Shortest Paths with Curvature and Torsion

Petter Strandmark[1] Johannes Ulén[1] Fredrik Kahl[1,2] Leo Grady[3]
[1]Lund University, Sweden [2]Chalmers University of Technology, Sweden [3]HeartFlow Inc.
{petter,ulen,fredrik}@maths.lth.se lgrady@heartflow.com

Abstract

This paper describes a method of finding thin, elongated structures in images and volumes. We use shortest paths to minimize very general functionals of higher-order curve properties, such as curvature and torsion. Our globally optimal method uses line graphs and its runtime is polynomial in the size of the discretization, often in the order of seconds on a single computer. To our knowledge, we are the first to perform experiments in three dimensions with curvature and torsion regularization. The largest graphs we process have almost one hundred billion arcs. Experiments on medical images and in multi-view reconstruction show the significance and practical usefulness of regularization based on curvature while torsion is still only tractable for small-scale problems.

1. Introduction

In differential geometry, the fundamental theorem of curves states that any regular curve in three dimensions with non-zero curvature has its shape completely determined by its curvature and torsion [12]. Therefore, for curve reconstruction problems, it makes sense to regularize the solution with curvature and torsion priors. So, why is it that the majority of approaches in the literature only consider length regularization? The reason is of course due to computational complexity. In this paper, we demonstrate that with today's modern CPUs and clever implementation choices, it is actually possible to solve inverse problems involving both curvature and torsion priors at reasonable running times.

Curvature regularization has been shown to be important for a number of applications in computer vision. In [10], it is shown that Euler's elastica – the line integral of the squared curvature – conforms better to intuitive completion of boundary curves. Other applications where curvature plays an important role include saliency [17], inpainting [9], stereo [21], region-based image segmentation [15] and surface reconstruction [19].

Torsion measures how much a curve deviates from its osculating plane. In many applications, the reconstructed curve should follow closely the shape of an underlying surface which is locally planar. In such situations, it may be advantageous to penalize torsion. We are not aware of any work that has previously used this as a prior. Figure 1 shows a drawing of a heart. The coronary arteries are neither short nor have low curvature, but they do have low torsion. Penalizing high torsion would be an ideal prior.

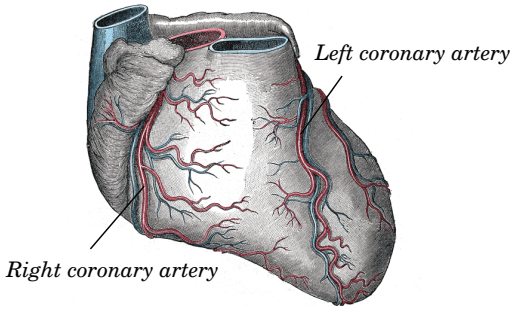


Figure 1: A drawing of a heart. The thinner coronary vessels are long with high curvature, but have low torsion as they lie approximately in a plane.

Related work. We follow a classic approach for computing the global minimum of an active contour model based on shortest paths [2, 3]. This is a flexible and frequently used tool for extracting 1D structures in both 2D and 3D images, especially in medical image analysis [8]. Most of these methods use only weighted length regularization, even though the idea of applying shortest paths to higher-order functionals involving curvature is not new. One early example of minimizing a curvature-based energy can be found in [1]. However, in order to obtain reasonable running times, only a small band around the initial contour is considered. In [16], the elastic ratio functional is proposed for edge-based 2D image segmentation. Still, the run times are up to several hours for medium-sized images on the CPU. An extension of the live-wire framework with curvature priors is presented in [20]. The algorithm is applied to image squares of up to 80×80 pixels with 8-connectivity. In [11], the standard 2D shortest path is lifted to 4D by incorporating 12

discrete orientation angles and radii in the estimation, and hence implicitly penalizing curvature. Many nice examples of the benefits with curvature are given in both [11] and [20], but no running times are reported. In [7], a heuristic method is developed for minimizing a pseudo-elastica for 2D segmentation. The computational complexity of their algorithm is attractive, but at the price of approximate solutions. Our algorithm has comparable running times and the same asymptotic complexity $\mathcal{O}(d^2 n \log nd)$ where n is the number of pixels and d depends on the neighborhood size.

Contributions. Our contribution is a global method to find a curve of minimal curvature between two points using line graphs. We are the first to demonstrate practical curvature experiments in 3D and our two-dimensional running times are in the order of seconds. Our approach extends to higher-order properties of space curves such as torsion. As the grid resolution and the neighborhood size increase, the discretization errors of the continuous problem tend to zero. This fact is experimentally verified.

We segment blood vessels in 2D (Section 3.2) and 3D (Section 3.3), and we perform multi-view 3D reconstruction (Section 3.4), which previously only have been possible using local optimization [6]. The framework has been implemented in highly optimized C++ code and is made publicly available to facilitate further research.

