Random Walks for Interactive Organ Segmentation in Two and Three Dimensions: Implementation and Validation

Leo Grady[1], Thomas Schiwietz[1], Shmuel Aharon[1], and Rüdiger Westermann[2]

[1] Department of Imaging and Visualization
Siemens Corporate Research
755 College Rd. East
Princeton, NJ, USA
[2] Technische Universität München
Lehrstuhl für Informatik 15
Boltzmannstrasse 3
85748 Garching, Germany

Abstract. A new approach to interactive segmentation based on random walks was recently introduced that shows promise for allowing physicians more flexibility to segment arbitrary objects in an image. This report has two goals: To introduce a novel computational method for applying the random walker algorithm in 2D/3D using the Graphics Processing Unit (GPU) and to provide quantitative validation studies of this algorithm relative to different targets, imaging modalities and interaction strategies.

1 Introduction

A general-purpose, automatic, segmentation engine for medical images is extremely challenging due to the drastic changes in image data as a result of pathology and changes in radiologist preference. Therefore, efforts have continued toward providing *interactive* tools that allow a physician to quickly obtain an image/volume segmentation meeting their specific goals and criteria. Recently, a promising new interactive approach was introduced that uses random walks to define segmentation boundaries given user-placed seeds indicating K objects [1] (for arbitrary K). It was shown in [1] that this random walks algorithm is robust to weak boundaries and image noise. With the theory developed in [1], the current work introduces a faster, GPU-based, method of computation and offers a quantitative validation of the segmentations produced by this method.

The major computational hurdle of the random walker algorithm in [1] is the solution to a sparse system of equations for which a generic conjugate-gradients approach is employed. Although achieving reasonable speeds for moderately sized images (four seconds for a 256×256 2D image), the size of modern medical volumes requires a more efficient implementation to achieve an interactive speed. Therefore, we introduce a GPU-based implementation that offers over an order of magnitude speed increase for the processing of 2D and 3D datasets.

Validation of a general-purpose, interactive segmentation tool is difficult. Since this tool will provide an arbitrary segmentation with enough user interaction (i.e., if the user seeds every pixel), the main concerns are: 1) How sensitive are the results to exact seed placement? 2) How sensitive are the results to the quantity of seeds placed? 3) How much time, both user and computer, is required to perform a segmentation? 4) What is the subjective quality of the segmentations across different imaging modalities and different segmentation targets? These are the four questions that we address in the validation section.

The major interactive techniques for general-purpose organ segmentation are graph cuts, intelligent scissors and level sets. Graph cuts [2] treat the image as a graph where each pixel is associated with a node and a lattice edge structure is imposed, weighted to reflect intensity changes. Although performing well in many situations, there are a few concerns associated with this technique. For example, if a small number of seeds are used, the algorithm will often return the smallest cut as the cut that minimally separates the seeds from the rest of the image. Therefore, a user often has to continue placing seeds in order to overcome this "small cut" problem. Additionally, the K-way graph cuts problem is NP-Hard, requiring use of a heuristic to obtain a solution. The intelligent scissors algorithm [3] again views the image as a graph and employs Dijsktra's algorithm to compute the shortest path between user-defined points, treating this path as the object boundary. Unfortunately, a low-contrast or noisy boundary may require the specification of many points and the algorithm is inapplicable to 3D boundaries. Although the family of active contours and level sets is large [4] a user is generally asked to place a contour near the desired boundary and the algorithm evolves the boundary to a local energy minimum. The main problems with level set methods are difficulty of implementation (often requiring specification of several free parameters) and difficulty in fixing an incorrect solution, especially if the desired contour does not correspond to a local energy minimum.

This paper is organized as follows: In Section 2 we review the random walker algorithm and Section 3 details our novel GPU implementation in 2D/3D. Section 4 provides the results of our validation studies with respect to the questions raised above. Section 5 follows with a conclusion.

