Interactive GPU-based Segmentation of Large Medical Volume Data with Level-Sets

Oliver Klar*

VRVis Research Center and University of Koblenz-Landau

Abstract

Automatic or semi-automatic segmentation of anatomical structures in three-dimensional datasets is still an ambitious challenge in medical image computation. The scope of application contains visualization, diagnosis, and preplanning of operations. This paper presents a 3D GPU-based level-set solver for segmentation which focuses on segmenting and rendering large volumes. Neither the volume nor the level-set have to be resident on the GPU in their entirety. Only those parts of the volume which are relevant for computation are held in memory and are dynamically swapped in and out of GPU memory. The level-set solver is integrated in a high-quality GPU based ray caster.

Keywords: Segmentation, Level Sets, Bricking, Volume Rendering

1 Introduction

An important challenge in medical image analysis is tracking the progress of diseases. For instance identification of a tumor in medical datasets it is essential to identify the exact position and to obtain an accurate description of the anatomical structures. Determining which parts of an image or volume belong to a given object of interest is called segmentation. To find the tumor‘s shape, segmentation approaches which are based on surface representations are very powerful. A technique for deforming and evolving implicit surfaces is the level-set method.

This paper presents a segmentation tool based on the level-set method which can be used for applications like operation planning and diagnosis. We introduce a 3D level-set solver which is combined with a high quality GPU raycaster. This combination provides immediate visual feedback of the segmentation process in real time and with high quality.

While the level-set computation is in progress, the user must be able to adjust the segmentation parameters if the level-set evolves in an undesired way. Therefore, the system must instantly react to all changes done by the user. To speed up the application, hardware based acceleration techniques for rendering and segmentation are used. Furthermore, our system focuses on segmentation and rendering of large volumes, where neither the volume nor the level-set have to be resident on the GPU in their entirety.

We introduce a dynamic 3D memory management which handles large medical datasets and overcomes the restrictions of GPU memory. Only those parts of the volume which are relevant for computational purposes are held in memory and are dynamically swapped in and out of GPU RAM.

In contrast to previous work, especially [5, 6, 8], our system works entirely on 3D textures and computes the level-set evolution on the same format that is concurrently used for rendering. In previous approaches, bricking has only been used to speed up the level-set computations. In our work, we use a general bricked virtual memory scheme that is employed for both the density volume and the level-set volume, which allows scalable and flexible memory management for large volumes. A system for high-quality ray-casting [2] has been extended to simultaneously render the density volume and the embedded semi-transparent iso-surface that is the level-set, with accurate intersections between the areas rendered with direct volume rendering and the surface.

In the next section we will give an overview of related work. Afterwards, Section 2 introduces the main idea of the level-set approach. The 3D level-set solver with its three main steps is presented in Section 3. Furthermore, Section 4 gives an overview of the interactive segmentation application and the workflow. Finally, performance numbers and some results are discussed.

Related Work

The level-set method was introduced by [12], as a versatile method for computing and analyzing the motion of implicit curves or surfaces in two or three dimensions. It is used in a wide field of applications. The approach has been used to solve several problems [1], [11], [10]. Because of the fast evolution of commodity graphics hardware and their highly parallel architecture, it is worthwhile trying to implement the level-set method on the GPU. The use of level-sets for segmentation purpose on commodity graphics hardware was first proposed in [13]. They presented a two dimensional level-set solver with a time-invariant speed function.

*klar@uni-koblenz.de

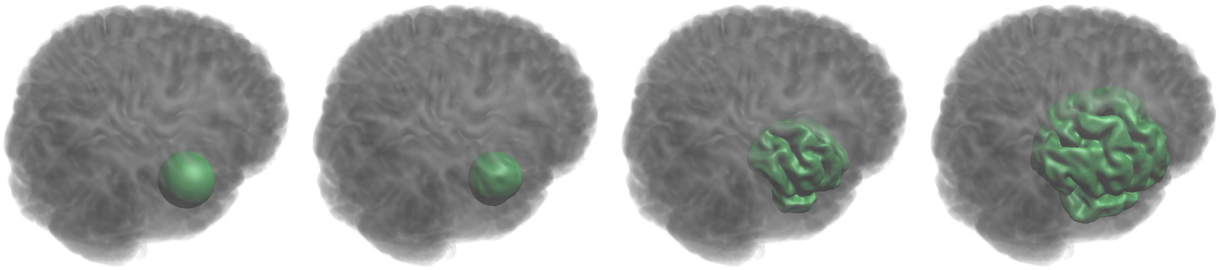


Figure 1: These images show the segmentation of a human brain. The implicit isosurface deforms into the shape of the brain.