2. Shortest Paths with Curvature and Torsion

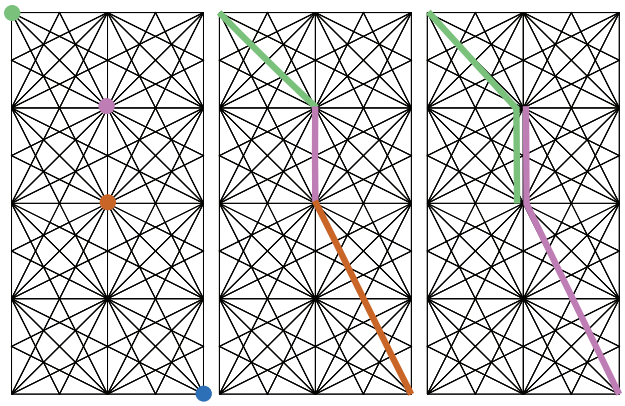
We are interested in finding a curve γ in two or three dimensions which minimizes an image-dependent functional of its length, curvature and torsion, or more generally:

$$\inf_{\gamma, L} \int_0^L f(I, \gamma(s), \gamma'(s), \gamma''(s), \gamma'''(s))\, ds, \quad \text{subject to } \gamma(0) \in E_{\text{start}} \text{ and } \gamma(L) \in E_{\text{end}}, \tag{1}$$

where $\gamma : [0, L] \to \mathbf{R}^{2 \text{ or } 3}$ is parametrized by arc-length [12]. The curve has to start in the set E_{start} and end in E_{end}. The image I is arbitrary. This problem has a finite solution if $f \geq 0$ everywhere. We will restrict our attention to positive functionals f of the form

$$I(\gamma(s)) + \rho + \sigma\kappa(s)^2 + \nu\tau(s)^2. \tag{2}$$

Here $\kappa(s)$ and $\tau(s)$ denote the curvature and the torsion, respectively; see Section 2.1. The scalars ρ, σ and ν are weighting factors of length, curvature and torsion, respectively and can be made image-dependent as well. In the framework of geodesic active contours, the first two terms are merged and regarded as a general Riemannian metric. Our approach is to solve (1) globally on a predefined discrete mesh. Local refinement is always possible as a post-processing step.



(a) Using points (b) Using edges (c) Using edge pairs

Figure 2: The same path through a mesh (shown in black) represented in three different ways. The graph nodes (shown in color) correspond to (a) points, (b) edges, and (c) edge pairs in the mesh.

Terminology. The predefined *mesh* consists of *points* and *edges*. The shortest path computation is carried out in a *graph*, consisting of *nodes* and *arcs*. These two objects are generally distinct.

Length. The mesh is stored explicitly in memory and is the same no matter which regularization is used. Figure 2 shows the mesh in black. The structure and size of the graph, however, depend on the regularization used. The simplest case is length regularization ($\sigma = \nu = 0$), for which the graph is identical to the mesh. Each node in the graph then corresponds to a point in the mesh. The arc weights of the graph are computed by simply integrating (2) over each arc.

Curvature. A pair of points always lie on a straight line. It follows that we need to consider at least three points to determine the curvature cost. To achieve this, a line graph is constructed in which each node corresponds to an edge in the mesh. Now any arc connecting two nodes corresponds to the interaction of three points (one pair of edges). In this manner we can compute the curvature cost of any path through the original mesh. Figure 2b shows the same path as before with the nodes of this graph highlighted.

Torsion. Because three points always lie in the same plane, at least four points are needed to determine the torsion cost. The process of constructing the line graph can be iterated one more time to incorporate torsion. Figure 2c shows the same path as before, with each node in the graph corresponding to an edge pair in the mesh. The graph grows very rapidly in size, but we never construct it explicitly; only the mesh is explicitly created and stored.

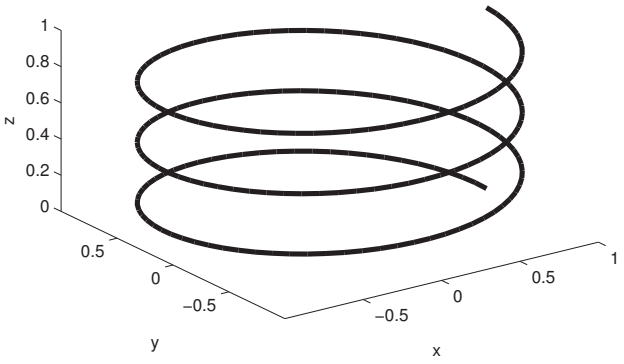


Figure 3: Example curve $(x, y, z) = (\cos 6\pi t, \sin 6\pi t, t)$.

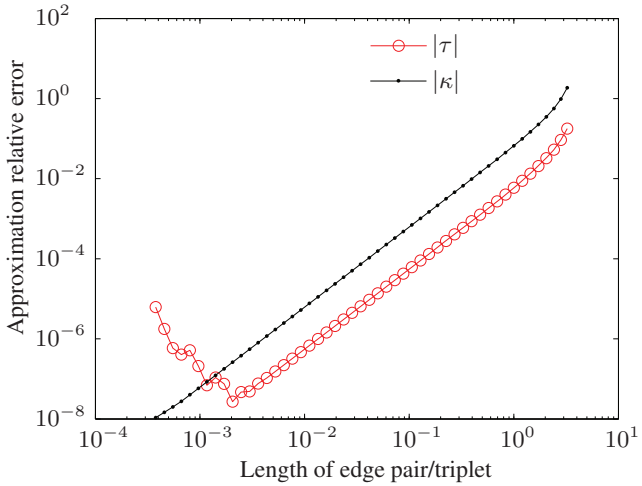


Figure 4: Error when computing the curvature and torsion using the middle point of a spline fit to two or three edges on the curve in Figure 3.

