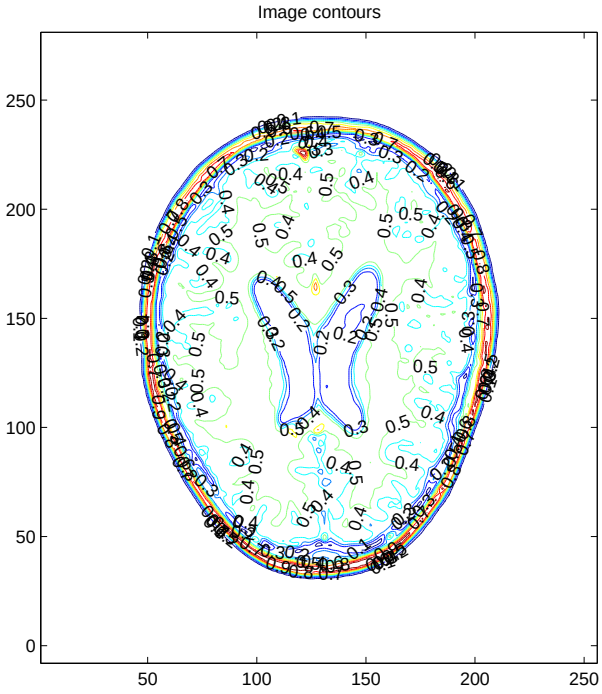

Figure 5.1: Intensity contours of a blurred and normalized MR image.

This can be accomplished using the level set method, with a speed function $F(x, y)$ that depends on the image intensity $I(x, y)$ at each point:

$$F(x, y) = v(x, y) - w\kappa \tag{5.1}$$

where $w > 0$ is a constant, κ is the curvature, and

$$v(x, y) = 1 - \frac{1}{E} I(x, y) \tag{5.2}$$

where $I(x, y)$ is the normalized intensity of the image with $0 \leq I(x, y) \leq 1$, and E is the image intensity at the ventricular walls. Thus, at points inside the ventricles, $I < E$ and so $v > 0$ and the curve expands. At points outside of the ventricles, $I > E$ and so $v < 0$ and the curve shrinks. At points on the ventricular walls, $I = E$ and $v = 0$, so the curve remains stationary. Here it is assumed that the second term in (5.1) is small compared to the first term. The second term is a curvature term, and controls the smoothness of the curve. In [15], it is intended to prevent the active contour/surface from growing into the

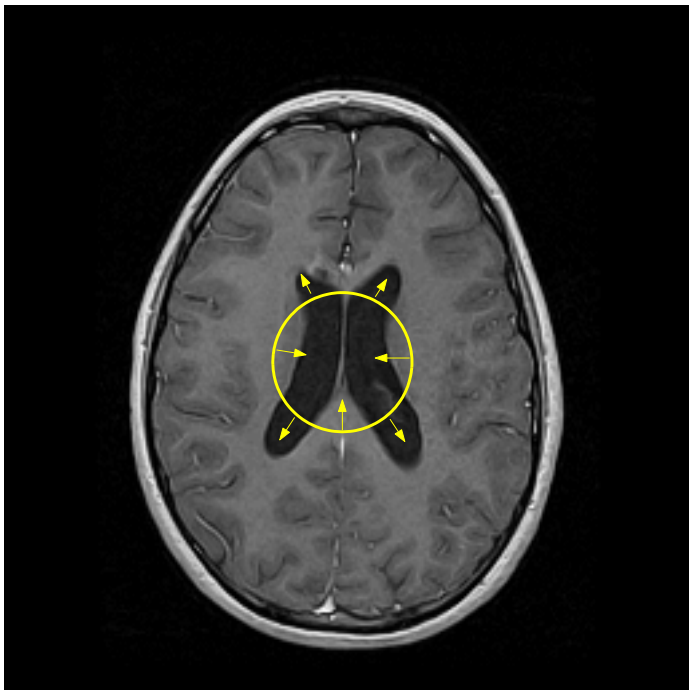

Figure 5.2: Illustration of an active contour method.

outer pial surface near the skull. However, we have never encountered difficulties with this, so $w = 0$ unless otherwise noted.

5.2 Implementation Details

Initialization

Initially, a circle intersecting the ventricles is defined. This can be done by asking the user to click on a point inside and outside of the ventricles. The first point defines the centre of the circle, and the second point defines a point on the circle. These two points fully define the circle. To get the LS function ϕ_{ij}^0 for this circle, one can simply calculate the distance from each grid point to the centre of the circle and then subtract the radius.

If the ventricles are defined by more than one contour (examples to follow), that is, if the ventricular area is not simply connected, then the circle-defining procedure can be repeated for each "piece" of the ventricles, and the LS functions for each circle can be

combined into one LS function simply by taking the minimum value at each grid cell:

$$\phi^0_{ij} = \min\left\{\phi^{0_1}_{ij}, \phi^{0_2}_{ij}, ..., \phi^{0_N}_{ij}\right\}$$

where $\phi^{0_n}_{ij}$ is the LS function for the n^{th} circle.

In three dimensions, either the 3D MRI is treated as a collection of 2D MRI's, each segmented separately then combined together, or the user defines a *sphere* intersecting the ventricles in a 3D MRI, and a analogous 3D version of the program is used. With the help of an isosurface plot of the 3D MRI (figure 5.3), the user can specify a coordinate for the centre of the sphere and the radius. If neccessary, many spheres can be combined together quite easily.

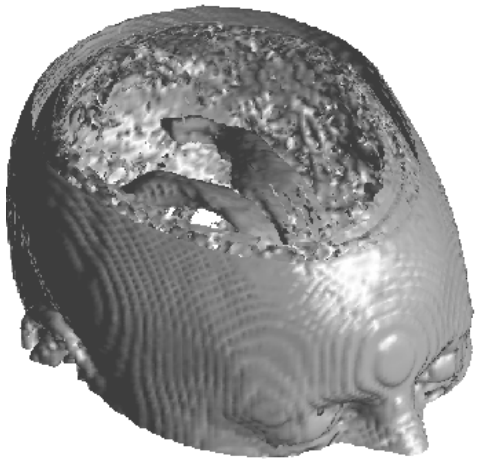

Figure 5.3: Isosurface plot of a 3D MRI, showing the ventricles.

Propagation

The initial LS function is deformed according to the speed function given in (5.1) and (5.2) by numerically solving the same PDE used for front propagation:

$$\phi_t + v\left|\nabla\phi\right| = w\kappa\left|\nabla\phi\right|. \tag{5.3}$$

Again, the term on the right side is approximated by central differences. The homo-

geneous equation

$$\phi_t + v\,|\nabla\phi| = 0$$

is solved numerically using the schemes from chapter 3. Note that the speed function $v(x,y) = 1 - \frac{1}{E}I(x,y)$ is non-convex in general, so the scheme (3.18) with flux (3.19) does not apply. In [30], the non-convex modification is given by

$$\begin{aligned}\phi_{ij}^{n+1} = \phi_{ij}^{n} - \Delta t \Big[& g\left(D_x^-\phi, D_x^+\phi, D_y^-\phi, D_y^+\phi\right) \\ & -\frac{1}{2}\alpha_u\left(D_x^+\phi - D_x^-\phi\right) - \frac{1}{2}\alpha_v\left(D_y^+\phi - D_y^-\phi\right)\Big]\end{aligned}$$

where the numerical flux function g is the same as before, and

$$\alpha_u = \max_{i,j}\left|D_x^0\phi\right|$$

$$\alpha_v = \max_{i,j}\left|D_y^0\phi\right|.$$

However, our experience suggests that the two new terms introduce too much diffusion into the solution. Using the non-convex scheme, there appears to be a strong curvature term present, even though $w = 0$. This makes it difficult for the active contour to converge to areas of high curvature. It turns out that the convex scheme used earlier works well, so that is what will be used.

Re-initialization

As the active contour/surface is propagated, ϕ will not remain a signed distance function. Due to the non-uniformity of the image intensities, and hence of the speed field, the level sets of the image will get closer in some places ($|\nabla\phi| > 1$) and spread out in other places ($|\nabla\phi| < 1$). If $|\nabla\phi|$ becomes too large (for example, greater than $5h$, where h is the mesh spacing), then the calculation can become unstable. If $|\nabla\phi|$ becomes too small (less than $0.1h$, say), instability can result in the curvature term, and also a small error in the value of ϕ can result in a large displacement of the contour $\phi = 0$.

