

Level Set Methods: An Act of Violence

*Evolving Interfaces in Geometry,
Fluid Mechanics, Computer Vision
and Materials Sciences*

J.A. Sethian ¹

Boundaries abound. Dynamic boundaries change position and shape in response to the particular physics at work: examples are breaking waves in the ocean, dancing flames in the fireplace, and milk swirling in a cup of tea. Static boundaries, such as tumors in medical scans and cartoon characters against a background animation, can be just as perplexing: try finding edges in a picture of a dalmatian lying on a rug with spots! Surprisingly enough (as we shall soon see), less familiar problems can be also cast in the setting of evolving boundaries, including negotiating a piano through a cramped apartment and finding the shortest path over a mountain range.

While the physics and chemistry that drives a boundary (or “interface”) may be formidable, it can be difficult enough to simply follow the shape of an evolving interface, even if its speed and direction of motion are well understood. The first concern is the formation of sharp corners, as witnessed by the intricate patterns in a snowflake. Second, distant edges can blend together: the “edge” of a forest fire changes as separate fires burn together and sparks carried by the wind ignite distant regions. And third, in three dimensions (and higher), even finding a nice way to represent (let alone move) an undulating boundary is a challenge.

Hermann Weyl, a vocal and well-known figure in mathematics, once said “The introduction of a coordinate system to geometry is an act of violence.”²

¹James Sethian is Professor of Mathematics at the University of California at Berkeley, and Head of the Applied and Computational Mathematics Dept. at the Lawrence Berkeley National Laboratory. He may be reached at sethian@math.berkeley.edu.

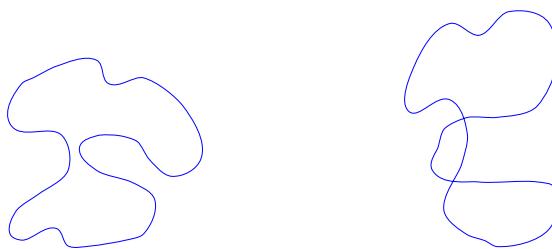
²Weyl relished his own opinionated views about mathematics. The actual quote is: “The introduction of numbers as coordinates by reference to the particular division schemes

Level Set Methods commit this violence, and, in doing so, provide mathematical and computational tools for tracking evolving interfaces with sharp corners and cusps, topological changes, and three dimensional complications. Along the way, they efficiently compute optimal robot paths around obstacles, extract clinically useful features from the noisy output of medical images, and model the manufacturing steps that transfer a streetmap of circuitry onto a tiny piece of silicon.

It will require a bit of background to explain these techniques. We will start with a example of a moving interface, and indeed show why a coordinate system seems so unnatural. Unfortunately, the traditional alternative, known as an “intrinsic parameterization”, will turn out to have its own problems; it is ill-suited for sharp corners and topological change. The right way to view things, at least for computational purposes, comes from recasting the problem in a higher dimensional space, where, paradoxically, a coordinate system offers salvation. Armed with these level set techniques, we can efficiently compute solutions to problems in geometry, fluid mechanics, computer vision, and materials sciences.

An Opening Example

Take a piece of rope, glue the two ends together, and drop it on the ground, making sure that the rope doesn’t cross over itself (this is known as a **simple closed curve**.)

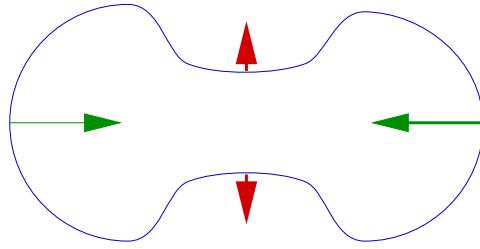


A Simple Closed Curve Not A Simple Closed Curve

of the open one-dimensional continuum is act of violence whose only practical vindication is (its) special calculatory mangeability...”. Along the same lines, ”....the coordinate system remains as the necessary residue of the ego-extinction.”

One defining characteristic of a curve is its curvature, which measures how fast the curve bends at any spot. For example, a circle has a constant curvature because it always is turning at the same rate; a smaller circle has a higher constant curvature because it turns faster.

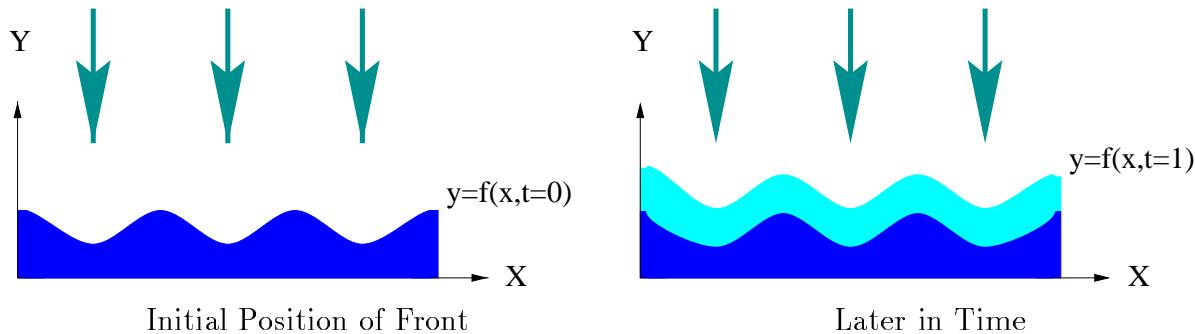
Now, suppose each piece of the curve moves perpendicular to the curve with speed proportional to the curvature. Since the curvature can be either positive or negative (depending on whether the curve is turning clockwise or counterclockwise), some parts of the curve move outwards while others move inwards. In the figure below, the red arrows are where the curvature is negative, and the green arrows are where the curvature is positive: the arrows are of different lengths because the magnitude (or “strength”) is larger at the green arrows than it is at the red ones.



