Fast approximate curve evolution

James Malcolm[1] Yogesh Rathi[2] Anthony Yezzi[1] Allen Tannenbaum[1]

[1]Georgia Institute of Technology, Atlanta, GA
[2]Brigham and Women's Hospital, Boston, MA

ABSTRACT

The level set method for curve evolution is a popular technique used in image processing applications. However, the numerics involved make its use in high performance systems computationally prohibitive. This paper proposes an approximate level set scheme that removes much of the computational burden while maintaining accuracy.

Abandoning a floating point representation for the signed distance function, we use the integral values to represent the interior, zero level set, and exterior. We detail rules governing the evolution and maintenance of these three regions. Arbitrary energies can be implemented with the definition of three operations: initialize iteration, move points in, move points out.

This scheme has several nice properties. First, computations are only performed along the zero level set. Second, this approximate distance function representation requires only a few simple integer comparisons for maintenance. Third, smoothness regularization involves only a few integer calculations and may be handled apart from the energy itself. Fourth, the zero level set is represented exactly removing the need for interpolation off the interface. Lastly, evolution proceeds on the order of milliseconds per iteration using conventional uniprocessor workstations.

To highlight its accuracy, flexibility and speed, we demonstrate the technique on standard intensity tracking and stand alone segmentation.

Keywords: Level set methods, active contours, image segmentation real time tracking

1. INTRODUCTION

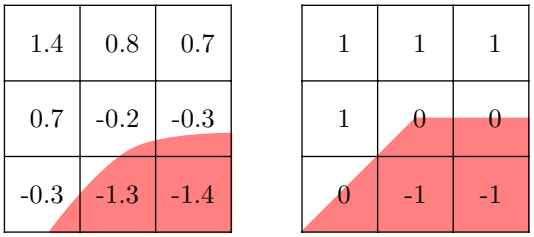
The level set method is a popular technique used in image processing applications. Casting the problem in an energy-based formulation, gradient descent is performed. However, the numerical considerations in this approach add a significant burden to stable and accurate implementations. For example, care must be taken in the choice of differencing scheme, maintaining the signed distance function, and interpolating values off the underlying grid. Such considerations introduce significant computational overhead into the technique.

Several techniques have been proposed to address these drawbacks. Since only the zero level set is of interest, significant performance improvement was initially achieved by reducing the domain of computation to the area immediately surrounding this interface; however, the same numerical considerations must be applied to this reduced domain.[1,2] Alternatively, changing the underlying distance function representation from floating point to binary allows for ignoring the maintenance of the distance function; however, solutions may contain inaccuracies and computation is still performed on the entire domain.[3] Another alternate representation is to maintain a list of points just inside and just outside the interface and move according to the sign of the force calculations; however, this necessitates interpolation of the interface and computation of the force twice along the interface.[4]

This paper introduces an approximate level set scheme that removes much of the computational burden while maintaining accuracy of the solution. Abandoning a floating point representation for the signed distance function, we use the integral values {-1, 0, 1} to represent the interior, zero level set, and exterior, respectively. We detail rules governing the evolution and maintenance of these three regions. Also, we show that arbitrary energies can be implemented with the definition of three operations: initialize iteration, move points in, move points out.

Corresponding author: James Malcolm (malcolm@gatech.edu, www.ece.gatech.edu/∼malcolm)

Real-Time Image Processing 2008, edited by Nasser Kehtarnavaz, Matthias F. Carlsohn,
Proc. of SPIE-IS&T Electronic Imaging, SPIE Vol. 6811, 68110L, © 2008 SPIE-IS&T · 0277-786X/08/$18



(a) Floating point (b) Discrete

Figure 1. An example of an implicit surface on a 3x3 grid using both floating point and approximate discrete representations. Interior regions are shaded. The floating point representation resolves to sub-pixel accuracy while the discrete version approximates the interface as crossing the center of each pixel.

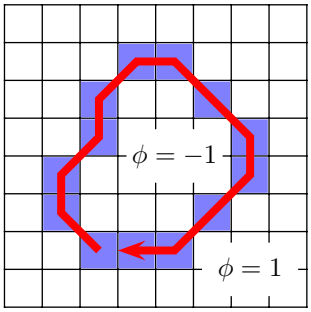


Figure 2. Discrete approximation of signed distance function. Inside ($\phi = -1$) and outside ($\phi = 1$) regions are separated by interface points ($\phi = 0$, *blue*). Implicit ordered curve is indicated *(red)*

.

