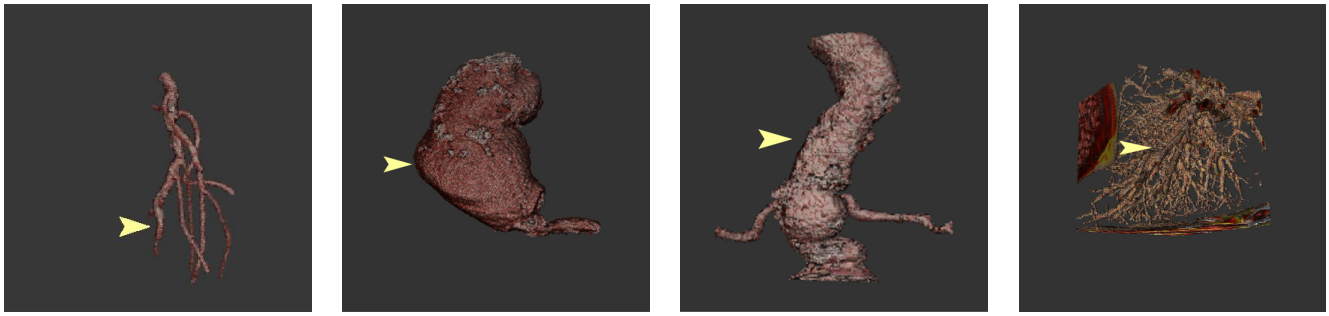
Fast Volume Segmentation With Simultaneous Visualization Using Programmable Graphics Hardware

Anthony Sherbondy Mike Houston Sandy Napel*
Stanford University



(a) (b) (c) (d)

Figure 1: These four volume renderings utilize a fully opaque transfer function, but are segmented using the method discussed in this paper. The segmented volumes show: (a) abdominal aortic branch vessels, (b) an aortic aneurysm, (c) an aorta, and (d) peripheral blood vessels in the lung. The yellow arrows indicate the location of the user's initial seeds that were evolved to form the presented segmentations.

Abstract

Segmentation of structures from measured volume data, such as anatomy in medical imaging, is a challenging data-dependent task. In this paper, we present a segmentation method that leverages the parallel processing capabilities of modern programmable graphics hardware in order to run significantly faster than previous methods. In addition, collocating the algorithm computation with the visualization on the graphics hardware circumvents the need to transfer data across the system bus, allowing for faster visualization and interaction. This algorithm is unique in that it utilizes sophisticated graphics hardware functionality (i.e., floating point precision, render to texture, computational masking, and fragment programs) to enable fast segmentation and interactive visualization.

CR Categories: I.4.6 [Segmentation]: Region Growing, Edge and Feature Detection; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Volume Rendering; I.3.8 [Computer Graphics]: Applications

Keywords: region growing, diffusion, segmentation, graphics processor, streaming computation

*{sherbond,mhouston,snapel}@stanford.edu

1 Introduction

In current radiological practice, highly trained medical imaging specialists segment anatomical regions of interest from computed tomography (CT) and magnetic resonance (MR) volumes for further analysis. The segmentation process involves the trained operator selecting the voxels that belong to the anatomy of interest by drawing contours around cross-sectional views and linking these cross-sections together.

This manual procedure can be an extremely tedious chore because of the complexities of anatomical structures. For instance, arterial vessels may take many odd shapes as they twist and turn throughout the anatomy making them very difficult to track along the multitude of cross-sections of the volume. The vessels may also not have clearly defined edges separating them from nearby objects of similar intensity (i.e., bone for contrast enhanced images and soft tissue otherwise) and may have multiple regions where the anatomy stops and then begins again due to constrictions or scanner artifacts. Manually segmented data is impacted by the user's training and judgement. For example, one must decide whether or not to include calcium deposits, thrombus, constricted regions, and different portions of the vessel.

Many contributions have been made to the field of automatic segmentation. However, the complicated structures found in medical imaging offer several unsolved challenges to automated algorithms, including the lack of global defining morphological characteristics, scanner noise and artifacts, and an incomplete or weak separation between voxels representing neighboring tissue. Any of the existent automated algorithms can be shown to fail on certain datasets for reasons specific to each algorithm [Kirbas and Quek 2003; Leventon et al. 2000].

