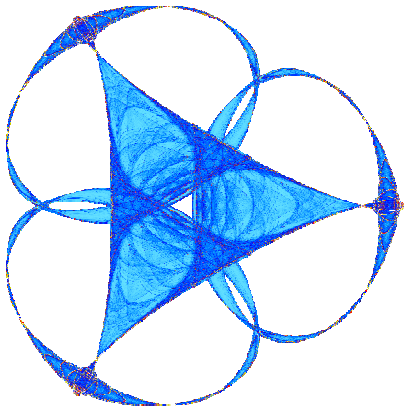
GEOMETRIC AND APPEARANCE MODELING OF VASCULAR STRUCTURES IN CT AND MR

By

Qichuan Bai, Brittan Farmer, Eric Foxall, Xing (Margaret) Fu, Sunnie Joshi, Zhou Zhou

IMA Preprint Series #2392

(May 2012)



INSTITUTE FOR MATHEMATICS AND ITS APPLICATIONS
UNIVERSITY OF MINNESOTA
400 Lind Hall
207 Church Street S.E.
Minneapolis, Minnesota 55455-0436
Phone: 612-624-6066 Fax: 612-626-7370
URL: http://www.ima.umn.edu

Geometric and Appearance Modeling of Vascular Structures in CT and MR

Qichuan Bai, Brittan Farmer, Eric Foxall,
Xing (Margaret) Fu, Sunnie Joshi, Zhou Zhou

August 11, 2011

1 Introduction

Medical imaging devices such as CT and MRI scanners allows doctors to view their patient's internal organs noninvasively. However, it can be difficult to see the structure of organs and tissues in the raw intensity data. Segmentation of the intensity data can be helpful for describing the shape of such objects. Vital Images creates software for visualizing medical images. They have algorithms for determining the centerline of a blood vessel as well as determining the contours of the vessel lumen and vessel wall in two-dimensional slices through the three-dimensional image; see for example Figure 1. The centerline and the contours give a good idea of the vessel's shape, but when meshed directly, they can lead to a rough surface with self-intersections. The primary goal of our project was to give a more complete description of the geometry using the existing segmentation data.

Four approaches are described here. The first three are level set methods, in which the target surface (either the vessel lumen or the outer wall) is viewed as the zeroset of a function defined on a space around the vessel, and the goal is to find such a function. The fourth method is a geodesic method for surface reconstruction; an underlying smooth surface is assumed, and an approximate parametrization is obtained by computing the tangent space at each data point and shooting trajectories along the surface.

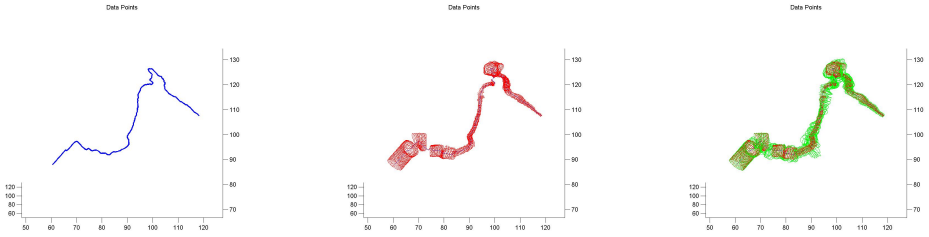


Figure 1: (*Left to right:*) The centerline, lumen, and vessel wall data from Vital Images' current algorithms.

2 The Weighted Minimal Surface Model

Rather than try to determine the connectivity between contours, we can view the given contours as a sparse subset of the desired surface. In [3], Hongkai Zhao and Stanley Osher describe a method for using such data to obtain an implicit representation of the surface using the level set method. This representation allows for dynamic deformation of the surface and the surface can easily be visualized by creating a triangulated surface from an isosurface, e.g. with MATLAB's `isosurface` command. We describe their method in what follows.

One begins with a set of data points $\mathcal{S} = \{x_i \in \mathbb{R}^d, i = 1, \ldots M\}$, which in our case is a set of points evenly sampled on the given contours. The first step in the construction is to create a distance function $d(x) = dist(x, \mathcal{S})$ to the data points. For our computational domain, we shall consider a rectangular grid that contains all the data points. One could explicitly calculate the distance at all grid points, but this could be very computationally expensive. If there are M data points and N grid points, this calculation would require $O(MN)$ computations. Another option is to solve the Eikonal equation

$$|\nabla d| = 1, \quad d(x) = 0, x \in \mathcal{S}.$$

As recommended in [3], we solve this equation using the fast sweeping method, which uses upwinding to propagate distances away from the data points, and uses 2^d Gauss-Seidel iterations in order to propagate this information in all characteristic directions. For example, for an $I \times J$ grid in two dimensions with grid size h, we discretize the Eikonal equation as

$$[(d^h_{i,j} - d^h_{xmin})^+]^2 + [(d^h_{i,j} - d^h_{ymin})^+]^2 = h^2, \quad i = 2, \ldots, I-1, j = 2, \ldots, J-1,$$

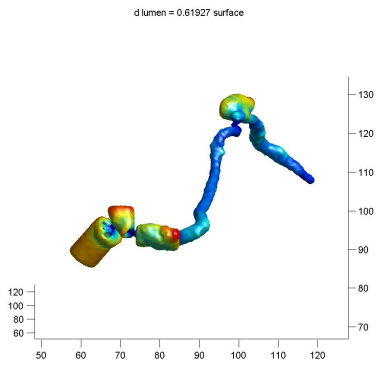


Figure 2: A contour of the distance function to the lumen data.