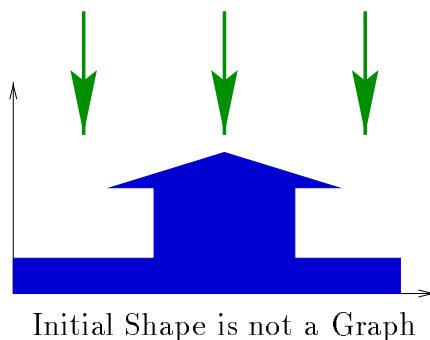
What happens to this curve as it moves according to this “motion by curvature”? If the initial curve is a circle, it’s easy to convince yourself (by looking at the symmetry of the problem) that each point on the curve races in towards the center, and the interface must collapse stay a circle and collapse to a single point. In the figure above, you can probably also convince yourself that the shape relaxes itself and smooths out and becomes more circular. In fact, for any simple closed curve, there is a remarkably tidy mathematical answer that explicitly predicts an interface’s future from its initial shape: we will come to that theorem shortly. But since motion by curvature is one component of many physical phenomena (for example, surface tension in a soap bubble and freezing rates at the edge of a snowflake both depend on the curvature at a point), let’s try to build a computer model of what happens to an evolving interface moving “under its curvature”.

Representing an Evolving Interface: Functional vs. Parametric

In order to move an interface, we need first a good way to describe it. A standard way to describe a curve is as the graph of a function, so let's begin with that. Recall that a function f has the property that every input produces a unique (that is, one and only one) output. Imagine now a steady snow falling straight down on a hilly terrain, and let the function $y = f(x, t = 0)$ describe the initial height y at any point x at time $t = 0$. As the snow accumulates, the height changes in time above each point x .



Unfortunately, this is limited view; there is no guarantee that the initial position of the front can be always written as the graph of a function. For example, the initial shape below does not have a unique height for every point x , nonetheless the accumulation of snow on this shape can be easily predicted.

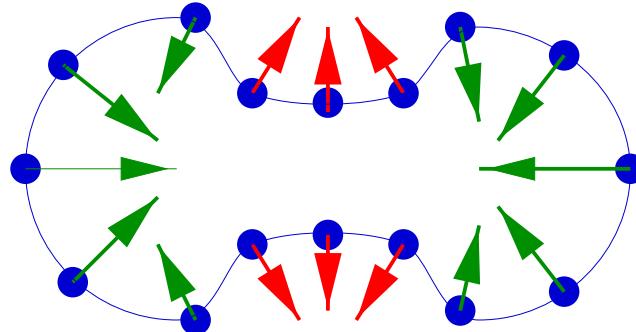


The limitation here, to recall our earlier quote, is our use of a coordinate system; it has nothing to do with the problem, but has severely restrained our options.

If you were a differential geometer, you would most likely abandon this functional representation of an interface for the “parameterized” view. Imagine you are walking along the interface, calling out both your x and y coordinates as you move: at the same time that you walk, a friend is plotting those points on the xy plane. The speed at which you walk is known as the “parameterization”; the curve drawn by your friend is known as the “image”. Regardless of whether you walk quickly or slowly, your friend will trace out the same boundary. The advantage of this approach is that the dependence on the coordinate system has vanished; you can orient the figure any way you’d like, and things still work.

A First Attempt at a Numerical Algorithm

Suppose we try to use this parameterized representation of an interface as the backbone of a numerical algorithm. We can walk around the curve, and plant a blue buoy at regular intervals. These buoys, together with the ropes that connect them together, form a discrete view of the boundary. A discretized version of the earlier motion by curvature example would then look like the following figure.

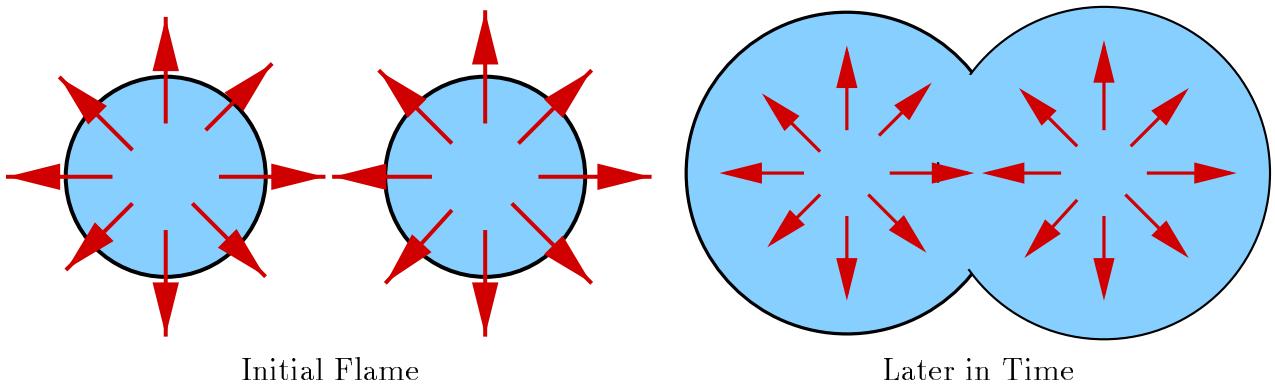


Recall that the length and direction of the arrows is determined by the local curvature. The strategy is to advance the positions of the buoys according to the arrows, recalculate new arrows, and then advance the buoys again; the hope is that by using more buoys, a more accurate answer will emerge.