Another major drawback of the automated algorithms to date is that they offer limited "trial and error" based user interaction. The user often sets global parameters and runs the algorithm hoping to get the correct results. If the desired result is not achieved, the user attempts to adjust the parameters and runs the algorithm again.

This iterative, indirect approach can be quite time consuming and, because of the often global effect of these adjustable parameters, the desired results may never be achieved.

This paper proposes a fast segmentation method that leverages the computational power of modern programmable graphics hardware to combine some of the successful components of automated algorithms efficiently with real-time animation of the segmentation's progress. The segmentation algorithm is based on seeded region growing with the merging criteria based on intensity and gradient values, with the gradient sensitivity scaled using nonlinear diffusion.

Since the proposed method allows the human operator to observe the segmentation in real time, it removes the necessity of manually marking edges along hundreds of cross-sections through a CT volume. The speed of the segmentation combined with direct visualization offers the ability for the operator to interact with an evolving volume segmentation. Our technique facilitates local control for "expert-based" input and a rapid segment/view/edit cycle, providing practical improvements that are applicable to diagnostic medical imaging.

2 Related Work

There are many useful formulations of the segmentation problem for intensity images. Adams and Bischof separate the techniques into the following four categories: threshold based, boundary localization, region growing, and hybrid approaches that are combinations of the previous methods [1994]. Others have noted that these categories may represent low-level or voxel-level approaches to the problem and that one may also add top-down approaches [Zucker 2001]. One example is to compare the image to be segmented with a database of images with known segmentations and then to segment the image by mapping it to its best match in the database [Leventon 2000; Zucker 2001]. In addition, hybrid approaches have been developed that bridge top-down and bottom-up approaches [Leventon et al. 2000].

To date, no algorithm has been proven to be perfect at automatically segmenting regions of interest from surrounding volume data. In many of these algorithms, user input often involves restarting a fast algorithm with new parameters [Felkel 2000]. Even extremely fast algorithms perform slowly in such "trial and error" based approaches because many iterations, with careful operator scrutiny of the results, may be required to achieve an adequate result.

Because the programmability and performance of modern graphics hardware continues to increase at such a rapid pace, many researchers are beginning to adapt computationally intensive algorithms to run on GPUs. These modern high performance graphics processors, such as the ATI Radeon 9800 [2003] and the nVidia GeForce FX [2003], are already capable of outperforming current CPUs in certain compute intensive applications, and the performance difference is expected to increase in the future [Khailany et al. 2003]. Matrix operations [Larsen and McAllister 2001; Thompson et al. 2002], non-linear diffusion [Rumpf and Strzodka 2001b], and many other computationally intensive algorithms have been shown to run faster on GPUs than similar implementations on CPUs. While these first attempts to utilize graphics hardware provided excellent performance, they suffered from limited precision (8-bit fixed point) provided by the hardware at the time. But as faster hardware with floating point precision has become available, it has become possible to implement high precision algorithms on graphics hardware.

To capitalize on the performance available in graphics hardware, several graphics hardware segmentation approaches have been proposed. Yang et al. implemented image segmentation and morphology operations on an nVidia GeForce4 that was 3-5 times faster for segmentation than an optimized version running on a 2.2GHz Intel P4 [2003]. This approach falls into the category of providing fast segmentation using global parameters and thresholding alone, which is not a successful technique for many segmentation problems [Kirbas and Quek 2003].

Rumpf et al. wrote a 2D level set segmentation algorithm on graphics hardware utilizing the image processing operations provided by the SGI Onyx2 and also the GeForce4 [2001a]. Despite the 8-bit computation limitations of their platform, they were able to produce visually accurate results. Our approach to the segmentation problem differs in that we are concerned with simultaneous visualization and evolution of a 3D hybrid seeded region growing algorithm.

Lefohn et al. proposed an efficient 3D level set solver with curvature flow integrated with a GPU volume renderer for interactive feedback which they implemented on an ATI Radeon 9700 [2003]. Their implementation was between 10-15 times as fast as an equivalent CPU implementation of the same algorithm. In order to limit computation to active voxels, they use an intricate packing scheme from which they evolve and render their level set. Our method differs mainly in our choice of segmentation algorithm, the way in which a user interacts with the segmentation, and method of limiting computation. Our method of limiting computation is similar to [Purcell et al. 2002], which proposed the notion of preventing computation at certain pixels, as part of a GPU-based ray tracer. Our paper presents "computation masks," a generalization of this idea, to eliminate unnecessary computation on a per pixel basis.