2 Random Walks for Image Segmentation

In this section we review the random walker image segmentation algorithm introduced in [1]. In the case of two labels, we determine the label for a non-seed pixel by asking: Given a random walker starting at this pixel, what is the probability that the random walker will reach a foreground seed before it reaches a background seed? If this probability is above one-half, then we assign this pixel to the foreground and if it is below one-half, we assign this pixel to the background. If more than two labels are used, the pixel is assigned the label for which a random walker is most likely to reach first. In [1], a Gaussian weighting function was used to represent the image structure as random walker biases (i.e., edge weights). This function has a single free parameter (representing the

only free parameter in the algorithm), β, that was set to $\beta = 1500$ for all 2D experiments and $\beta = 4000$ for all 3D experiments.

The equivalence between the random walker problem and the Dirichlet problem from potential theory was exploited in [1] to transform the computation of the random walker probabilities into the solution to system of linear equations. Although the system is sparse, symmetric and positive definite, allowing application of fast solvers like conjugate gradients, interactive speeds were not achieved for high-resolution images in [1]. For this reason, we present in the next section a GPU-based method that the solves the linear system (i.e., calculates the random walker probabilities) at interactive speeds for higher-resolution images and volumes.

3 GPU implementation

Commodity graphics cards have been successfully used in recent years to enhance the speed of computation for image segmentation algorithms [5, 6]. The computation of the random walker algorithm fits perfectly with the GPU in three respects: 1) All necessary linear algebra operators are computed extremely fast due to the inherit parallelism of the GPU and efficient caching, 2) Since a GPU processes four channels (RGBA), each channel may be used to represent a set of probabilities, allowing four labels to be solved simultaneously, 3) The segmentation may be continuously updated on the screen for the benefit of the user. An additional benefit of simultaneously solving the system of equations for multiple labels is that one may stop the computation when three of the four labels have converged and simply subtract the converged probabilities from unity to obtain the probabilities for the unconverged label.

Either a 2D or 3D dataset may be processed using the GPU. However, the limited size of the on-board memory of today's GPUs (256MB) constrains us to images with a resolution of 1024×1024 or volumes of resolution $128 \times 128 \times 128$. The implementation of a conjugate gradients method requires only two non-trivial operations: a sparse-matrix vector multiply and a vector inner product. The sparse-matrix vector multiply is described below and the vector inner product is described by Bolz *et al.* [7] and Krüger *et al.* [8]. Each row of the 2D-Laplacian matrix represents one pixel and the four values on the sub-diagonals indicate the weights of that pixel to each of its neighbors. Since the diagonal contains redundant information (i.e., the diagonal entry equals the negative sum of the off-diagonals), we can represent the Laplacian matrix as a 2D, four-channels, texture with size equal to the number of pixels in the image. The matrix-vector multiplication is therefore executed by multiplying the four channels of a given pixel (row) by the values of the four neighboring pixels of the vector. The diagonal value of the matrix is retrieved by summing and negating the four channel values. This value is then multiplied by the corresponding vector pixel. The five multiplication values are then summed together to provide the current output vector element results.

