Fast and Adaptive Pairwise Similarities for Graph Cuts-based Image Segmentation

Baris Sumengen
UC, Santa Barbara
Santa Barbara, CA 93106
sumengen@ece.ucsb.edu

Luca Bertelli
UC, Santa Barbara
Santa Barbara, CA 93106
lbertelli@ece.ucsb.edu

B. S. Manjunath
UC, Santa Barbara
Santa Barbara, CA 93106
manj@ece.ucsb.edu

Abstract

We introduce the use of geodesic distances and geodesic radius for calculating pairwise similarities needed in various graph cuts based methods. By using geodesics on an edge strength function we are able to calculate similarities between pixels in a more natural way. Our technique improves the speed and reliability of calculating similarities and leads to reasonably good image segmentation results. Our algorithm takes an edge strength function as its input and its speed is independent of the feature dimension or the distance measure used.

1. Introduction

Starting with the introduction of normalized cuts method [8], image segmentation using graph partitioning techniques has become very popular within the past few years. There are two main steps involved in this segmentation process: 1) building a similarity graph by calculating pairwise similarities between pixels, 2) partitioning this graph to obtain an image segmentation. In this paper we are proposing an efficient and effective way of calculating pairwise similarities. Our technique is based on finding geodesic distances between pixels using a variation of Fast Marching Method (FMM) [6].

Regardless of the cost function used for the graph cut, the image segmentation result is closely related to how pairwise similarities are generated. In the past, most effort has been on finding better ways of partitioning the graph. Designing better pairwise similarities did not receive as much attention. Traditionally similarity between pixels is reduced with increasing spatial and feature distances– $W = e^{-\frac{\|p_i - p_j\|^2}{\sigma_P} - \frac{\|\vec{F}_i - \vec{F}_j\|^2}{\sigma_F}}$, where p_i and $\vec{F}_i$ denote the location and feature vector of pixels. More recently the idea of intervening contour is introduced [4] that reduces similarity if there are image edges along a line connecting two pixels.

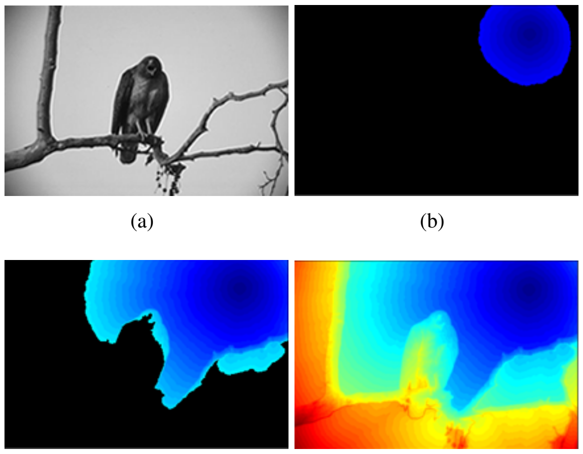
Our approach in this paper combines all these ideas in a very natural and mathematically sound way. As can be seen, intervening contours and other rules used in calculating similarities can be thought of as a crude way of estimating geodesics on a suitable manifold. Here we introduce a more formal way of estimating geodesics on an image. Intuitively our method can be explained as follows: 1) First an edge strength function is generated from the pixel features (gray-scale, color or texture). Values of this edge function represent the cost of crossing a pixel when connecting two pixels along a shortest path. We make sure that there is a minimum cost everywhere even if the edge strength is zero. This is to reduce the similarity if two pixels are spatially distant. 2) Calculate geodesics from each pixel to all pixels. Pairwise similarity is then a monotonically decreasing function with respect to the geodesic distances. In Fig. 1 the behavior with increasing geodesic radius is demonstrated.

Without loss of generality, in the rest of the paper, we focus our analysis and discussion to normalized cuts method due to its popularity. In principle, algorithms and methods we introduce in this paper are independent of the graph partitioning method used.