3 Algorithm

In this paper, the segmentation problem is mainly posed as one of growing the regions that the user initializes with seed data. The seeds placed by the user indicate to the algorithm the structure that the user would like to separate from the volume. The seeds are then evolved and either diffuse into other regions or move away from regions based on a Perona and Malik nonlinear diffusion metric [Adams and Bischof 1994; Perona and Malik 1990]. The implementation of this algorithm consists of four stages: seed selection, segmentation evolution, optional image smoothing, and computational masking.

3.1 Seed Selection

The current system interface is shown in Figure 2. The user paints seeds by drawing on the sectional views of the volume. The user can select a sub-volume from a larger volume for segmentation and rendering. Mouse coordinates are sent to a fragment program on the graphics hardware that adds seeds to voxels, or turns voxels on, based on the geometry of the brush, usually a sphere with a user definable radius. Let $S(t,x,y,z)$ represent the number of seeds at any position $x,y,z \in \Omega_v$ at some evolution state t, where $t = 0$ denotes the initial or user defined state and Ω_v represents the span of all possible voxel locations. The maximum number of seeds at any voxel location and evolution instance is 2^{16}.

3.2 Segmentation

The merging criteria for the seeded region growing algorithm is based on the Perona and Malik nonlinear diffusion metric. This metric allows the seeds to smoothly merge into regions of similar intensity, while slowing the diffusion into voxels against high gradients. Therefore, the rate of diffusion is governed by the underlying image characteristics. Let $V(t,x,y,z)$ represent the image intensity value of a voxel at the location $x,y,z \in \Omega_v$ and $S(t,x,y,z)$ be the number of seeds per voxel. Then the diffusion equation governing

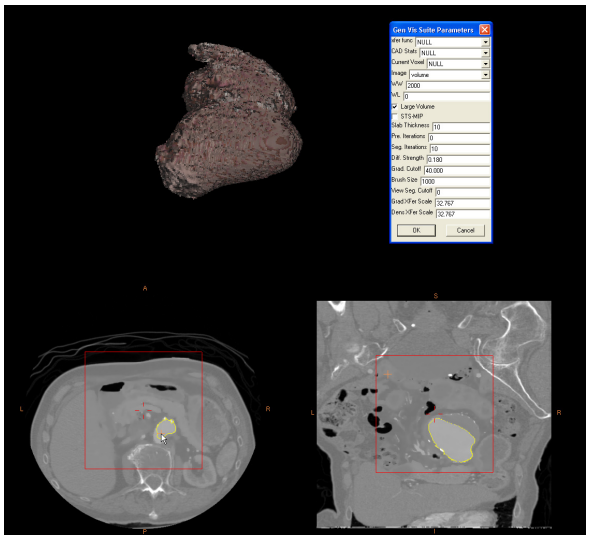


Figure 2: This figure shows our interface for visualizing and interacting with the evolving segmentation. The bottom of the screen presents an axial image section (intersects z axis) and also a coronal image section (intersects y axis) of the volume. The red box indicates the current active sub-volume for volume rendering and segmentation that can be moved to the desired region of interest in the larger volume. The presented data is from a 512x512x494 CT scan and the active sub-volume is 256^3. The user can interact with the segmentation by drawing on the sectional views. The volume renderer uses the evolving segmentation to choose the active voxels to render. The user can choose between multiple pre-defined transfer functions. The current transfer function labels all voxels as opaque and allows the segmentation to define the volume of interest.

the seed flow can be written as:

$$\frac{\partial S(t,x,y,z)}{\partial t} = \operatorname{div}(g(|\nabla V(t,x,y,z)|)\nabla S(t,x,y,z)) \quad (1)$$
$$\text{where } g(s) = \nu \cdot \exp^{-\frac{s^2}{K^2}}$$

Note that t is a variable that represents the current evolution state. K is a cutoff term that controls how fast $g(s)$ goes to zero for high gradients. ν is a regularization term that controls the sensitivity to noise and speed of grouping nearby objects [Catté et al. 1992]. The discretization of this equation will also add a regularization factor, but we place it into the continuous equation to directly show the influence it has on the edge stopping criteria. In addition to the metric imposed by the diffusion, there is an added maximal drift criteria that will not allow seeds to diffuse into regions that are beyond a user definable intensity difference from the last voxel directly selected by the user's brush.

