ECE 6560 Final Project Report
Thickness Computation

Yifei Fan

April 28, 2017

1 Introduction

Thickness, or width, is an indicator for the health of the subject, such as human tissue or pavement cracks. In medical image analysis, measuring the thickness could provide useful information on the functionality of an organ [1], as well as the potential disease [2]. In pavement preservation and management, width is a key factor for pavement distress classification [3].

In this final project, we first introduce and explain the Eulerian PDE approach for computing thickness propose in [4]. Then we experiment with this method on series of synthetic images and try to capture the characteristic of this method. Analysis and discussion will be given in the end.

2 Mathematical Formulation

2.1 Thickness Definition

Previous definition on thickness works well on regular and uniform shapes, however, problems occur at shapes with more complex geometry. In [4], thickness is defined as the length of correspondence trajectory passing through the point by taking account the following concerns:

- The trajectories shall not intersect
- The trajectories shall be orthogonal to both boundaries
- The trajectories should be as short as possible to make it efficient

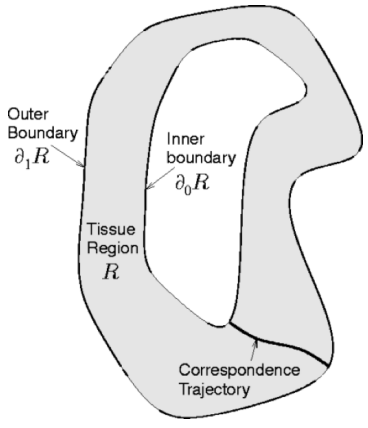

Figure 1: Thickness defined using correspondence trajectory, from [4]

These correspondence trajectories connect the inner boundary $\partial_0 R$ and the outer boundary $\partial_1 R$ in a bijective manner, meaning that there is a one to one correspondence between the points on inner and outer boundaries.

2.2 Eulerian PDE Approach

For a point x within the region R, let $L_0(x)$ denote the length of the correspondence trajectory from the point x to inner boundary $\partial_0 R$, and $L_1(x)$ denote the length from x to outer boundary $\partial_1 R$. Then the thickness W of point x can be simply expressed as $W(x) = L_0(x) + L_1(x)$.

To solve $L_0(x)$ and $L_1(x)$, we first construct a unit vector field that coincides with the tangent vectors of the correspondence trajectories. We adopt the normalized gradient of the unique harmonic function suggested by [5], and obtain the harmonic function u by solving the following Laplace equation

$$\Delta u = 0 \tag{1}$$

with the Dirichlet boundary conditions

$$u(\partial_0 R) = 0 \quad \text{and} \quad u(\partial_1 R) = 1 \tag{2}$$

where Δ is the Laplace operator. Then, the tangent field $\vec{T}$ is calculated via

$$\vec{T} = \frac{\nabla u}{\|\nabla u\|} \tag{3}$$

where ∇ is the gradient operator According to differential geometry, L_0 and L_1 must satisfy the following first order PDEs.

$$\nabla L_0 \cdot \vec{T} = 1, \quad L_0(\partial_0 R) = 0 \tag{4}$$

$$\nabla L_1 \cdot \vec{T} = -1, \quad L_0(\partial_1 R) = 0 \tag{5}$$

The whole procedure is given below.

Algorithm 1 Eulerian PDE Approach

procedure COMPUTE-THICKNESS
  Solve $\Delta u = 0, u(\partial_0 R) = 0, u(\partial_1 R) = 1$
  Compute $T = \frac{\nabla u}{\|\nabla u\|}$
  Solve $\nabla L_0 \cdot T = 1, L_0(\partial_0 R) = 0$ and $\nabla L_1 \cdot T = -1, L_0(\partial_1 R) = 0$
  return $L_0 + L_1$

3 Numerical Implementation

Here we present all the discretization in 2-D to make it more concise, it should be easy to generalize them to 3-D.

3.1 Harmonic Function and the Tangent Field

There are many algorithms for solving (1), in the interest of time, we choose the easiest one. Since Laplace equation describe the steady state of the linear heat equation, we can obtain u with a diffusion process whose initial state is given as (2) and $u(R) = 0.5$. In fact, we can choose any initial value between 0 and 1 for the region.