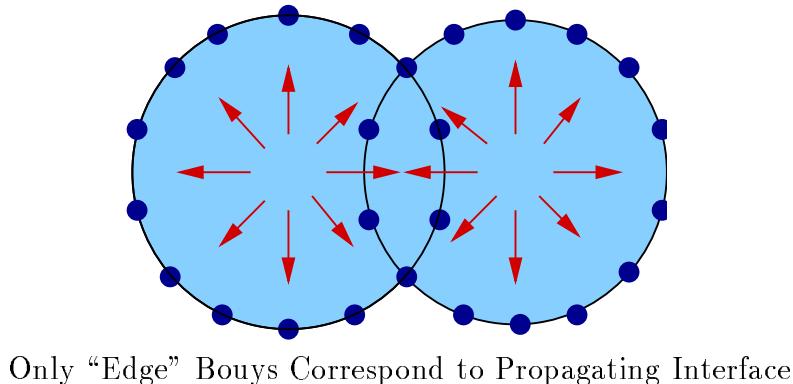
Unfortunately, there are several flaws in this approach, some inconvenient,

some fatal. A look at the figure reveals an inconvenient one: the bouys try and cross over themselves, and it becomes hard to keep the connecting ropes organized. A remedy is to stop the advancement periodically, re-walk along the curve, and drop new equi-spaced bouys. However, doing this for a propagating surface in three dimensions is, at the very least, unpleasant.

A more serious problem comes when the evolving boundary attempts to change its topology. Taking a slight detour, consider two separate circular flames, each burning outwards at a constant speed: the shape of the evolving interface is easily predicted.



As the two separate flames burn together, the evolving interfaces merge into a single propagating front. However, a numerical algorithm based on a discrete parameterization runs into real trouble: in the figure below, the two pairs of bouys located inside the burned region must somehow be removed if we want to track the true “edge” of the expanding flame.



Trying to systematically determine which bouys to remove is a confusing task: doing so in three dimensions is overwhelming. (A detailed, technical discussion of these issues may be found in [7]).

Paradoxically enough, the path to an efficient and versatile representation of propagating interfaces leads directly to the violent act of introducing a coordinate system. However, the trick is to do so in one higher dimension: this is the fundamental idea behind level set methods, introduced in [6], and based on earlier work in [7].

A Level Set Representation

Rather than follow the interface itself, the level set approach instead takes the original interface and adds an extra dimension to the problem. Recall the previous interface, which consisted of two expanding circular blue flames. We are going to re-introduce a coordinate system, using the xy plane which contains the interface, and a z direction to measure height.

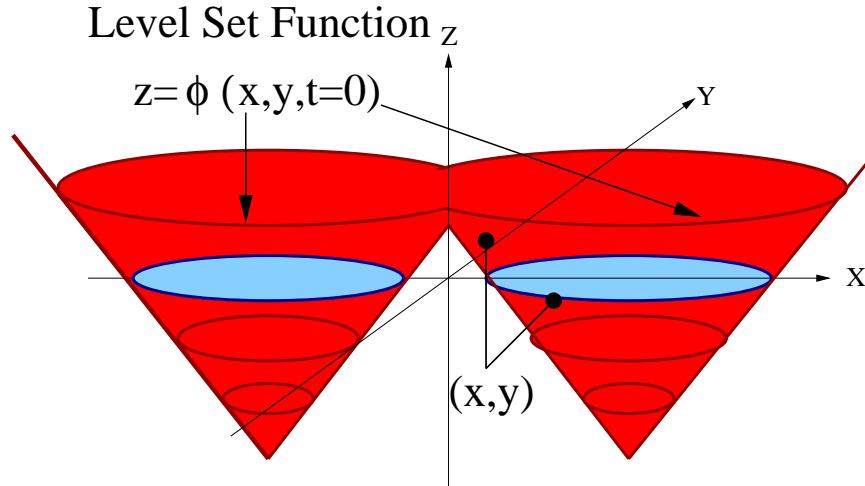
Suppose we invent a function $z = \phi(x, y, t = 0)$, just as was done previously, to take as input a point (x, y) , and assigns a height z . This time, however, assign as height z the distance from (x, y) to the interface at time $t = 0$.

This builds a surface (shown in red) with the property that it intersects the xy plane **exactly** at the interface. The red surface is called the level set function, because it accepts as input any point in the plane and hands back a height as output. The blue interface is called the zero level set, because it is the collection of all points that are at height zero.

Another way to see why this is called a “level set surface” is to imagine a saw that can cut a slice of the surface and then drop it onto the xy plane. However, the slice has to be perfectly level.

- If the saw cuts the red level set surface at height zero above the xy plane, the slice that will drop to the xy plane will be the original interface corresponding to the pair of blue flames.
- If the saw cuts at some other height, a different slice will drop down, producing one of the red curves instead.

Yet another way to view this level set function is to think of a topographic

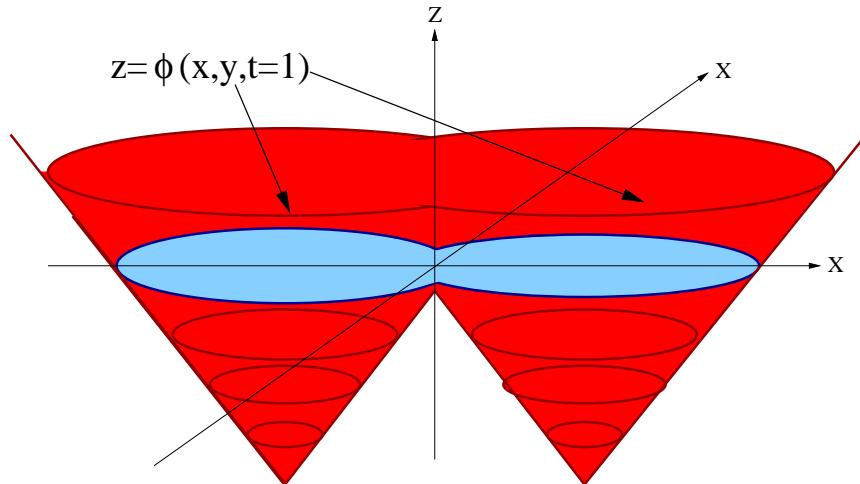


The Level Set Surface (in red) plots the distance from each point (x, y) to the Interface (in blue)

hiking map which gives surface elevations. We have chosen a map such that sea level always corresponds to the edge (or edges!) of our interface, with oceans inside the interface, and mountains outside the interface.