2.1. Calculating Curvature and Torsion

Let (x_i, y_i, z_i) for $i = 1, 2, 3$ be three ordered points. The quadratic B-spline associated with these points is

$$\boldsymbol{r}(t) = \frac{1}{2} \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} t^2 \\ t \\ 1 \end{bmatrix}. \quad (3)$$

The curvature is defined as $\kappa(t) = \frac{\|\boldsymbol{r}'(t) \times \boldsymbol{r}''(t)\|}{\|\boldsymbol{r}'(t)\|^3}$ [12]. It is straightforward to analytically compute the squared curvature as well as the length element $ds = \|\boldsymbol{r}'(t)\|dt$ of the spline using symbolic mathematics software. There is no closed form for the integral $\int_0^1 \kappa(t)^2 ds(t)$, so numerical integration is required to compute it. Since we cache all values, the execution time will not be affected by the choice of numerical scheme.

When approximating torsion, four points and derivatives of up to degree 3 are needed. Hence, cubic B-splines are suitable:

$$\boldsymbol{r}(t) = \frac{1}{6} \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}. \quad (4)$$

Similar to curvature, the torsion [12]

$$\tau(t) = \det \begin{bmatrix} \boldsymbol{r}'(t) & \boldsymbol{r}''(t) & \boldsymbol{r}'''(t) \end{bmatrix} / \|\boldsymbol{r}'(t) \times \boldsymbol{r}''(t)\|^2, \quad (5)$$

of the B-spline can also be analytically calculated.

Figure 3 graphs an example curve (a helix), for which the exact curvature and torsion are well known: they are both constant. Figure 4 shows the error when approximating the curvature and torsion using three and four points, respectively. The experimental section will demonstrate convergence of the entire system (Figure 7).

2.2. Our Implementation

Dijkstra's algorithm for computing the shortest path is well known. An important property is the fact that it does not require the entire graph to be stored. Two things are required: (i) The number of nodes along with a start set and a end set. (ii) An oracle that given a node returns its neighbors and arc weights. Using an oracle means that the weight does not have to be computed for every arc in the graph. As Section 2.1 shows, computing the arc weights can become quite involved.

A common improvement to Dijkstra's algorithm is A*, which changes the node visitation order. It requires a function l, which for each node returns a lower bound of the distance to the end set. Any such l yields a correct algorithm, and if $l \equiv 0$, then A* becomes Dijkstra's.

When computing the path with shortest length, the distance "as the crow flies" can be used as a lower bound, ignoring any obstacles in the graph. For curvature, however, computing a useful lower bound is more difficult (it has to be done really fast to make a difference). One option is to set $\sigma = \nu = 0$ in (2), solve (1) for all nodes (this is fast), and use the result as a lower bound. We will evaluate this heuristic in Section 3.1.

We use the same implementation of Dijkstra's algorithm for all experiments in the paper. It uses the C++ standard library for all data structures. Many of the graphs have billions of arcs, which puts big demands on the memory efficiency. The following items are stored.

- An array of points, where each point contains its coordinates and a set of its neighbors.
- An array of edges, where each edge contains the in-

dices of its two points.

- An array of edge pairs, where each pair contains the indices of its three points.

In our C++ implementation, "array" and "set" mean `std::vector`. Edges and edge pairs are represented by `std::tuples` of integer point indices. Each edge and edge pair in the mesh then require 8 and 12 bytes, respectively.

The array of edges is sorted after the mesh is created, which makes it possible to find the index of a given edge in logarithmic time. The neighbors of a given edge are computed with an oracle. It first fetches the neighbors of the end point of the edge – this takes constant time. Then it looks up the edge index for each pair of points in the sorted edge array. This procedure can be made faster if the neighbors of each edge are stored in the mesh. However, we only found it to be slightly faster and it requires much more memory. Storing the neighbors of each edge pair is out of the question. Instead, the neighbors of an edge pair are computed via the edge neighbors of its last edge as above.

We use caches to avoid having to perform the same computation twice. For example, the curvature cost between two edges is the same even if both edges are translated by the same amount. Thus, the integrated curvature needs only be computed once for each configuration. The cache has to be fast, since when working with torsion there are millions of different configurations of two edge pairs. Our implementation uses an `std::map` with the coordinates (mean subtracted) as keys.

Our implementation works with arbitrary meshes. If only regular meshes with fixed point neighborhoods are desired, memory can be saved by not even storing the mesh explicitly.