This scheme has several nice properties. First, computations are only performed at the zero level set instead of on the entire domain or even a narrow band. Second, this approximate distance function representation requires only a few simple integer comparisons for maintenance. Third, smoothness regularization involves only a few integer calculations and may be handled apart from the energy itself. Third, the zero level set is represented exactly removing the need for interpolation off the interface. Fourth, points comprising the zero level set are maintained in order automatically allowing for curve parameterized flows. Lastly, evolution proceeds on the order of milliseconds per iteration using conventional uniprocessor workstations.

2. PROPOSED ALGORITHM

This section describes the discrete algorithm for evolution and extensions to incorporate smoothing and improve speed. Each iteration of evolution involves the core routines for propagation and cleanup and the user-provided callback routines for force computation and bookkeeping as points cross the interface. Pseudo code listings are provided: the core loop in Procedure 1, propagation in Procedure 2, and cleanup in Procedure 3. We now give a brief discussion of the overall algorithm before covering those core and user-provided routines. Section 3 provides definitions of user-provided routines for various energies.

To approximate the signed distance function ϕ, this paper uses a uniform 2D grid with integer values {-1, 0, 1} to represent the interior, zero interface, and exterior regions, respectively. Additionally, we assume the curve to be closed, the points of which are maintained as an ordered list. Note that we use the terms "points" and "pixels" interchangeably. Figure 2 illustrates such a setup.

2.1 Mechanics of evolution

The core algorithm is listed in Procedure 1. Each iteration of evolution proceeds in two phases: contraction and dilation. This two phased evolution was developed to address a problem that develops upon convergence due to the approximation of the interface crossing at the pixel center. Suppose the curve is to truly settle slightly off the center of a particular grid pixel, hence the force will be slightly nonzero to correct for the approximated midpoint

Procedure 1 Main algorithm

for each iteration **do**
{*Contraction*}
Callback: initialize iteration
Restrict to contraction (only allow positive forces)
Propagate (Procedure 2)
Cleanup (Procedure 3)
Callback: move points in and out

{*Dilation*}
Callback: initialize iteration
Restrict to contraction (only allow negative forces)
Propagate (Procedure 2)
Cleanup (Procedure 3)
Callback: move points in and out
end for

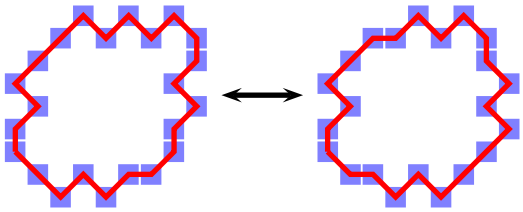


Figure 3. In an unfazed approach, jagged interfaces may develop when the interface oscillates upon convergence. Notice that the shape and position of the interface remain relatively stable despite oscillation.

.

crossing. However, due to our only checking the sign of the speed when propagating (See Procedure 2), the curve is still pushed a full pixel off. In the next iteration, the force compensates sending the pixel back one full pixel and the process repeats. Depending on the arrival of the interface along that line of convergence, this can lead to a fractal interface; with each iteration, points oscillate between sides of that true interface. Figure 3 illustrates such an interface oscillating. In a fully numerical scheme, the curve can settled with subpixel accuracy. To solve this, the phased approach restricts one direction of movement with each phase. This gives the interface a chance to catch up, not allowing the jagged oscillation. Conversely now with the two phased approach, whole smooth sections oscillate which gives a more appropriate result. Section 4.2 describes a faster, stripped down version of the algorithm which involves fewer speed computations but often results in a somewhat jagged interface.

The algorithm for curve propagation is listed in Procedure 2. For each point with nonnegative force it pushes the curve to its immediate neighbors in the appropriate direction. This is where the curve maintains its order (if desired) as new neighbor points are added to the interface. Figures 4 and 5 illustrate the process of determining neighbors and propagating in each phase. Note that after dilation, it is important to drop unnecessary points to maintain a minimal interface so artifacts do not develop. With a minimal interface, we need only look to the four neighbors of any point when making decisions during evolution.

The final core routine for maintaining a minimal interface is listed in Procedure 3. Points along the interface are considered unnecessary if, in addition to touching other interface points, they only touch interior or only touch exterior points. Conversely, points are considered necessary if they touch both interior and exterior points. Figure 6 illustrates cleanup of the contracted interface in Figure 5.

2.2 Energy specific callbacks

Having described the core evolution mechanics, it remains to discuss the user-provided callbacks for force computation and bookkeeping as points cross the interface. This is the energy specific aspect of the curve evolution.