In [9], [7] the first GPU-based level-set solver with curvature flow is introduced. The speed function in their approach is created by combining a pre-computed data-driven speed with mean curvature. Lefohn [4] presented a general purpose segmentation tool that relies on interactive deformable models implemented as level sets. It is the first GPU approach that utilized the sparse nature of level-sets.

The possibility of intervention and steering the process of segmentation by the operator is a very desirable goal. Integrating the segmentation application in a real-time volume renderer can solve this problem [4], [8]. Additionally, the application introduced there uses the advantages of GPUs by packing the level-set iso-surface data into a dynamic sparse texture format. If the contour is evolving, this sparse data structure is updated via a GPU to CPU message passing scheme that uses automatic mipmap generation to create compact encoded messages. The data is packed in a single huge 2D texture.

2 The Level-Set Method

In this paper we use the level-set approach for segmentation. Isolating objects in images or volumes often means finding a good approximation of the boundary of an examined object. The idea of model based segmentation approaches is to start with a simple initial contour and deform it over time until it approximates the desired object with sufficient accuracy. The level-set method represents an n-dimensional contour by embedding it in a $(n+1)$-dimensional function $\phi(\mathbf{x})$.

The contour is deformed by solving a partial differential (PDE) equation on a voxel grid. One can find a detailed description of level-set techniques in [11]. To propagate $\phi(\mathbf{x})$ over time we use the simple convection equation:

$$\frac{\partial \phi(\mathbf{x})}{\partial t} = F\left|\nabla \phi(\mathbf{x})\right|, \tag{1}$$

where F is the so called *speed function* and $|\nabla\phi(\mathbf{x})|$ is the gradient magnitude of $\phi(\mathbf{x})$. However, solving Equation 1 for every voxel is computationally demanding. Therefore in this paper the narrow band approach is used to accelerate the level-set computation. The central idea of the narrow band method is to solve the PDE only in the neighborhood of the tracked level-set instead of the whole volume [1].

Level-Set Speed Function

The propagating level-set will expand or contract depending on the speed function F. This function can consist of arbitrary complex terms that influence the evolution. Our choice of speed function consists of two terms. The first parameter is based one the underlying density volume. Thus, the propagating front will expand or contract depending on the gray values of the medical data set.

The second term that is used for the level-set solver is computed from the level-set itself and basically is the mean curvature of the level-set at that point. This term is necessary to keep the deformable surface together and avoid singularities. The speed function F used in this work is defined as:

$$F(\mathbf{x}) = \left(\alpha D(I(\mathbf{x})) + (1-\alpha)\, div \frac{\nabla\phi}{|\nabla\phi|}\right). \tag{2}$$

The first term D represents the influence of the density volume and is defined as

$$D(I(\mathbf{x})) = \varepsilon - |I(\mathbf{x}) - T|. \tag{3}$$

$I(x)$ is the image intensity at a given position x.T approximates the average intensity of the object of interest, ε controls the range around T that is considered to still be inside the object. Therefore, a small ε yields a lower variance around T.

3 3D Level-Set Solver

This section describes the implemented 3D level-set solver with its main steps shown in Figure 2. The first step represents the setup of the solver. It performs the initialization of the volume as a distance field, representing a sphere, see Figure 3. Step two includes the subsequently described kernel execution for each grid point in the computational domain. Step three presents the update of the computational domain. Step two and three are executed for each iteration.

3.1 Signed Distance Field

In our level-set solver $\phi(\mathbf{x})$ represents a signed distance function that is scaled and biased to fit into the range $[0,1]$ in order to be easily compatible with texture format. A simple implicit form of the sphere equation, depicted in Figure 3 initializes the level-set volume. Thus, the computational domain contains distance values. The values $\phi(\mathbf{x}) < 0$, which define the exterior of the initial sphere, will be clamped to zero. The values $\phi(\mathbf{x}) > 1$ are set to one. This is a helpful property for deciding at which side of the level-set a given position is located.