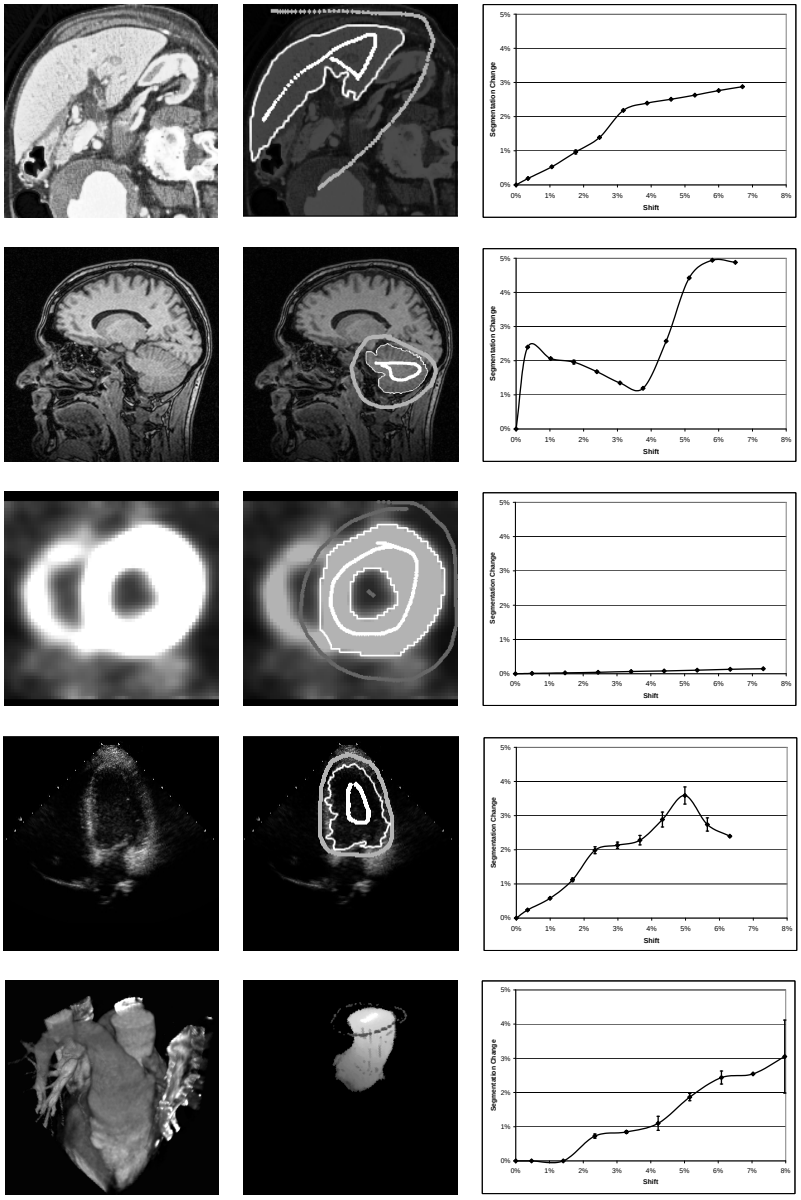


Fig. 1. Sensitivity analysis of segmentation to random shifts of seed placement in direction and magnitude over 1,000 trials. Left: Original 2D datasets from each of the four major imaging modalities and a 3D (CT) dataset. Middle: Foreground and background seeds are given by the gray markers. The white line represents the initial segmentation boundary (in 2D datasets). Right: Experimental results. The x-axis represents the ratio of the shift measured in pixels to the horiztonal image resolution. The y-axis represents the ratio of the pixels that switched label to the number of pixels originally labeled foreground. The plotted line represents the mean, with error bar indicating one standard error of the mean.

The Laplacian matrix for a 3D lattice has six sub-diagonals. However, due to matrix symmetry, one need only store three out of six sub-diagonals. The other three sub-diagonals can be retrieved by sampling neighboring matrix entries. In 2D and 3D a matrix-vector operation requires one rendering pass only. In 3D we put all slices of a volume in one 2D flat texture side-by-side. Operations on a flat texture have proven to be much faster than on a stack of slices (textures) [9].

4 Validation

The original exposition of the random walker technique demonstrated that the algorithm is capable of finding weak (i.e., low-contrast or no-contrast) boundaries, behaving robustly to noise and giving good quality results. However, since a user may achieve an arbitrary segmentation with the placement of enough seeds, a strict error measure is practically meaningless. An ideal interactive segmentation algorithm would not require an excess of seeds or require user precision to carefully choose the seed locations. An additional criterion is obviously speed of computation and user time. Therefore, in this section we provide a quantitative validation of the algorithm by studying the following questions: 1) How sensitive are the results to exact seed placement? 2) How sensitive are the results to the quantity of seeds placed? 3) How much time, both user and computer, is required to perform a segmentation? We examine the first two questions with a 2D example from each of the major imaging modalities of MR, CT, PET and Ultrasound as well as a 3D (CT) volume. The third question is addressed by applying the algorithm to a range of targets in images of differing image modalities. For simplicity, all of these experiments were conducted on a lattice with a 4-connected topology using a two-label scenario. However, these results should extend to the multilabel setting since the probabilities for each label are computed by effectively viewing the label as the foreground opposed to the background of all of the other labels.