where $d^h_{xmin} = \min(d^h_{i-1,j}, d^h_{i+1,j})$, $d^h_{ymin} = \min(d^h_{i,j-1}, d^h_{i,j+1})$ and $(x)^+ = \max(x, 0)$. At the boundary, we use one-sided differences as needed. To initialize the distance function, we calculate the exact distance at all grid points in a small neighborhood around each data point, and hold

these distances fixed. For all other grid cells, the distance function is initialized to a large positive value which will be updated as the algorithm progresses. At each grid location we calculate the solution of (2), denoted $\bar{d}$ and then update the value of d at the grid cell to be the minimum of its current value and $\bar{d}$. We sweep through the grid consecutively in the following orders:

$$(1) i = 1 : I, j = 1 : J \quad (2) i = I : 1, j = 1 : J$$
$$(3) i = I : 1, j = J : 1 \quad (4) i = 1 : I, j = J : 1.$$

Computing the distance function in this way can be done in $O(N)$ operations. One can visualize the data using surfaces $d = \epsilon$ for $\epsilon > 0$. However, these surfaces do not pass through the original data and they can be quite rough. See figure 2.

In order to obtain a smooth representation of the surface, Zhao and Osher [3] introduce the weighted minimal surface model. They define the following surface energy

$$E(\Gamma) = \left[\int_\Gamma d^p(x)\, ds \right]^{\frac{1}{p}}, \quad 1 \le p \le \infty, \tag{1}$$

where Γ is an arbitrary surface and ds is the surface area. The gradient flow

of this energy functional is

$$\frac{d\Gamma}{dt} = -\left[\int_\Gamma d^p(x)\, ds\right]^{\frac{1}{p}-1} d^{p-1}(x)\left[\nabla d(x)\cdot n + \frac{1}{p}d(x)\kappa\right] n,$$

where n is the outward unit normal and κ is the mean curvature. This gradient flow represents an attraction toward the surface regularized by the curvature term. In two dimensions, the steady state is a polygon with the data points as its vertices, but in three dimensions the steady state is smoother than a polyhedron. Since the implementation of this gradient descent currently requires a time explicit method, it is necessary to take a small time step $\Delta t = O(h^2)$. So one must begin with a good initial guess of the surface. First, we bound the lumen surface with a tube around the centerline. If we let $d(x)$ be the distance function to the centerline and r be the maximum radius of the lumen contours, then $d = r$ will be such a surface. See the plot on the left of figure 3. We then employ the tagging procedure proposed in [3] to march this surface toward the data. Our goal is basically to tag the grid points on the boundary and in the interior of the lumen. We begin with a temporary boundary at $d = r$ and consider the point in this set that is farthest from the data. If it has an interior neighbor which is further away, we tag it as a final boundary point. If it does not, then we tag it as an exterior point. We continue this process until the furthest temporary boundary point is less than some specified distance from the data. We then construct the signed distance function to this estimated interior. This provides us with an initial level set function for a surface that is "shrinkwrapped" around the data. See the plot on the right of figure 3.

The gradient flow of the weighted minimal surface model can be solved using a level set formulation. We choose a level set function $\phi(x)$ such that $\Gamma = \{x : \phi(x) = 0\}$. (We also use the convention that $\phi < 0$ on the interior of the surface and that $\phi > 0$ on the exterior.) Then, when extended to all level sets, the gradient flow of the surface energy (1) (with p=1) becomes

$$\frac{\partial \phi}{\partial t} = -|\nabla\phi|\left[\nabla d\cdot\frac{\nabla\phi}{|\nabla\phi|} + d\nabla\cdot\frac{\nabla\phi}{|\nabla\phi|}\right].$$

To preserve a well-behaved level set function, we initialize the above evolution with a signed distance function, as described previously, and reinitialize the level set every time step. Reinitialization simply turns a general level set function into a signed distance function describing the same surface. There

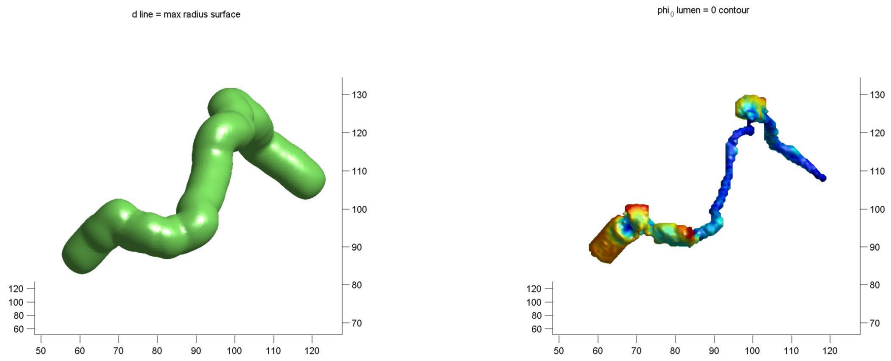


Figure 3: (*left:*) A contour of the distance function to the centerline which contains the lumen surface. (*right*) The initial guess for the lumen surface obtained by marching the tube toward the data.

are various methods for this procedure. We fix the level set function ϕ_0 and consider the evolution of the PDE

$$\phi_t = \text{sgn}(\phi_0)(1 - |\nabla\phi|).$$

In the steady state, ϕ will be a signed distance function with the same zero level set as ϕ_0. However, when implemented numerically, the sgn function must be regularized, and this causes some smoothing of the zero level set. We perform one time step of this reinitialization PDE for each time step of the gradient descent PDE. After the steady state is found, we perform complete reinitialization to find a signed distance function to the surface. See the figure on the right of figure **??** for the results.