One way to prevent these problems from arising is to monitor $|\nabla\phi|$ and occasionally re-initialize ϕ to a signed distance function. One way to do this is by first extracting the curve $\phi = 0$, then standing at each grid cell and computing the signed distance to the

curve. This is a very computationally expensive procedure, especially in three dimensions.

A more efficient technique for re-initialization is to numerically solve the PDE

$$\phi_t + S(\phi^0, \phi)(|\nabla\phi| - 1) = 0$$

to steady-state, where $|\nabla\phi| = 1$ (in practice, iterate until $\max_{i,j} |\nabla\phi(i\Delta x, j\Delta y)|$ is close to 1, say $||\nabla\phi|_\infty - 1| < \varepsilon$ for some ε). The function $S(\phi^0, \phi)$ is a switch function controlling the direction of propagation of the level sets, and is positive for ϕ (or ϕ^0)> 0 and negative for ϕ (or ϕ^0)< 0. Here, ϕ^0 represents ϕ before re-initialization. Some choices (from [24]) of the switch function are

$$\begin{aligned} S_1(\phi^0, \phi) &= sign(\phi) = \begin{cases} +1, \, if\, \phi > 0 \\ -1, \, if\, \phi < 0 \end{cases} \\ S_2(\phi^0, \phi) &= \frac{\phi^0}{\sqrt{(\phi^0)^2 + h^2}} \\ S_3(\phi^0, \phi) &= \frac{\phi}{\sqrt{\phi^2 + |\nabla\phi|^2 h^2}} \end{aligned}$$

The reasons for each of these choices are given in [24]. While this method of re-initializing ϕ is relatively fast computationally (typically, 20 iterations are needed to satisfy $||\nabla\phi|_\infty - 1| < 0.5$), the front $\phi = 0$ does move slightly, especially near areas of high curvature, and especially for the first switch function above. This is because of the "crudeness" of the switch functions (see [30]). The other two switch functions are more advantageous. However, after some thought and experimentation, we have chosen to use

$$S(\phi, \phi^0) = \min(1, \phi)\, sign(\phi)$$

which tends to zero as $\phi \to 0$, hence the fronts near $\phi = 0$ will not move much. In practice, $\phi = 0$ moves slightly, but this switch function appears to be more successful than the others, at least for the examples we have tried. Also, the convex scheme is used instead of the non-convex scheme for the same reason as before (to eliminate excess diffusion).

This is certainly not the full story on re-initialization. For other re-initialization

methods, and ways to avoid the need for re-initialization, see [30], [24], and [14].

Convergence criterion

The calculation should stop when the entire curve has reached the desired boundary. To quantify the "closeness" of the curve to the boundary, consider the 2D case for simplicity, although the following idea also applies in 3D. One idea is to look at the speed $v(x, y)$ near $\phi = 0$. If $|v(x, y)|$ is small near $\phi = 0$ then the calculation should stop. Since $v_{ij} = v(ih, jh)$ depends on the image intensity I_{ij} and the edge value E, one can use these parameters to estimate convergence.

Let $\Delta = \{(i, j) | \, |\phi_{ij}| < \delta\}$ where $\delta > 0$ is a small tunable parameter. Then Δ represents the set of grid point indices about $\phi = 0$ in a band of width δ. That is, Δ is the set of grid points "close" to $\phi = 0$. Then, define the weighted average of intensity deviations

$$R_\delta^{avg} = \frac{\sum\limits_{(i,j)\in\Delta} \omega_{ij} \, |I_{ij} - E|}{\sum\limits_{(i,j)\in\Delta} \omega_{ij}}$$

where

$$\omega_{ij} = \frac{1}{|\phi_{ij}| + a}$$

are the weights which penalize the distance from $\phi = 0$, so that points closer to $\phi = 0$ are considered to be more important. The parameter a is a positive real number, typically equal to 1 for simplicity.

When the contour $\phi = 0$ is directly on the contour $I(x, y) = E$, then $R_\delta^{avg} = 0$. So, one can choose a small parameter ε, and let the calculation stop at $R_\delta^{avg} < \varepsilon$, at which point convergence is declared. By trial and error, $\delta = 0.5h$ and $\varepsilon = 0.05$ work well most of the time.

Due to the fact that $|\nabla\phi|$ changes and occasional re-initialization is needed, complications can occur where convergence is almost reached when ϕ is re-initialized, and the re-initialization procedure changes ϕ enough so that, by the time convergence is almost reached again, it is time for another re-initialization. This can lead to oscillations where ϕ never quite converges. As of now, the author sees no simple remedy and has chosen to simply impose a time limit on the calculation. This seems acceptable since the user

should visually inspect the segmentation for accuracy anyway, and hence choose either a smaller parameter ε or a larger time limit if the program tends to stop prematurely.

Note that, as long as δ and ε are sufficiently small, the evolving contour should—barring re-initialization complications—converge to the correct contour, and not to some intermediate contour. The reason for this is that $I_{ij} < E$ everywhere in the region inside of the desired contour, and $I_{ij} > E$ almost everywhere in the region outside of the desired contour and inside of the initial circle (outside of the initial circle this can be violated in some places such as around the skull; this is the reason for the requirement that the initial circle does not intersect the pial surface). In contrast, older active contour methods use the image gradient $|\nabla I|$ as a stopping criterion, so that the curve stops where a strong image gradient exists. These strong gradients usually, but not always, correspond to the desired edges in the image. For example, the ventricular walls sometimes appear "soft" and have a weak image gradient in some areas. In short, the advantage of the approach we have taken is that the speed of the active contour depends directly on the image intensities and not on the image gradients.

This being said, the method presented in this chapter can be improved upon. See [15] and [2], for example. In [15], a fully automatic procedure is presented (the user need not determine the edge value E or click inside and outside of the ventricles). In [2], the active contour automatically detects interior contours, such as a small amount of brain matter in the middle of the ventricles, which our method does not detect unless the initial circle intersects that interior patch. Nonetheless, the model used here suffices for now, and saves many hours of manual segmentation for graduate students, neurosurgeons, and radiologists alike.

Algorithm

To summarize the steps in evolving the active contour/surface to convergence, the following algorithm is used:

```
prepare image (blur, normalize intensities, scale size if desired)
show contours and get edge value E, if not known
ask user to specify initial circle(s), and initialize the corresponding LS function φ^0
repeat until R_δ^avg < ε or time limit reached
  update φ according to (3.18) or (3.20) with speed function given in (5.2) or its 3D equivalent
```

```
    if |∇φ| < 0.1h or |∇φ| > 5h anywhere in the domain, then
      re-initialize φ
    end if
  end repeat
```

For more details, see the Matlab code in the appendix.

5.3 Results

Two dimensional examples, showing the evolution of the active contour, are shown in figures 5.4 and 5.5. A three dimensional example is shown in figure 5.6. Visual comparisons between manually traced ventricles, slice-by-slice segmented ventricles (in which each slice is segmented using the 2D algorithm) and fully 3D segmented ventricles are made in figure 5.7.

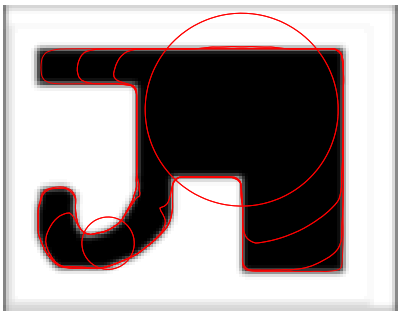

Figure 5.4: Benchmark example, showing the active contour converging to the desired boundary.

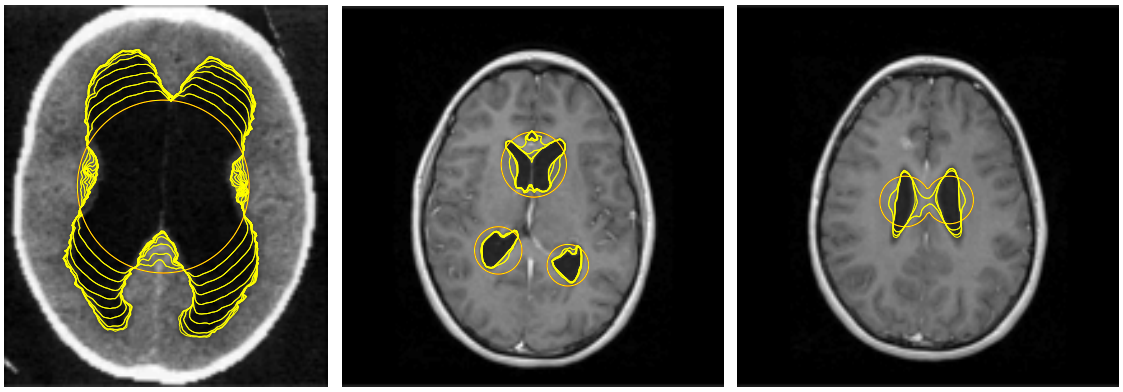

Figure 5.5: Some real examples of two dimensional segmentation, showing the active contour as it approaches the desired boundary.