Obviously, the segmentation obtained with the random walker algorithm will depend on the location of the seeds. If this were not true, no seeds would need to be placed. However, a small difference in the placement of the seeds within the same object should not result in a significant change in the computed segmentation. Ideally, the user should be free from a requirement that the seeds are drawn very carefully or placed in prescribed locations. We performed five experiments in seed placement using a 2D example from each of the four imaging modalities described above and a 3D (CT) dataset. For each of these seed placements, the locations of the foreground seeds were shifted (as a group) in a random direction with random magnitude. Although no range of magnitudes was pre-established, we rejected any perturbation that would have moved foreground seeds into the background. After the seed placements were perturbed, the segmentation was recomputed and the change in pixel labeling was reported as the ratio of pixels that switched labels to the number of pixels originally la-

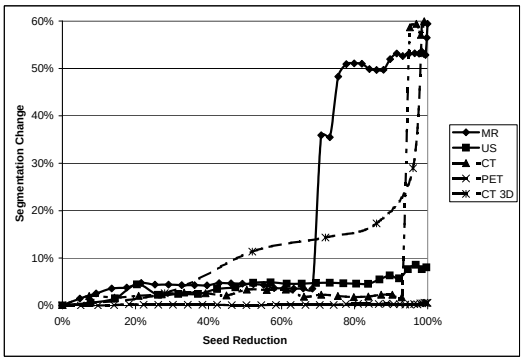


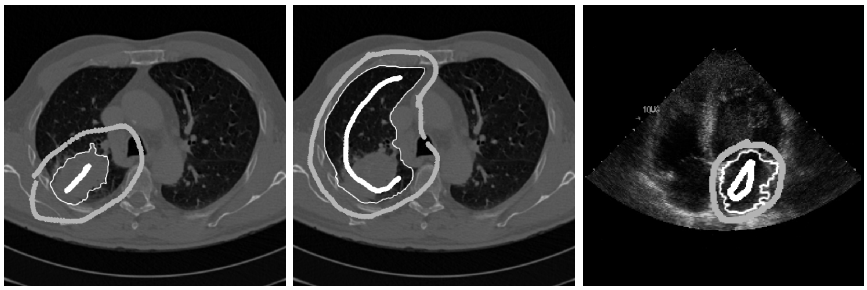
Fig. 2. Sensitivity analysis of segmentation to the numbers of seeds used. For each image in 1, we initially filled the inside of the initial segmentation with foreground seeds and the outside with background seeds. Then, we performed morphological erosion until the seeds were entirely removed and tracked the change in segmentation after each erosion step. The x-axis indicates the percentage of seeds remaining (with respect to the initial seeding) and the y-axis represents the ratio of the number of seeds that changed label to the total number of foreground seeds. Since this experiment was deterministic, the resulting value is simply reported after each erosion operation. Seed reductions of 50% or more are seen to produce only minor changes in the resulting segmentation. Note that the 100% value is not actually reported — the experiment was terminated when erosion would have removed the final seed.

beled as foreground. The original images, initial segmentations and experimental results of the perturbation studies are shown in Figure 1.

As one might expect, a greater magnitude shift produces a greater change in the segmentation. However, the algorithm does appear to be stable in the traditional sense that a small change in input (i.e., seed placement) produces a small change in output (i.e., the segmentation). In fact, perturbations within the original object all produced changes in the segmentation under 5%. However, image content does influence the segmentation response to seed perturbation, as may be seen by comparing the plots in Figure 1. For example, the low-contrast boundary of the cerebellum in this particular MR image results in a greater sensitivity to seed location than the more straightforward cardiac PET image, although all segmentations remain within 5% of the original. Therefore, we conclude that a user need not be very accurate in placing seeds, since a small deviation will not be expected to make a large difference in the results.

A second important question is: How many seeds does a user need to place in order to achieve the desired segmentation? If many seeds are required, the algorithm has little to offer over a purely manual segmentation. As above, we have used a foreground/background scenario for simplicity. We used following design in examining this question quantitatively: We start with the initial segmentations of Figure 1, filling the object with foreground seeds and the background with background seeds. Then, we shrink the two seed groups by applying a morphological erosion operation to each group and track the change in the seg-