Level set functions are a convenient way of describing objects that can change their topology. In our case, this is a disadvantage, since we know that segments of the lumen or wall surface (with their ends capped) have a spherical topology, and we would also like our surface representation to have this property. One approach would be to use a topology-preserving method such as the one proposed by Han, Xu, and Prince in [2]. Since we have some initial information about the geometry of the object, we use a different approach. We know that the centerline is inside the lumen, so we enforce this condition during the evolution of the lumen surface by using an inequality constraint. Let d_c be the distance function to the centerline, and ϕ_ℓ be the level set function for the lumen. Then, $\Omega_c := \{x : d_c(x) \leq \epsilon\}$

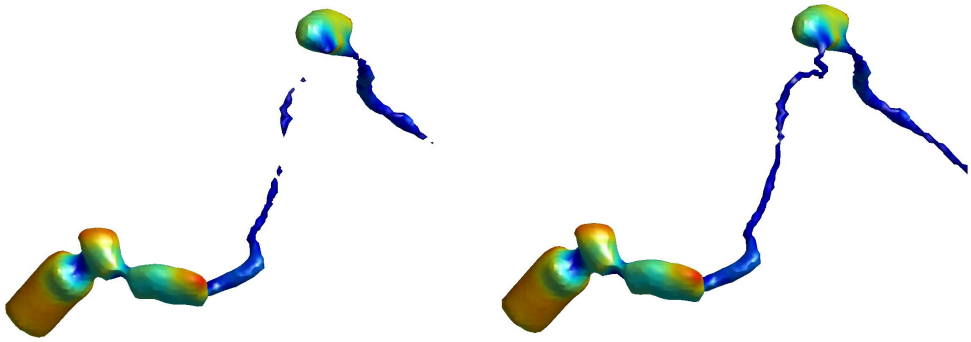


Figure 4: (*left:*) The reconstructed lumen surface without constraints. (*right:*) The reconstructed 0lumen surface with constraints.

is a thin cylinder around the centerline, and $\Omega_\ell := \{x : \phi_\ell(x) \leq 0\}$ is the region enclosed by the lumen surface. We want to ensure that $\Omega_c \subset \Omega_\ell$. In terms of the level set function, we can express this as $\phi_\ell(x) \leq 0$ for $x \in \Omega_\ell$. Therefore, in each step of our algorithm, for grid points $x \in \Omega_\ell$, we replace $\phi_\ell(x)$ with $\min(\phi_\ell(x), 0)$. Similarly, we constrain the wall surface to always contain the interior of the lumen. In figure 4, we show the effect of including these constraints.

2.1 Numerical Simulations

For our numerical simulations we solve the eikonal equation with the fast sweeping method as described above. The gradient flow of the weighted minimal surface model is computed using the finite difference method. We use the forward Euler method in time. We discretize advection terms using first-order differences and upwinding. For the curvature terms, we use second order centered differences for the first and second derivatives. Approximate reinitialization is performed once every time step in the manner described above, using Godunov flux functions to discretize the derivatives. Neumann boundary conditions were used. To save computations, we only update the value of ϕ at grid points near the surface. This is the so-called narrow band method.

2.2 Conclusion

The method of Zhao and Osher described in [3] allows one to obtain an implicit representation of a surface given an unorganized set of points on the surface. We were able to apply this to the vessel segmentation data from Vital Images and obtain a description of the lumen and wall surfaces. It is possible to use other forcing functions to evolve the surface and obtain a better estimation of the vessel surfaces. One possibility is to assign weights to the initial estimated contours that would attract the surface toward contours that the radiologist has edited or given a higher confidence rating0. Another option would be to use the original image data to drive the reconstructed surface toward edges or other image features. Finally, it may be possible to gather statistics from a large number of vessels and to use this as prior information for the segmentation. Since blood vessels vary greatly in shape, it is unlikely that a global shape prior can be used. However, local information, such as the distance between the lumen and the wall, may exhibit trends which can be exploited in the segmentation. We would recommend that Vital Images explore these possibilities.

3 Direct Construction through Linear Interpolation

3.1 2D Implementation

We first implement the linear interpolation schemes on a two dimensional plane. Given the sampling data points of a contour, $\mathcal{S} = \{(x^{(n)}, y^{(n)} \in \mathbb{R}^2, n = 1, \ldots M\}$ and its center point $\{x^c, y^c\}$, we want to find a function $f : \mathbb{R}^2 \to \mathbb{R}$ whose zero level set $\{(x, y)|f(x, y) = 0\}$ gives a smooth fitting to desired contour. Bilinear interpolation is carried out as the following schemes:

$$f(x, y_1) \approx \frac{x_2 - x}{x_2 - x_1} f(x_1, y_1) + \frac{x - x_1}{x_2 - x_1} f(x_2, y_1) \quad (2)$$

$$f(x, y_2) \approx \frac{x_2 - x}{x_2 - x_1} f(x_1, y_2) + \frac{x - x_1}{x_2 - x_1} f(x_2, y_2) \quad (3)$$

$$f(x, y) \approx \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y - y_1}{y_2 - y_1} f(x, y_2), \quad (4)$$

where $x_{1,2}$ and $y_{1,2}$ define the grid that a specific (x, y) lies in. As shown in (4), $f(x, y)$ can be represented as a linear combination of $f(x_1, y_1)$, $f(x_1, y_2)$,

$f(x_2, y_1)$ and $f(x_2, y_2)$. After imposing $f(x^{(n)}, y^{(n)}) = 0$ on each sampling contour data $(x^{(n)}, y^{(n)}) \in \mathcal{S}$ and $f(x^c, y^c) = -1$ on the center point, we end up solving a linear system

$$Af = b. \tag{5}$$

Here two problems may show up. First, when we discretize the domain using $N \times N$ grids, there are N^2 variables that need to be determined. By increasing N, we can get a finer result. However, given the limited number of sampling data we have, the linear system would become highly under determined. Secondly, in order to get a smooth contour, we also need to put appropriate conditions on the $f(x, y)$ and its derivatives.