Therefore $|\nabla\phi(\mathbf{x})|$ is nonzero near the level-set 0.5. Even though ϕ is defined on the whole domain, the idea of the narrow band will keep the computational cost of the distance field creation small. The distances are only computed on a narrow band around the level-set.

3.2 Level-Set Calculation

This section describes how the discretized level-set equation is solved in a fragment program for every voxel. The discretization of the curvature and the final update of the PDE for one time step is based on the work presented in [5].

$$\phi(t+\Delta t) = \phi(t) + \Delta t F |\nabla\phi| \tag{4}$$

To solve Equation 4 for one voxel, where F is the speed function and $|\nabla\phi|$ is the gradient magnitude, the upwinding described in [11] and the curvature are computed on a 3×3×3 neighborhood. First, for the center voxel of the 3×3×3 mask, the forward, backward and central differences are computed. Afterwards the components of the speed function, see Equation 2, are processed. The target density T and ε required to compute D, see Equation 3, adjusted by the operator in the GUI, are transferred as uniforms to the fragment program. With the difference of normals method used in [5] the mean curvature is computed by the derivatives mentioned above. This method uses two differently approximated normals n^+ and n^-, which are basically computed from forward and backward differences respectively. From these, the kernel then computes the components of the divergence in Equation 2 as

$$\frac{\partial \mathbf{n}_x}{\partial x} = \mathbf{n}_x^+ - \mathbf{n}_x^-, \tag{5}$$

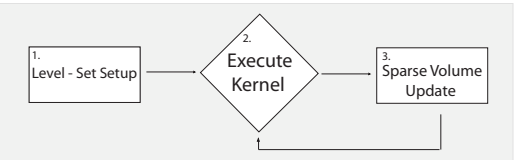


Figure 2: Overview of the main steps of the three dimensional level-set solver. In the setup step the signed distance field is initialized. The kernel solves the partial differential equation for every time step. After each iteration the data structure has to be updated for the next computation pass.

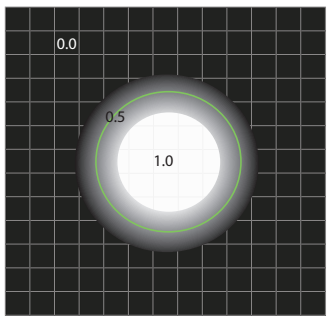


Figure 3: The distance field is initialized by a signed distance function of a sphere. Values in the interior are clamped to 1.0 and values in the exterior to 0.0. Only in a narrow band the values are in between [0.0,1.0].The tracked level-set is 0.5.

$$\frac{\partial \mathbf{n}_y}{\partial y} = \mathbf{n}_y^+ - \mathbf{n}_y^-, \tag{6}$$

$$\frac{\partial \mathbf{n}_z}{\partial z} = \mathbf{n}_z^+ - \mathbf{n}_z^-. \tag{7}$$

Then, the mean curvature of the implicit surface at that location is approximated as

$$\kappa = 0.5\left(\frac{\partial \mathbf{n}_x}{\partial x} + \frac{\partial \mathbf{n}_y}{\partial y} + \frac{\partial \mathbf{n}_z}{\partial z}\right). \tag{8}$$

The upwinding approach decides depending on the sign of the speed function F, how to approximate $\nabla\phi$.

Therefore, as presented in [5] the next step in the kernel execution is to compute $\nabla\phi_{min}$ and $\nabla\phi_{max}$. For $F > 0$, the shader solves the PDE with $|\nabla\phi| = |\nabla\phi_{max}|$. Otherwise, $\nabla\phi$ is computed by $|\nabla\phi| = |\nabla\phi_{min}|$. With the correct choice of the gradient magnitude and κ for the speed function, a fragment program is executed for every voxel of every active brick in the level-set cache. Bricking is described in the following section.

3.3 GPU Level-Set Data Structures

Because the level set is constricted to a subset of the entire volume, it is not necessary to compute the PDE on the whole voxel grid. Therefore, restricting the computational effort to areas where it is absolutely required is a very important part of the system presented here. Thus we introduce a dynamic memory management that handles large datasets, decouples the segmentation application from the GPU memory restrictions and caches relevant data in GPU.