(a) Brain tumour (b) Brain (c) Cerebellum



(d) Lung tumour (e) Lung with tumour (f) Left atrium

Fig. 3. User and computer time for image segmentations. User time indicates the estimated time taken to place seeds. Computer time is given with both a CPU and GPU implementation. a) User: 2s, CPU: 10s, GPU: 0.7s, b) User: 2s, CPU: 6s, GPU: 1.5s, c) User: 2s, CPU: 83s, GPU: 0.3s, d) User: 3s, CPU: 3s, GPU: 0.9s, e) User: 5s, CPU: 23s, GPU: 1.3s, f) User: 4s, CPU: 3s, GPU: 0.8s.

mentation as the number of seeds are reduced. Figure 2 illustrates the change in segmentation as the number of seeds are reduced. As desired, small reductions in the numbers of seeds preserve the object boundaries almost exactly. In fact, it is only after the number of seeds has dropped dramatically, to less than 50% of the original, that the effects of reduced seeds are noticeable. Therefore, we conclude that only a fraction of the seeds necessary to specify a manual segmentation are required to produce a nearly identical segmentation.

The time required by a user to obtain a desired segmentation (with editing) is more crucial than the computation time. Figure 3 shows the results of segmenting several different targets in different imaging modalities. The caption details the total user time required to place the seeds (including edits) and the computation time on the CPU and GPU. The experiments were performed on a Pentium 4 with 2.8 GHz and a ATI Radeon X800 XT graphics card. User time ranged from 2–5s, CPU computation time from 3–83s and GPU time from 0.3–1.5s. As a comparison, we note that segmenting the 3D volume of Figure 1 required 35s for the CPU, 1s for the GPU and 5s of user time to place seeds.

5 Conclusion

Anatomical differences between patients and the patient pathology often prevent a fully automatic algorithm from producing high-quality segmentations without user interaction. The random walker method presented in [1] has proven a useful tool for general-purpose, interactive segmentation.

Since a diligent user could achieve an arbitrary segmentation by placing enough seeds, it is important to validate this technique using metrics that measure the speed and ease of use for a naive user. Our experiments above examined how sensitive the segmentation is to seed placement and how many seeds are needed to obtain a quality segmentation. Furthermore, we have presented a simple method for reducing the computation time of the random walker algorithm by over an order of magnitude on commodity graphics hardware and measured the total time required for a user to obtain a desired segmentation. Our experiments support the statement that the random walker algorithm allows a user to quickly obtain a desired segmentation without concern for placing an excess of seeds, placing them very carefully or in a prescribed pattern. Ultimately, we believe that these qualities will lead to the widespread use of the random walker segmentation algorithm in varied applications.

References

1. Grady, L., Funka-Lea, G.: Multi-label image segmentation for medical applications based on graph-theoretic electrical potentials. In Šonka, M., Kakadiaris, I.A., Kybic, J., eds.: ECCV 2004 Workshops CVAMIA and MMBIA. LNCS3117, Prague, Czech Republic, Springer (2004) 230–245
2. Boykov, Y., Jolly, M.P.: *Interactive graph cuts* for optimal boundary & region segmentation of objects in N-D images. In: International Conference on Computer Vision. Volume I. (2001) 105–112
3. Mortensen, E., Barrett, W.: Interactive segmentation with intelligent scissors. Graphical Models in Image Processing **60** (1998) 349–384
4. Sethian, J.A.: Level Set Methods and Fast Marching Methods. Cambridge Monograph on Applied and Computational Mathematics. Cambridge University Press (1999)
5. Lefohn, A.E., Cates, J.E., Whitaker, R.T.: Interactive, GPU-based level sets for 3D segmentation. In: Medical Image Computing and Computer Assisted Intervention (MICCAI). (2003) 564–572
6. Sherbondy, A., Houston, M., Napel, S.: Fast volume segmentation with simultaneous visualization using programmable graphics hardware. In: IEEE Visualization 2003, Seattle, WA, IEEE (2003) 171–176
7. Bolz, J., Farmer, I., Grinspun, E., Schröder, P.: Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. In: ACM Transactions on Graphics. Volume 22 of SIGGRAPH. (2003) 917–924
8. Krüger, J., Westermann, R.: Linear algebra operators for GPU implementation of numerical algorithms. In: ACM Transactions on Graphics. Volume 22 of SIGGRAPH. (2003) 908–916
9. Harris, M.J.: Real-time cloud simulation and rendering. Technical Report TR03-040, University of North Carolina (2003)