Given the discussion above, we impose second order derivative conditions on the interior nodes and first order on the boundary nodes, i.e.

$$\begin{aligned} \Delta f(x, y) &= 0 \qquad (x, y) \in \Omega & (6) \\ \nabla f(x, y) \cdot \overrightarrow{\mathrm{n}} &= 0 \qquad (x, y) \in \partial\Omega. & (7) \end{aligned}$$

Note that this is actually a Laplace's equation with Neumann boundary conditions. According to uniqueness theorem, Laplace's equation has one unique solution once boundary conditions are fixed. In our case, the unique solution would be zero. However, we do not solve Laplace's equation exactly. With the imposed function values on the sampling data, we want to solve a least square problem such that the solution vector f would satisfy the function value constraints $f(\mathcal{S}) = 0$ and $f(x^c, y^c) = -1$, and be as smooth as possible. We use finite difference to discretize (8) and (9).

One could also impose a Dirichlet boundary condition on the boundary i.e.

$$\begin{aligned} \Delta f(x, y) &= 0 \qquad (x, y) \in \Omega & (8) \\ f(x, y) &= c \qquad (x, y) \in \partial\Omega. & (9) \end{aligned}$$

where c is a constant. This will prevent sharp spikes and help smooth out the level set function.

When implementing this method, we first mesh the domain with a $N \times N$ grid,

$$\begin{aligned} x_{\min} = x_1 < \ldots < x_N = x_{\max} \\ y_{\min} = y_1 < \ldots < y_N = x_{\max}, \end{aligned}$$

and find the specific square unit that each sampling point lies in, i.e. the vertices $[x_{1,2}^{(n)}, y_{1,2}^{(n)}]$ for $n = 1, \dots M$ in (2)–(4). Then vectorize the discretized 2D function $f(x_i, y_j)$ first along x then along y. Construct the coefficient matrix by inserting the four nonzero coefficients as defined in (2)–(4) into each row that corresponds to the sampling points. Assign corresponding function values to the righthand side vector.

The discretized smoothing constraints (8)–(9) can also be represented as a coefficient matrix. Combining it with the data point coefficient matrix above, we can get a complete linear system $Af = b$. In effect, adding smoothing constraints on every grid point largely increases the matrix rank. The row size of the complete matrix A is the total number of constraints that we put onto the function, including function value 0 and -1 on sampled contour and center point, and derivative constraints. It also reveals how much information we have about the function. The column size of A is N^2 depending on the grid size. Note that A is a highly sparse matrix.

By solving the linear equation, we get $f(x, y)$ on each grid point. Finally, apply bilinear interpolation in (4) to approximate the original function over the entire domain.

3.2 Numerical Results in 2D

We test this linear interpolation method on 281 cross-intersections of a vessel. Each cross-intersection gives two sampling data sets for the vessel's outer wall and lumen respectively. We expect that after interpolated over discrete sampling points of either the outer wall or the lumen, the zero level set of the bilinear function $\{(x, y)|f(x, y) = 0\}$ is continuous and passes through the samples. Here are some examples showing our method work well in 2D problem, even when the sampling data reflects extreme curvature.

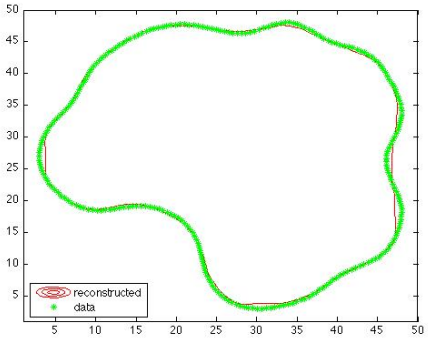


Figure: contour plot

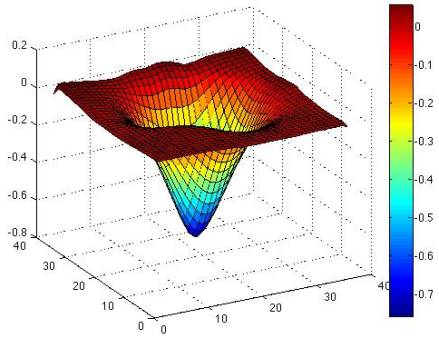


Figure: Surface plot of $\phi(x)$

3.3 3D Implementation

The linear interpolation and smoothing constraints can be translated into 3D. Now given 281 cross-intersections and a sampled center line in 3D, we want to construct a smooth surface that coves all the cross-intersections and have the center line passing through inside the surface. We define this desired surface to be a zero level set of a function $f : \mathbb{R}^3 \to \mathbb{R}$, $\{(x, y, z)|f(x, y, z) = 0\}$.

