Vis3D+: An Integrated System for GPU-Accelerated Volume Image Processing and Rendering

I. Nisar and T. McInerney(✉)

Department of Computer Science, Ryerson University,
Toronto, ON M5B 2K3, Canada
tmcinern@ryerson.ca

Abstract. This paper presents a prototype 3D image processing and rendering system that extends an existing interactive 3D image visualization system. The extensions consist of software modules implemented using graphics processing unit (GPU) programs known as "compute shaders". Compute shaders are able to utilize the massively parallel, general-purpose computing capabilities provided by modern GPUs and can also be tightly integrated as new stages in a GPU-based volume and surface rendering pipeline. The compute shaders in this paper are designed to support the execution of volume image processing algorithms, as well as to support the interactive editing of the algorithms' output. An example volume image processing algorithm known as level set segmentation is implemented and demonstrated. A new editing module is developed that enables user modification of the segmentation algorithm's output by extending a pre-existing volume "painting" interface.

1 Introduction

The general purpose computing capability of GPUs, known as GPGPU, is well-suited for efficient processing of medical volume images. Volume images are typically represented as a 3D grid of *voxels* (i.e. volume elements), and many volume image processing algorithms are data-parallel in nature, requiring repeated operations on individual voxels or on a small local neighborhood of voxels. Modern volume visualization systems provide real-time volume rendering typically by programming the fragment shader stage of the GPU-based rendering pipeline to accommodate the 3D grid structure of a volume image. However, few of these systems are able to integrate general volume image processing algorithms, such as image filtering and segmentation, into this GPU pipeline in a flexible and modular manner. Recently, a new programmable stage of the OpenGL [1] pipeline has been made available on graphics hardware called a "compute shader". Compute shaders can execute general purpose numerical calculations and can be inserted into various stages of the rendering pipeline.

In this paper we describe extensions to our existing interactive volume visualization system [2]. The extensions consist of volume image processing capabilities which are added to the system in a tightly integrated yet modular fashion.

© Springer International Publishing Switzerland 2015
G. Bebis et al. (Eds.): ISVC 2015, Part I, LNCS 9474, pp. 830–841, 2015.
DOI: 10.1007/978-3-319-27857-5_74

Our existing system is based on a well-known open source visualization framework called *ImageVis3D* [3], and combines both volume rendering of 3D medical images with surface rendering of polygonal meshes, and allows users to define 3D regions within the volume, delineated by surface envelopes, using an intuitive "volume painting" style interface. The volume image processing extensions are implemented using general compute shader modules. In this paper, we describe and demonstrate example volume image processing compute shaders for performing image filtering and segmentation, both necessary components for visualizing noisy volume images and for performing image analysis. We have also extended the existing system's front-end volume painting interface. This extension supports the editing of labelled 3D regions outputted by the segmentation algorithm (or other volume image processing modules). Finally, we also add a user-controllable 3D image slice that is rendered together with the volume, allowing the user to edit 3D regions in a slice-by-slice manner and providing precise painting and editing control in noisy volume images.

2 Related Work

Graphics hardware is typically structured such that the rendering process is executed in a staged pipeline fashion and many of the stages are now programmable using a high-level programming language. These programs are commonly referred to as "shaders" and examples are vertex, geometry and fragment shaders. The graphics hardware has quickly evolved resulting in a "unified" shader architecture that provides one large grid of general data-parallel floating-point processors that can be used by the various stages. This hardware advance coincided with the emergence of general purpose computing (GPGPU) on the graphics card, along with API's to create GPGPU programs such as CUDA [4]. It is possible to mix CUDA programs and OpenGL programs in several steps including mapping and unmapping of a buffer into formats understood by CUDA and by OpenGL. As mentioned, compute shaders are a recently released stage of the OpenGL graphics pipeline that not only provide similar general-purpose computation functionality as that of CUDA, but also can be more tightly and seamlessly integrated into the pipeline.

As volume images continue to grow in size due to advances in scanning technology, highly efficient image processing algorithms that can filter and label a 3D image are becomingly increasingly important. Examples of processing algorithms that have been implemented on the GPU include median filtering [5], an implementation of Canny edge detection [6], nonlinear anisotropic diffusion-based 3D image denoising using CUDA [7], and level set segmentation [8]. For a recent and thorough survey of medical volume image processing on the GPU, the reader is referred to Eklund et al. [9].