We discretized the diffusion using a standard explicit forward Euler approach. Without loss of generality, we present a discretized version of Equation (1) along the x axis:

$$S_i^{n+1} = S_i^n + h \cdot \left((S_{i+1}^n - S_i^n) \cdot g\left(\frac{V_{i+1}-V_i}{\Delta x}\right) - (S_i^n - S_{i-1}^n) \cdot g\left(\frac{V_i - V_{i-1}}{\Delta x}\right) \right) \quad (2)$$

where $S_i^n \approx S(n \cdot \Delta t, i \cdot \Delta x)$, $V_i = V(i \cdot \Delta x)$, $0 \le n \le N$, $0 \le i \le W, \Delta x = 1/\mathrm{W}$, and $h = \Delta t / \Delta x^2$.

Note that our spatial discretization of x,y,z leads to a kernel that spans the six neighbors of the voxel being computed. N is the total number of iterations for the segmentation computation, and W is the total width of the image. S_j^{n+1} and S_k^{n+1} are similarly derived.We recognize that the explicit kernel solution to the PDE limits the "speed" of the evolution (such that $h < \frac{1}{2}$) in order to maintain stability [Morton and Mayers 1994]. The explicit Euler scheme in Equation (2) avoids communication overhead between the GPU and the CPU that may be required for an implicit discretization. This savings reduces the total time required for interactively changing parameters and running the algorithm again as communication across the system bus is not required. In addition, we are able to limit computation to active voxels by utilizing computation masks described in Section 3.4.

The number of iterations of the diffusion is controllable by the user. Each iteration through the volume is broken down into steps in the 3D texture's z direction. Each z slice defines one rendering pass and the results of that pass are rendered to the opposite or output 3D texture for that iteration, as shown in Figure 3. After every 3D iteration, a 3D texture-based real time volume rendering engine generates a view of the segmentation seeds and the image data, each with its own transfer function.

3.3 Image Smoothing

This optional stage provides a scaling to the image that allows the algorithm to perform well under noisy conditions, with ill-defined edges or other factors that may affect the connectedness of voxels within regions. We introduce scaling through the Perona and Malik nonlinear model, which can be considered a piecewise smoothing process, where we expect to evolve the image toward separate, piecewise contiguous regions [Weickert 1997]. As stated in Section 3.2, $V(t,x,y,z)$ represents the image intensity value of a voxel at the location $x,y,z \in \Omega_v$. If t is an arbitrary term that represents the current iteration of the diffusion, then the evolution of the image smoothing can be represented by the following equation:

$$\frac{\partial V(t,x,y,z)}{\partial t} = \operatorname{div}(g(|\nabla V(t,x,y,z)|)\nabla V(t,x,y,z))$$

If the image smoothing stage is run, it only requires a small number of iterations as it converges quickly towards the minimal energy solution [Leclerc 1989]. When applied, the image smoothing stage is performed before the segmentation evolution and lies outside the segmentation iteration loop.

3.4 Computational Masking

The explicit scheme used for the segmentation algorithm allows the computation of the segmentation to iterate without using any intermediate representation of the volume data. Consequently, the algorithm may compute on voxels that have no chance of a seed entering the voxel in the current iteration. In order to compensate for this, a "viable" subvolume must be tracked and the computation on voxels outside of this subvolume prevented. The most conservative subvolume, without knowing any particular seed values in the current iteration, is a dilation of the current segmented volume by one voxel in each direction. We use this dilation to create a mask that represents the currently active voxels.

4 Implementation

We implemented our proposed algorithm on the ATI Radeon 9800 Pro 256MB using the ARB fragment program extensions [2002] and the board's unique support for render to 3D texture. Our