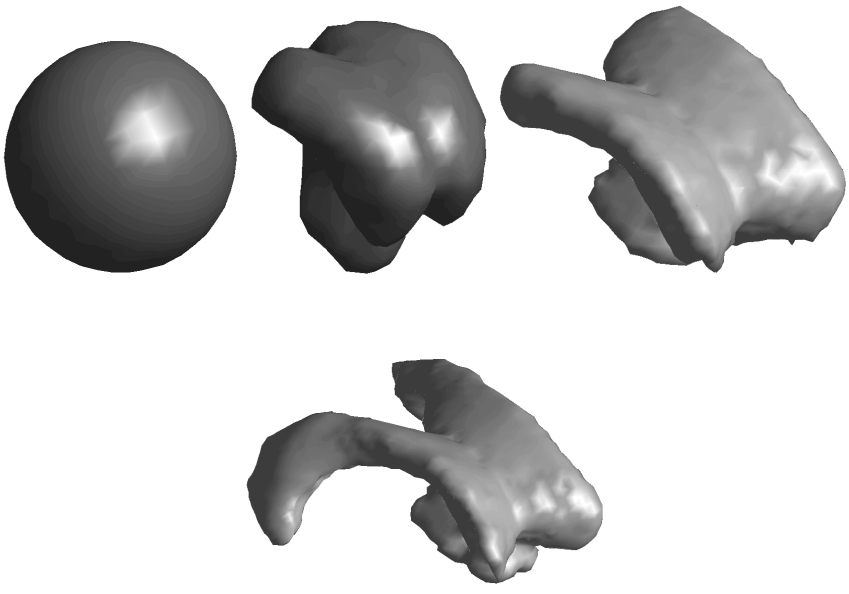

Figure 5.6: Three dimensional segmentation as the calculation progresses.

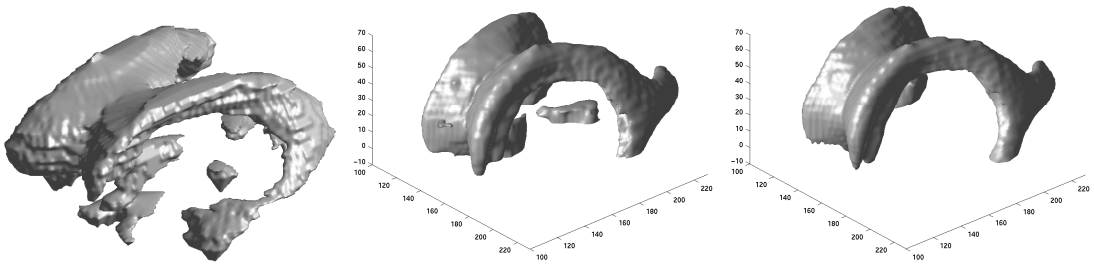

Figure 5.7: Slice-wise manual (left) and semi-automatic (right) segmentation as compared to the fully 3D semi-automatic segmentation (bottom).

Chapter 6

Summary, Discussion and Conclusions

Summary

In this work, we have applied the Level Set Method (LSM) to propagate curves and surfaces representing the ventricles in two and three dimensions respectively, with the goal of predicting the geometry of the ventricles some time after shunt treatment.

In chapter 1, some background and motivation was provided, followed by a brief discussion of curve propagation in chapter 2. Chapter 3 discussed the theory and numerical methods of the LSM, and chapter 4 presented the algorithms used and the results obtained. A new transformation method was also presented, motivated by the need for a realistic, non-constant speed. Chapter 5 dealt with the semi-automatic segmentation of the ventricles from 2D and 3D brain images.

Discussion and Conclusions

In this work, a very simple model for curve evolution was used, as well an a new extrapolation method for predicting the final shape, in the framework of known level set methods. The constant speed case $F = -1$ predicts the final shape of the contours amazingly well given its simplicity (recall figure 4.8). However, the eight data sets shown are the only time-series data currently available to us, so quantitative measures of success and statistically valid conclusions will have to wait until more data becomes available.

The numerical issue of accuracy has not been rigorously dealt with here, as the literature does a good job of this, and it is not neccessary until the final stages of implemen-

tation. At this point, it is more important to focus on making the model more realistic, than it is to polish an oversimplified model.

Nonetheless, this work at least gives us a feel for what we are ultimately working towards: a computer simulation which predicts the final shape of the ventricles after shunt treatment. Of course, many important factors have been ignored, such as the mechanical properties of the brain, the intra-cranial and extra-cranial pressure, the location of the shunt, the growth of the patient, and the exact location and orientation of 2D scans. But the journey of a thousand miles starts with a single step.

Future Work

Future work could include further development of the transformation method presented in section 4.3.2, as well as the empirical determination of the speed field F from the preceding section. One could even use active contours (chapter 5) to propagate the initial curve to the final curve, generating time-series data in the process. Furthermore, when the data becomes available, there is no reason why one could not apply the transformation in three dimensions. Also, when more data becomes available, the methods used in this work could be tested more thoroughly, and statistically valid conclusions could be sought.

Another idea for future work includes coupling the LSM with realistic governing equations for brain matter, such as the viscoelastic constitutive equations. In [32], the LSM is combined with the Navier-Stokes equations to simulate the motion of an air bubble in water, where the level curve $\phi = 0$ gives the interface over which the density and viscosity are discontinuous. Perhaps one could numerically solve the viscoelastic constitutive equations that describe the brain outside of the ventricles and use the Navier-Stokes equations to describe the cerebrospinal fluid (CSF) inside of the ventricles. Since it is unclear how to couple two different sets of governing equations, one could also treat the CSF as a viscoelastic material, where certain parameters are chosen near the fluid limit. (In one limit, a viscoelastic material is an elastic solid, in the other limit, it is a fluid). Then perhaps the problem could be solved in a similar way as the air bubble problem, which has the same governing equations inside and outside of the bubble.

The above ideas may be useful in studying other phenomenon involving the motion of a boundary, such as tumor growth.

Finally, it is important to note that the LSM is not neccessarily the best technique

available. In fact, our research group is currently exploring the idea of using the Finite Element Method to simulate the motion of the ventricles. The author has admittedly become very attracted to the robustness and simplicity of the LSM, but in the future plans to heed the wise advice given at the end of [30]:

> "The reader is reminded that, given only a hammer, everything starts to look like a nail. As a general rule, a good, varied, and somewhat dispassionate tool chest is invaluable."

Appendix A

Matlab Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 2D Level Set Algorithm

function ans = LevelSet2d(NC, Psi, h, T, dt, Fo, epsilon);
%  ans = output (Psi at final time)
%  NC = 1 (for non-convex scheme) or 0 (for convex)
%  h = [hx hy] = mesh spacing
%  dt = timestep
%  Fo = constant speed term
%  epsilon = curvature parameter

NumContours = 5;

% Plot initial contour
figure(1); clf; hold on;
contourplot(Psi,1,'k',1.5);
axis equal;
drawnow;
hold on;

% Start propagating

for t = T(1)+dt:dt:T(2)
   display(['time = ' num2str(t)]);

   Psi = UpdatePsi2d(NC,Psi,h,dt,Fo,epsilon);

   if mod(round(t/dt), ceil(diff(T)/dt/NumContours)) == 0
      contourplot(Psi,1,'k',1.5);
      axis equal; %axis([N size(Psi,1)-N  N size(Psi,2)-N]);
      drawnow;
   end;
   % Check if a re-initialization is needed
   [Dx Dy] = gradient(Psi);
   MaxGrad = sqrt(max(max(Dx.^2+Dy.^2)));
   disp(MaxGrad);
   if MaxGrad > 5
      Psi = ReInit2d(Psi,h);
   end;

end;

ans = Psi;