The rest of the paper is organized as follows. In Section 2 we review Fast Marching Method [6] and our implementation of it. In Section 3 we introduce an extension of Fast Marching to calculate all-pairs geodesics and discuss how we generate pairwise similarities from geodesics. Section 4 compares our methods with the previous methods of generating similarities and show some image segmentation results. In Section 5 we introduce our multi resolution approach. We conclude in Section 6.

2. Fast Marching Method (FMM) for Geodesic Distance Calculation

A fast algorithm for finding shortest paths on a graph is Dijkstra's algorithm [3]. If we create a graph from an image using 4-neighbors of pixels, the shortest paths on graph



(a) (b)

(c) (d)

Figure 1. a) original image of size 321×481, b) local geodesics up to geodesic radius $\alpha = 100$ around the point $(50, 400)$, c) local geodesics up to geodesic radius $\alpha = 300$. d) Global geodesics for the whole image.

will calculate L_1 distance, *e.g.* the diagonal of a unit square is 2 (instead of $\sqrt{2}$) regardless of how much we refine the grid. This is because Dijkstra's algorithm is designed to work on a graph and does neither consider that we are on a rectangular grid, nor tries to approximate the shortest path in continuous domain. In contrast, Fast Marching Method introduced by Sethian [6] calculates a first order approximation of the geodesics in continuous domain.

While conceptually Fast Marching is quite different from Dijkstra's algorithm, implementation of both algorithm are very similar. Our implementation of Fast Marching uses a heap based priority queue due to its simplicity and has a complexity of $O(n \log n)$ for all geodesics from a single point, where n is the number of pixels in the image. Very recently Yatziv *et al*. introduced [9] an $O(n)$ algorithm to calculate Fast Marching by using an untidy priority queue. For the rest of the discussion we will assume that single source Fast Marching has $O(n)$ complexity.

Fast Marching Method is an efficient algorithm to calculate geodesic distances by solving an *eikonal equation*. The general expression of the eikonal equation in 2D is the following:

$$|\nabla T(x,y)| = F(x,y). \tag{1}$$

The function $F(x,y)$ gives the local weights used in arc length calculation: $ds^2 = F^2(x,y)(dx^2 + dy^2)$. If F is constant over the whole domain, the solution of the eikonal equation is exactly the Euclidean distance. In our case F is an edge strength function and the solution T gives us the geodesic distances. The boundary conditions are given as a set of points for which $T(x,y)$ is known.

In our implementation of FMM we utilize three static matrices–T, S and P–and one priority queue Q. In the matrix T we store the solution of the eikonal equation (geodesic distances). The matrix S tells us the state of each point: 1 corresponds to *decided*, 0 to *far away* and -1 to *close*. The matrix P contains the positions of *close* points in the priority queue. This allows $O(1)$ access to the points in the priority queue when we need to update their distances.

To initialize the algorithm we set $S(x,y) = 0$ and $T(x,y) = \infty \ \forall \ x,y$. Then for every point (x,y), whose value $T_0(x,y)$ is known, we set $T(x,y) = T_0(x,y)$, we update their state as *decided* $(S = 1)$ and we add the 4-neighbors to the priority queue, updating their state to *close* $(S = -1)$. For every point in the queue we solve a quadratic equation to update their value and their position in the queue.

After this initialization phase, the main cycle of the algorithm begins:

Algorithm 1 Main Loop of Fast Marching Algorithm.

```
while Q is not empty do
    Extract q=minimum(Q).
    Add it to the decided Set ($S(x_q, y_q) = 1$).
    Set the state of the neighbors of q as close ($S = -1$).
    Add 4-neighbors of $q$ to $Q$ or if a neighbor is already
    in the queue then update its distance accordingly.
end while
```