We mesh the 3D domain into $N \times N \times N$ cubics. Now the interpolation scheme is

$$f(x, y_{1,2}, z_{1,2}) = \frac{x_2 - x}{x_2 - x_1} f(x_1, y_{1,2}, z_{1,2}) + \frac{x - x_1}{x_2 - x_1} f(x_2, y_{1,2}, z_{1,2}) \quad (10)$$

$$f(x, y, z_{1,2}) = \frac{y_2 - y}{y_2 - y_1} f(x, y_1, z_{1,2}) + \frac{y - y_1}{y_2 - y_1} f(x, y_2, z_{1,2}) \quad (11)$$

$$f(x, y, z) = \frac{z_2 - z}{z_2 - z_1} f(x, y, z_1) + \frac{z - z_1}{z_2 - z_1} f(x, y, z_2), \quad (12)$$

where $[x_{1,2,3}, y_{1,2,3}, z_{1,2,3}]$ defines which cube that a specific point (x, y, z) lies in. And the constraints are

$$\Delta f(x, y, z) = 0 \qquad (x, y, z) \in \Omega \quad (13)$$

$$\nabla f(x, y, z) \cdot \overrightarrow{\mathrm{n}} = 0 \qquad (x, y, z) \in \partial\Omega. \quad (14)$$

We also set the function values at the sampled cross-intersections to be zero and -1 at the sampled center line. As illustrated in 3.1, we first vectorize the discrete $f(x, y, z)$ and then construct the coefficient matrix A and the

righthand side vector b correspondingly. As in the 2D case, we construct the coefficient matrix by inserting the eight nonzero coefficients as defined in (10)–(12) into each row that corresponds to the sampling points. After discretizing the smoothing constraints and assembling the sparse matrix A, we solve the complete linear system $Af = b$. As mentioned before, the system is large and highly under determined, and if we use MATLAB's backslash operator to solve the system, numerical errors are accumulated, which might not give the desired solution. In order to fix this, we use a *multi grid* approach to solve the linear system. The method is described in the next section.

3.4 Nested Iteration for $Af = b$

One essential difficulty with linear interpolation method is that the linear system is giant, especially when we are in multi-dimension and want to refine the grid. Moreover, since the matrix may be underdetermined, the numerical error of solve the system directly would be large and may even completely bury the real result. In Section 3.3, we have come across with this problem. The computational cost is so large and the result becomes unrecognizable due to numerical error. Thus, we use nested iteration to solve the system efficiently and accurately.

The idea of nested iteration comes from multigrid method. We first discretize the domain by a coarser grid of small grid size N and f is a N^3 vector in 3D. We use an iterative solver `lsqr` in MATLAB to solve for f. `lsqr` solves least square problem especially for a rectangular matrix A and it allows us to input an initial estimation f_0. After getting f from solving the linear system, we interpolate f on a finer grid of size $2N$ to extend it into a $4N^2$ vector. In the next step, reuse the solver `lsqr` and set the interpolated $4N^2$ vector f to be our initial estimation. `lsqr` would return a new solution vector f. We continue doing this until a satisfying result is obtained. The pseudo code is provided as below:

- Nested Iteration
  - Set `grid_size` $= N$ and construct A^N and b^N.
  - $f^N :=$ `lsqr`$(A^N, b^N, f_0^N = 0)$
  - While Iteration $<$ MAX
  - Set up new grid with `grid_size` $N := 2N$.

– Interpolate f on new grid size N to get f^N and reconstruct A^N and b^N.

– Update $f^N := \texttt{lsqr}(A^N, b^N, f^N)$.

– End

3.5 Numerical Results in 3D

We test the 3D linear interpolation for 30 of the 281 given sample sections. The results are promising for the 30 sections, however, to reconstruct the vessel with the entire 281 sections, a more powerful machine is required since the linear system is huge and computationally expensive.

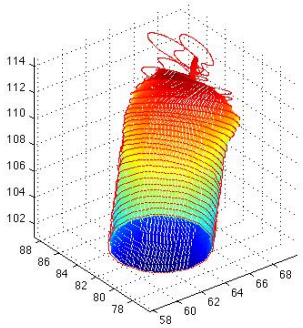


Figure: Data with 30 sections

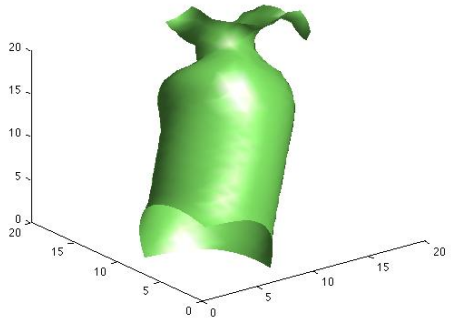


Figure: Reconstruction

4 Polynomial Method

Suppose we already know some 3D data points on the surface of a vessel, and our goal is to build the surface of the vessel. Here we use the zero set of a polynomial to represent the surface of the vessel.

Let's fix the order of the polynomial and let it be n. Thus, the polynomial can be written as:

$$P(x, y, z) = \sum_{\substack{i,j,k \geq 0 \\ i+j+k \leq n}} a_{ijk} x^i y^j z^k$$

and we hope to find a suitable value of $a = (a_{ijk})_{i,j,k\geq 0}^{i+j+k\leq n}$, such that the zero set of the polynomial can fit the real surface of the vessel well. Our main idea is to use the least square method. We want to make the distance from the data and the zero set of the polynomial as small as possible. So we may use some criterial to minimize the total distance.