% UpdatePsi2d.m -- the 2d version, based on the 1st-order space
% convex or non-convex scheme.
% This scheme gives the solution to
%  dPsi/dt + Fo|grad(Psi)| = epsilon*kappa*|grad(Psi)|
% over one timestep.
% see Sethian's book, pages 60-74.
% Implemented by Joe West, Jan-Mar 2003.


function PsiNew = UpdatePsi2d(NC,Psi,h,dt,Fo,epsilon);

% NC = 0 for convex scheme, 1 for non-convex scheme
% Psi = Level Set function at time t
% PsiNew = Level Set function at time t+dt
% h = [dx dy] mesh spacing
% dt = timestep (0 for automatic timestepping)
% Fo = velocity field ( should have size = size(Psi)-[2,2] )
% epsilon = curvature weight

if dt<0 display('Warning:  Negative timestep');end;

p = size(Psi,1);
q = size(Psi,2);
if length(h)==1
   dx = h; dy = h;
else
   dx = h(1);  dy = h(2);
end;

O = zeros(p-2,q-2);

% Derivatives:

i = [2:p-1];
j = [2:q-1];

Dxm = 1/dx * (  Psi(i,j) - Psi(i-1,j)  );
Dxp = 1/dx * (  Psi(i+1,j)-Psi(i,j)    );
Dx  = 1/(2*dx) * ( Psi(i+1,j)-Psi(i-1,j) );
Dym = 1/dx * (  Psi(i,j) - Psi(i,j-1)  );
Dyp = 1/dx * (  Psi(i,j+1) - Psi(i,j)    );
Dy  = 1/(2*dx) * ( Psi(i,j+1)-Psi(i,j-1) );
```

```
kappa = 0;
if epsilon ~= 0 % need second derivatives for curvature
 Dxx = 1/(dx^2)* (  Psi(i+1,j)-2*Psi(i,j)+Psi(i-1,j)  );
 Dyy = 1/(dx^2)* (  Psi(i,j+1)-2*Psi(i,j)+Psi(i,j-1)  );
 Dxy = 1/(4*dx*dy)* ( Psi(i+1,j+1)-Psi(i+1,j-1)...
       -Psi(i-1,j+1)+Psi(i-1,j-1)  );

 Numerator = Dxx.*Dy.^2 - 2*Dy.*Dx.*Dxy + Dyy.*Dx.^2;
 Denominator = (Dx.^2+Dy.^2).^(3/2);

 INDX = find(Denominator); % where Denominator is not zero

 kappa(INDX) = Numerator(INDX)./Denominator(INDX);

 % set all undefined (c/0) values equal to zero:
 kappa(~INDX) = 0;

end;

GradPlus = sqrt( max(Dxm,0).^2 + min(Dxp,0).^2 + ...
 max(Dym,0).^2 + min(Dyp,0).^2 );
GradMinus = sqrt( max(Dxp,0).^2 + min(Dxm,0).^2 + ...
  max(Dyp,0).^2 + min(Dym,0).^2 );

% Make sure Fo is right size
if size(Fo) ~= [p-2,q-2]
 if size(Fo) == [1,1]
   Fo = Fo*ones(p-2,q-2);
 elseif size(Fo) == size(Psi)
   Fo = Fo(2:p-1,2:q-1);
 else
   disp('Error:  Fo is not the right size');
 end;
end;


% Check CFL condition (and check for automatic timestepping)
% If CFL condition violated or automatic timestepping,
% recursively call this function with 'small enough' timesteps.

hm = min(h);
a = 0.5;
if max(max(abs(dt*(Fo-epsilon*kappa)))) > a*hm  | dt == 0

   dtNew = a*hm/max(max(abs(Fo-epsilon*kappa)));

   if dt ~= 0

     % adjust dtNew so it does not get too small:
     dtNew = max(dtNew, 0.01*hm);

     % recursively update remaining part of timestep
     if dt-dtNew > 0
        Psi = UpdatePsi2d(NC, Psi,hm, dt - dtNew, Fo, epsilon);
     else
       % if new timestep is larger than original timestep, then
       % forget this correction.
       dtNew = dt;
     end;

   end;

   % use this dt value to update below
   dt = dtNew;

end;


% Approximation to Hamiltonian
Ham = GradPlus.*max(Fo,0) + GradMinus.*min(Fo,0);
%Ham2 = Fo*(Dx.^2+Dy.^2).^(1/2);

if NC == 0  % Hamiltonian is convex
   alphaU = 0;
   alphaV = 0;
else % Hamiltonian is not convex
   % Bounds on Hamiltonian (see p.69 Sethian's text)
   [Hx Hy] = gradient(Ham,dx,dy);
   alphaU = max(max(abs(Hx)));
   alphaV = max(max(abs(Hy)));
end;

% update all of Psi except for elements on the edge of array
PsiNew(i,j) = Psi(i,j) - dt*( Ham  ...
- 1/2*alphaU.*(Dxp-Dxm)  - 1/2*alphaV.*(Dyp-Dym)...
- epsilon*kappa.*(Dx.^2+Dy.^2).^(1/2) );

PsiNew(1,:)=PsiNew(2,:);
PsiNew(p,:)=PsiNew(p-1,:);
PsiNew(:,1)=PsiNew(:,2); %    <-- ghost-cell boundary conditions
PsiNew(:,q)=PsiNew(:,q-1);      %        (see [Sethian], p. ???)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  3D level set algorithm

function ans = LevelSet3d(NC, Psi, h, T, dt, Fo, epsilon);
% ans = final Psi
% NC = non-convex scheme (1) or convex (0)
% h = mesh spacing
% T = [t1 t2] = time interval
% dt = timestep
% Fo = constant speed term
% epsilon = curvature parameter

figure(1); clf;
isosurface(Psi,0);
N = 0;
axis equal; axis([N size(Psi,1)-N  N size(Psi,2)-N  N size(Psi,3)-N]);
%view([1 0 1]);
camlight;  drawnow;

NumContours = 9;

% Start propagating

for t = T(1)+dt:dt:T(2)
   display(['time = ' num2str(t)]);
   Psi = UpdatePsi3d(NC,Psi,h,dt,Fo,epsilon);

  if mod(round(t/dt), ceil(diff(T)/dt/NumContours)) == 0

   clf;
   isosurface(Psi,0);
   axis equal; axis([N size(Psi,3)-N  N size(Psi,1)-N  N size(Psi,2)-N]);
   %view([1 0 1]);
   axis off;
   camlight;  drawnow;

   [Dx Dy Dz] = gradient(Psi(3:end-2,3:end-2,3:end-2));
   AbsGrad = sqrt(max(max(max(Dx.^2+Dy.^2+Dz.^2))));
   disp(AbsGrad);
```

```
   if AbsGrad > 5       % Re-initialize psi
      Psi = ReInit3d(Psi, h);
   end;

  end; %if

end;

ans = Psi;

% UpdatePsi3d.m -- the 3d version, based on the 1st-order space
% convex or non-convex scheme.
% This scheme is the solution to
%  dPsi/dt + Fo|grad(Psi)| = epsilon*kappa*|grad(Psi)|
% over one timestep.
% see Sethian's book, pages 60-74.
% Implemented by Joe West, Jan-Har 2003.


function PsiNew = UpdatePsi3d(NC,Psi,h,dt,Fo,epsilon);

% NC = 0 for convex scheme, 1 for non-convex scheme
% Psi = Level Set function at time t
% PsiNew = Level Set function at time t+dt
% h = mesh spacing
% dt = timestep (0 for automatic timestepping)
% Fo = velocity field ( should have size = size(Psi)-[2,2,2] )
% epsilon = curvature weight

if dt<0 display('Warning:  Negative timestep');end;

p = size(Psi,1);
q = size(Psi,2);
r = size(Psi,3);
if length(h)==1
   dx = h; dy = h; dz = h;
else
   dx = h(1); dy = h(2); dz = h(3);
end;

0 = zeros(p-2,q-2,r-2);

% Derivatives:

i = [2:p-1];
j = [2:q-1];
k = [2:r-1];