Because the level-set deforms and evolves over time, the dynamic memory layout of the cache has to be updated after each iteration. One has to determine which voxels are active concerning the expansion of the surface, and which voxels can be ignored according to the width of the narrow band in the updating process.

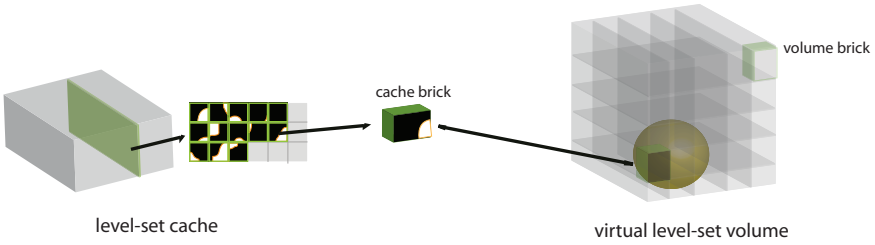


Figure 4: The relationship between the level-set cache and its corresponding virtual volume is shown. A dynamic mapping scheme between both volumes ensures that the 3D level-set solver fetches the correct neighbors while processing.

3.3.1 Cache Layout

The 3D level-set solver manages and operates on two cached volumes simultaneously: The cached density volume and the level-set cache. The caches are realized as one 3D texture each and exist only in GPU memory. They contain the actually active data and are divided into bricks of size 34^3. A brick consists of a 32^3 interior area and a one voxel wide border area on all sides. The border areas are required for filtering and for the neighbor activation which is described subsequently. The bricks in the cache are stored in an arbitrary order. Therefore, neighboring voxels are not necessarily adjacent in the cache anymore. However, for computation it is required to know the neighbors of every active brick. Thus, an additional volume is required. We call this *virtual volume* because it holds no data. The virtual volume is divided into 32^3 tiles. The relationship between this volume and the cache is depicted in Figure 4.

The important difference between the cache and the virtual volume is that adjacent tiles in the virtual volume are in opposite to adjacent bricks in the cache real neighbors. This information is encoded in a dynamic mapping scheme between bricks in the cache and bricks in the virtual volume that ensures that the level-set solver fetches the right neighbors while processing. This mapping is depicted in Figure 4. Only those tiles of the virtual volume which enclose the narrow band are mapped to bricks in GPU memory.

While the process is running and the tracked level-set evolves over time, the cache is filled up step by step with more bricks. Naturally, bricks that are not necessary anymore are flushed out of memory. Thus, the content of the cache is dynamically changing and with it the mapping between the two volumes.

3.3.2 Indirection Textures

The mapping between bricks of the cache and bricks of the virtual volume is stored in an additional texture. For every active brick in the cache these indirection textures hold an offset to the corresponding position in the virtual volume. An indirection texture stores one voxel for each brick and vice versa. Furthermore it holds additional information.

During one iteration, the GPU analyzes a flag stored in the alpha channel of the indirection texture. This flag comprises the status information of the current brick.There are three different life time cycles. A brick can have the status:

- active
- inactive
- must be initialized

According to this flag each brick is treated in a different manner in the computation shader. If the brick has active status, the kernel is executed. If the flag is inactive, the algorithm skips for all fragments of this brick. Otherwise the brick is in the initialization phase and all its voxels have to be initialized.

3.3.3 Brick Activation

In the beginning, only the bricks that are initialized by the signed distance function in the setup step have valid values. If the update step of the level-set solver decides to map a new brick because the narrow band enters a previously inactive area, the values of this new brick are not initialized.

Depending on which side of the narrow band the new brick is located, the interior of this brick is initialized by 1.0 or 0.0. To ensure that the level-set can cross the border between an active brick which has triggered the memory request, and the new adjacent brick, the borders of these bricks must have the same values. Therefore border areas of new bricks are initialized by the borders of their neighbors, for a detailed description see [3].