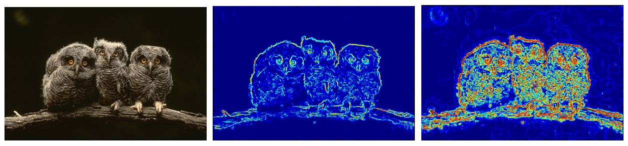
3. Pairwise Geodesics as Similarities

The Fast Marching algorithm described in the previous section can be used to find the geodesic distances of one point (x_0, y_0) of the image to all the other points, simply by setting the initial condition $T(x_0, y_0) = 0$ and solving (1) for T. In our particular setup we use an edge strength function as local weight function F:

$$F = \frac{k(\beta + \max(e))e}{\max(e)(\beta + e)} + 1, \tag{2}$$

where β is a constant parameter, which controls the enhancement or attenuation of the weak edges with respect to the strong ones. Fig 2 shows change in the edge function with respect to β. e is the gradient magnitude of the smoothed image. k is chosen such that $(k+1)$ is the cost of crossing the strongest edge. The (minimum) cost of moving by one pixel where there are no edges is 1. We normalize e between 0 and 1 using $e_N = \frac{e}{\beta + e}$. Afterwards, rescaling this function between 1 and $(k+1)$ leads to (2).

This 2D function $F(x,y)$ is calculated only once at the beginning and then used for the evaluation of all the pairwise geodesics. Running FMM for each pixel results in a distance matrix of size $hw \times hw$ where h and w are height and width of the input image. Commonly, due to memory



(a) (b) (c)

Figure 2. a) An image of owls, b) edge strength F evaluated with $\beta = 100$, c) edge strength F with $\beta = 10$. k is selected as 50.

and computational limitations similarities are limited to local neighborhoods of each pixel. In our case we limit the calculation of distances and similarities to a geodesic radius α. This can be achieved simply by checking if the distance is less than α when we extract a new element from the priority queue and stopping FMM otherwise. Geodesic distances are then converted into similarities using $s = 1 - \frac{d}{\alpha}$. As can be seen, by adjusting α we can control the sparsity of the similarity matrix. Since we are running FMM n times, the complexity of the all-pairs geodesics algorithm is $O(n^2)$. In the rest of this section we will discuss important implementation and speed issues.

3.1. Implementation Issues with (Matlab) Sparse Matrices and Complexity Analysis

Usually pairwise similarities are passed to the graph partitioning algorithm as a sparse matrix. This sparse matrix is of size $hw \times hw$. To form the indices of the sparse matrix all pixels of the image are ordered as a one dimensional array with an index $i = r + h(c - 1)$ where r is row and c is the column index of the image. In our case, we need to pay attention to this process since the indices of the elements that FMM returns are not ordered. In the rest of this section, we discuss the complexity of filling a Matlab sparse matrix using similarities returned by the FMM method. Most sparse matrix packages and representations use the same or similar format as Matlab's, so the discussion in this section is widely applicable.

Matlab utilizes three one dimensional arrays–s_1, s_2, and s_3–to store the contents of a 2D sparse matrix. s_1 has length N_{max}, where N_{max} is the number of nonzero elements in the sparse matrix, and contains the values of the non zero elements ordered in a column by column scan. s_2 of length N_{max}, contains the row indices of the nonzero elements (again visited in column by column order). Elements of s_3 indicate the number of non zero elements up to the start of each column. Because of this notation, we need to order the indices of FMM results before inserting them to the sparse matrix.

Since we are dealing with sorting integers (the position indices of the pixels), which are bounded up by $n = hw$ and down by 1, *Distribution Sort* [1, Section 3.6] can be used for sorting with $O(n)$ complexity. As a consequence, single source FMM still has $O(n)$ complexity allowing us to keep $O(n^2)$ complexity of the all-pairs algorithm.

3.2. Exploiting Symmetry of the Similarities

In our discussion so far we haven't considered the fact that pairwise similarities are symmetric. Say FMM is already run for pixel p_i. When computing FMM for another pixel p_j, we already know the geodesic distance from p_j to p_i and should not recalculate it. As discussed in Section 2, FMM can accept a number of *decided* points for which the geodesic distances are known (in addition to the distance at starting point p_j, which is 0). By reusing previously calculated geodesic distances in the following runs of FMM we can easily avoid recalculating the same distances. This process requires some amount of bookkeeping. We use a length n array of linked lists to keep track of previously calculated distances.