The diffusion proceeds according to the linear heat equation.

$$u_t = \Delta u = u_{xx} + u_{yy} \tag{6}$$

Here we discretize the PDE with Taylor series approach. For those points on the boundary, we can pad its "outward" neighbor (the one that does not belong to region R) with the same value. The discretization is given below.

$$u_{xx} = \frac{u(x + \Delta x, y) - 2u(x, y) + u(x - \Delta x, y)}{(\Delta x)^2} \tag{7}$$

$$u_{yy} = \frac{u(x, y + \Delta y) - 2u(x, y) + u(x, y - \Delta y)}{(\Delta y)^2} \tag{8}$$

We can write out the update formula as

$$u(x, y, t + \Delta t) = u(x, y, t) + \Delta t \ (u_{xx} + u_{yy}) \tag{9}$$

where u_{xx} and u_{yy} is given by (8) and (9). Here we set the default value for Δx and Δy as 1. With this setting, the Courant–Friedrichs–Lewy (CFL) condition of this 2D linear heat equation is $\Delta t \leq \frac{1}{4}(\Delta x)^2 = \frac{1}{4}$.

The corresponding tangent field $\vec{T}$ is then computed using (3), with the symmetric central difference as well. Since $\vec{T}$ shows up as a constant in the PDE and is not something we try to solve for, it does not matter which difference scheme to use as long as it is meaningful.

$$u_x(x, y) = \frac{u(x + \Delta x, y) - u(x - \Delta x, y)}{2}$$

$$u_y(x, y) = \frac{u(x, y + \Delta y) - u(x, y - \Delta y)}{2}$$

$$T(x, y) = \left(\frac{u_x(x, y)}{\sqrt{u_x^2(x, y) + u_y^2(x, y)}}, \frac{u_y(x, y)}{\sqrt{u_x^2(x, y) + u_y^2(x, y)}} \right) \tag{10}$$

Since u is a harmonic function which satisfies the maximum and minimum principle, there is no need to worry about the denominator being 0.

3.2 Upwind Differencing for the First Order PDEs

The two first order PDEs (4) and (5) have a similar form as the transport PDE, as we can treat one of the spacial variable in (4) and (5) as the temporal variable in transport PDE. The CFL condition for transport equation requires us to use the "upwind" difference scheme, so that the value of L (either L_0 or L_1) only depends on the values of L in the backward direction along the characteristic passing through that point. One good thing about first order PDEs is that they do not intersect with each other, which ease the consideration for shocks and entropy conditions.

We keep the same naming convention as the original paper [4]. The standard notation of backward and forward difference is given below.

$$D_x^- L[i, j] = \frac{L[i, j] - L[i - 1, j]}{h_x}, \qquad D_x^+ L[i, j] = \frac{L[i + 1, j] - L[i, j]}{h_x}$$

$$D_y^- L[i, j] = \frac{L[i, j] - L[i, j - 1]}{h_y}, \qquad D_y^+ L[i, j] = \frac{L[i, j + 1] - L[i, j]}{h_y}$$

For the point $[i, j]$, the upwind direction is given by $-T[i, j]$, thus PDE (4) can be discretized as

$$\begin{aligned} 1 = & T_x[i, j] \left\{ \begin{array}{ll} D_x^- L_0[i, j], & -T_x[i, j] < 0 \\ D_x^+ L_0[i, j], & \text{otherwise} \end{array} \right\} \\ & + T_y[i, j] \left\{ \begin{array}{ll} D_y^- L_0[i, j], & -T_y[i, j] < 0 \\ D_y^+ L_0[i, j], & \text{otherwise} \end{array} \right\} \end{aligned} \tag{11}$$

Assuming $h_x = h_y = 1$, we solve (11) and obtain the update formula.

$$L_0[i, j] = \frac{1 + |T_x| L_0[i \mp 1, j, k] + |T_y| L_0[i, j \mp 1, k]}{|T_x| + |T_y|} \tag{12}$$

where

$$i \mp 1 = \begin{cases} i-1, & T_x[i,j] > 0 \\ i+1, & T_x[i,j] < 0 \end{cases}$$

$$j \mp 1 = \begin{cases} j-1, & T_y[i,j] > 0 \\ j+1, & T_y[i,j] < 0 \end{cases}$$