Our plan is to figure out how to change the height of the surface $\phi(x, y, t)$ in time to match the evolution of the interface. The goal is to let the level set function expand, rise, fall, and do all the work: to find out where the interface is at any time, we can simply cut the surface at zero height, in other words, plot the zero contour.

At first glance, it might seem crazy to take the problem of a moving curve and trade it in for a moving surface! More dimensions usually mean more work. The reason the extra dimension is so powerful is that, rather than track buoys around which can collide and stretch apart, we can now stand at each point xy and adjust the height of the level set function. This means, for example, that our topological problems have vanished; two expanding flames which merge into one simply means that the zero level set at a particular time becomes one curve rather than two.



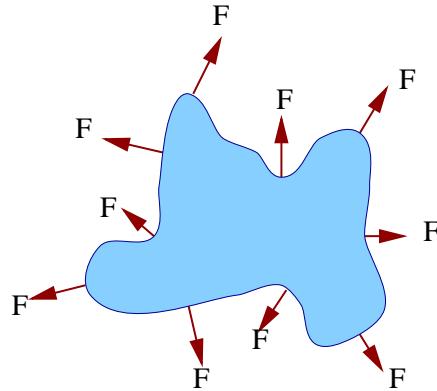
Later in Time: Red Level Set Surface has moved, yielding new Blue Interface

To summarize, level set methods exchange a geometric, moving coordinate representation for a fixed coordinate perspective where each point (x, y) adjusts its value to measure the distance to the evolving interface. Using terminology from basketball, marker particle/bouy methods are a man-to-man coverage, while level set methods are a zone defense.

One of the most striking aspects of the level set approach is that nothing is changed for interface problems in three (or more!) dimensions: while slightly harder to visualize, the strategy is still the same. First, embed the evolving surface in one higher dimension. In the case of a propagating surface, this would mean using a time-dependent function $\phi(x, y, z, t = 0)$ in four-dimensional space. Then, adjust this higher dimensional function corresponding to motion of the interface, and compute the “zero” level set to find the position of the propagating interface. All together, the trick of embedding the front in a higher dimensional function is well worth the added cost (in fact, with some work, that cost can be made the same as that of marker techniques).

Level Set Methods: Technical Details

The story would be incomplete with explaining how to actually move the level set surface. Suppose you are given an interface separating one region from another (either a closed curve in two dimensions or a closed surface in three dimensions), and a speed F that tells you how to move each point of the interface. Here, F can depend on all sorts of complex physics, such as heating on either side of the interface, or fluid mechanical effects. Regardless, we shall assume that the speed F is handed to us, and gives the speed in the direction perpendicular to the interface (observe that any tangential component will have no effect on the position of the front).



Front Propagating with Speed F

We build an initial value for the level set function $\phi(x, y, t = 0)$ based on the signed distance d from each point (x, y) to the initial front, choosing a positive distance if we are outside the blue region, and a negative if we are inside. This constructs an initial value for the level set function ϕ ; all that remains is to figure out how to adjust its value in time to match the evolving interface.

Fortunately, this is strikingly easy. Imagine a bouy sitting on the blue front, and let its position be described by $(x(t), y(t))$, where t is time. Then in order for this bouy to always ride on the edge of the blue interface as the surface moves, it must always be true that:

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

since the blue interface always corresponds to the place where $\phi = 0$. Now, all

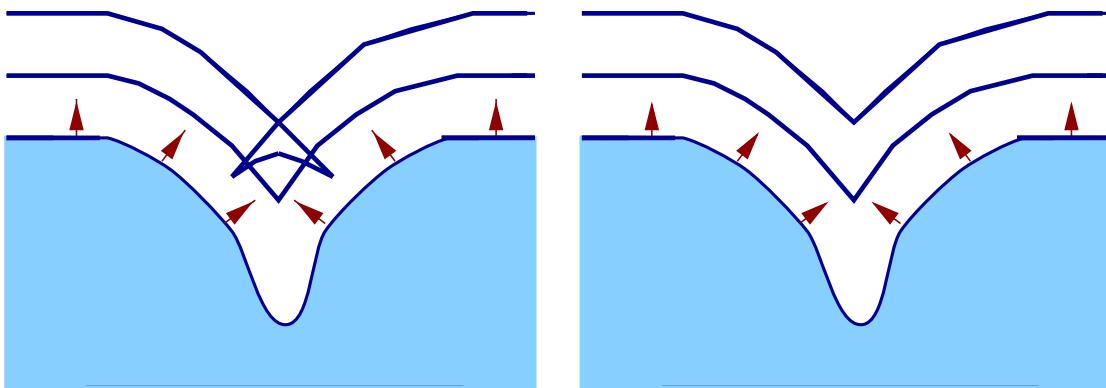
we need to do is apply the chain rule from calculus: take the time derivative of both sides, and substitute the speed function F which tells how the front moves. A little bit of algebraic manipulation produces the level set equation, namely

$$\phi_t + F (\phi_x^2 + \phi_y^2)^{1/2} = 0.$$

where the subscripts let us know that we are taking partial derivatives.

The above level set equation is called an *initial value partial differential equation*: initial value because it describes the time-evolution of a solution on the basis of an initial state, and partial differential because the equation contains partial derivatives. It transforms pure geometry problems into the language of partial differential equations, where theoretical results about existence and uniqueness of solutions may be used to analyze solutions for different speed functions F .