FMM is an approximation of the geodesic distances. Therefore, in practice the distance from p_i to p_j comes out slightly different than the distance from p_j to p_i due to the discretization errors. As a consequence, an implementation of all-pairs geodesics without exploiting the symmetry would not necessarily result in a symmetric sparse matrix, which is not desired. However, by exploiting the symmetry of geodesic distances we are also enforcing the sparse matrix to be symmetric.

Our experiments show that symmetric algorithm described in this section results in about two times increase in speed, which shows that there is very little overhead in our symmetric algorithm.

4. Comparison of Geodesic Similarities with Radial Similarities

Normalized cuts method [8] is based on finding smallest eigenvalues starting with the second smallest up to the k'th eigenvalue and the corresponding eigenvectors of the matrix $D^{-1/2}(D - W)D^{-1/2}$, where W is the similarity matrix and D is a diagonal matrix with the diagonals $d_i = \sum_j w(i, j)$. For the efficiency of the eigensolver, it is desirable that the matrix is sparse. For this purpose, normalized cuts method usually restrict similarities to small neighborhoods such that $w(s_i, s_j) = 0$ if $\|p_i - p_j\| > r$, where $\|.\|$ is the spatial distance and r is a constant, *e.g.* 10 pixels. In this case, there are equal number of nonzero similarities for each pixel. In practice, for a radius r similarities are calculated in a square of size $2r \times 2r$.

In our case, we also limit ourself to local similarities by stopping the Fast Marching algorithm when a geodesic radius of α is reached. For an indication we give the radius for the equivalent density in our result. Since the number of pixels visited by FMM depends on F, the total number of similarity connections from a pixel vary from pixel to pixel.

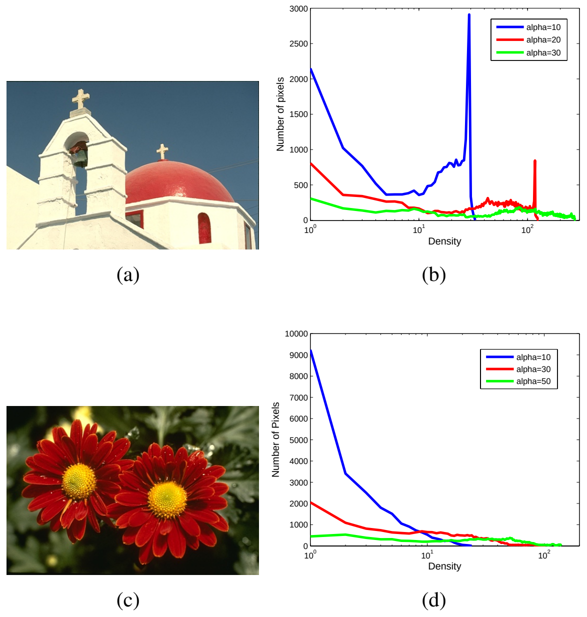


Figure 3. Histogram of density of similarities shown on a logarithmic grid. Density is defined as the number of nonzero similarities given a pixel. a) An image of a church with large homogenous regions, b) corresponding density histogram. c) A flower image, which contains a lot of edges, d) corresponding density histogram.

Let us define *density* for a pixel p_j as the number of nonzero similarities from p_j to all the pixels. Fig. 3 shows the distribution of densities for two images: a church image with large homogenous areas, and a flower images with large number of edges. It is expected that the pixels of the church image has larger densities than the flower image for a given α. In Fig. 3(b) we see that for $\alpha = 10$ and $\alpha = 20$, there is a peak towards right. This shows that we have stopped Fast Marching too early. Increasing α to 30 shows that this peak disappears and the densities are more evenly distributed. In contrast, for the flower image we don't observe such a peak even for $\alpha = 10$, which suggests that a large α may not be necessary.