Parallelization. Computing the shortest path between two nodes efficiently in parallel is not a trivial task. Therefore Dijkstra's is sequentially run and the neighbors of each node are sequentially computed. However, the oracle can compute the arc weights for all neighbors in parallel on a multi-core CPU. Since these computations are a large portion of the total computational cost (even with the above-mentioned cache), some parallelization is obtained. The curvature or torsion cache is filled before the computation starts. With this partial parallelization, using two CPU cores was about 30% faster in our experiments.

3. Experiments

We have performed a number of experiments. First, we would like to demonstrate the differences between length, curvature, and torsion with a couple of synthetic experiments. We then proceed with experiments in medical image analysis and 3D multi-view reconstruction.

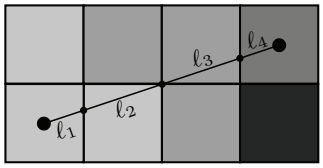


Figure 5: The interpolated data cost for an edge from the lower right pixel to the upper right pixel. The cost is $\sum_{i=1}^{4} \ell_i U(i)$, where $U(i)$ is data cost for voxel i.

$\sigma = 0.25$	$\sigma = 50$	$\sigma = 500$
886,966 (3.45 s)	1,328,045 (9.70 s)	916,061 (9.14 s)
196,553 (0.86 s)	885,415 (6.67 s)	740,683 (7.54 s)

Table 1: Number of nodes visited using Dijkstra's (first row) and A* (second row) for the curvature experiments in Figure 6. Naturally, the heuristic works best for low curvature regularization.

Data-dependent term. The function g in (2) is the data term and it depends on the image. We view the image as a function which is piecewise constant on all pixels (or voxels). The integral $\int g(I, \gamma(s))\, ds$ is then equal to a sum; see Figure 5.

Connectivity. All of our 2D (3D) experiments are performed with a mesh whose points are arranged in a square (cubic) lattice. All points within a specified distance d are connected with an edge. For example, $d = 2.5$ gives each point a degree of 16.

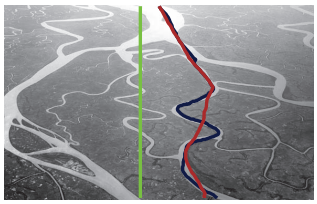
3.1. Synthetic Experiments

The first experiments highlight the differences between length and curvature. We manually sampled the river image in Figure 6 at different locations and calculated the data cost for every pixel as the shortest (Euclidean) distance in L*a*b* space to any of the color samples of the river delta. The start set is the upper boundary and the end set is the lower boundary of the image.

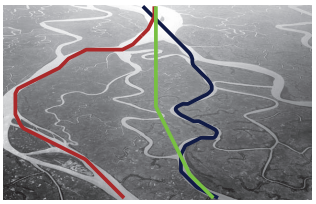
Figures 6a and 6b present the results. Curvature regularization yields a long path which does not turn much. No amount of length regularization is able to find that path as indicated by Figure 6a. The running times are compared in Table 1 for Dijkstra's and A*, and it is evident that large speed-ups are obtained with A*.

Figure 7 shows an experiment where some edges have been removed around the start and end sets in such a way that the optimal analytical solution is a circle of maximum radius. This experiment shows that low regularization radii introduce significant bias when minimizing the squared curvature. The first result that looked approximately like a circle was using 32-connectivity ($d = 4$).

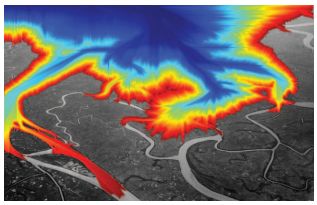
Finally, in Figure 10 an experiment indicating the dif-



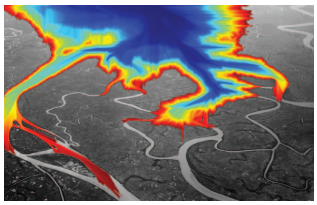
(a) Length regularization; mean time: 0.33 s.



(b) Curvature regularization; mean time: 5.0 s.



(c) Visit order using Dijkstra's.



(d) Visit order using A*.

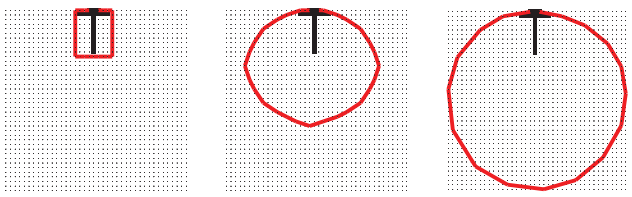
Figure 6: Segmentation of the Irrawaddy river delta in Burma. The underlying mesh has 220,443 points (373×591) and 3,509,756 edges (16-connected). For both the length and curvature regularization, we show the results with three different strengths of the regularization. The blue path corresponds to low strength, red stronger regularization, and green very strong regularization. Figures (c) and (d) show the order in which the nodes were visited for medium curvature with and without A*, from blue (early) to red (late). Photo by Jan Ševčík.

ference between a curve with curvature regularization and torsion regularization is given. High torsion regularization forces the curve to stay within a plane, but within the plane, the curve may have high length and curvature. The graph used for the torsion regularization had about 75 billion arcs. The technique of not storing the graph explicitly is of course imperative for such a graph.