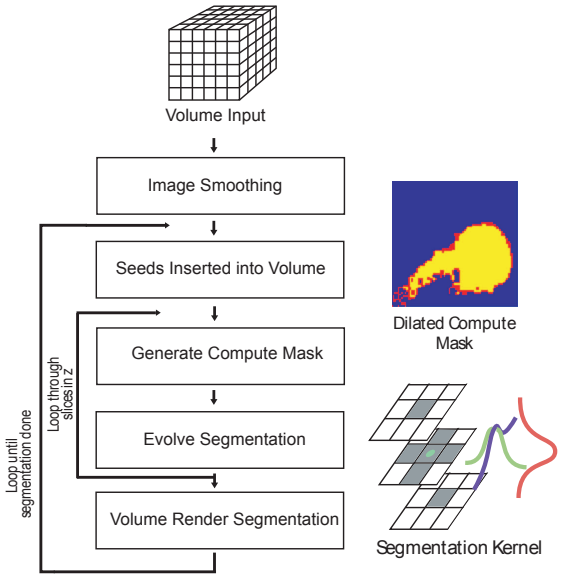


Figure 3: This flow chart represents a simplified implementation of our segmentation algorithm. The image smoothing stage is optional and only included for noisy images. The dilated computation mask inset to the right shows the current seeds in yellow, the six neighbor dilation in red, and the masked out portion in blue. The segmentation inset describes the explicit diffusion kernel in which the six neighbors are weighted by scaled Gaussians that differ in shape based on each neighbor's direction to calculate the value of the current pixel. Note that the computation renders to sections of the 3D texture and therefore must iterate through all sections in the z direction before a complete volume iteration is completed.

method is implemented as depicted in Figure 3. The volume data is loaded into a 3D texture on the graphics card before the algorithm starts. The 3D texture is stored using 32 bits/voxel with 16 bits allocated to the intensity from the CT image data (ALPHA) and 16 bits allocated to the seeds (LUMINANCE). Therefore, each voxel may contain multiple seeds and the maximum number of seeds at any voxel is 2^{16}. It should also be noted that arithmetic operations on the ATI 9800 are limited to 24-bit precision.

The image smoothing fragment program was written using 30 instructions composed of 7 3D texture lookups (6-neighbors plus the current voxel), 10 4-vector operations and 13 scalar operations. Six of the scalar operations are power (POW) operations that expand into three hardware instructions each on the ATI 9800 Pro. The segmentation evolution fragment program is similar to the image smoothing program, as they are both discretized with the scheme described in Equation (2), but it requires slightly more instructions. The segmentation program requires 39 total instructions. These instructions expand to 56 floating point operations with 8 of those being 4-vector operations and 24 being scalar operations. These algorithms require only one rendering pass per slice of the volume, which allows for efficient execution.

Before the computation of the segmentation is run on the current section, an additional fragment program is run to calculate the current computational mask. The computational masking step alters the depth buffer on a per pixel basis by placing a high depth value for any voxel that contains seeds or is a six-neighbor of a voxel with seeds. If the current voxel fails this test, then the depth buffer value

Data Set	Iterations	Time (s)	Rate (Mvox/s)
C. Vessels	450	9.5 (14.7)	99 (64)
Aorta 1	150	3.7 (4.9)	85 (64)
Aorta 2	350	7.8 (11.5)	94 (64)
Lung	305	9.1 (10.0)	70 (64)

Table 1: The results of our segmentation algorithm on various 128^3 datasets: C. Vessels (Figure 1a), Aorta 1 (Figure 1b), Aorta 2 (Figure 1c), and the Lung (Figure 1d). Each segmentation was seeded in the location depicted in Figure 1. The numbers in parenthesis represent the performance without computational masking. Notice the improvement that computational masking provides.

will be set to a very low value. The segmentation computation will then fail a depth test at any location that has the very low value. Since the depth test occurs before the fragment program runs, early z-kill can be done by the hardware, preventing the fragment program from executing on masked-out portions.

An important note is that the ATI Radeon 9800 Pro's implementation of render to 3D texture does not allow reading from and writing to the same texture. Because of this, the algorithm must 'ping-pong', or write between, two 3D volumes. This requires twice the amount of memory as the volume we want to segment, limiting the maximum volume resolution to 256^3.

5 Results

All results were collected using an ATI Radeon 9800 Pro with 256MB of onboard memory on a 2.4GHz P4 with a 533MHz front side bus. We utilized a 128^3 volume of 16-bit intensity data from a CT scan for all presented results.

Table 1 presents the results of running the segmentation algorithm while continuously pumping seeds into the already segmented voxels, to test the maximum throughput of the algorithm. The results of the table show the benefits of the computation mask.