Procedure 2 Perform one iteration of curve propagation

```
for each point x on curve do
  if nonzero force then
    if negative force then
      φ(x) ← 1
    else
      φ(x) ← −1
    end if
    Drop x from interface
    for each neighbor y of x having opposite sign as φ(x) do
      φ(y) ← 0
      Insert along interface (maintaining curve order)
    end for
  end if
end for
```

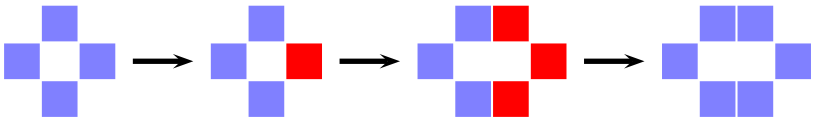


Figure 4. Curve dilating outward: initial, point to move, outward neighbors, final *(left to right)*.

We have found the above evolution algorithm to be generic and applicable to a wide range of curve evolution energies.

At every iteration, the user must compute the force, only the sign of which matters. The user is provided with the underlying ϕ and ordered curve points. Notice that since ϕ only describes the zero level set, it only makes sense to compute (approximate) derivatives $\nabla\phi$ and the force only along the interface. Since the zero level set is indicated explicitly, there is no need for interpolation onto the interface.

Additionally at each iteration, the user is provided an opportunity to adjust regional statistics, etc., in response to points crossing the interface, *i.e.* interface points with $\phi = 0$ moving in to become $\phi = -1$ or moving out to become $\phi = 1$.

3. EXAMPLE ENERGIES

In order to implement an energy, we must define callback routines for force computation and bookkeeping. To illustrate formulating these callbacks, we briefly examine two types of intensity-based segmentation energies found in the literature.

3.1 Mean intensity separation

One of the most common segmentation strategies involves separating the mean intensity of two regions.[5,6] Assuming $\phi < 0$ to be the interior and using the Heaviside function $H(\cdot)$ to describe support, a typical continuous representation of this energy might be:

$$E = \int (I(x) - v)^2 H(\phi(x)) + (I(x) - u)^2 H(-\phi(x)) dx, \tag{1}$$

where u and v are the mean intensities of inside and outside, respectively. A simple form of its gradient is

$$\nabla E = \delta(\phi) \left[(I(x) - v)^2 - (I(x) - u)^2 \right]. \tag{2}$$

Force computation simply involves computing this expression at each point along the interface. Further, since we are computing it directly on the interface where ϕ is defined to be zero, we can ignore $\delta(\phi)$. Bookkeeping

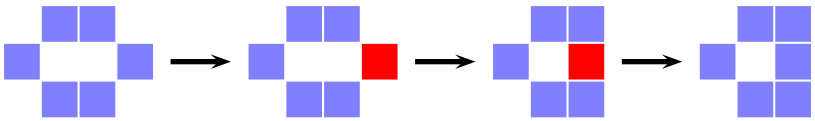


Figure 5. Curve contracting inward: initial, point to move, inward neighbors, final *(left to right)*. The result includes unnecessary points violating minimal interface principle; these are removed during cleanup (see Figure 6).

Figure 6. During cleanup, unnecessary interface points *(red)* are detected and removed to ensure minimal interface.

then simply involves adjusting the regional means upon the movement of pixels, a computation that can be done recursively. For example, for a point moving from inside to outside at time t use

$$u_{t+1} = \frac{u_t A_u - I(x)}{A_u - 1} \qquad \text{and} \qquad v_{t+1} = \frac{v_t A_v + I(x)}{A_v + 1}, \tag{3}$$

where A_u and A_v are the areas of inside and outside, respectively.

3.2 Distribution separation

Another common energy for separating regions involves maximizing the Bhattacharyya distance between two intensity distributions.[7,8] A typical continuous representation of this energy might be:

$$E = d^2(\mathbf{p}, \mathbf{q}) = \int_{\mathcal{Z}} \sqrt{\mathbf{p}(z)\mathbf{q}(z)}dz, \tag{4}$$

where $\mathcal{Z}$ is the set of possible intensities and $\mathbf{p}, \mathbf{q}$ are the distributions for inside and outside, respectively. The gradient of this expression is found to include nested integrals:

$$\nabla E = \underbrace{\frac{d^2(\mathbf{p}, \mathbf{q})}{2}\left(\frac{1}{A_u} - \frac{1}{A_v}\right)}_{\text{global}} + \underbrace{\frac{\delta(\phi)}{2}\int_{\mathcal{Z}} K(z - I(x))\left(\frac{1}{A_v}\sqrt{\frac{\mathbf{p}(z)}{\mathbf{q}(z)}} - \frac{1}{A_u}\sqrt{\frac{\mathbf{q}(z)}{\mathbf{p}(z)}}\right)dz}_{\text{local}}, \tag{5}$$