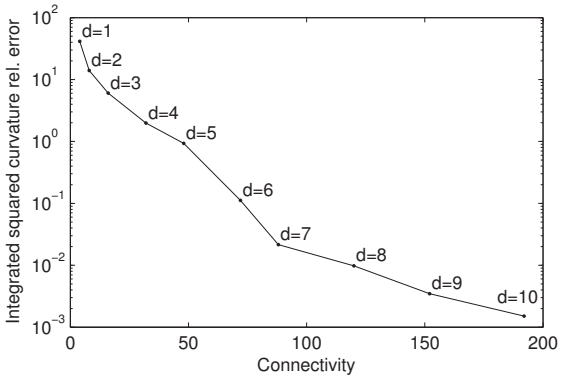
3.2. 2D Retinal Images

Automated segmentation of vessels in retinal images is an important problem in medical image analysis, which, for example, can be used when screening for diabetic retinopathy. We have investigated whether curvature regularization is useful for this task on a publicly available dataset [18]. Figure 8 shows the experiment. We iteratively computed shortest paths from a user-provided start point to any point on the image boundary. A curve could start anywhere along the previously found curves, but not end close to a previous end point.

We performed experiments with different amounts of length regularization (Figures 8b–8e). No or low length regularization resulted in noisy, wiggling paths. This problem expectedly disappeared for medium regularization, but sharp turns were still present in the solution (sharp turns usually change direction in the vessel tree). When using too high regularization (Figure 8e), the solution almost ignores the data term and prefers paths along the image boundary.



(a) $d = 1$ (b) $d = 4$ (c) $d = 10$



(d) Convergence of integrated squared curvature.

Figure 7: Circle experiment with a mesh size of 40×40. All points within a distance d of each other are connected. The start and end points have been chosen in such a way that the analytical solution when minimizing the integrated squared curvature is a circle with cost $\pi/10$.

In contrast, curvature regularization is able to more correctly capture the vessel tree. We knew this would be the case from a theoretical analysis; this experiment shows that the segmentation is feasible in practice.

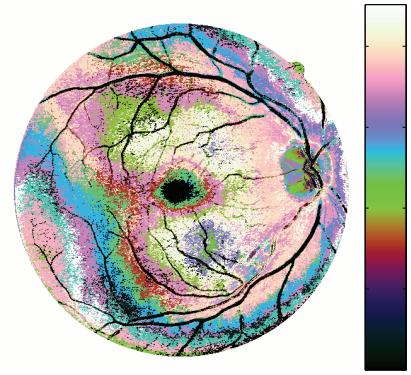
3.3. 3D Coronary Artery Centerline Extraction

Finding the centerlines of coronary arteries is of high clinical importance [14], but is very time consuming when performed manually. We use a public data set [14] consisting of 32 CT angiography scans to evaluate our method.

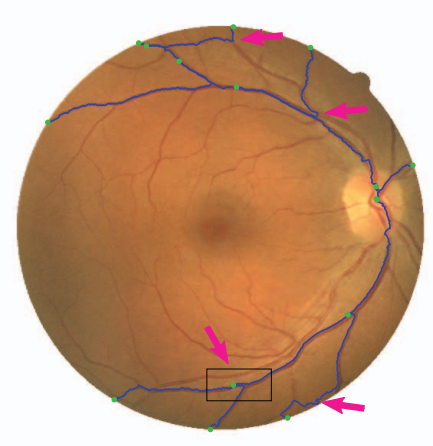
The CT volume size can be halved in each dimension without the loss of any information [5]. Each segmentation is initialized by manually specifying the start and end of each vessel – information which is included in the dataset. To further speed up the process we cut out a bounding box encapsulating the start and end set.

To model the vessel we adopt the speed image S from [5], which combines the probability of a voxel being a vessel, measured by "vesselness" [13] and a soft threshold of the image. We use $(1 - S)^2$ as our data cost.

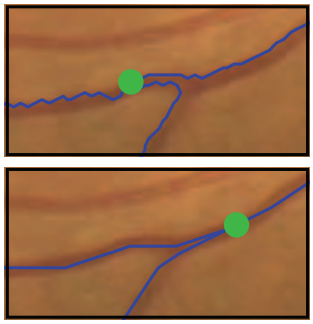
The accuracy of the segmentation is reported as overlap [14], which is the fraction of the segmented curve inside the ground truth vessel. The previously reported results in the database vary between 70-98% overlap, with different amounts of human interaction and model complexity. Most reported methods use some variant of shortest paths. Ta-



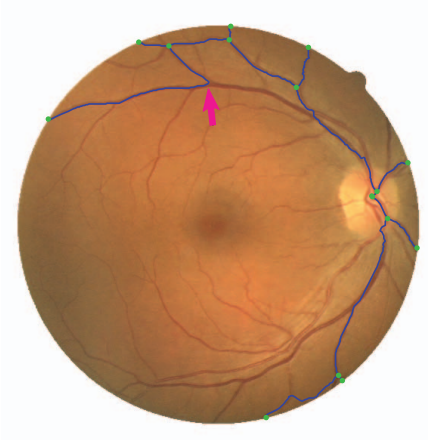
(a) Data term.