3.4 Sparse Volume Update

After each iteration of the level-set calculation the computational domain has to be updated. For efficiently managing and updating the GPU memory a communication scheme between CPU and GPU is necessary. This communication consists of two main tasks:

- GPU-CPU Memory Request
- CPU-GPU Memory Allocation

While the segmentation process is running, the level-set cache holds all bricks required for solving the PDE in one time step. The dynamic nature of this sparse data structure requires updating the mapping and the cache. The CPU tells the GPU which parts of the virtual domain are active. The algorithm has to request the required data for solving the partial differential equation, always one time step before they are needed.

The communication in the direction from GPU to CPU consists of two steps. A reduction step and a neighbor activation step. The result of both is passed from the GPU to the CPU encoded as a bit state. Afterwards the CPU analyzes this bit code, activates the right neighbors in virtual volume space and allocates memory for the next iteration.

3.4.1 Reduction Step

After the level set solver has finished one iteration, it has to be detected if the current cache layout has to be updated. For this reason, the reduction step computes for every brick in cache memory a flag which gives a active-inactive status feedback about all the bricks currently in the cache. This flag is computed by packing the cache in the three main axes, as seen in Figure 5. First it is reduced in z-direction, depicted in Figure 5 (a). To accelerate the packing two slabs of bricks are reduced simultaneously. The result of two adjacent slabs is written into the *rgba* channels of one texel of a 2D destination buffer. In pass two, shown in Figure 5 (b), the result of the previous step is packed along the x-direction. Finally, the cache has to be reduced in the vertical dimension, see Figure 5 (c). The resulting flag for each brick is evaluated by the CPU and decides for every brick if it is required in the next segmentation pass.

3.4.2 GPU Neighbor Activation

If the narrow band is coming closer to the border of an active brick, the neighbors, depending on which side the narrow band leaks out, have to be activated. Additionally, if the level-set has left a cache brick, the brick has to be deactivated. The more bricks are activated the higher the computational cost in the next pass is. For this reason, the challenge is to activate the smallest number of bricks. However, reducing the computational cost requires a complicated case differentiation. The neighbor activation computes a bit state of active-inactive flags for all bricks in the cache. In Figure 6 for one side of a brick in the cache, all cases where the contour could leak out are shown in different shades of green. Depending on the case, a different number of neighbors has to be activated for the following pass. To reduce the number of bricks to be activated, twenty six cases, composed of eight corners, twelve edges and six faces, plus the interior has to be checked. This information is extracted from the border areas of every active brick. If the level-set flows from one brick into another it naturally passes its borders. The GPU detects this alteration in the border areas and notifies the CPU by sending this information encoded in a bit state. The CPU again decodes it to update the cache with adjacent bricks. The algorithm that we introduce in this paper performs the neighbor activation, with all 27 cases mentioned above, by testing only the six brick faces on the GPU and determining all cases from their intersections on the CPU.

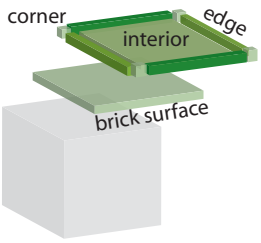


Figure 6: The algorithm has to determine where the evolving surface leaks out. The goal is to activate the minimal number of bricks for the next pass. Therefore, for each face of a brick nine boundary cases have to be checked where the evolving front could cross the border.

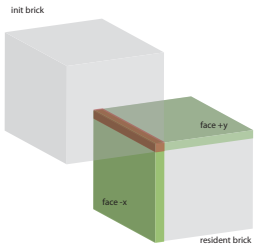
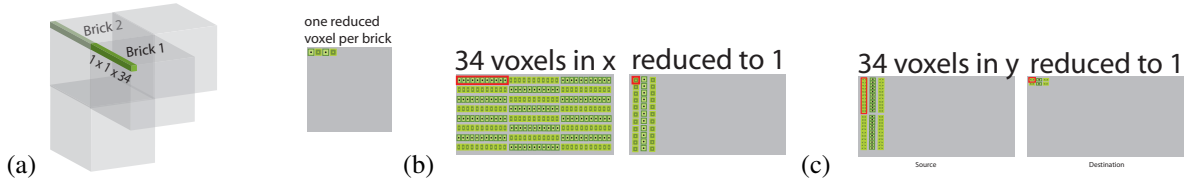


Figure 7: 27 cases where the level-set could cross the border of each brick are determined by processing only six faces on the GPU. The edge of the currently processed brick in red color activates the new brick because of set bits in face -x and face +y. For the corner case three adjacent faces must have their bits set.