where $K(\cdot)$ is for a kernel density estimate. Since we are already working in an approximate framework, we may be able to take the liberty of further approximations to implement this gradient. For example, if we can assume that the distributions are left unsmoothed we are left with

$$\nabla E = \frac{d^2(\mathbf{p}, \mathbf{q})}{2}\left(\frac{1}{A_u} - \frac{1}{A_v}\right) + \frac{\delta(\phi)}{2}\left(\frac{1}{A_v}\sqrt{\frac{\mathbf{p}(z)}{\mathbf{q}(z)}} - \frac{1}{A_u}\sqrt{\frac{\mathbf{q}(z)}{\mathbf{p}(z)}}\right). \tag{6}$$

As an implementation note, notice there is a global term and a local term that depends only on possible intensities encountered, both of which can be precomputed each iteration. Computing the speed along the interface merely entails looking up the local value for each intensity encountered and adding the global term.

4. EXTENSIONS

Here we demonstrate two simple extensions to the basic algorithm. The first extension is smoothing using integer Gaussian kernels similar to,[4] the second is an additional speed enhancement at the expense of a rougher interface.

4.1 Smoothing

Typically, smoothing is incorporated as an additional term in the energy that penalizes curve length. This penalty takes the form of a Laplacian of ϕ the evolution of which equates to Gaussian filtering the signed distance function.[4] They propose to use the Gaussian filter response similar to the force thereby moving points if the sign of $\phi(x)$ differs from the filter response. This can be decoupled from the energy into a subsequent phase of evolution shown in Procedure 4. Here, the main algorithm alternates between the two phased evolution and smoothing.

Procedure 3 Drop unnecessary points to ensure minimal interface

```
for each point x on curve do
  if neighbors' include only interior and interface then
    φ(x) ← 1
    Drop x from interface
  end if
  if neighbors' include only exterior and interface then
    φ(x) ← −1
    Drop x from interface
  end if
end for
```

Procedure 4 Main algorithm with smoothing.

```
for each iteration do
  for each iteration of evolution do
    Callback: initialize iteration
    Restrict to contraction (only processes positive forces)
    Propagate, cleanup (Procedures 2, 3)
    Callback: move points in and out

    Callback: initialize iteration
    Restrict to contraction (only processes negative forces)
    Propagate, cleanup (Procedures 2, 3)
    Callback: move points in and out
  end for

  for each iteration of smoothing do
    Compute integer kernel along interface (use as force)
    Propagate, cleanup (Procedures 2, 3)
    Callback: move points in and out
  end for
end for
```

4.2 Further speed increase

As mentioned in Section 2.1, portions of the interface can become jagged as they oscillate upon convergence (see Figure 3). The two phased approach addressed the jagged interface. However, if the smoothness of the interface is not critical to the application, the algorithm can be further simplified to one phase with no restriction on direction (see Procedure 5). This approach can be significantly faster with monotonically advancing fronts. Under such circumstances the phased approach would discard the work of every other force computation.

5. EXPERIMENTS

Intensity segmentation was performed using both mean and full distribution separation. Measuring the time to complete each iteration (after the image is loaded), speeds of roughly 10 KHz were achieved in some cases.

Figure 7 shows selected frames from one of the tracking sequences used. Here we used mean separation for segmentation without smoothing. To investigate the tradeoffs resulting from the approximation, we attempted to segment the classic zebra image found in the literature.[7] Figure 8 shows the final segmentation using the Bhattacharyya distance with smoothing. Compared to the results found in the literature, the only discrepancy is near the hind end where two of the strips are not captured. Lastly, Figure 9 demonstrates segmentation where topology changes. Here, the background is split to include the truck window, and so the interface is no longer a single sorted curve. The discrete nature of this algorithm makes it suitable for discrete topology preserving techniques.[9]

Procedure 5 Faster unfazed version without smoothing.

```
for each iteration do
  for each iteration of evolution do
    Callback: initialize iteration
    Propagate, cleanup (Procedures 2, 3)
    Callback: move points in and out
  end for
end for
```

Figure 7. Street sequence showing path of target.

6. CONCLUSION

The proposed method is a simple form of curve evolution suitable for high performance surface propagation where subpixel accuracy is unnecessary.