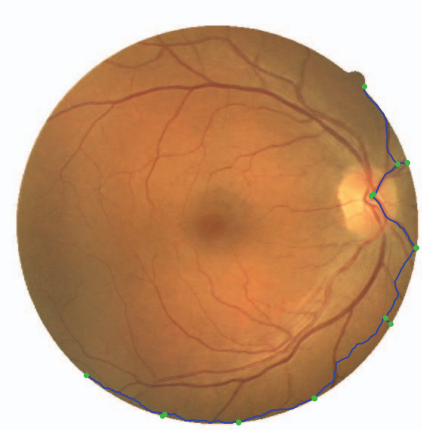
(b) $\rho = \sigma = 0$. Mean time: 0.41 s.



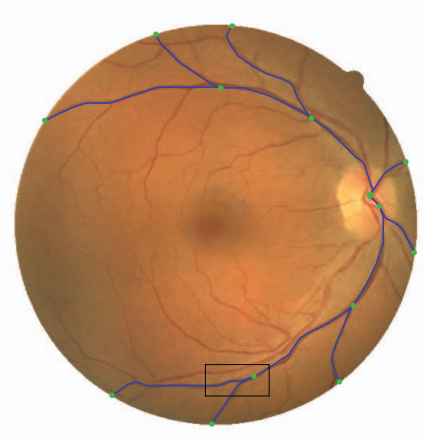
(c) Details of (b) and (f), respectively.



(d) $\rho = 0.1, \sigma = 0$. Mean time: 0.56 s.



(e) $\rho = 1, \sigma = 0$. Mean time: 0.66 s.



(f) $\rho = 0, \sigma = 100$. Mean time: 10.1 s.

Figure 8: Segmenting a vessel tree with 8 leaves in a 2D retinal image from the DRIVE [18] database. The green dots show the computed best starting points for the new branches. The underlying mesh has 329,960 points (584×565) and 10,496,766 edges (32-connected). The arrows point at sharp turns where the segmented vessel changes direction in the vessel tree and at particularly noisy parts. All length regularizations (b-d) have various issues and curvature regularization (e) finds a reasonable tree.

ble 2 presents quantitative result on the training data of [14] and compares length to curvature regularization. A example where curvature regularization outperforms length is given in Figure 9.

3.4. Multi-View Reconstruction of Space Curves

Curvature has previously been used to reconstruct space curves from multiple calibrated views of a static scene [6], but only using local optimization. Given start and end points, we are now able to reconstruct the curves optimally. Figure 11 shows an experiment on the same data as [6]. We reconstructed a tree iteratively in the same way as in Figure 8 and, as expected, length regularization introduces similar artifacts. Integrating the image along the projected 3D edge gives the edge cost for a single view. The data term we used for an edge is simply the maximum of the cost over all views.

	Overlap Median (mean)	Wins	Regularization	
Length	91% (74%)	10	$\rho = 0$	$(3 \cdot 10^{-4})$
Curv.	94% (72%)	11	$\sigma = 10^{-4}$	$(7.2 \cdot 10^{-3})$

Table 2: Length and curvature regularization for coronary vessel segmentation. We segmented each vessel in the training set of [14] with length and curvature regularization. Like in Figure 9 we tested different regularization strengths (10). For both length and curvature we choose the strength giving the best result and applied it to all vessels. The second column reports the number of vessels for which length and curvature performed best, respectively, not counting any vessel with less than 50% overlap (6 out of 28 vessels) or ties (5 vessels). The number of wins is calculated using the median regularization.