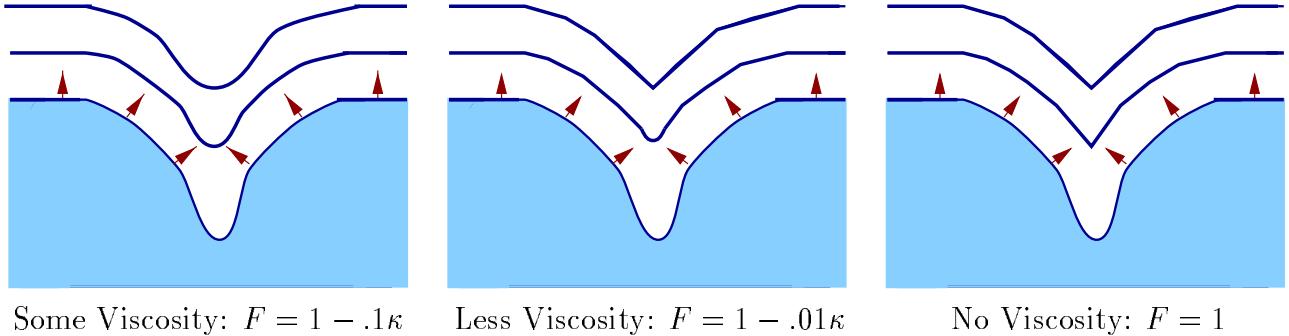
An illustration of the power of partial differential equations comes from an additional example. Starting from the simple case of a constant speed function $F = 1$, and a sinusoidal initial interface, consider two solutions to the problem.



The Swallowtail Solution

The Leading Wave Solution

The two solutions are the same until a corner develops in the propagating interface, at which point one of them overlaps itself, while the other chooses only the leading wave. Let's name the solution on the left the "swallowtail" solution, and the one on the right the "leading wave" solution. Intuitively, the "leading wave" solution seems like the physically correct one, especially in light of the earlier discussion about removing markers which don't lie on the boundary between inside and outside.



But which solution is chosen by the level set partial differential equation? As soon as the evolving front develops the sharp corner, all bets are off: we can't evaluate the partial derivative at a place when the slope makes a sudden jump in direction.

The answer comes from the mathematical theory of *viscosity solutions*. Loosely speaking, viscosity measures the ability of a fluid right damp sharp transitions and mute sudden changes. If you drop a marble in a jar of honey, the viscosity of the honey slows it down. We will use this idea of viscosity to smooth out the corner in our propagating interface.

Recall that motion under curvature acts to smooth out sharp corners; we can think of this as adding a little viscosity. With this in mind, let's consider a speed of the form $F = 1 - 0.1\kappa$, where κ is the curvature. We can substitute this speed into our level set equation to produce

$$\phi_t + (\phi_x^2 + \phi_y^2)^{1/2} = 0.1\kappa.$$

That little bit of curvature acts to smooth out the sharp corner; a little bit less curvature ($F = 1 - .01\kappa$) smooths it out even less. Observe that even though there is very little smoothing going on with such a small amount of curvature, as long as some positive non-zero amount is added, it is enough to guarantee that a corner never develops.

The theory of viscosity solutions leads to a remarkable fact: if we take a sequence of problems, each with ever smaller viscosity, they will head towards our corner “leading wave” solution. This means that all we need to do is solve for the viscosity solution of our level set equation, and we are guaranteed to pick out the right topological evolving front.

We've said very little about how one actually devises a numerical algorithm to solve the level set equation. Briefly, numerical schemes discretize xy

space into a grid of points, like a perfect map of streets and avenues. Each city block holds a value for the level set function, and updates its value as the surface moves using neighboring values to determine the necessary partial derivatives in the level set equation. In practice, the most sophisticated level set algorithms employ two critical improvements:

- Rather than update the value of the level set function everywhere, work is confined to a thin region around the evolving front: this is known as the narrow band method, see [1].
- In the case of fronts that always move forwards under some particular speeds, an extraordinarily efficient fast marching method is preferred, see [8].

Motion by Curvature

As a first application of level set methods, we can revisit motion by curvature, and examine what happens to a closed curve moving with speed $F = -\kappa$ (the minus sign is chosen so that convex parts move in, and concave parts move out). We've seen that a circle must collapse smoothly to a point before it disappears, and argued that more complicated simple closed curves must smooth out. In fact, Grayson, (see [3]), proved that every simple closed curve collapses smoothly to a single point, without crossing over itself. This is a remarkable theorem: no matter how complicated or convoluted a curve might be, it quickly relaxes itself into a circular object and shrinks down to a point. As illustration, in the figure below we show one such curve shrinking to a circular object; from there, it is easy to believe that it shrinks to a point and disappears.



Motion under Curvature: Collapse of a Curve to a Single Point

If you'd like to try this out, a java applet is available which allows you to draw in any curve you'd like, and follow its evolution, see

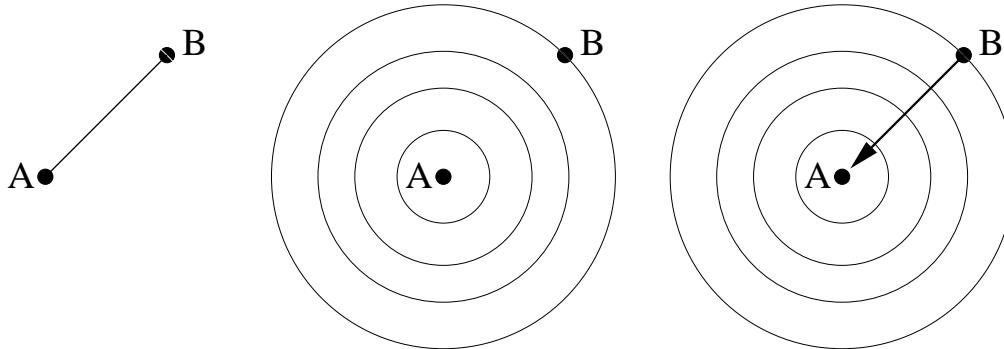
http://math.berkeley.edu/~sethian/level_set.html

Negotiating a Piano Through a Cramped Apartment:

Suppose you live in an apartment with lots of corridors and long skinny halls. You've just bought a piano, which you need to get from the front door to the back room. The first question is, is it actually possible to twist and turn the piano in such a way as to get it back there? And second, if it is, what is the shortest path?