Similarly, we can obtain the update formula for L_1. Notice that the upwind direction for L_1 is defined by $\vec{T}$ this time.

$$L_1[i,j] = \frac{1 + |T_x| L_1[i \pm 1, j, k] + |T_y| L_1[i, j \pm 1, k]}{|T_x| + |T_y|} \tag{13}$$

where

$$i \pm 1 = \begin{cases} i+1, & T_x[i,j] > 0 \\ i-1, & T_x[i,j] < 0 \end{cases}$$

$$j \pm 1 = \begin{cases} j+1, & T_y[i,j] > 0 \\ j-1, & T_y[i,j] < 0 \end{cases}$$

3.3 Iterative Procedures

We can solve the two linear PDEs in an iterative procedure. In the interest of time, we only implement the simplest one in the paper, the iterated parallel relaxation method.

Algorithm 2 Iterative Relaxation Method

1: **procedure** ITERATIVE-RELAXATION
2: Initialize $L_0 = L_1 = 0$ for all points on the grid
3: Update L_0 and L_1 for points inside the region R, using (12) and (13)
4: Repeat step 2 until the values of L_0 and L_1 converge.

The update process follows a Jacobian style, meaning that the values we obtain within an iteration do not interact with each other as all of them are calculated using the old values from last iteration. Although this could be slower, we can still work simultaneously for L_0 and L_1 , since the update formulas for the two PDEs are designed to look at exact opposite directions.

As suggested by the paper [4], we can get fast convergence by visiting the points in a optimized order, which is similar to the "fast marching method". The algorithm is given as the following. A

Algorithm 3 Ordered Traversal Method

1: **procedure** ORDERED-TRAVERSAL
2: Initialize all grid points in region R as *UNVISITED*
3: Solve L_0 for points next to the inner boundary $\partial_0 R$ and re-tag these points as *VISITED*.
4: Find the *UNVISITED* point with the smallest value of L_0. Pop up the point from the list and re-tag it as *SOLVED*.
5: Update L_0 values using (12) for whichever neighbors of this point that are not yet tagged as *SOLVED*. If any of these neighbors are currently tagged as *UNVISITED*, re-tag them as *VISITED* and add them to the currently list of *VISITED* points.
6: Stop if all points within the region R have been tagged *SOLVED*, else goto step 3

min-heap can be used to keep the list of *VISITED* , but not yet *SOLVED* points. If the value of a point in the heap is updated and become smaller, we should percolate it up. In cases where the upwind neighbor is not yet VISITED and no value is available, we ignore the terms associated to this neighbor in both numerator and denominator of (12) and (13). The update formula can be interpreted as a weighted average among the upwind neighbors, thus we need to completely reduce the impact of

the neighbor if it is not yet available. In another point of view, we can still keep moving in the right direction by following some of the orthogonal component in the tangent field. If we are not careful enough to remove the term in denominator, the value we obtain becomes smaller, which could mess up the order in fast marching.

4 Experiments and Analysis

In this section, we manage to understand the characteristic of our method with several experiments. For simplicity, we use synthetic images which all follow the same convention: green for inner boundary, red for outer boundary, blue for region pixels, black for inside, and gray for outside. The method is implemented in python with Anaconda environment, all materials are accessible at https://github.com/thelittlekid/PDE-thickness

4.1 Qualitative Evaluation

We test the method on different images with different shapes and geometry, sample results can be found at Figure 2, where warmer color is used for larger thickness.