Table 1 and Fig. 4 display and compare the calculation times for Fast Marching-based similarities, simple gray-scale based ($W_I = e^{-\|I_i - I_j\|/\sigma_I}$) similarities, and similarities based on intervening contours (W_C). Implementations are in C++, compiled as a mex dll and called from Matlab. We obtained a fairly optimized implementation of intervening contours based similarities (W_C) from Jianbo Shi's web site: http://www.cis.upenn.edu/~jshi/software/. In this case we are timing the C++ function *affinityic()*, which accepts an edge function and a list of pairwise coordinates for which W_C is calculated[1]. Our implementation of Fast Marching Method is based on $O(n \log n)$ algorithm. Similarities based on gray scale values (W_I) are calculated using our hand optimized code. We avoid the spatial term $e^{-\frac{\|p_i - p_j\|^2}{\sigma_P}}$ since its complexity is negligible. Symmetry is not exploited for any of the similarities calculated here.

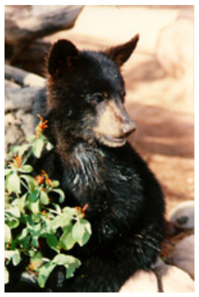
[1]We disable the random sub sampling to avoid heuristics in our benchmarks.

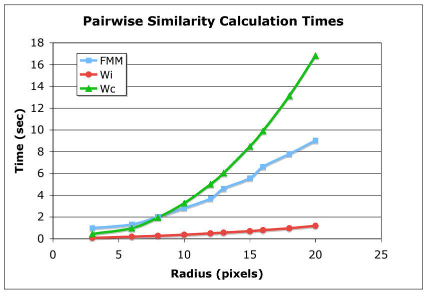
Image Size 255 × 148					
α	avg. density	radius	FMM	W_I	W_C
10	73.65	4	9.98s	1.21s	2.30s
20	264.19	8	14.16s	1.58s	8.16s
30	547.66	12	20.27s	2.43s	21.68s
40	909.28	15	27.97s	3.70s	37.76s
50	1331.41	18	36.84s	4.94s	Failed
Image Size 112 × 74					
α	avg. density	radius	FMM	W_I	W_C
10	41.05	3	0.99s	0.10s	0.45s
20	134.17	6	1.32s	0.19s	0.98s
30	260.21	8	2.02s	0.27s	1.96s
40	411.04	10	2.82s	0.38s	3.28s
50	568.59	12	3.69s	0.50s	5.02s
60	727.95	14	4.60s	0.57s	6.05s
70	923.53	15	5.57s	0.71s	8.49s
80	1087.1	16	6.62s	0.80s	9.91s
90	1309.6	18	7.79s	0.97s	13.13s
100	1545.9	20	9.03s	1.19s	16.82s

Table 1. Running times for calculating pairwise similarities. Benchmarks are run on the bear image in Fig. 4(a). Fast Marching-based (FMM) similarities, simple gray-scale based ($W_I = e^{-|I_i - I_j|}$) similarities, and similarities based on intervening contours (W_C) are given.

As can be seen in Fig. 4, generating W_I is quite fast. On the other hand, note that these timings are for gray scale values and Euclidean distance. Higher dimensional features such as color or texture and more complicated distance measures such as *earth mover's distance* [5] may increase the time to create W_I significantly. For geodesic similarities this is not an issue. In our case we only need to create the edge function once. FMM is independent of the dimensionality of the feature vectors and the distance measure used. Similarly, W_C is also independent of the feature vectors or distance measures used and just require an edge function as its input. Fig. 4(b) shows that calculating W_C is slower than FMM-based method and time increases much faster when radius is increased. Note that, in [2] the suggested way of calculating pairwise similarities is a combination of W_C and W_I and formulated as: $W = \sqrt{W_I \times W_C} + 0.1 W_C$.

Another observation is that the running times and the number of iterations required for the eigensolver is less for FMM-based pairwise similarities than the intervening contour-based pairwise similarities. Fig. 5(b) shows that, for the sea star image of Fig. 5(a) even though FMM-based





(a) (b)

Figure 4. Comparison of computation times of the pairwise similarities. a) An image of bear of size 112×72. b) Corresponding plots of pairwise similarity calculation time with respect to increasing radius (increasing density) for Fast Marching-based (FMM) similarities, simple gray-scale based ($W_I = e^{-\|I_i - I_j\|^2/\sigma_I}$) similarities, and similarities based on intervening contours (W_C).