This is a problem in robotic navigation with constraints; the **navigation** part is to find the shortest path; the **constraint** part is the requirement that you don't accidentally add a few new holes in the walls as you move the piano. With a little work, this can be recast as a problem involving propagating interfaces: here, we briefly summarize the application of level set methods and fast marching methods to optimal path navigation developed in [4].

To see how this becomes an evolving interface, let's start with a simpler problem. Suppose you are standing in a parking lot. You are at point A, your car is at point B, and there are no other cars in the lot. If you want the shortest path to your car, you can just draw the straight line shown in the figure on the left below.

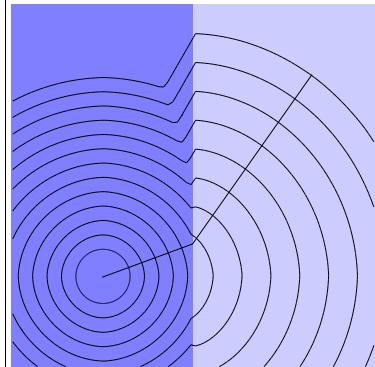
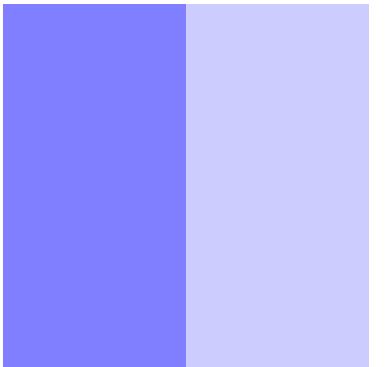


Straight Line from A to B Expanding Front Around A Trace Back to Find Path

But there's a different way to find this path. Imagine a front expanding from point A in all directions. Since it doesn't "cost" you any more to walk in one direction over another, let the front expand with speed 1 in all directions (Using our earlier terminology, let the speed $F(x, y) = 1$, which means that the speed in the normal direction is always unity) That means that the expanding front will be a growing circle around point A, which will

eventually touch point B. Once the front touches point B, you can find the shortest path by starting at point B and proceeding **backwards** along the path that is always perpendicular to the expanding front. If you do this, you'll get the straight line shown in the figure on the right above.

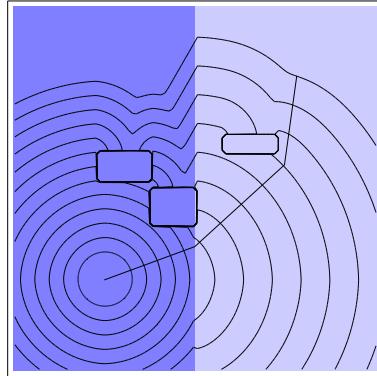
Now, let's imagine that one half of the parking lot is full of snow, and it's slower to walk through snow. Furthermore, you're standing on the snowy side, and your car is on the dry side. In this case, the "shortest" path (that is, the one that takes the least time), as shown in the figure on the right, is not always a straight line. But we can still use our front propagation technique: we expand a front around point A, only this time the front expands faster when it is on dry pavement than it does over the snow. To build this into our front propagation problem, we let the speed $F(x, y)$ be $1/2$ if the point (x, y) is on the snow, and 1 if the point (x, y) is on dry pavement. Once the front hits the car, we again trace backwards from B to A, always going perpendicular to the front, and construct the shortest (in time) path.



Left=snow, Right=dry Expand Front Around A: Trace Back to Find Path

Now, let's add other cars to the lot. We can represent those cars as places where the speed function $F(x, y) = 0$; this means that it takes forever to walk through a car. Again, we solve our front propagation problem, trace back, and construct the optimal path. The key here is to have an algorithm which allows the evolving interface to split into two or more fingers, and then merge back together.

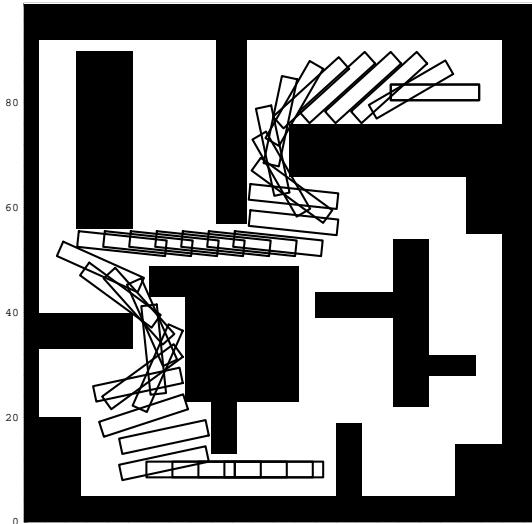
And finally, suppose you are carrying a ladder; this means that you must angle the ladder between cars, where some orientations fit while others do not. In terms of our evolving interface perspective, this means that we will add an extra dimension to the problem, and now the speed function F depends



Optimally Efficient Path Around Obstacles

on three variables: two for position (x, y) and one for the orientation θ of the ladder.

As illustration, we show the optimal trajectory of a piano around an apartment: here we specify the initial and final configurations (position and angle) of the piano, and then use the fast marching method to compute the optimal trajectory.



Optimal Trajectories for Piano Movers

For further details about the application of level set methods to problems in robotic navigation, see [4].