Let $(x_i, y_i, z_i)_{i=1}^m$ be the data set. For each data point(x_i, y_i, z_i), we can get a corresponding row vector $\alpha_l = (x_l^i y_l^j z_l^k)_{i,j,k\geq 0}^{i+j+k\leq n}$. So the algebraic distance from this data point to the zero set of the polynomial is given by $|\alpha_l \cdot a|$. Let

$$A = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{bmatrix}$$

be a matrix. Then each element of the column vector $A \cdot a$ represent the algebraic distance between the corresponding data point and the zero set. Now we apply the least square method to measure the total distance between the data points and the zero set. In other words, we want to minimize $\|A \cdot a\|_2$. However, we find that a can be zero. So we add $\|a\|_2 = 1$ as a restriction on a, and our problem becomes:

$$min_{\|a\|_2=1} \|A \cdot a\|_2$$

Then we use the Lagrangian to solve this problem. Let

$$\mathcal{L} = a^T A^T A a + \lambda(a^T a - 1)$$

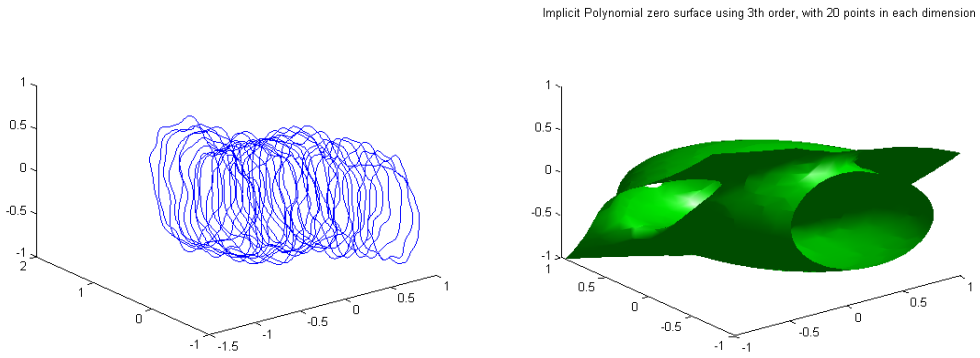
then

$$\frac{\partial L}{\partial a} = 2A^T A a + 2\lambda a = 0 \Rightarrow (A^T A + \lambda I)a = 0$$

Here $\frac{\partial L}{\partial a} = \left(\frac{\partial L}{\partial a_{ijk}}\right)_{i,j,k\geq 0}^{i+j+k\leq n}$ is a column vector. Thus we can see that a is an eigenvactor of $A^T A$. So we can get the value of a among these unit eigenvactors by comparing the corresponding norm $\|Aa\|_2$.

However, since the least square only represent the algebraic distance, not the geometric distance.

So we may make some adjustment of our least square method. If we assume that the gradient of $P(x, y, z)$ doesn't change much in certain neighborhoods of the data points. Then roughly speaking, the geometric distance



(a) (b)

Figure 5: (a) a collection of contours. (b) third-order polynomial fit.

between the data point (x_l, y_l, z_l) and the zero set of the polynomial is approximately given by $(\alpha_l \cdot a)/\left\|\nabla P(x_l, y_l, z_l)\right\|_2$. Since we don't know a in advance, we can only use the value of a from what we have already get using the least square without adjustment. So for each row vector α_l of A, we add weight $1/\left\|\nabla P(x_l, y_l, z_l)\right\|_2$. I.e., let

$$W = diag(\left\|\nabla P(x_1, y_1, z_1)\right\|_2^{-1}, \left\|\nabla P(x_2, y_2, z_2)\right\|_2^{-1}, \cdots, \left\|\nabla P(x_m, y_m, z_m)\right\|_2^{-1})$$

be the weighted matrix. We solve our new least square problem:

$$min_{\|a\|_2=1} \left\|(AW) \cdot a\right\|_2$$

hoping to get a better result.

To sum up, our mean steps are as follows:

step 1: solve $min_{\|a\|_2=1} \left\|A \cdot a\right\|_2$ to get a;

step 2: use the value of a we get from the last step to get the weighted matrix W;

step 3: solve $min_{\|a\|_2=1} \left\|(AW) \cdot a\right\|_2$ to get a new a;

step 4: repeat step 2 and step 3 until the program satisfies a certain stopping criterion.

We use our vessel data to test this kind of algorithm. An example of a third order fit is shown in Figure 5. Possible issues with the method include the following:

1. we haven't considered the singularity of the matrix A;

2. we haven't considered the stability of these equation systems;

3. we don't know if the weighted matrix W is accurate enough, and if a gotten from each step converges finally;

4. we fix the order of the polynomial. Maybe we can try some implicit polynomial methods.

So if possible, we will make several corresponding adjustments to our algorithm in the future.

5 Geodesic Method

Given are a centerline and given a collection of cross-sectional contours of either the vessel lumen or of the vessel outer wall. Considering only centerline and vessel lumen, the corresponding data are a $3 \times L$ matrix `centerline` and $L \times 256$ matrices `xl, yl, zl` (the lowercase `l` stands for "lumen"), where L is the number of cross-sections and 256 is the default number of data points, in 3D space, for each contour. In other words, to each point c on the centerline there corresponds

- a tangent vector v to the centerline at c
- a cross-sectional plane containing c and orthogonal to v
- a single cross-sectional contour lying in the cross-sectional plane

and the cross-sectional contours are exactly the data `xl, yl, zl`. The contour data are assumed to lie approximately on an underlying (i.e., not explicitly given) smooth, 2D surface which is diffeomorphic to a cylinder. Also, it is possible that the individual cross-sectional contours intersect or overlap in certain places. The goal is to obtain a discrete reconstruction of the underlying 2D surface, i.e., a set of matrices `Sx, Sy, Sz` describing the coordinates of a grid of points on the surface. One way to do this is to use the contour data to estimate the tangent planes to the surface. The reconstructed surface can then be obtained by evolving a collection of trajectories, as described below.