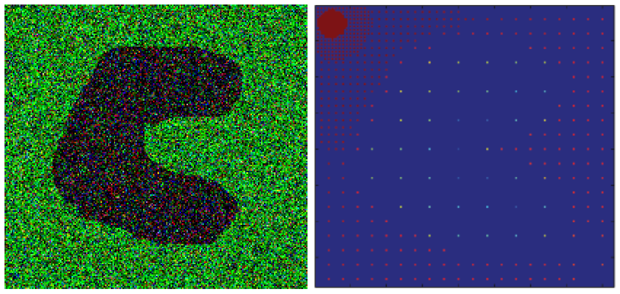
similarity matrix is 100 times more dense, it takes less time to find the eigenvectors. This is mainly because our method creates better connections on the graph and diffusion on the graph converges quicker with less iterations as seen in Fig. 5(c).

Fig. 6 shows some experimental results. We used normalized cuts-based clustering software from http://www.cis.upenn.edu/∼jshi/software/ to obtain our similarities that are used to generate the results. Results are obtained on a set of images of size 128×192, with various parameter configurations. The edge strength function, which is the input of the FMM, is computed in the CIE-L*a*b* color space. We used the algorithm from [10] to calculate color gradients of the smoothed image. The segmentation results are somewhat better but very similar to the intervening countours based approach in terms of accuracy of capturing the region boundaries. In the next section we will show that a multi-resolution approach based on FMM will significantly improve the efficiency and effectiveness of the segmentations.

5. Multi-scale Geodesic Similarities

It is well known that storing and processing large number of pairwise similarities is expensive. It is also known that segmentation results improve with increasing radius (decreasing sparsity). On the bear image of size 112×72, eigensolver goes out of memory if we increase α more than 100 (equivalent radius is 20). Multi-scale approaches [7, 2] are proposed in the past to address this issue.

In our case we generate an adaptive grid around each pixel so that similarities are sampled with increasing sparsity as the geodesic radius increases. Again we stop at α but we are able to increase alpha much further with high sparsity. Our algorithm is as follows. The inputs to the algorithm are α_0 and α, where $\alpha_0 \ll \alpha$. Given a pixel at (x, y), FMM is run up until α_0 at which point k pairwise similarities are generated on the dense grid. We reduce the grid resolution by two and evolve FMM by k point on the coarser grid. We repeat this process and continuously coarsen the grid. Values from the fine grid are reused for the coarser grid so that there is no computational overhead. We stop either when we reach geodesic radius α or when we reach the image boundary. Fig 9 shows a comparison of single and multi scale approaches. Fig 7, 8, and 10 show some examples of the multi-scale grid and segmentation results.



(a) (b)

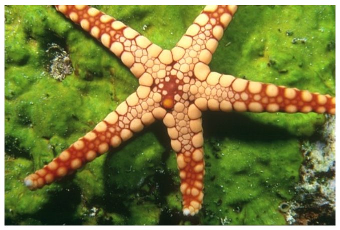
Figure 7. a) A synthetic image of size 208×195, b) multi-scale geodesic similarities for pixel $(10, 10)$.

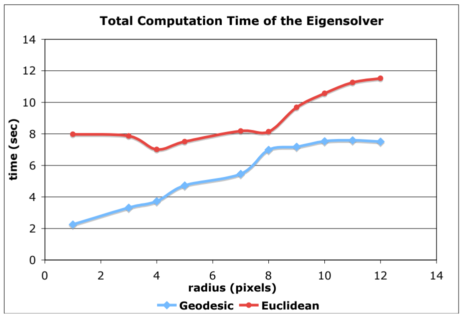
In [2], three characteristics of pairwise similarities are observed:

- *Similarities decrease as the distance between pixels are increasing.* Our method automatically handles this as we have a minimum cost of traveling the grid.
- *There is less variance for short and long range connections among themselves but a large variance in mid-range connections.*. Consider a binary black rectangle (size 20 by 50) image on a white background. For a pixel at the center, short connection (less than 10) and long connection (more than 25) won't have much variance, but mid range connections vary for euclidean radii. Geodesic radii handle this case naturally.
- *Variance in similarities between pixels in one small neighborhood to pixels in another small neighborhood decreases as the two neighborhoods are further apart from each other.* It should be noted that our coarsening strategy fits into this observation.