Depending on the resulting bits a different number of adjacent bricks has to be activated. One case of the neighbor activation is shown in Figure 7. The result yields a valid bit for the brick face $+y$ and a second valid bit for the face $-x$ of the currently processed brick. The combination of these bits leads to the conclusion, that the edge colored in red has to activate the new brick in Figure 7. The information for the faces is encoded on the GPU and sent to the CPU. The six bits for the six faces are stored in one floating point value.

3.4.3 CPU Neighbor Activation

The filling of the cache for the next pass is done by evaluating the request from the GPU. The request includes the bit state for every brick for neighbor activation and further the status flag for each currently active brick. The CPU pro-



(a) (b) (c)

Figure 5: The reduction step computes a flag which gives a active-inactive status feedback for every brick in the cache in three steps. The cache is reduced first in z-direction, shown in (a), afterwards in x-direction, depicted in (b) and finally in the y-direction (c).

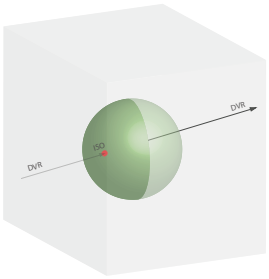


Figure 8: The raycaster displays a combination of direct volume rendering (DVR) in front of the iso-surface, iso-surface-shading and DVR behind the shaded surface. The opacity of the iso-surface is adjustable.

cesses the request from the GPU by decoding the passed information.

If a required brick is actually in the cache, nothing has to be done. Otherwise, it has to be paged down into GPU ram for the next iteration. As previously described new bricks have to be initialized in the calculation shader.

3.5 DVR-ISO-DVR Rendering

To be able to follow the propagating interface embedded in the underlying density volume the iso-surface, i.e., the level-set, and the volume must be shown simultaneously in the 3D view. Only this gives the feedback whether the parameter settings work well and the level-set evolves towards the object borders of interest. For visualization, a special rendering mode renders a combination of unshaded DVR in front of a shaded iso-surface in the middle and unshaded DVR behind the surface, see Figure 8.

The algorithm contains three main steps. Along each generated viewing ray it takes samples in regular intervals of the cached density texture and accumulates them until a density value of the packed level-set volume exceeds the iso-threshold. It is important to mention that the combination of these two rendering approaches is done in one fragment program. Therefore, while iterating along the ray, the fragment program operates on both cached volumes simultaneously. If the iso-threshold is exceeded, the gradient at that position is estimated for shading the level-set surface. The determined location of the iso-surface is taken as starting position for the last DVR step. The raycaster now continues until the ray leaves the volume or any other early ray termination criterion has been met such as fully accumulated opacity. The final output color consists of the composited color of the DVR step in front of the surface, the shaded iso-surface itself, and the color of the unshaded DVR behind the level-set surface.

4 Interactive Segmentation Application

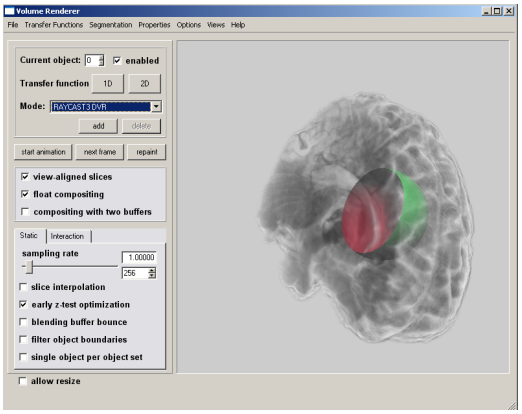


Figure 9: The 3D view is generated by the high quality raycaster. Properties like the sampling rate or different render modes are adjustable.

This section gives an overview of the developed segmentation tool and its user interface. Furthermore, we show the work-flow of the presented tool with a segmentation example. The level-set segmentation application consists of three main panels. A 2D slice viewer, depicted in Figure 10, the segmentation panel shown in Figures 11 and 12 and the 3D view generated by the raycaster shown in Figure 9.

The segmentation panel consists of two main panels. The setup panel depicted in Figure 11 includes all features for the initialization of the segmentation process and the level-set calculation panel is depicted in Figure 12. In the following, a short overview of the work-flow of the segmentation process is given with a example segmentation of a human brain.