We also measured the performance of the individual fragment program components. Each execution of the segmentation fragment program requires 0.22 ms. Just doing the 3D texture lookup from within a fragment program takes 0.09 ms of that time. To perform any computation, we need to flip-flop textures, which costs an additional 0.04 ms. This gives a total time of 0.26 ms for each slice of the volume, or 33 ms per full volume iteration. We measured the time for the building the computation mask to be 0.14 ms per slice in addition to the time for the segmentation evolution. This means that the crossover point for gaining performance from computational masking occurs when computation on about half of the voxels in the volume can be prevented. These numbers scale linearly for different volume/slice sizes up to the maximum 256^3 volume limited by onboard memory. Without considering the savings from computational masking, the segmentation algorithm is capable of processing 64 Mvoxels/s, or 128 MB/s, of volume data.

We also measured our performance when simultaneously displaying the progress of the segmentation in a 3D texture hardware-based volume renderer. Recent 3D texture-based volume renderers are now capable of rendering at interactive rates on programmable graphics hardware [Engel et al. 2001; Gelder and Kim 1996]. Using Blinn-Phong shading and calculating gradients on the fly for lighting, the volume renderer used for this paper is able to render 128^3 volumes at 15-25 fps (40-66 ms per frame) and 256^3 volumes at 5-15 fps (66-200 ms per frame) at useful screen resolutions.

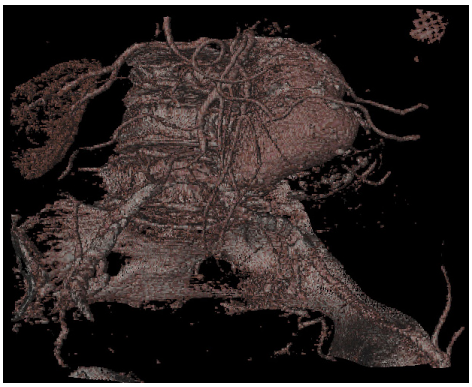


Figure 4: This figure shows the results of thresholding a 256^3 subvolume of a contrast enhanced abdominal CT centered at 270 Hounsfield Units. In addition to the aorta, many other parts of the lower abdomen anatomy are picked up by the threshold operation.

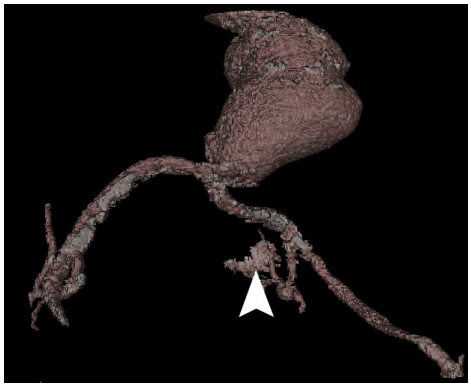


Figure 5: This figure depicts a segmentation of the aortic aneurysm as well as the large vessels distal to the bifurcation from the same subvolume shown in Figure 4. The arrow in the volume rendering image points to where one of the tiny vessels makes contact with bone and the segmentation leaks through the weak interface between the bone and the vessel.

6 Discussion

The animation of the segmentation evolution does not substantially affect the volume rendering performance. In fact, a full iteration in the evolution of the volume segmentation runs roughly the same speed or faster than the volume rendering. This allows for effective visualization of the evolving segmentation, and the shared data structures between the computation and rendering allow for a simple and efficient implementation.

Our GPU-based volume segmentation algorithm achieves a speedup of ten to twenty times over a SSE2 optimized CPU-based solution. It should be noted that the CPU implementation used 32-bit precision while the GPU implementation is limited to 24-bit precision. Whereas we are able to process 64 Mvoxels/s without computational masking, the CPU is only able process 3 Mvoxels/s. Using computational masking, our GPU-based algorithm is more than twenty times faster than the CPU-based solution when we can prevent computation on about half of the voxels. When we can no longer limit our computation on voxels as efficiently as the CPU-based solution can, at the worst point, we are still more than an order of magnitude faster (33 Mvoxels/s vs. 3 Mvoxels/s).