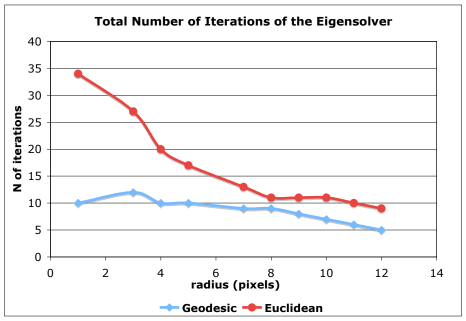
6. Discussion and Conclusion

In this paper we introduced the use of geodesic distances and geodesic radius for pairwise similarity calculation. The computation of these distances between pixels is based upon a variation of the Fast Marching Method. An


(a)

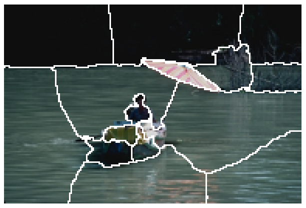


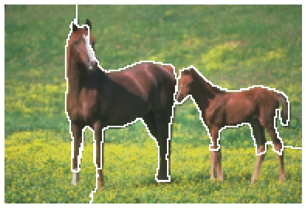
(b)

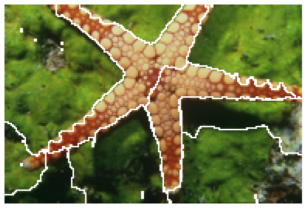


(c)

Figure 5. Efficiency of the eigensolver is compared for similarities calculated using FMM and using intervening contours (W_C) a) An image of a sea star of size 96×144. b) Running time of the Matlab's sparse eigensolver with increasing radius (descreasing sparsity). 10 eigenvectors are returned. c) Number of iterations needed for the eigensolver with increasing radius (descreasing sparsity).


(a) $\alpha = 20$, 15 segments


(b) $\alpha = 30$, 5 segments


(c) $\alpha = 30$, 10 segments

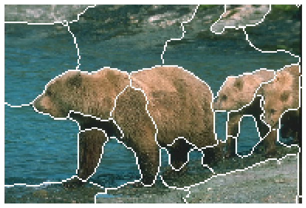

(d) $\alpha = 30$, 25 segments

Figure 6. Segmentation results using geodesic pairwise similarities within normalized cut framework. (Images are best viewed in color)

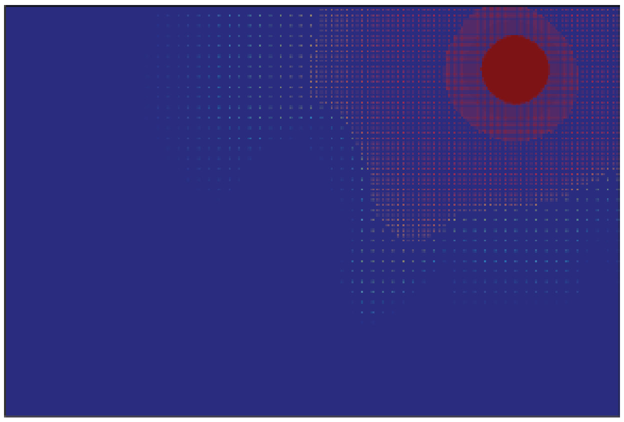


Figure 8. Multi-scale geodesic similarities for the bird image in Fig. 1a) on the coarse grid.

edge strength function, representing the cost of crossing a pixel, is the input of the FMM which allows us to compute pairwise similarity as a monotonically decreasing function with respect to the geodesics distances. Applying then a graph partitioning method on the similarity matrix we get the desired image segmentation.

As shown in Section 4 the calculation of Fast Marching-based similarities is faster compared to Intervening Contours-based, and independent of the dimensionality of the feature vector or the distance measure used. It also reduces the running time and the number of iterations of the graph partitioning eigensolver. Our multi-scale approach offered further speed and adaptivity in similarity calculations. In conclusion we showed that our technique improves speed and flexibility of similarity calculation.