Shape Recovery in Medical Imaging

Another application of level set methods stems from the need to extract useful anatomical features from medical images such as MRI or CAT scans. While a skilled eye can pick out the desired boundaries from a noisy image, even those delineated by slight changes in image intensity, asking a physician to draw by hand outlines on each scan is both time-consuming and inexact. Automatic “edge detection” has drawn considerable attention, however, educating a piece of software to both ignore noise and avoid introducing non-existent boundaries is quite a challenge. One idea is to look for places where there is a big jump in intensity between neighboring pixels. However, it is hard to pick a good value for the jump; too small and you get extra boundaries; too large and you miss the everything. As an example, given the digital subtraction angiogram (DSA) below, a goal is to extract the outline of the artery.

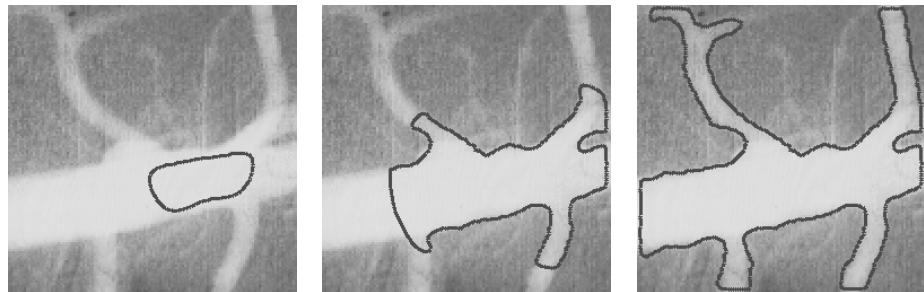


Digital Subtraction Angiogram

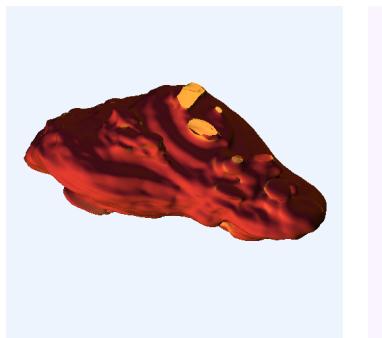
A different strategy comes from a level set approach [5], in which an imaginary front is allowed to propagate from an initial speed. The trick is to adjust the speed F to detect the edge of the shape:

- When the interface passes over places where the image gradient (that is, the change in value from one pixel to the next) is small, we assume we are not near a boundary, and we let the curve expand quickly.
- When the curve passes over places where the image gradient is large, we suspect we are near the boundary, and slow the curve down.

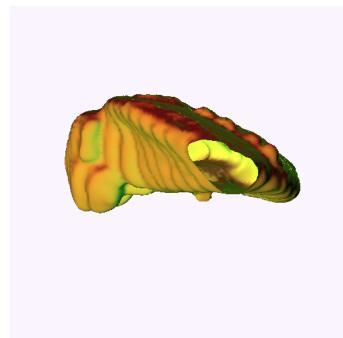
In addition, a little surface tension (in the form of motion by curvature) is included to slightly retard the expanding contours. Again, we use the fast marching version of the level set method to obtain optimal performance; three-dimensional edge detection is equally straightforward.



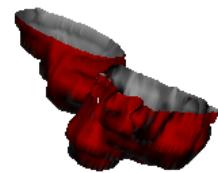
Evolving Front Driven by Function of Image Gradient



3D Liver



3D Spleen

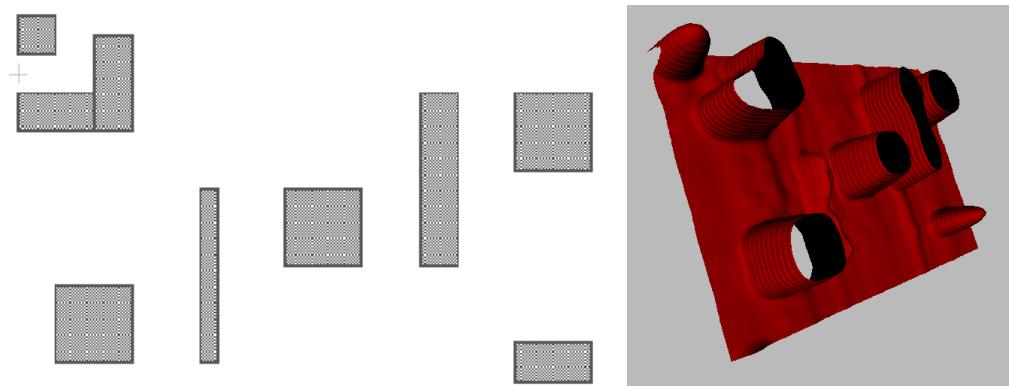


3D Heart Chambers

Semiconductor Manufacturing: Etching, Deposition, and Photolithography

Part of the process of manufacturing a computer chip consists of digging holes in silicon wafers and coating/shaping them with materials. As devices sizes get smaller and smaller, this becomes an increasingly complicated and expensive process. Level set methods (see [1]), have provided computational models provide to track the evolution of the surface profile during various stages of the manufacturing process.

For example, in photolithography, which resembles silk-screening, a pattern mask is placed on a silicon wafer which is then exposed to light: this weakens the ability of the exposed material to resist an etching. During the photoresist development stage, the material is etched away, leaving deep holes that form initial paths in the silicon.

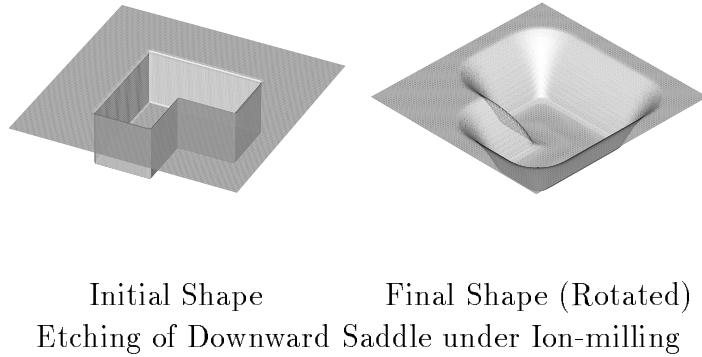


On Left: Masking Pattern/On Right: Evolution of Lithographic Profile using the Fast Marching Method and Technology Modeling Associates' [DEPICT](#)

One of the more complex manufacturing processes involves ion-milling, in which a beam of reactive ions acts like a sandblaster and etches away at a surface. This etching rate depends on, among other things, the angle at which the beam hits the surface. Just like a bad golf swing, the most effective etching angle is not always directly straight down; you can often dig out more grass with a glancing blow from the side. The "yield function" relates how much material is removed as a function of the incoming angle.