5.1 Implementation

The following method describes the construction of the surface grid `Sx, Sy, Sz`.

5.1.1 Geodesic Trajectory

Here we describe the determination of trajectories from approximate tangent planes at each data point. Note that "geodesic" is used loosely here, as the resulting trajectory may not be an exact geodesic on the surface.

Let $(x_k)_{k=1,\ldots,N}$ denote an N-step trajectory. Contour data points are denoted by $p \in \mathbb{R}^3$, and corresponding centerline points and centerline tangent vectors are denoted by $c_p \in \mathbb{R}^3$ and $v_p \in \mathbb{R}^3$. The tangent space at p is denoted T_p, and the projector onto the tangent space at p is denoted $P_p : \mathbb{R}^3 \to T_p$. Note that the tangent *space* T_p is a subspace of $\mathbb{R}^3$, i.e., it contains the origin, whereas the tangent *plane* $p + T_p$ at p is a plane in $\mathbb{R}^3$ which contains the point p, and is not necessarily a subspace of $\mathbb{R}^3$. The initial point x_1 is taken to be a contour data point at one extremity of the vessel. The update $x_i \mapsto x_{i+1}$ is performed as follows. Let p be a contour data point such that $\|p - x_i\|$ is minimal. Then, the following two steps are performed:

$$\begin{aligned} x_i &\mapsto x_{i+1/2} = p + P_p(x_i - p) \\ x_{i+1/2} &\mapsto x_{i+1} = x_{i+1/2} + \Delta t \cdot P_p(v_p)/\|P_p(v_p)\| \end{aligned}$$

where it is in general assumed that $\|P_p(v_p)\| \neq 0$, i.e., that the tangent vector to the centerline point c_p is not orthogonal to the tangent space T_p. The first step projects x_i onto the tangent plane $p + T_p$ of p, the contour data point which is closest to x_i. The second step advances $x_{i+1/2}$ in the direction obtained by projecting the tangent vector v_p onto the tangent space T_p, and normalizing to unit length.

5.1.2 Computation of the tangent plane

The tangent plane is determined from its normal vector. A naive approach to computing a normal vector is taken here. We take a collection of points

$(p_i)_{i=1,\ldots,n}$ in the contour data which are close to p. Assuming the contour data is in fairly good order, if (i, j) is the index of p in the contour data matrix, we may take data points with indeces $(i \pm w, j \pm w)$ for some $w \in \mathbb{Z}^+$. Then, we compute cross-products of some number of pairs $(p_i - p)$ and $(p_j - p)$ with $i \neq j$. From these cross-products, a collection of vectors is obtained. By averaging over this collection, an approximate normal to the surface at p is determined.

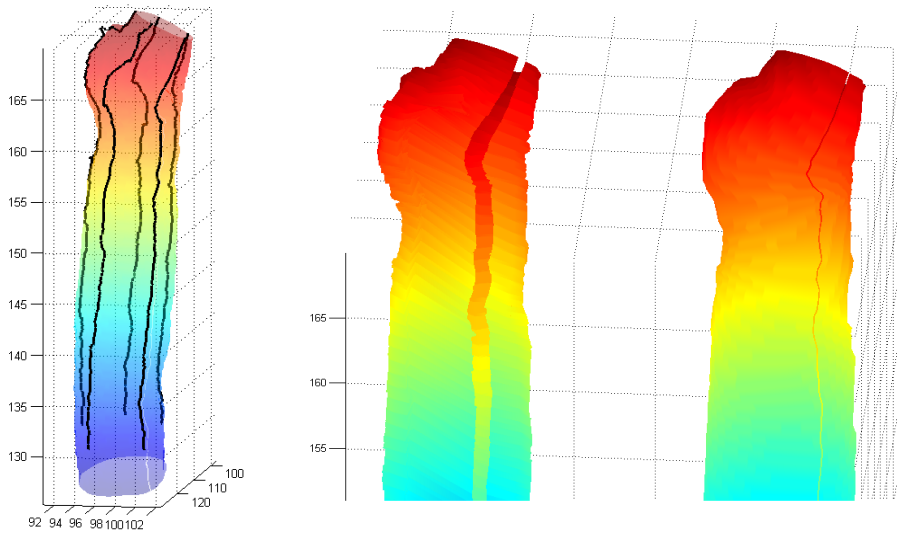
5.1.3 Surface Reconstruction

The parametrized surface `Sx, Sy, Sz` can be obtained by evolving a collection of trajectories beginning at equally spaced points around the edge of the contour data, as seen in Figure 6 (a). In that figure, there are five trajectories beginning at the top of the coloured surface, at points equally spaced around the opening of the surface. In that picture, the coloured surface is a visualization of the contour data. By taking enough initial points equally spaced around the edge, the trajectories can be "stitched together" into a surface. An example of such a surface is shown in Figure 6 (b), in which the contour data (right) is compared to a reconstructed surface made up of trajectories (left). Explicitly, each row of the matrices `Sx, Sy, Sz` corresponds to a single trajectory, and together, the trajectories make up the reconstructed surface.

5.2 Results/Discussion

The geodesic method is implemented on blood vessel lumen data. Images are shown in Figures 6, 7 and 8.

The geodesic method does a good job of staying close to the surface and of moving along the surface as shown in Figure 6 (a), since the update direction is linked to the centerline direction. When the contour data are already smooth and well-behaved, an accurate reconstruction of the surface is obtained, as shown in Figures 6 (b) and 7. The trajectory udpate rule described in 5.1.1 is only position-dependent. If we assume that the update rule corresponds to a smooth underlying vector field, then aside from discretization error the trajectories should in principle not intersect.