References

[1] L. Ammeraal. *Algorithms and Data Structures in C++*. John Wiley and Sons, 1996. 3

[2] F. Benezit, T. Cour, and J. Shi. Spectral segmentation with multi-scale graph decomposition. In *CVPR*, 2005. 4, 5

[3] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1:269271, 1959. 1

[4] J. Malik, S. Belongie, J. Shi, and T. Leung. Textons, contours and regions: cue integration in image segmentation. In *ICCV*, pages 918–925, September 1999. 1

[5] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, pages 99–121, November 2000. 4

[6] J. A. Sethian. A marching level set method for monotonically advancing fronts. *Proc. Nat. Acad. Sci.*, 93(4), 1996. 1, 2

[7] E. Sharon, A. Brandt, and R. Basri. Fast multiscale image segmentation. In *CVPR*, pages 70–7, 2000. 5

[8] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 888–905, August 2000. 1, 3

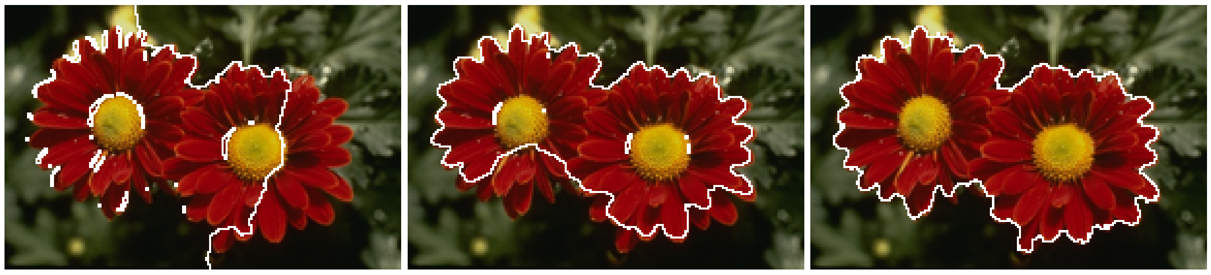
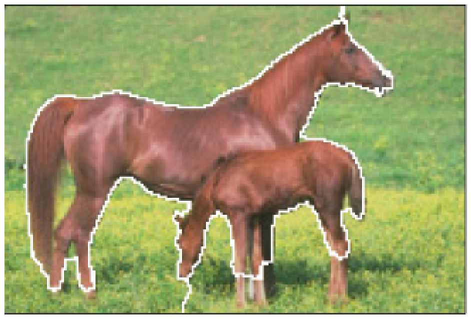

(a) (b) (c)

Figure 9. Comparison between single and multi scale methods. Segmentations are conducted using two segments. a) and b) show results in single resolution and c) shows multi resolution. a) has density 192 per pixel with $\alpha = 23$, b) has density 295 with $\alpha = 30$, which is the memory limit for the eigensolver to function. c) has density of 184 with $\alpha_0 = 10$ and $\alpha = 400$.


(a)

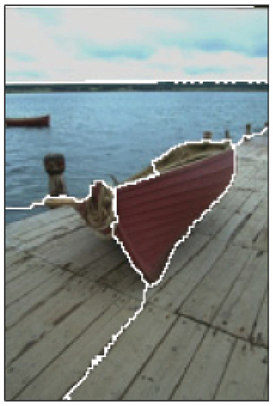

(b)

Figure 10. a) segmentation of a horse image (3 segments, $\alpha_0 = 10$, $\alpha = 200$, average density is 275), b) segmentation of a boat image (4 segments, $\alpha_0 = 10$, $\alpha = 150$, average density is 405).

[9] L. Yatziv, A. Bartesaghi, and G. Sapiro. O(n) implementation of the fast marching algorithm. *Journal of Computational Physics*, September 2005 (In Press, Available online). 2

[10] S. D. Zenzo. A note on the gradient of a multi-image. *Computer Vision, Graphics and Image Processing*, pages 402–407, 1986. 5