A qualitative comparison of thresholding and our segmentation method is shown in Figures 4 and 5. In Figure 4 the thresholded image picks up the desired aortic aneurysm as well as bone and other tissue, which were present in the selected subvolume and were similar in intensity. Using the same transfer function for rendering as Figure 4, Figure 5 shows the result of an iterative use of the segmentation algorithm on the aorta. One can clearly see the aneurysm and the stenosis below the aneurysm as well as the vessels that feed from it. However, the cursor on the volume rendering points to a leak from one of the small vessel branches into the bone. The leak of seeds across weak boundaries is a common difficulty with edge based algorithms in general and is motivation for having an interactive segmentation process.

In comparison to the packing technique employed by Lefohn et al. [2003], the computation masks allow more straightforward computation and visualization. In addition, as the arithmetic intensity of the computation stage increases, e.g. by adding geometric constraints or other operations per evolution, the cost of building the computation masks would have a smaller impact on the total performance of the algorithm, thus continuing to provide speedups for much larger active regions. Furthermore, by employing more aggressive culling techniques, such as only tracking the evolving front or "most active" voxels, one may stay well below our current fifty percent cutoff for computation savings.

The performance advantages of the computational masking suggest that our algorithm would benefit from improved support for building computation masks. Currently, the depth of a fragment cannot be set from within a program outputting floating point values. This requires us to take an additional rendering pass, comprised mostly of the same texture lookups as in the segmentation evolution pass, to build the computation mask. Besides the cost of the redundant texture lookups, there is also some overhead involved with setting up and running an additional fragment program. Currently, our conservative application of computational masking improves performance if more than one half of the volume computation can be saved. As better support for computational masking becomes available, the added cost of building the masks can be reduced, substantially increasing the benefits of tracking the subvolume.

7 Future Work and Conclusion

A segmentation algorithm with interactive visual feedback allows human operators to segment complex 3D structures in real time. New mechanisms for interacting with an evolving 3D scene would allow the user to more effectively harness the speed of such algorithms. Alternative 3D scene painting techniques could enhance the user's local control provided by this algorithm. For instance, the user could paint simple 3D strokes, suggesting structure, that the algorithm would interactively evolve into a detailed volume boundary. The segmentation process would also benefit from carefully crafted visualizations that facilitate discerning the changing segmentation surface from the volume data.

This future proposal for user steering of the segmentation is similar to other researchers' [Falcao et al. 2000; Mortensen and Barrett 1995] approaches for 2D tracing utilizing a live-wire method. Our method also involves user guidance; however, ours would be a 3D approach, and rather than having the user's attention on the boundary, we allow the user to force structure segments to be included or not by painting seeds in the volume. This input would help the automatic part of the segmentation grow through structures and/or avoid "leaking" into adjacent structures.

This paper has shown that computational masking can be an effective and simple way to save computation and speed up GPU-based algorithms. Although we only explore one method for creating computational masks in this paper, a simple dilation of the segmentation, more aggressive methods could be considered. For example, one could mask out all voxels containing seeds not likely

to evolve in future time steps, i.e. voxels surrounded by voxels with identical seed values. By employing more aggressive methods, one could limit the number of active voxels to be below the computational mask crossover point.

This paper presents a segmentation method that allows interactive visualization and control because of its computation speed and coordination with a hardware accelerated volume renderer. In addition, using the GPU for general computation will become even more advantageous as graphics processors continue to outpace the performance of CPUs. Also, through the use of computational masking we are able to use the computational resources of modern graphics hardware more effectively. With interactive 3D segmentation available, researchers can now explore volumetric segmentation with human intervention.

8 Acknowledgments

We would like to thank James Percy at ATI for all of his help with render to 3D texture and all of the help getting beta drivers and also Mark Segal from ATI for giving us early access to the R300 and R350 which made this work possible. We would also like to thank Tim Purcell, Ian Buck and Pradeep Sen from Stanford University for thoughts and ideas on optimizing our fragment programs and many conversations about computational masking. Also, we appreciate the support of everyone at the 3D Radiology Lab for their thoughts on interactive segmentation and its application. We would especially like to thank Pat Hanrahan for the many conversations that sparked the original idea and supported our continued efforts.

References

ADAMS, R., AND BISCHOF, L. 1994. Seeded Region Growing. *IEEE Transactions on Pattern Analysis and Machine Intelligence 16*, 6 (June), 641–647.

ATI, 2003. RADEON 9800. http://www.ati.com.

CATTÉ, F., LIONS, P., MOREL, J., AND COLL, T. 1992. Image Selective Smoothing and Edge Detection by Nonlinear Diffusion. *SIAM Journal of Numerical Analysis 29*, 7, 182–193.