Interestingly enough, this process produces beveled, rounded edges in some areas, and sharp cusps in others. While these are difficult problems

to model, they are handled easily by level set methods, due to their reliance on numerical schemes which construct the correct viscosity solution of the differential equation.



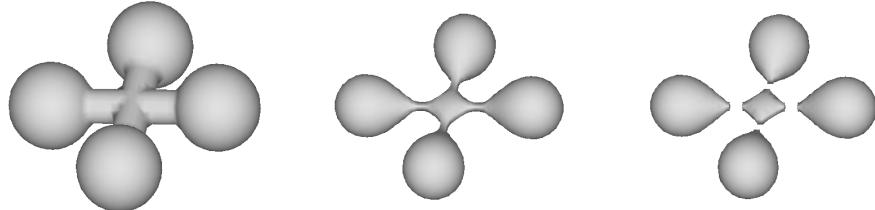
Returning Full Circle: Motion by Curvature

We started this story with motion by curvature, demonstrated the remarkable theorem that all simple closed curves collapse to a point, and then followed with more involved interface motions. Not surprisingly, we haven't scratched the surface of problems that have been tracked using level set methods, including rising bubbles with surface tension, fluid mixtures, combustion and flame propagation, fracture mechanics, and seismic travel time calculations. Nonetheless, the "simple" geometry problem of motion by curvature holds a few more mysteries.

What happens in three dimensions? First, we need a definition for curvature. Standing at point on a surface, the curvature of any path depends on the direction we travel. For example, standing in the center of a horse's saddle, one curvature is positive since it bends up, while the other one bends down and hence is negative. The **mean** curvature of a surface is defined as the average between the biggest and smallest such curvatures.

Now, if we take a sphere and let it collapse under its mean curvature, once again, by symmetry the sphere must collapse to a point. But for more complex surfaces, this is not true. A dumbbell, if it has a narrow enough

handle, will split into two pieces! In the figure below (see [2]), a single surface splits into five pieces before each one collapses to a point.



Collapse of a Two-handled Dumbbell

This is not just a mathematical oddity. Since surface tension is proportional to curvature, at the core, this “non-theorem” sheds light on why drops of fluid, when perturbed, often break into multiple parts.

Finally, let’s come full circle, and return to our example of a simple closed curve collapsing under its curvature. Grayson’s theorem proves that all such curves shrink to a single point: but which point? For circles, ovals, and similarly symmetric curves, the location of the shrinkage spot can be deduced.

Beyond that, for general curves, no one knows.

Further Information and Bibliography

More information about level set methods and fast marching methods may be found in a recent book “Level Set Methods” [8]. A web page devoted to the topic, complete with movies, interactive java applets, and discussions of applications in computer vision, materials, sciences, fluid mechanics, and geometry, may be found at http://math.berkeley.edu/~sethian/level_set.html.

This work has been supported by the Applied Mathematics Subprogram of the Office of Energy Research under contract DE-AC03-76SF00098, and Division of Mathematical Sciences as the National Science Foundation. The author would also like to thank Barry Cipra for his help with this article.

Bibliography

- [1] Adalsteinsson, D., and Sethian, J.A., *A Unified Level Set Approach to Etching, Deposition and Lithography I, and II*, J. Comp. Phys., 120, 1, pp. 128-144, 1995, and 122, 2, pp. 348-366, 1995.
- [2] Chopp, D.L., and Sethian, J.A., *Flow Under Curvature: Singularity Formation, Minimal Surfaces, and Geodesics*, Jour. Exper. Math., 2, 4, pp. 235-255, 1993.
- [3] Grayson, M., *The heat equation shrinks embedded plane curves to round points*, J. Diff. Geom., Vol. 26, 285, 1987.
- [4] Kimmel, R., and Sethian, J.A., *Fast Marching Methods for Robotic Navigation with Constraints*, Center for Pure and Applied Mathematics Report, Univ. of California, Berkeley, submitted for publication, IEEE Transactions on Robotics, 1996.
- [5] Malladi, R., and Sethian, J.A., *A Unified Approach to Noise Removal, Image Enhancement, and Shape Recovery*, IEEE Trans. on Image Processing, 5, 11, 1554-1568, 1996; see also Malladi, R., Sethian, J.A., and Vemuri, B.C., *Shape Modeling with Front Propagation: A Level Set Approach*, IEEE Trans. on Pattern Analysis and Machine Intelligence, 17, 2, pp. 158-175, 1995.
- [6] Osher, S., and Sethian, J. A., *Fronts propagating with curvature dependent speed: Algorithms based on Hamilton-Jacobi Formulations*, Jour. Comp. Phys., Vol. 79, pp. 12-49, 1988.
- [7] Sethian, J.A., *Curvature and the evolution of fronts*, Commun. in Math. Physics, Vol. 101, pp. 487-499, 1985.
- [8] Sethian, J.A., *A Fast Marching Level Set Method for Monotonically Advancing Fronts*, Proc. Nat. Acad. Sci., 93, 4, 1996.
- [9] Sethian, J.A., *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision, and Materials Science*, Cambridge University Press, 1996.
- [10] Weyl, H., *The Philosophy of Mathematics and Natural Science*, Atheneum, New York, 1963.