As with many techniques, the proposed technique trades accuracy for speed. One of the largest tradeoffs is in the unit propagation. For example, this may affect applications that are highly sensitive to the rate of surface propagation. This was found to occur in a global energy where one portion of the curve evolving faster than another rapidly influencing the global statistics and in doing so retarded the speed of the slower.[10] Here, under the discrete evolution both portions moved equally fast and so influenced the global statistics equally so sometimes leading to incorrect segmentation. Unit propagation also caused the problem of jagged interfaces upon convergence which resulted in the two phased approach. The faster version of Procedure 5 is most useful if when computing the energy you detect for subpixel convergence and zero out forces; this can eliminate oscillation.

Faster yet, if the requirement for an ordered interface is removed, evolution becomes highly parallelized, so much so that it may be possible to reformulate it in terms of cellular automata.

Both C and Matlab versions of this algorithm are provided online.*

ACKNOWLEDGMENTS

This work was supported in part by grants from NSF, AFOSR, ARO, MURI, as well as by a grant from NIH (NAC P41 RR-13218) through Brigham and Women's Hospital. This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149. Information on the National Centers for Biomedical Computing can be obtained from `http://nihroadmap.nih.gov/bioinformatics`. The authors would also like to thank Starbucks coffee providing much of the motivation behind this paper's development.

REFERENCES

1. D. Adalsteinson and J. A. Sethian, "A fast level set method for propagating interfaces," *J. of Computational Physics* **118**(2), pp. 269–277, 1995.
2. R. Whitaker, "A level-set approach to 3D reconstruction from range data," *Int. J. of Computer Vision* **29**(3), pp. 203–231, 1998.
3. F. Gibou and R. Fedkiw, "A fast hybrid k-means leve set algorithm for segmentation," in *Int. Conf. Statistics and Mathematics*, pp. 281–291, 2005.

*`www.ece.gatech.edu/∼malcolm`

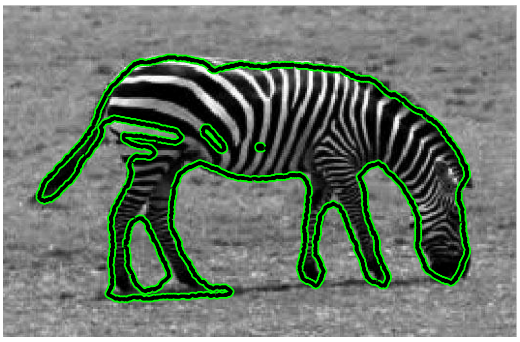


Figure 8. Classic zebra image segmented using the Bhattacharyya distance.[7] The final result is similar to that found in the literature except for a few hind end stripes.

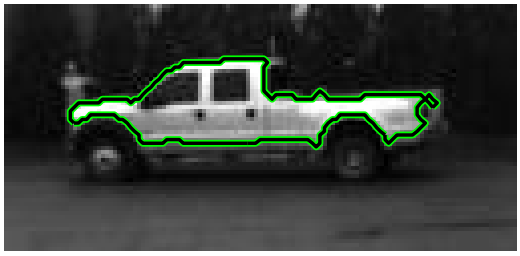


Figure 9. Truck body segmented with mean intensity without smoothing. Notice that the topology breaks up around the window demonstrating the natural ability of the level set method to split and merge.

4. Y. Shi and W. Karl, "A fast level set method without solving PDEs," in *Int. Conf. on Acoustics, Speech, and Signal Processing*, pp. 97–100, 2005.
5. T. Chan and L. Vese, "Active contours without edges," *Trans. on Image Processing* **10**(2), pp. 266–277, 2001.
6. A. Yezzi, A. Tsai, and A. Willski, "A statistical approach to snakes for bimodal and trimodal imagery," in *Int. Conf. on Computer Vision*, pp. 898–903, 1999.
7. Y. Rathi, O. Michaeilovich, J. Malcolm, and A. Tannenbaum, "Seeing the unseen: Segmenting with distributions," in *IASTED Conf. on Signal and Image Processing*, 2006.
8. T. Zhang and D. Freedman, "Tracking objects using density matching and shape priors," in *Int. Conf. on Computer Vision*, 2003.
9. X. Han, C. Xu, and J. Prince, "A topology preserving level set method for geometric deformable models," *Trans. on Pattern Analysis and Machine Intelligence* **25**(6), 2003.
10. S. Dambreville, A. Yezzi, M. Niethammer, and A. Tannenbaum, "A variational framework combining level-sets and thresholding," in *British Machine Vision Conf.*, 2007.