Dxm = 1/dx * (  Psi(i,j,k) - Psi(i-1,j,k)  );
Dxp = 1/dx * (  Psi(i+1,j,k)-Psi(i,j,k)    );
Dx  = 1/(2*dx) * ( Psi(i+1,j,k)-Psi(i-1,j,k) );
Dym = 1/dy * (  Psi(i,j,k) - Psi(i,j-1,k)  );
Dyp = 1/dy * (  Psi(i,j+1,k) - Psi(i,j,k)    );
Dy  = 1/(2*dy) * ( Psi(i,j+1,k)-Psi(i,j-1,k) );
Dzm = 1/dz * (  Psi(i,j,k) - Psi(i,j,k-1)  );
Dzp = 1/dz * (  Psi(i,j,k+1) - Psi(i,j,k)    );
Dz  = 1/(2*dz) * ( Psi(i,j,k+1)-Psi(i,j,k-1) );

kappa = 0;
if epsilon ~= 0 % need second derivatives for curvature
 Dxx = 1/(dx^2)* (  Psi(i+1,j,k)-2*Psi(i,j,k)+Psi(i-1,j,k)  );
 Dyy = 1/(dy^2)* (  Psi(i,j+1,k)-2*Psi(i,j,k)+Psi(i,j-1,k)  );
 Dzz = 1/(dz^2)* (  Psi(i,j,k+1)-2*Psi(i,j,k)+Psi(i,j,k-1)  );
 Dxy = 1/(4*dx*dy)* ( Psi(i+1,j+1,k)-Psi(i+1,j-1,k)...
    -Psi(i-1,j+1,k)+Psi(i-1,j-1,k)  );
 Dxz = 1/(4*dx*dz)* ( Psi(i+1,j,k+1)-Psi(i+1,j,k-1)...
    -Psi(i-1,j,k+1)+Psi(i-1,j,k-1)  );
 Dyz = 1/(4*dy*dz)* ( Psi(i,j+1,k+1)-Psi(i,j+1,k-1)...
    -Psi(i,j-1,k+1)+Psi(i,j-1,k-1)  );

 % Use mean curvature:

 Numerator = (Dyy+Dzz).*Dx.^2 + (Dxx+Dzz).*Dy.^2 + (Dxx+Dyy).*Dz.^2 ...
             - 2*Dx.*Dy.*Dxy  - 2*Dx.*Dz.*Dxz  - 2*Dy.*Dz.*Dyz ;
 Denominator = ( Dx.^2 + Dy.^2 + Dz.^2 ).^(3/2);
 INDX = find(Denominator); % where Denominator ~=0

 kappa(INDX) = Numerator(INDX)./Denominator(INDX);

 % set all undefined (c/0) values equal to zero:
 kappa(~INDX) = 0;

end;

GradPlus = sqrt( max(Dxm,0).^2 + min(Dxp,0).^2 + ...
 max(Dym,0).^2 + min(Dyp,0).^2 + ...
 max(Dzm,0).^2 + min(Dzp,0).^2 );
GradHinus = sqrt( max(Dxp,0).^2 + min(Dxm,0).^2 + ...
  max(Dyp,0).^2 + min(Dym,0).^2 + ...
      max(Dzp,0).^2 + min(Dzm,0).^2 );

% Make sure Fo is right size
if size(Fo,1) == 1
  Fo = Fo*ones(p-2,q-2,r-2);
end;
if size(Fo) ~= [p-2,q-2,r-2]
 if size(Fo) == size(Psi)
   Fo = Fo(2:p-1,2:q-1,2:r-1);
 else
   disp('Error:  Fo is not the right size');
 end;
end;


% Check CFL condition (and check for automatic timestepping)
% If CFL condition violated or automatic timestepping,
% recursively call this function with 'small enough' timesteps.

hm = min(h);
a = 0.5;
if max(max(max(abs(dt*(Fo+epsilon*kappa))))) > a*hm  | dt == 0

   dtNew = a*hm/max(max(max(abs(Fo-epsilon*kappa))));

   if dt ~=0

     % adjust dtNew so it does not get too small:
     dtNew = max(dtNew, 0.01*hm);

     % recursively update remaining part of timestep
     if dt-dtNew > 0
       Psi = UpdatePsi3d(NC, Psi,hm, dt - dtNew, Fo, epsilon);
     else
       % if new timestep is larger than original timestep, then
       % forget this correction.
       dtNew = dt;
     end;


   end;
```

```
   % use this dt value to update below
   dt = dtNew;

end;


% Hamiltonian
Ham = GradPlus.*max(Fo,0) + GradHinus.*min(Fo,0);


if NC == 0  % Hamiltonian is convex
   alphaU = 0;
   alphaV = 0;
   alphaW = 0;
else % Hamiltonian is not convex
   % Bounds on Hamiltonian (see p.69 Sethian's text)
   [Hx Hy Hz] = gradient(Ham,dx,dy,dz);
   alphaU = max(max(max(abs(Hx))));
   alphaV = max(max(max(abs(Hy))));
   alphaW = max(max(max(abs(Hz))));
end;

% update all of Psi except for elements on the edge of array
PsiNew(i,j,k) = Psi(i,j,k) - dt*( Ham  - 1/2*alphaU.*(Dxp-Dxm) ...
- 1/2*alphaV.*(Dyp-Dym) - 1/2*alphaW.*(Dzp-Dzm) ...
- epsilon*kappa.*(Dx.^2+Dy.^2).^(1/2) );

PsiNew(1,:,:)=PsiNew(2,:,:);
PsiNew(p,:,:)=PsiNew(p-1,:,:);
PsiNew(:,1,:)=PsiNew(:,2,:); %    <-- "ghost cell" boundary cond.
PsiNew(:,q,:)=PsiNew(:,q-1,:);      %        (see [Sethian] p. ???)
PsiNew(:,:,1)=PsiNew(:,:,2);
PsiNew(:,:,r)=PsiNew(:,:,r-1);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ReInit2d.m -- the 2d version, based on the 1st-order space non-convex
% scheme.  Re-initializes Psi so that |grad(Psi)| = 1 by solving
%            dPsi/dt = min(1,Psi)sign(Psi)(1-|grad(Psi)|)
% to steady-state.  (See Sethian's book, page 138-139)
% By Joe West, Jan 2003.  Updated/fixed Mar 2003.

function PsiNew = ReInit2d(Psi,h);

Verbose = 0;
epsilon = 0.5;

% Automatically choose timestep
hm = min(h);
dt = 0.5*hm;

PsiNew = Psi;
Psi_0 = Psi;

p = size(Psi,1);
q = size(Psi,2);
if length(h)==1
   dx = h; dy = h;
else
   dx = h(1);  dy = h(2);
end;

0 = zeros(p-2,q-2);

% Derivatives:
i = [2:p-1];
j = [2:q-1];


% Plot 'before' picture:
if Verbose
 figure(4);  clf;
 contour(Psi,[0 0],'g');
 title('Re-initialization (green=before, red=after)');
 drawnow; hold on;
end;

% Repeat until abs(|grad(Psi)|-1) < epsilon for all x and y, but exit if
% it blows up (say > 50)
[PsiX PsiY] = gradient(Psi(3:end-2,3:end-2),dx,dy);
HagGradPsi = (PsiX.^2+PsiY.^2).^(1/2);
HaxGrad = max(max(HagGradPsi));
HinGrad = min(min(HagGradPsi));
%Dev = max(max(abs(HagGradPsi-1)));
Dev = abs(HaxGrad-1);

while Dev>=epsilon & HaxGrad < 50

if Verbose; disp([HinGrad HaxGrad Dev]); end;

Dxm = 1/dx * (  Psi(i,j) - Psi(i-1,j)  );
Dxp = 1/dx * (  Psi(i+1,j)-Psi(i,j)    );
Dx  = 1/(2*dx) * ( Psi(i+1,j)-Psi(i-1,j) );
Dym = 1/dy * (  Psi(i,j) - Psi(i,j-1)  );
Dyp = 1/dy * (  Psi(i,j+1) - Psi(i,j)    );
Dy  = 1/(2*dy) * ( Psi(i,j+1)-Psi(i,j-1) );


GradPlus = sqrt( max(Dxm,0).^2 + min(Dxp,0).^2 + ...
 max(Dym,0).^2 + min(Dyp,0).^2 );
GradHinus = sqrt( max(Dxp,0).^2 + min(Dxm,0).^2 + ...
  max(Dyp,0).^2 + min(Dym,0).^2 );


Fo = min( ones(size(Psi)), abs(Psi) ).*sign(Psi);  Fo = Fo(i,j);
%Fo = sign(Psi_0);  Fo = Fo(i,j);