4.1 Initialization Parameter Settings

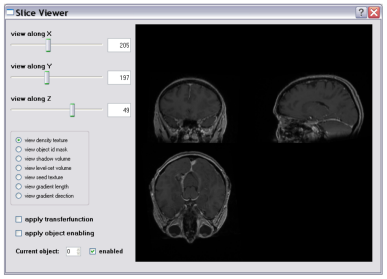


Figure 10: The sliceviewer in the segmentation tool provides a detailed view on every layer of the volume. With a mouse click in the desired region of interest, the user sets the center of the sphere in the setup step.

After a data set is loaded it is shown in the 3D view by the raycaster, as well as in the 2D slice viewer. The operator has the possibility to find the region of interest in both views. When the area to be segmented is roughly defined, the level-set surface can be initialized by the spherical drawing tool. The center of the sphere is defined by clicking in the 2D slice viewer. The radius is adjustable in the level-set setup panel depicted in Figure 11. Furthermore, the user can define the width of the narrow band in that panel. When the painted sphere approximates the region of interest insufficiently, a reset of the spherical initialization can be done.

4.2 Speed Function Adjustment

To adjust the parameters of the speed function, which defines the motion of the evolving front, is the next step. The first parameter, depicted in Equation 9, which has to be set is a weight alpha that determines the influence of the density and the curvature, respectively:

$$F = \alpha * Density + (1-\alpha) * curvature \qquad (9)$$

The curvature setting influences the smoothness of the level-set surface. If the α is set to 0.0, the speed function is solely curvature-driven speed. Thus, it defines a weighting between the fraction of density driven speed and curvature. The second parameter in the level-set panel adjusts

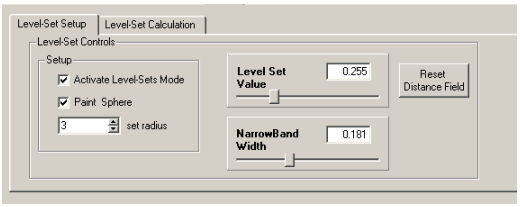


Figure 11: The radius of the initial sphere and the width of the narrow band are adjustable is adjustable in the first panel.

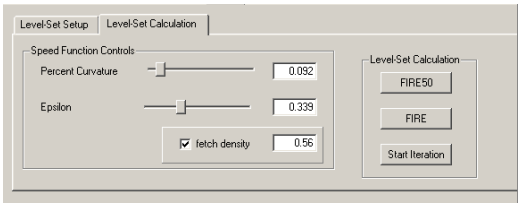


Figure 12: The second segmentation panel contains the speed function controls and the calculation controls.

the epsilon region. This parameter determines the variance around the density value of interest. The target density is specified in the same manner as setting the sphere's center, i.e. by a mouse click in the slice viewer. Both together, the density and the epsilon, let the level-set surface evolve if the underlying densities are in the interior of that epsilon range, otherwise it contracts, see Equation 3.

4.3 Segmentation Process

After the adjustments are complete the segmentation process can be started. If the front leaks out, the operator is able to adjust the segmentation process by changing the parameters at runtime.

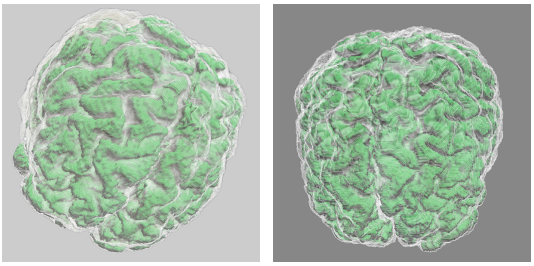


Figure 13: Level-Set segmentation of a human brain (512x512x184). The deformable model expands over time until it converges into the shape of the brain. Different views of the level-set shape are shown.

5 Performance

LEVEL-SET SOLVER STEPS	ELAPSED TIME
sphere setup	0.08
one iteration	0.68

Table 1: $512x512x184$ MR-brain is the tested dataset for the time measurements of this table. It shows the performance numbers for the setup step including the distance field computation and the time in seconds for one iteration of the 3D level-set solver.