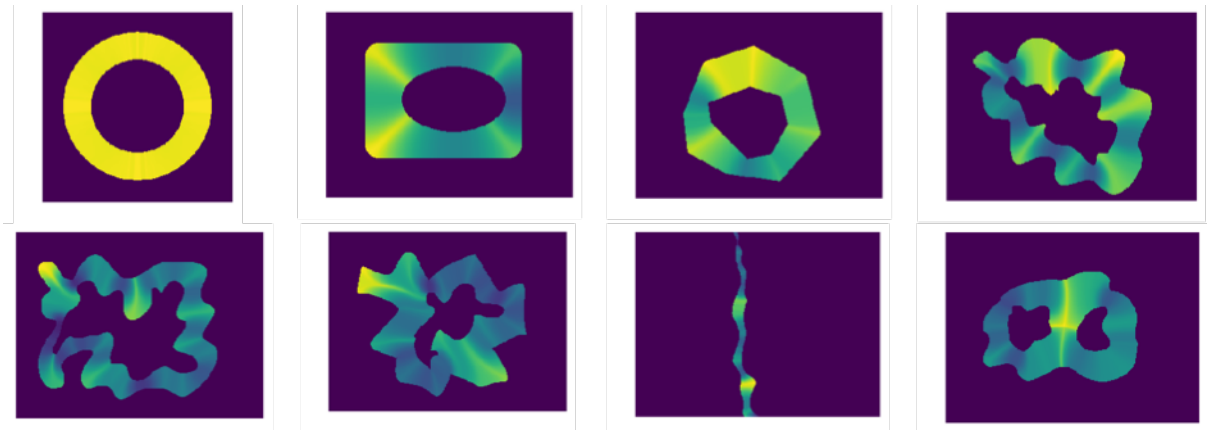

Figure 2: Qualitative results on different images, (warmer for higher thickness)

If we take a careful look at the results, we can find the interesting effect around the corner area, especially when the corner is sharp. The thickness values are higher all of a sudden at sharp corners, which leads to discontinuity. We will explorer more on this in 4.4.

4.2 Solving the Harmonic Function

The solution of the harmonic function u is obtained through a linear heat diffusion process. To save computation time, we assume the steady state is reached once the L_∞ norm of the difference between the results from two adjacent iterations is less than a tiny threshold $\delta = 10^{-8}$. In this section, we test if different initial values and time steps could affect the convergence speed. Figure 3 illustrates the number of iteration required under different initial values and time steps.

According to the test result, setting the initial value to somewhere near the middle point could save some iterations, however, the difference is not quite obvious compared to the absolute number of iteration, especially when accuracy requirement is high, i.e. δ is small. On the other hand, we do benefit by using a larger time step which is allowed by the CFL condition. In a linear heat diffusion (9), a larger time step means that the future value of a pixel depends less on itself and more on its neighbors. At the extreme case, if we pick the maximum time step possible, the future value of a pixel is completely determined by the current value of its four neighbors. The less weight we put on the pixel itself, the faster the diffusion process.

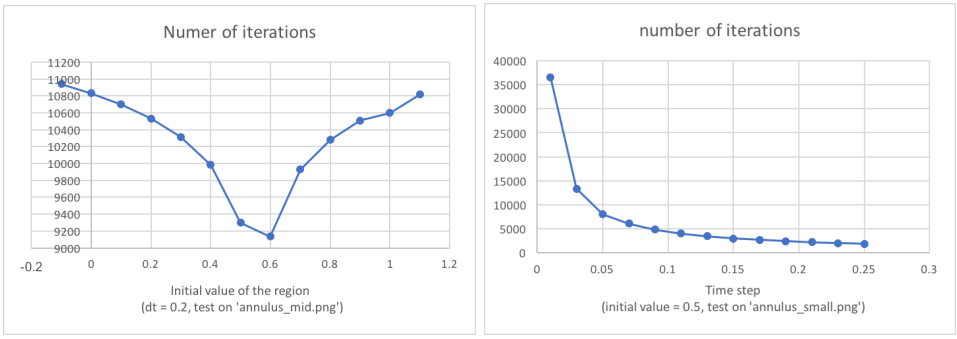

Figure 3: The impact of initial value (left) and time step (right) on convergence speed

4.3 Evolution of the Linear PDEs

As the annulus is almost the only case whose ground truth can be easily obtained, we use *annlus_ mid.png* (with thickness around 36.5) to illustrate the evolutions of the two PDEs. At the end of each iteration, the Frobenius norm of the difference between current thickness and the ground truth is computed. The evolution of such difference is plotted at Figure 4.