3 Vis3D+

In this section we describe the various GPU-based modules that we integrated to extend the existing interactive volume visualization system. The new modules

consist of the following: compute shaders providing basic volume image filtering in the form of Gaussian smoothing and edge detection, a compute shader that implements a variant of the level set segmentation algorithm [10], and compute shaders and modifications to an existing volume rendering fragment shader to extend the existing system's 3D ROI painting mechanism for use in ROI editing.

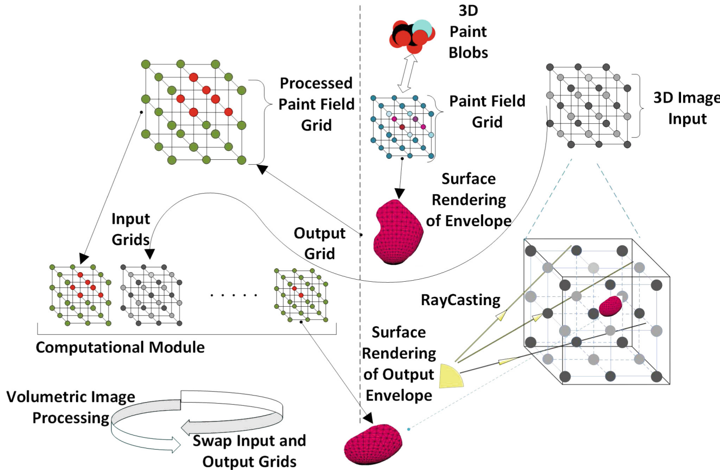


Fig. 1. High-level overview of the extended framework.

Volume processing algorithms, such as filtering, can be inserted into the processing pipeline of the extended system as long as they are parallelizable at the 3D grid point level. Furthermore, the filters can be cascaded - the output of one filtering stage can be used as input to the next. In this paper we implemented Gaussian Filtering to smooth a volume image and edge detection using a simple image gradient magnitude calculation. The output of these cascaded image filtering stages are used as input to a level set segmentation algorithm. We have chosen level set segmentation [10] to showcase our GPU-based volume image-processing support because it can segment topologically complex objects, is a highly parallelizable algorithm and fits well with the existing 3D painting interface. We have used Shader Storage Buffer Objects [1] as buffers, using their binding points as input or output hooks, allowing different shaders to pick up the same buffer and process them as they see fit, avoiding copying or moving data around on the GPU.

Figure 1 shows the extended system, with the left side of the vertical dotted line showing the extensions added in this paper. A compute shader module (Fig. 1 lower left) accepts a volume image grid and algorithm global parameters as input, iteratively executes the algorithm, and generates an output grid which is then used for volume rendering, if desired. In the extended system, the 3D painting interface can be used to define 3D regions of interest as an optional input to any volume image-processing algorithm. The upper middle and upper

left part of Fig. 1 shows an optional input grid generated from the result of painting a 3D ROI. More complex compute shader programs, such as the level set segmentation compute shader, can make use of this input. In the case of the level set segmentation algorithm for example, the algorithm refines the 3D region and labels this region in an output grid (Fig. 1 lower left). The output grid can then be optionally input to a geometry shader where a Marching Cubes algorithm will generate a boundary surface representation of the labeled 3D region.

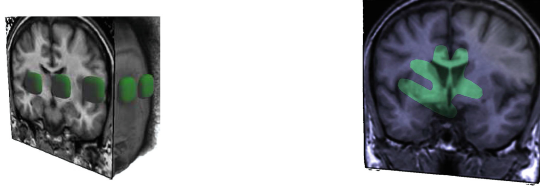


Fig. 2. Left: paint brush "tip" blob, rendered opaque here, can slide along a 3D slice clip plane. Right: applying paint strokes to a slice plane (blob envelope is transparent).

We have extended the system's original 3D painting interface to support painting a 3D ROI on a 3D user-orientable image slice plane. We alter the volume ray casting algorithm in the fragment shader to generate a 3D slice plane that is rendered along with the volumetric data during the volume rendering stage (Fig. 2), achieving the effect of a clipped volume rendering of the data. The volume ray casting is altered by computing the start of each ray from where it intersects the clip plane. These starting ray sample points ensure that everything in front of the plane is clipped away. The starting ray sample points are then used to look up the corresponding image intensity value in the volume image via interpolation. These volume image samples are mapped, using a simple transfer function, to a color and an opacity value and each sample corresponds to a fragment which will appear as a screen pixel. The fragments are shaded using the normal vector of the clip plane rather than a normal vector computed from the volume image. If the fragments are mapped to an opacity equal to 1.0, the ray casting algorithm is terminated for this ray; otherwise, the ray casting algorithm continues as usual.