% Bounds on Hamiltonian (see p.69 Sethian's text)
Ham = GradPlus.*max(Fo,0) + GradHinus.*min(Fo,0);
[Hx Hy] = gradient(Ham,h);
alphaU = 0; %max(max(abs(Hx)));
alphaV = 0; %max(max(abs(Hy)));
% (It turns out, in practice, that the convex scheme works, and with less
% diffusion and more stability, so these are set to zero)

% update all of Psi except for elements on the edge of array
PsiNew = Psi;

PsiNew(i,j) = Psi(i,j) ...
         - dt*( Ham  ...
- 1/2*alphaU*(Dxp-Dxm) - 1/2*alphaV*(Dyp-Dym) ...
- Fo );

PsiNew(1,:)=PsiNew(2,:);
PsiNew(p,:)=PsiNew(p-1,:);
PsiNew(:,1)=PsiNew(:,2); %    <-- "ghost cell" boundary cond.
PsiNew(:,q)=PsiNew(:,q-1);

Psi = PsiNew;
```

```
[PsiX PsiY] = gradient(Psi(3:end-2,3:end-2),dx,dy);
HagGradPsi = (PsiX.^2+PsiY.^2).^(1/2);
HaxGrad = max(max(HagGradPsi));
HinGrad = min(min(HagGradPsi));
Dev = abs(HaxGrad-1);

end;

% Plot 'after' picture:
if Verbose
 figure(4);
 contour(Psi,[0 0],'r');  drawnow;
end;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ReInit2dSlow.m:  Re-initializes psi the expensive-yet-accurate way.

function ans = ReInit2dSlow(psi,h);
% psi = 2d level set array
% h = mesh spacing

figure(99); drawnow;

ans = Inf*ones(size(psi));
[x s] = contour(psi,[0 0]);  x = x';
I = find(x(:,1)==0);
I(end+1) = length(x(:,1))+1;
for i = 1:length(I)-1
    X = x(I(i)+1:I(i+1)-1,1);
    Y = x(I(i)+1:I(i+1)-1,2);
    ans = min(ans,InitPsiFull(X,Y,size(psi,1),size(psi,2),h));
end;




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% InitPsiFull.m  -- initialize Psi on entire image area, preserving coordinate locations
%
% Initalizes Psi according to Sethian:
% Psi(j,k) = plus (if outside) or minus (if inside) euclidean distance
%            between the point (jh,kh) and the curve.

function Psi = InitPsiFull(y, x, jmax, kmax, h);

if length(h)>1
  hx = h(1); hy = h(2);
else
  hx = h; hy = h;
end;

% assumes coordinates x and y are in first quadrant

Psi = zeros(jmax, kmax);

% for each row of points, we determine which points are inside, which are
% outside, and set Psi accordingly
for k = 1:kmax
 X=[1:jmax];
 Y=k*ones(1,length(X));
 in=inpolygon(X,Y,x/hx,y/hy);

 Xin=X(in)*hx;
 Yin=Y(in)*hy;
 Xout=X(~in)*hx;
 Yout=Y(~in)*hy;

 for q = 1:length(Xin)
   Psi(round(Xin(q)/hx),round(Yin(q)/hy))...
  = -(min((x-Xin(q)).^2+(y-Yin(q)).^2))^(1/2);
 end;

 for q = 1:length(Xout)
   Psi(round(Xout(q)/hx),round(Yout(q)/hy)) ...
                 =  (min((x-Xout(q)).^2+(y-Yout(q)).^2))^(1/2);
 end;

 %disp(['Percentage done: ' int2str(round(k/kmax*100))]);
end;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ReInit3d.m -- the 3d version, based on the 1st-order space non-convex
% scheme.  Re-initializes Psi so that |grad(Psi)| = 1 by solving
%            dPsi/dt = min(1,Psi)sign(Psi)(1-|grad(Psi)|)
% to steady-state.  (See Sethian's book, page 138-139)
% By Joe West, Jan 2003.  Updated/fixed Mar 2003.

function PsiNew = ReInit3d(Psi,h);

hm = min(h);
dt = 0.5*hm;

PsiNew = Psi;

p = size(Psi,1);
q = size(Psi,2);
r = size(Psi,3);
if length(h)==1
   dx = h; dy = h; dz = h;
else
   dx = h(1); dy = h(2); dz = h(3);
end;

O = zeros(p-2,q-2,r-2);

% Derivatives:

i = [2:p-1];
j = [2:q-1];
k = [2:r-1];

% Plot 'before' picture:
figure(4);  clf;
isosurface(Psi,0);  title('before');
drawnow; hold on;


% Repeat until |grad(Psi)| < 2
% (ideally should equal 1, but anything < 1.5 is good enough)
[PsiX PsiY PsiZ] = gradient(Psi(3:end-2,3:end-2,3:end-2),dx,dy,dz);
HagGradPsi = (PsiX.^2+PsiY.^2+PsiZ.^2).^(1/2);

while abs( max(max(max(HagGradPsi))) - 1) > 1

%disp(max(max(max(abs(HagGradPsi)))));

Dxm = 1/dx * (  Psi(i,j,k) - Psi(i-1,j,k)  );
Dxp = 1/dx * (  Psi(i+1,j,k)-Psi(i,j,k)    );
```

```
Dx  = 1/(2*dx) * ( Psi(i+1,j,k)-Psi(i-1,j,k) );
Dym = 1/dy * (  Psi(i,j,k) - Psi(i,j-1,k)  );
Dyp = 1/dy * (  Psi(i,j+1,k) - Psi(i,j,k)    );
Dy  = 1/(2*dy) * ( Psi(i,j+1,k)-Psi(i,j-1,k) );
Dzm = 1/dz * (  Psi(i,j,k) - Psi(i,j,k-1)  );
Dzp = 1/dz * (  Psi(i,j,k+1) - Psi(i,j,k)    );
Dz  = 1/(2*dz) * ( Psi(i,j,k+1)-Psi(i,j,k-1) );

GradPlus = sqrt( max(Dxm,0).^2 + min(Dxp,0).^2 + ...
                 max(Dym,0).^2 + min(Dyp,0).^2 + ...
                 max(Dzm,0).^2 + min(Dzp,0).^2 );
GradHinus = sqrt( max(Dxp,0).^2 + min(Dxm,0).^2 + ...
                  max(Dyp,0).^2 + min(Dym,0).^2 + ...
                  max(Dzp,0).^2 + min(Dzm,0).^2 );

%Fo = sign(Psi);  Fo = Fo(i,j,k);
Fo = min( ones(size(Psi)), abs(Psi) ).*sign(Psi);  Fo = Fo(i,j,k);

% Bounds on Hamiltonian (see p.69 Sethian's text)
Ham = GradPlus.*max(Fo,0) + GradHinus.*min(Fo,0);
%[Hx Hy Hz] = gradient(Ham,h);
alphaU = 0;%max(max(max(abs(Hx))));
alphaV = 0;%max(max(max(abs(Hy))));
alphaW = 0;%max(max(max(abs(Hz))));

% update all of Psi except for elements on the edge of array
PsiNew = Psi;

PsiNew(i,j,k) = Psi(i,j,k) ...
        - dt*( Ham - 1/2*alphaU*(Dxp-Dxm)  - 1/2*alphaV*(Dyp-Dym) ...
    - 1/2*alphaW*(Dxp-Dxm)  - Fo );

PsiNew(1,:,:)=PsiNew(2,:,:);
PsiNew(p,:,:)=PsiNew(p-1,:,:);
PsiNew(:,1,:)=PsiNew(:,2,:);          %    <-- "ghost cell" boundary condfigure(2); hold on; contour(psi,[0 0],'r');  drawnow;
PsiNew(:,q,:)=PsiNew(:,q-1,:);
PsiNew(:,:,1)=PsiNew(:,:,2);
PsiNew(:,:,r)=PsiNew(:,:,r-1);

Psi = PsiNew;

[PsiX PsiY PsiZ] = gradient(Psi(3:end-2,3:end-2,3:end-2),dx,dy,dz);
HagGradPsi = (PsiX.^2+PsiY.^2+PsiZ.^2).^(1/2);

end;

% Plot 'after' picture:
figure(5); clf;
isosurface(Psi,0); title('after');  drawnow;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Segmentation function    by Joe West, Harch 2003

function psi = EdgeDetect(Im, EdgeValue,  sigma, w, InitialPsi, epsilon, Verbose);
% Im = image
% EdgeValue = value of edge of ventricles in 'normalized' image
% sigma = blur factor
% w = curvature weighting factor (for smoothness)
% InitialPsi = initial guess for psi
%              (usually a circle or collection of circles)
% epsilon = convergence parameter
% Verbose = 1 to display progress, 0 to display nothing