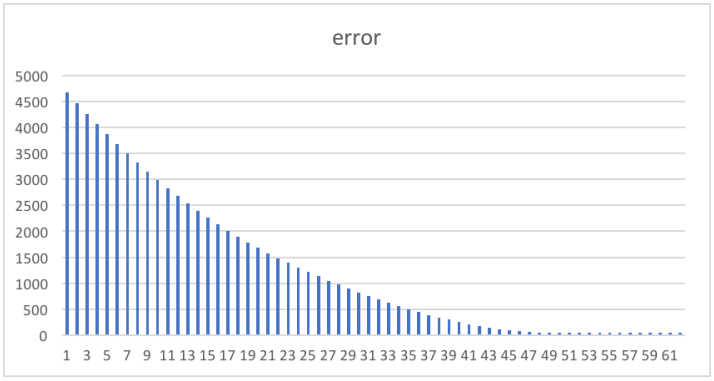

Figure 4: The error vs number of iterations

Based on the plot, the error drops much faster in early iterations. As the computed thickness approaches the true value, the enhancement we get from a single iteration becomes negligible. Therefore, it is reasonable to terminate the iteration at a point where the enhancement from an iteration drops below a threshold.

4.4 Robustness Test

As pointed in 4.1, there might be a discontinuity issue at sharp corners. We argue that this might result from the number of available pixels, i.e. the resolution of the image. The test is performed on a synthetic image (1024*768) as shown in Figure 5. We downsample the image at different rate (from 0.9 to 0.06) with "nearest neighbor" interpolation, compute the thickness, and quantitatively measure the standard deviation of thickness of the region within the yellow box. In addition to the standard deviation, we also compute the thickness gap between the end points of the longest trajectory and the second longest trajectory. However, since the implementation does not trace trajectories, those trajectories and representative points are selected manually.

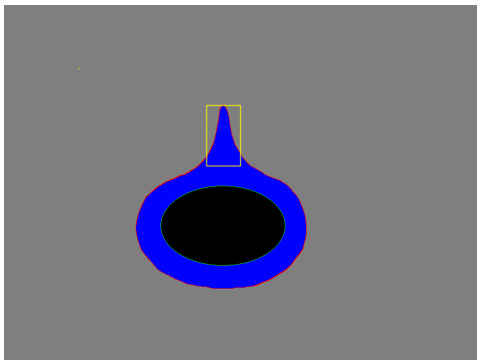

Figure 5: The synthetic image used for robustness test (the yellow box does not belong to the image)

Qualitative results on Figure 5 is given below. Since the trajectories cannot intersect with each other, they have fewer and fewer options as resolution goes down. As a result, the longest trajectory becomes more and more notable.

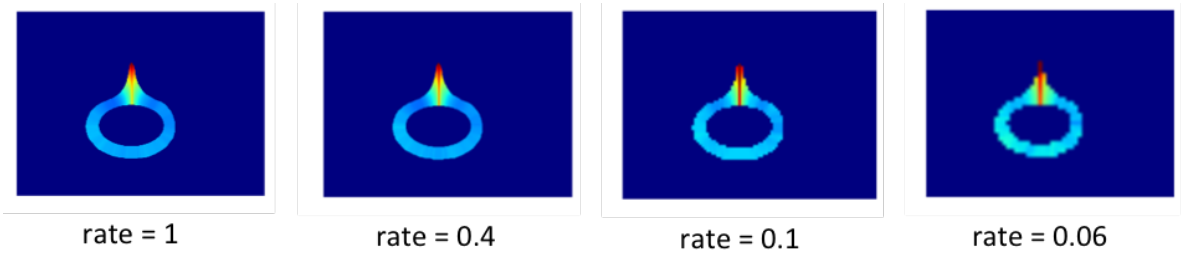

Figure 6: Qualitative results for Figure 5

The quantitative result on standard deviation and thickness difference is shown by Figure 7. The differences in thickness have been normalized to the original scale. The results show that the discontinuity issue becomes more and more severe as downsampling rate keeps dropping below the safety line (0.2 in this test), which indicates that the performance of this Eulerian PDE approach depends on how fine the grid is.