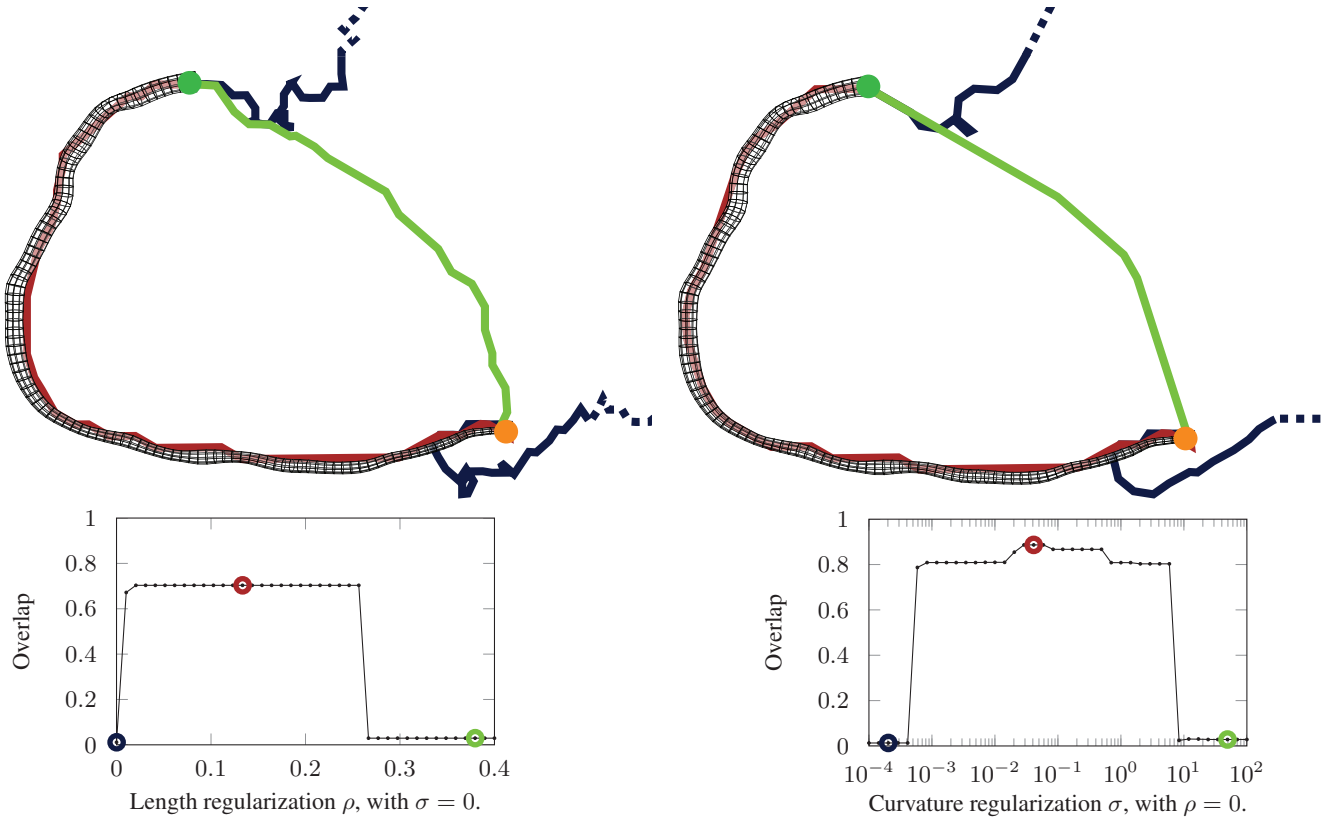


Figure 9: Example of coronary artery segmentation in 3D using length (left) and curvature (right). The start and end sets are indicated by orange and green dots, respectively. Each plot shows three different regularizations, where the color corresponds to the strengths displayed in the overlap plots. The black wireframe tube is the ground truth of the database [14]. Curvature regularization obtains a better solution (88.66% overlap) than any length regularization does (max. 70.33% overlap). The mesh has 841,662 points ($114 \times 107 \times 69$) and 60,271,574 edges. The average segmentation times were 1.8 s and 36.2 s for length and curvature, respectively.

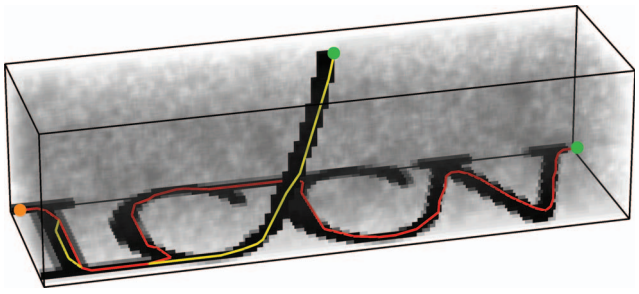
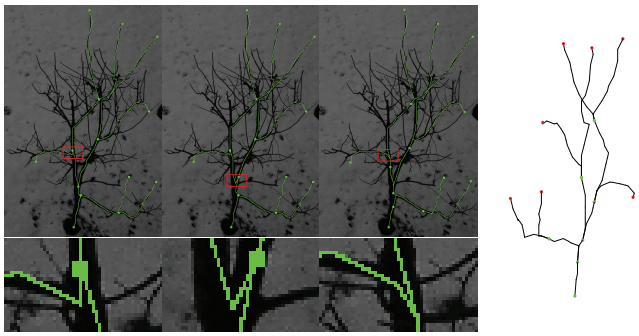


Figure 10: Synthetic 3D experiment for torsion with volume-rendered data cost. Darker regions correspond to lower cost. The start and end sets are indicated by orange and green dots, respectively. We run the segmentation with either very high curvature (yellow) or very high torsion regularization (red). The underlying mesh has 32,560 points ($22 \times 74 \times 20$), 4,118,200 edges (146-connected) and 538,717,792 edge pairs. Run times are 0.08 s for curvature and 10.6 s for torsion (harder problems will take longer time to solve).



(a) $\rho = 10$, $\sigma = 0$ (b) $\rho = 100$, $\sigma = 0$ (c) $\rho = 0$, $\sigma = 10$ (d) 3D

Figure 11: Reconstructing a tree in 3D from 4 different views (only one shown). Using length regularization gives similar artifacts (sharp turns) as in Figure 8. The underlying mesh has 125,000 points and 24,899,888 edges (218-connectivity). Run times are 4.8 minutes for length and 13.8 hours for curvature. Data from [6].

4. Conclusions

This paper has demonstrated the possibility of incorporating curvature and torsion to shortest path problems. The fact that the discretized problem converges to the underlying continuous problem follows from the fact that quadratic and cubic splines approximate second and third-order derivatives arbitrary well. We have also demonstrated convergence in practice (Figures 4 and 7).