In Table 5 performance numbers of the 3D level-set solver are depicted. The benchmarks are run on a GeForce

8800 GTS with 768 MB of video memory. The segmentation process is tested with a $512x512x184$ MR-brain dataset. The first row shows the time in seconds used for the sphere initialization. It comprises the distance field computation. The time for one full iteration of the 3D level-set solver is depicted in the second row. It includes the level-set computation and the update step with the GPU-CPU memory request and the CPU memory allocation.

6 Conclusions

This paper presents an interactive segmentation application that focuses on segmentation and rendering of large volumes. The application combines a high quality raycaster with a three dimensional sparse grid level-set solver. Both are working on the same packed data format. Thus, no intermediate data structure is required for rendering. The challenge was how to overcome the problem of the sparse nature of level-sets on the one hand, and exploiting the power of graphics hardware when working on large continuous data streams on the other hand. To reach interactivity the conceptual design of the level-set solver is focused on packing the sparse data structure for efficient processing on the GPU.

The level-set brick caching algorithm stores only the active subset of the virtual level-set volume. This cache can be processed either by the level-set solver or directly be used for rendering by the raycaster. Solving the level-set equation is done completely by the fragment processor on the GPU. For updating the modified cache, a GPU-CPU communication scheme was presented. This communication ensures that the subset of the entire volume that is required for the next computation pass is available to the fragment processor. A customized render mode has been introduced, which is a combination of direct volume rendering and iso-surface shading. It is well suited for the segmentation application because the operator can directly follow the evolution of the level-set shaded iso-surface.

7 Acknowledgments

This work has been carried out at the VRVis Research Center for Virtual Reality and Visualization in Vienna (http://www.vrvis.at). The medical data sets are courtesy of the Department of Neurosurgery at the Medical University Vienna and Agfa Vienna.

References

[1] David Adalsteinsson and James A. Sethian. A fast level set method for propagating interfaces. *J. Comput. Phys.*, 118(2):269–277, 1995.

[2] Henning Scharsach Markus Hadwiger Andre Neubauer, Stefan Wolfsberger and Katja Buehler. Perspective isosurface and direct volume rendering for virtual endoscopy applications. In *EUROVIS - Eurographics /IEEE VGTC Symposium on Visualization*, pages 315–322, 2006.

[3] Oliver Klar. Interactive gpu based segmentation of large medical volume data with level-sets. Master‘s Thesis, VRVIS, 2006.

[4] Aaron Lefohn and et al. Interactive, gpu-based level sets for 3d segmentation. In *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2003, 6th International Conference, Montréal, Canada, November 15-18, Proceedings, Part I*, pages 564–572, 2003.

[5] Aaron E. Lefohn. Interactive computation and visualization of level-set surfaces: A streaming narrow band algorithm. Master‘s Thesis, 2004.

[6] Aaron E. Lefohn, J. Kniss, C. Hansen, and R. Whitaker. Interactive deformation and visualization of level set surfaces using graphics hardware. 2003.

[7] Aaron E. Lefohn, Joe M. Kniss, Charles D. Hansen, and Ross T. Whitaker. Interactive deformation and visualization of level set surfaces using graphics hardware. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 11, Washington, DC, USA, 2003. IEEE Computer Society.

[8] Aaron E. Lefohn, Joe M. Kniss, Charles D. Hansen, and Ross T. Whitaker. A streaming narrow-band algorithm: Interactive computation and visualization of level sets. 2004.

[9] Aaron E. Lefohn and R. Whitaker. A gpu-based, three-dimensional level set solver with curvature flow. In *University of Utah tech report*, pages 02–017, 2002.

[10] Frank Losasso, Tamar Shinar, Andrew Selle, and Ronald Fedkiw. Multiple interacting liquids. *ACM Trans. Graph.*, 25(3):812–819, 2006.

[11] Stanley Osher and Ronald Fedkiw. *Level set methods and dynamic implicit surfaces*. Springer, 2003.

[12] Stanley Osher and James A Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, pages 12–49, 1988.

[13] Martin Rumpf and Robert Strzodka. Level set segmentation in graphics hardware. In *Proceedings of IEEE International Conference on Image Processing (ICIP'01)*, volume 3, pages 1103–1106, 2001.