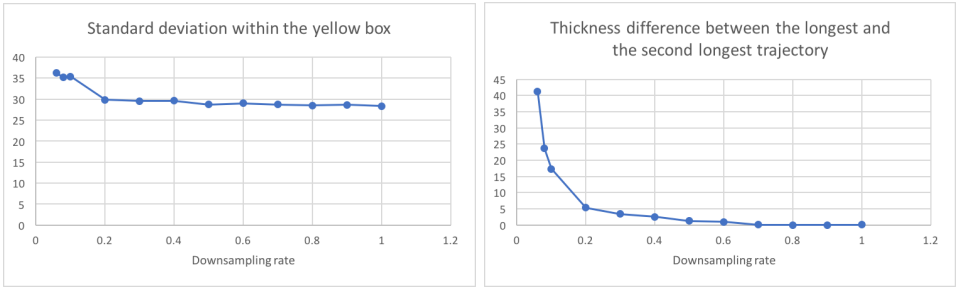

Figure 7: Quantitative results for Figure 5

5 Discussion

The Eulerian PDE approach for computing thickness works quite well on relatively smooth shapes, however, it may encounter discontinuity issue at sharp corners, especially at lower resolutions. This results from the design, which requires the trajectories travel from the one boundary to another without

intersecting each other. On one hand, a more reasonable approximation for thickness/width at the corner might be the radius as shown in Figure 8, but the radius only works for points that falls on the centerline of the region. We still have to come up with a scheme that can correlate all points that share a common trajectory. In this case, the start and end points can fall on the same boundary. On the other hand, lower resolution narrows the options for traveling across the region without intersecting other trajectories, which explains the sharp difference on adjacent paths in the corner.

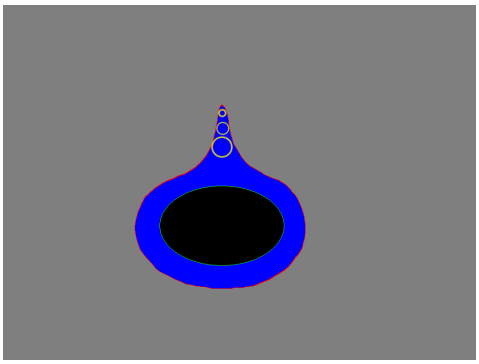

Figure 8: Radius as thickness near the corner

When it comes to solving PDEs, the most straight-forward and easy to implement method can be extremely slow. Although we can save a few iterations by choosing the largest time step possible or a more proper initial state, a more carefully design updating scheme can speed up the process to a large extent. Solving the transport equations with marching scheme does not require a CFL condition as we calculate the exact value of the points each time rather than evolving the solution in a dynamic way. In this case, the upwind neighbor is the one that has already been computed. Unfortunately I was not able to fix all the bugs in my implementation of the ordered traversal algorithm. To build up a clean fast marching scheme would be the first task to complete as it can be quite useful for solving different types of problems.

To further automate the process for crack width calculation, there are two more considerations: one is to detect the boundaries as a continuous curve rather than separated pixels; the other issue is that cracks can intersect each other in various ways, which leads to branches. The desired method need to include the process of determining the departure boundary and destination boundary for each point of interest. Starting from the target point, we may grow a circle until it hits two different boundary points that falls on different side of the target point, after which we can do a binary classification on the boundary points using the distance or connectivity with respect to the two hit points. This could also mitigate the discontinuity issue we found at sharp corners.

References

[1] M. E. DeBakey, *The new living heart*. Adams Media Corporation, 1997.

[2] B. Fischl and A. M. Dale, "Measuring the thickness of the human cerebral cortex from magnetic resonance images," *Proceedings of the National Academy of Sciences*, vol. 97, no. 20, pp. 11050–11055, 2000.

[3] K. H. McGhee, *A Guide to Evaluating Pavement Distress Through the Use of Video Images*. Virginia Department of Transportation, Maintenance Division, 1998.

[4] A. J. Yezzi and J. L. Prince, "An eulerian pde approach for computing tissue thickness," *IEEE transactions on medical imaging*, vol. 22, no. 10, pp. 1332–1339, 2003.

[5] S. E. Jones, B. R. Buchbinder, and I. Aharon, "Three-dimensional mapping of cortical thickness using laplace's equation," *Human brain mapping*, vol. 11, no. 1, pp. 12–32, 2000.