Although all of our methods find the global optimum in polynomial time, using torsion can not be considered feasible, as we have only used it for quite small problems. On the other hand, using curvature is not that expensive for many problems and we believe we have demonstrated its usefulness for medical imaging problems, both in 2D (Figure 8) and in 3D (Figure 9).

The quality of the solution depends a lot on the connectivity of the mesh. Previous works have used 8-connectivity [20] and the highest we have seen is 16 [7] with approximate solutions. In contrast, we think that 16-connectivity is the bare minimum and we have used 32-connectivity for our two-dimensional medical experiments. As demonstrated in Figure 7, low connectivities introduce large discretization errors; these errors are also visible in the solution curves.

Future work. The "vesselness" measures [4, 13] commonly used for blood vessel segmentation use the eigenvalues of the Hessian. An interesting avenue for future research is to include the eigenvectors with anisotropic curvature, resulting in a direction-aware curvature regularization. This is also possible using our open-source framework[1].

Acknowledgments. We gratefully acknowledge funding from the Swedish Foundation for Strategic Research (FFL), European Research Council (grant no. 209480) and Swedish Research Council (grant no. 2012-4215).

References

[1] A. Amini, T. Weymouth, and R. Jain. Using dynamic programming for solving variational problems in vision. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 12(9):855–867, 1990.

[2] L. Cohen and R. Kimmel. Global minimum for active contour models: A minimal path approach. *Int. Journal Computer Vision*, 24(1):57–78, 1997.

[3] M. Fischler, J. Tenenbaum, and H. Wolf. Detection of roads and linear structures in low-resolution aerial imagery using a multisource knowledge integration technique. *Comput Graph Image Process*, 15(3):201–223, 1981.

[4] A. F. Frangi et al. Multiscale vessel enhancement filtering. In *MICCAI*, Cambridge MA, USA, 1998.

[5] O. Friman, C. Kühnel, and H.-O. Peitgen. Coronary centerline extraction using multiple hypothesis tracking and minimal paths. In *MICCAI*, New York, USA, 2008.

[6] F. Kahl and J. August. Multiview reconstruction of space curves. In *Int. Conf. Computer Vision*, Nice, France, 2003.

[7] M. Krueger, P. Delmas, and G. Gimel'farb. Robust and efficient object segmentation using pseudo-elastica. *Pattern Recognition Letters*, 2013. In press.

[8] D. Lesage, E. Angelini, I. Bloch, and G. Funka-Lea. A review of 3d vessel lumen segmentation techniques: models, features and extraction schemes. *Medical Image Analysis*, 13(6):819–845, 2009.

[9] S. Masnou. Disocclusion: A variational approach using level lines. *IEEE Trans. on Image Processing*, 11(2):68–76, 2002.

[10] D. Mumford. Elastica and computer vision. In C. Bajaj, editor, *Algebraic Geometry and its Applications*. Springer, 1994.

[11] M. Péchaud, R. Keriven, and G. Peyré. Extraction of tubular structures over an orientation domain. In *Conf. Computer Vision and Pattern Recognition*, Miami, USA, 2009.

[12] A. Pressley. *Elementary Differential Geometry*. Springer, second edition, 2010.

[13] Y. Sato et al. Three-dimensional multi-scale line filter for segmentation and visualization of curvilinear structures in medical images. *Medical image analysis*, 2(2):143–168, 1998.

[14] M. Schaap et al. Standardized evaluation methodology and reference database for evaluating coronary artery centerline extraction algorithms. *Medical Image Analysis*, 13/5:701–714, 2009.

[15] T. Schoenemann, F. Kahl, S. Masnou, and D. Cremers. A linear framework for region-based image segmentation and inpainting involving curvature penalization. *Int. Journal Computer Vision*, 99(1):53–68, 2012.

[16] T. Schoenemann, S. Masnou, and D. Cremers. The elastic ratio: Introducing curvature into ratio-based globally optimal image segmentation. *IEEE Trans. on Image Processing*, 20(9):2565–2581, 2011.

[17] A. Shashua and S. Ullman. Structural saliency: The detection of globally salient structures using a locally connected network. In *Int. Conf. Computer Vision*, Tampa Florida, USA, 1988.

[18] J. Staal, M. Abramoff, M. Niemeijer, M. Viergever, and B. van Ginneken. Ridge based vessel segmentation in color images of the retina. *IEEE Transactions on Medical Imaging*, 23(4):501–509, 2004.

[19] P. Strandmark and F. Kahl. Curvature regularization for curves and surfaces in a global optimization framework. In *EMMCVPR*, St Petersburg, Russia, 2011.

[20] H. Wang. G-wire: A livewire segmentation algorithm based on a generalized graph formulation. *Pattern Recognition Letters*, 26(13):2042–2051, 2005.

[21] O. Woodford, P. Torr, I. Reid, and A. Fitzgibbon. Global stereo reconstruction under second-order smoothness priors. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 31(12):2115–2128, 2009.

[1] https://github.com/PetterS/curve_extraction