% Typical values of input:
% EdgeValue = 0.3
% sigma = 1
% w = 0
% InitialPsi = InitialGuess(Im)
% epsilon = 0.05
% Verbose = 1

TimeLimit = 15;
BeforeTime = clock;

% Scale Image and inital guess to a manageable size (say N cells vertically)
N = 100;
[p q] = size(Im);
Hag = N/p;
ImOriginal = Im;
Im = imresize(Im,Hag,'bicubic');
InitialPsi = Hag*imresize(InitialPsi,Hag,'bicubic');

Im = NormalizeIm(Im);

figure(1); clf; showIm(Im);
title('Original Image');

% Blur Image

Im = Blur(Im,sigma);
figure(2);  clf;  showIm(Im);
title('Blurred Image');  drawnow;

% Normalize Image:  (make all entries between 0 and 1)
Im = Im/max(max(Im));

% Show initial guess:
psi = InitialPsi;
psiNew = psi;
psi_lastPlotted=psi;

% speed field
v = 1-1/EdgeValue*Im;

% Stopping Criterion
delta=0.5; % width about zero level set to check
%epsilon=0.05;   % tolerance for difference between image value
                 % and edge value
a = 1;   % parameter used in computation

Close = find(abs(psi)<delta); % indices of elements of psi close to
% the level set
MeanDev = sum( abs(Im(Close)-EdgeValue) ./ (abs(psi(Close))+a) ) / ...
          sum( 1./(abs(psi(Close))+a) );
MaxSpeed = max(max(abs(v(Close))));
MaxSpeed = MeanDev;

%dt = 0.1; % timestep
n = 25; % draw contours every n updates in the loop
t=0;  ThisTime = 0;

% Repeat until stopping criterion is reached
while (MeanDev >= epsilon) & (ThisTime < TimeLimit)
   t = t + 1;

   psi = psiNew;

   v = 1-1/EdgeValue*Im;
   %psiNew = UpdatePsi2d(1, psi, 1, 0, v, w);
   psiNew = UpdatePsi2d(0, psi, 1, 0, v, w);
```

```
   %if abs( round(t/n)-t/n ) < 0.01*dt
   if abs( round(t/n)-t/n ) < 0.001 & Verbose
     figure(2);
     hold on; contour(psi_lastPlotted,[0 0],'r');
     psi_lastPlotted = psiNew;
     hold on; contour(psiNew,[0 0],'r'); drawnow;
   end;

   Close = find(abs(psiNew)<delta);
   HeanDev = sum( abs(Im(Close)-EdgeValue) ./ (abs(psiNew(Close))+a) ) / ...
             sum( 1./(abs(psiNew(Close))+a) );
   %HaxSpeed = max(max(abs(v(Close))));
   HaxSpeed = abs(psiNew-psi);
   HaxSpeed = max(max(HaxSpeed(Close)));

   [Dx Dy] = gradient(psiNew(3:end-2,3:end-2));
   AbsGrad = sqrt(max(max(Dx.^2+Dy.^2)));
   if Verbose; disp([HeanDev, HaxSpeed, AbsGrad]); end;

   if AbsGrad > 5 % Re-initialize psi
      if Verbose; disp('Re-initaializing...'); end;
      %psiNew = ReInit2dSlow(psiNew, 1);
      %psiNew = ReInit2d(psiNew,1);
      psiNew = psiNew/AbsGrad;
      if Verbose; disp('done re-initialization.'); end;
   end;

   AfterTime = clock;
   ThisTime = ElapsedMinutes(BeforeTime, AfterTime);
end;
psi = psiNew;

% Scale psi back to original image size
psi = 1/Mag*imresize(psi,[p q],'bicubic');

figure(1); clf; showIm(ImOriginal);
hold on; contourplot(psi,1,'r',1.5); drawnow;
title('Image with successive edge-detecting contours');


AfterTime = clock;
ThisTime = ElapsedMinutes(BeforeTime, AfterTime);

disp(ThisTime);

if ThisTime > TimeLimit
   display('TIMED OUT!');
   %psi(1,1)=999;
end;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Segmentation function (3d version) by Joe West, March 2003

function psi = EdgeDetect(Im, EdgeValue,  sigma, w, InitialPsi, epsilon, Verbose);
% Im = 3d image
% EdgeValue = value of edge of ventricles in 'normalized' image
% sigma = blur factor
% w = curvature weighting factor (for smoothness)
% InitialPsi = initial guess for psi
%              (usually a circle or collection of circles)

% Typical values:
% EdgeValue = 0.4;
% sigma = 1;
% w = 0;
% InitialPsi = InitialGuess(Im);

% Note: To read a dicom image:
% Im = dicomread('MR106000052.dcm');

TimeLimit = 300;
BeforeTime = clock;

Im = double(Im);

%.Blur Image

Im = Blur3d(Im,sigma);

% Normalize Image:  (make all entries between 0 and 1)
Im = Im/max(max(max(Im)));

% Show initial guess:
psi = InitialPsi;
psiNew = psi;
figure(2); clf; isosurface(psi,0); axis equal; drawnow;

% speed field
v = 1-1/EdgeValue*Im;

% Stopping Criterion
delta=0.5; % width about zero level set to check
%epsilon=0.05;   % tolerance for difference between image value
                 % and edge value
a = 1;   % parameter used in computation

Close = find(abs(psi)<delta); % indices of elements of psi close to
% the level set
HeanDev = sum( abs(Im(Close)-EdgeValue) ./ (abs(psi(Close))+a) ) / ...
            sum( 1./(abs(psi(Close))+a) );
HaxSpeed = max(max(max(abs(v(Close)))));


%dt = 0.1; % timestep
n = 2; % draw contours every n updates in the loop
t=0;  ThisTime = 0;

% Repeat until stopping criterion is reached
while (HeanDev >= epsilon) & (ThisTime < TimeLimit)
   t = t + 1;

   psi = psiNew;

   v = 1-1/EdgeValue*Im;
   %psiNew = UpdatePsi2d(1, psi, 1, 0, v, w);
   psiNew = UpdatePsi3d(0, psi, 1, 0, v, w);

   %if abs( round(t/n)-t/n ) < 0.01*dt
   if abs( round(t/n)-t/n ) < 0.001 & Verbose
      figure(2);
      clf; isosurface(psiNew,0);
   end;

   Close = find(abs(psi)<delta);
   HeanDev = sum( abs(Im(Close)-EdgeValue) ./ (abs(psi(Close))+a) ) / ...
             sum( 1./(abs(psi(Close))+a) );
   HaxSpeed = max(max(max(abs(v(Close)))));

   [Dx Dy Dz] = gradient(psiNew(3:end-2,3:end-2,3:end-2));
   AbsGrad = sqrt(max(max(max(Dx.^2+Dy.^2+Dz.^2))));
   if Verbose; disp([HeanDev, HaxSpeed, AbsGrad]); end;
```

```
   if AbsGrad > 5 % Re-initialize psi
      if Verbose; disp('Re-initaializing...'); end;
      %psiNew = ReInit3d(psiNew, 1);
      psiNew = psiNew/AbsGrad;
      if Verbose; disp('done re-initialization.'); end;
   end;

   AfterTime = clock;
   ThisTime = ElapsedMinutes(BeforeTime, AfterTime);
end;
psi = psiNew;

AfterTime = clock;
ThisTime = ElapsedMinutes(BeforeTime, AfterTime);

disp(ThisTime);

if ThisTime > TimeLimit
   display('TIMED OUT!');
   %psi(1,1,1)=999;
end;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initial guess generator for edge detection function:

function psi = InitialGuess(Im);

psi = inf*ones(size(Im));

% Im = image
Im = double(Im);

figure(1); clf;
showIm(Im);

disp('Click on point inside ventricles, then on point outside ventricles.');
disp('If there is more than one region, repeat this until done.');
disp('Double-click or hold shift key when selecting last point.');
disp('Press backspace to go back a point.');

% Get the inside/outside points:
[Y X] = getpts;
disp('Thanks.');

% For each pair of points, make a circle through them.  This will
% produce a circle that intersects the ventricular walls.
for i = 1:2:length(X)