ENGEL, K., KRAUS, M., AND ERTL, T. 2001. High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, ACM Press, 9–16.

FALCAO, A., UDUPA, J., AND MIYAZAWA, F. 2000. An Ultra-Fast User-Steered Image Segmentation Paradigm: Live Wire on the Fly. *IEEE Transaction on Medical Imaging 19*, 1 (January), 55–62.

FELKEL, P. 2000. Segmentation of Vessels in Peripheral CTA Datasets. VRVis Technical report TR-VRVis-2000-008, VRVis Center, Donau-City-Straße 1, A-1220 Vienna, Austria, www.vrvis.at, Dec.

GELDER, A. V., AND KIM, K. 1996. Direct Volume Rendering With Shading in Three-Dimensional Textures. In *Proceedings of the 1996 symposium on Volume visualization*, IEEE Press, 23–ff.

KHAILANY, B., DALLY, W., RIXNER, S., KAPASI, U., OWENS, J., AND TOWLES, B. 2003. Exploring the VLSI Scalability of Stream Processors. In *Proceedings of the Ninth Symposium on High Performance Computer Architecture*.

KIRBAS, C., AND QUEK, F. 2003. Vessel Extraction Techniques and Algorithms: A Survey. In *In Review: IEEE Conference Bio-Informatics and Bio-Engineering (BIBE)*.

LARSEN, E. S., AND MCALLISTER, D. 2001. Fast Matrix Multiplies using Graphics Hardware. In *Supercomputering 2001*.

LECLERC, Y. 1989. Constructing Simple Stable Descriptions for Image Partitioning. *International Journal of Computer Vision 3*, 73–102.

LEFOHN, A., KNISS, J., HANSEN, C., AND WHITAKER, R. 2003. Interactive Deformation and Visualization of Level Set Surfaces Using Graphics Hardware. In *To Appear in Proceedings of IEEE Visualization 2003*.

LEVENTON, M., GRIMSON, E., AND FAUGERAS, O. 2000. Statistical Shape Influence in Geodesic Active Contours. In *Computer Vision and Pattern Recognition (CVPR)*, 75–84.

LEVENTON, M., 2000. Statistical Models for Medical Image Analysis. MIT Ph.D. Thesis.

MORTENSEN, E., AND BARRETT, W. 1995. Intelligent Scissors for Image Composition. In *In SIGGRAPH'95*, 191–198.

MORTON, K., AND MAYERS, D. 1994. *Numerical Solution of Partial Differential Equations*. Cambridge University Press.

NVIDIA, 2003. nVidia GeForce FX. http://www.nvidia.com/.

OPENGL ARCHITECTURE REVIEW BOARD, 2002. ARB_fragment_program. http://www.opengl.org/.

PERONA, P., AND MALIK, J. 1990. Scale-space And Edge Detection Using Anisotropic Diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence 12*, 7 (June), 629–639.

PURCELL, T., BUCK, I., MARK, W., AND HANRAHAN, P. 2002. Ray Tracing on Programmable Graphics Hardware. *ACM Transactions on Graphics 21*, 3 (July), 703–712. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).

RUMPF, M., AND STRZODKA, R. 2001. Level Set Segmentation in Graphics Hardware. In *IEEE International Conference on Image Processing*, 1103–1106.

RUMPF, M., AND STRZODKA, R. 2001. Level Set Segmentation in Graphics Hardware. In *Proceedings of EG/IEEE TCVG Symposium on Visualization VisSym '01*, 75–84.

THOMPSON, C., HAHN, S., AND OSKIN, M. 2002. Using Modern Graphics Architectures for General-Purpose Computing: A Framework and Analysis. *International Symposium on Microarchitecture*.

WEICKERT, J. 1997. A Review of Nonlinear Diffusion Filtering. In *Scale-Space Theories in Computer Vision*, 3–28.

YANG, R., AND WELCH, G. 2003. Fast Image Segmentation and Smoothing Using Commodity Graphics Hardware. *To appear in the Journal of Graphics Tools Special Issue on Hardware Accelerated Rendering Techniques*.

ZUCKER, S. W. 2001. *Relaxation Labelling: 25 years and Still Iterating*. Kluwer Academic Publisher.