(a) (b)

Figure 6: (a) a collection of geodesic trajectories computed from the vessel lumen data. Coloured surface is a visualization of the contour data. (b) comparison of surfaces obtained from a set of good contour data (right) and from a set of 20 geodesic trajectories (left).

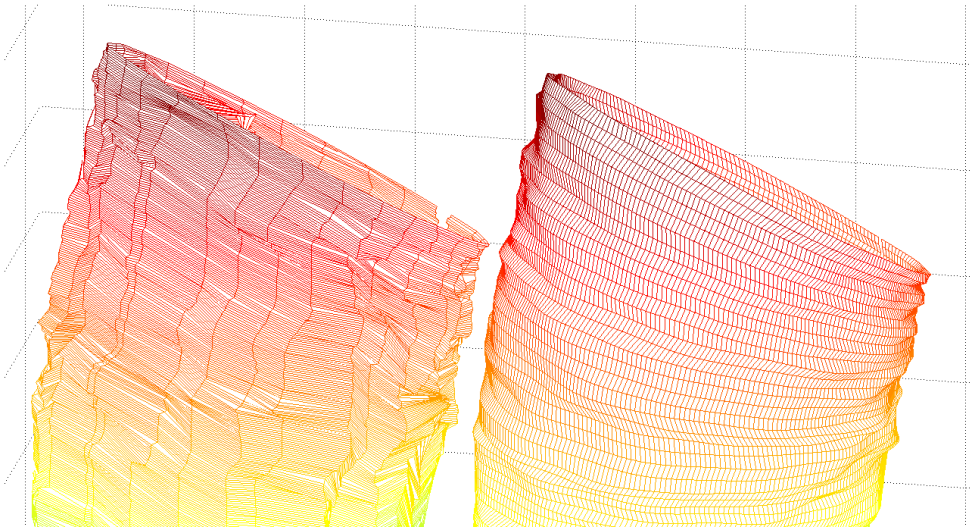


Figure 7: Comparison of meshes obtained from contour data (right) and from a set of 30 geodesic trajectories (left).

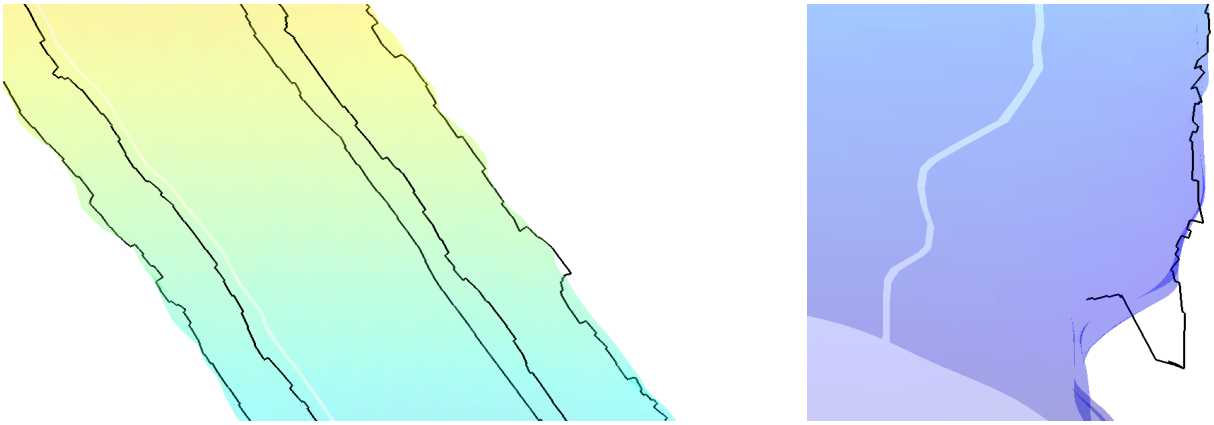
Some degeneracies in the trajectories are depicted in Figure 8. In Figure 8 (a) there are many sharp jumps which result from projection onto the tangent planes of different contour data points as the trajectory evolves. In Figure 8 (b) a trajectory shoots off from a sharp edge on the contour data, and continues along the same tangent plane for some time before returning to the contour data surface.

5.3 Recommendations for future improvement

Improvements can be made in the following areas:

- computation of the normal vector
- smoothness of trajectories
- well-conditioning of the parametrization

As it is presently computed, the normal vector is susceptible to sharp corners in the contour data. One way to improve this is to use an algorithm such as RANSAC [1] to diminish the influence of isolated sharp corners. This may



(a) (b)

Figure 8: (a) depiction of the sharp jumps resulting from projection at each step. (b) depiction of a geodesic "going off on a tangent".

lead to increased smoothness and better following of the trajectories, and a lower probability of effects such as in Figure 8 (b). A direct smoothing of the trajectories to avoid the small jumps observed in Figure 8 (a). The third bulleted item refers to the imposition of some constraint to ensure that the trajectories are equally distributed about the centerline as they travel down the vessel, since this ensure that trajectories do not "bunch up" in certain places along the surface.

References

[1] M.A. Fischler and R.C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *ACM*, 24:381–395, 1981.

[2] Xiao Han, Chenyang Xu, and J.L. Prince. A topology preserving level set method for geometric deformable mo'dels. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(6):755 – 768, june 2003.

[3] Hongkai Zhao and Stanley Osher. Visualization, analysis and shape reconstruction of sparse data. In Stanley Osher and Nikos Paragios, editors,

Geometric Level Set Methods in Imaging, Vision, and Graphics, pages 361–380. Springer New York, 2003.