   % centre and radius:
   xo = X(i);
   yo = Y(i);
   r = sqrt ( (X(i)-X(i+1))^2 + (Y(i)-Y(i+1))^2 );

   % make level set circle:
   ThisPsi = Circle(size(Im), xo, yo, r);

   % add it to the collection
   psi = min( psi, ThisPsi );

end;

figure(1);  hold on;
contour(psi,[0 0],'r');



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5
% look at contours of an image

function ImageContours(Im,sigma,i);

figure(i); clf;
Im = Blur(Im,sigma);
Im = imresize(Im,100/size(Im,1),'bicubic');
[c,r] = contour(Im); clabel(c,r); axis ij; axis equal;
title('Image contours');  drawnow;
```

Other Functions:

The following "helper" functions used in the above code have also been created. We describe them here.

- `Circle.m, Sphere.m`: Create LS functions for a circle and sphere, respectively.
- `ElapsedMinutes.m`: Calculates the number of minutes elapsed.
- `MagGrad.m, MagGrad3d.m`: Calculate $|\nabla\phi|$ in 2d and 3d, respectively.
- `contourplot.m, isoplot.m`: Combine matlab's plotting command `contour` or `isosurface` respectively with other commands to handle uneven mesh sizes and to plot the lines with varying thickness and colour in a single function call.

- `Blur.m, Blur3d.m:` Blurs an image with a Gaussian Kernel.
- `NormalizeIm.m:` Converts image to double precision format and normalizes intensity of each pixel/voxel so that $I_{ij(k)} \in [0, 1]$.
- `readIm.m, showIm.m:` Specialize matlab's `imread/dcmread` or `imshow` commands for our purposes.

Bibliography

[1] M.A. Biot. *General theory of three-dimensional consolidation.* Journal of Applied Physics, 12:155, 1941.

[2] T.F. Chan, L.A. Vese. *Active Contours Without Edges.* IEEE Transactions on Image Processing, vol. 10, no. 2, February 2001.

[3] A.J. Chorin and J.E. Marsden, *A Mathematical Introduction to Fluid Mechanics,* Spreinger-Verlag, New York, NY, 1980.

[4] M.G. Crandall, and P.L. Lions. *Viscosity Solutions of Hamilton-Jacobi Equations*, Tran. AMS, 277, pp. 1-43, 1983.

[5] C. Davatzikos, D. Shen, A. Mohamed, and S.K. Kyriacou. *A Framework for Predictive Modeling of Anatomical Deformations.* IEEE Transactions on Medical Imaging, vol. 20, no. 8, august 2001.

[6] J.M. Drake and Christian Sainte-Rose, *The Shunt Book,* Blackwell Science, 1995, p. 144.

[7] J.M. Drake, J.R.W. Kestle, and R. Milner et al. *Randomized clinical trial of cerebrospinal fluid shunt design in pediatric hydrocephalus.* Neurosurgery, 43:294-305, 1998.

[8] C.S. Drapaca. *Brain Biomechanics: Dynamical Morphology and Non-Linear Viscoelastic Models of Hydrocephalus.* PhD Thesis, University of Waterloo, 2002.

[9] A.E. Engin and H.C. Wang. *A mathematical model to determine viscoelastic behaviour of in vivo primate brain.* J. Biomechanics, 3:283-296, 1970.

[10] Engquist, B., and Osher, S.J., *Stable and Entropy-Satisfying Approximations for Transonic Flow Calculations*, Math. Comp., 34, 45, 1980.

[11] M.S. Estes and J.H. McElhaney. *Response of Brain Tissue of Compressive Loading.* ASME paper, 70-BHF-13:1-4, 1970.

[12] "Stress-Strain-History Relations of Soft Tissues in Simple Elongation," in Biomechanics: Its Foundations and Objectives. Y.C. Fung, N. Perrone, and M. Anliker (eds.) Prentice-Hall, 1972.

[13] J.E. Galford and J.H. McElhaney. *A viscoelastic study of scalp, brain, and dura.* J. of Biomechanics, 3:211-221, 1970.

[14] J. Gomes and O. Faugeras. *Reconciling Distance Functions and Level Sets.* Journal of Visual Communication and Image Representation 11, 209-223 (2000).

[15] X. Han, C. Xu, M.E. Rettmann, and J.L. Prince. *Automatic segmentation editing for cortical surface reconstruction.* Proceedings of SPIE, vol. 4322. Medical Imaging 2001, Image Processing, 2001, Image Processing, 19-22 Feb. 2001, San Diego, USA.

[16] E. Hopf. *The Partial Differential Equation* $u_t + uu_x = \mu u_{xx}$. Communications in Pure and Applied Mathematics, vol.3 [1950], pp. 201-230.

[17] M. Kaczmarek, R.P. Subramaniam, and S.R. Neff. *The biomechanics of hydrocephalus: steady-state solutions for cylindrical geometry.* Bull. of Math. Biol., 59(2):295-323, 1977.

[18] S.K. Kyriacou, A. Mohamed, K. Miller, and S. Neff. *Brain Mechanics For Neurosurgery: Modeling Issues.* Biomechanics and Modeling Mechanobiology, 1:2, pp. 151-164 (2002).

[19] Lax, P.D., *Hyperbolic Systems of Conservation Laws and the Mathematical Theory of Shock Waves,* SIAM Reg. Conf. Series, Lectures in Applied Math, 11, pp. 1-47, 1970.

[20] LeVeque, R.J., *Numerical Methods for Conservation Laws,* Birkhauser, Basel, 1992, p. 133.

[21] J. Martin, A. Pentland, and S. Sclaroff. *Characterization of Neuropathological Shape Deformations.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, no. 2, February 1998.

[22] K. Miller, K. Chinzei, G. Orssengo, P. Bednarz. *Mechanical properties of brain tissue in-vivo: experiment and computer simulation.* Journal of Biomechanics, 33 (2000) 1369-1376.

[23] T. Nagashima, N. Tamaki, S. Matsumoto, B. Horwitz, and Y. Seguchi. *Biomechanics of hydrocephalus: A new theoretical model.* Neurosurgery, 21(6):898-904, 1987.

[24] S. Osher and R. Fedkiw, Level Set Methods and Dynamic Implicit Surfaces, Springer-Verlag New York, Inc., 2003.

[25] M.R. Pamidi and S.H. Advani. *Nonlinear constitutive relations for human brain tissue.* Transactions of the ASME, 100:44-48, 1978.

[26] K.D. Paulsen, M.I. Miga, F.E. Kennedy, P.J. Hoopes, A. Hartov, and D.W. Roberts. *A Computational Model for Tracking Subsurface Tissue Deformation During Stereotactic Neurosurgery.* IEEE Transactions on Biomedical Engineering. vol. 46, no. 2, February 1999.

[27] G.W.M. Peters, J.H. Meulman, and A.A.H.J. Sauren. *The applicability of the time/temperature superposition principle to brain tissue.* Biorehology, 34(2):127-138, 1997.

[28] L.A. Platenik, M.I. Miga, et al. *In Vivo Quantification of Retraction Deformation Modeling for Updated Image-Guidance During Neurosurgery.* IEEE Transations on Biomedical Engineering, vol. 49, no. 8, august 2002.

[29] D.W. Schwendeman. *A front dynamics approach to curvature-dependent flow.* SIAM J. Appl. Math, 56 (1996), no. 6, p. 1523.

[30] J.A. Sethian. *Level Set Methods and Fast Marching Methods.* Cambridge University Press, 1999.

[31] L.Z. Shuck, R.R. Haynes, and J.L Fogle. *Determination of Viscoelastic Properties of Human Brain Tissue.* ASME paper, 70-BHF-12, 1-7, 1970.

[32] M. Sussman, P. Smereka, and S. Osher. *A Level Set Approach for Computing Solutions to Incompressible Two-Phase Flow.* J. Comp. Phys., vol. 114, 146-159 (1994).

[33] Sod, G.A., *Numerical Methods in Fluid Dynamics,* Cambridge University Press, 1985, p. 298.

[34] E.G. Takhounts. *Experimental Determination of Constitutive Equations for Human and Bovine Brain Tissue.* PhD Dissertation Persented to the Faculty of the School of Engineering and Applied Science, University of Virginia, 1998.

[35] G. Tenti, S. Sivaloganathan, and J.M. Drake. *Brain biomechanics: steady-state consolidation theory of hydrocephalus.* Can. Appl. Math. Q., 7, 1999.