It is often not possible to use a TF to isolate and volume render a target structure in noisy volumes, preventing the direct painting on the surface of the structure to create a surrounding envelope. This situation also occurs when the target structure is adjacent or connected to neighboring structures with similar intensity characteristics. In these cases, the user can use the image slice-plane painting approach, along with "flattened" superellipsoid paint blobs (Fig. 2 left), to define a 3D ROI that envelopes a cross-section of a target anatomical structure. The thickness of the flattened paint blobs can be precisely controlled by the user to range from a single slice thickness to many slices thick. The user can

paint thick envelopes (i.e. several image slices thick) on several cross-sectional slices of the target structure such that these slice-painted envelopes overlap. The slice-painted envelopes are automatically blended to form a single envelope tightly bounding the entire target structure.

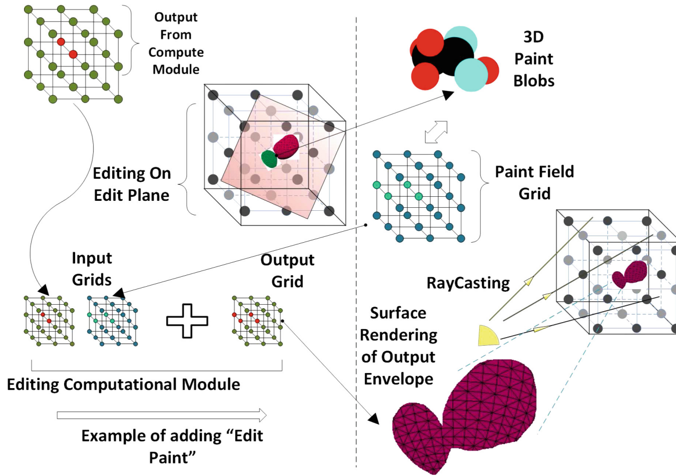


Fig. 3. High-level overview of editing a labelled 3D region via 3D slice paint painting.

Some image processing algorithms, including the level set segmentation algorithm, generate 3D grids with labeled regions. In our extended system, these labeled regions can be interactively edited by using the painting interface to erase parts of the region or to add "edit paint" to them. An illustration of this process is shown in Fig. 3. The left side of the dotted line depicts the slice plane painting and editing extensions. The volume image-processing algorithm for erasing and adding of labels to the voxels is described in Sect. 4.4. The appearance of the voxels within the labeled region can be controlled with a separate transfer function, allowing these voxels to be volume rendered using a distinctive "highlight" color. The altered volume ray-casting algorithm shades the color of voxels that are on the slice plane using the normal vector of the slice plane. The color of the labeled voxels is a blend of the highlight color and the color of the voxel intensity value assigned via the transfer function. The result is the labeled region voxels appear as semi-transparent and highlighted 2D "paint" (Fig. 2 right). Furthermore, the user can dynamically set the flattened superellipsoid paint blob thickness to be just thicker than a single slice plane and the surface of the paint blob can be made completely transparent. Any voxels on the slice plane that are inside the blob can therefore also be made to appear as semi-transparent and highlighted 2D paint (Fig. 2 right). Thus, this special slice-plane rendering capability gives the user the illusion of erasing and adding 2D paint to the labeled 3D region. The user visually discerns the boundaries of the target structure underneath the

semi-transparent 2D paint of both the labeled region and the paint blob. Corrections can be made to the labeled region on the current slice plane. The user can then continue to another oriented slice plane to make further corrections.

4 Implementation

This section provides an overview of the extended system implementation. Details can be found in [11].

4.1 Initial Level Set Construction

In the level set segmentation method, a surface (in 3D) is defined to be the zero level set of a continuous function, $\phi(t, x, y, z)$. The movement of the level set surface is governed by an evolution equation of this function. On a computer, the level set function ϕ is sampled at points on a regular 3D grid and is referred to as the ϕ-grid. Typically the ϕ-grid dimensions are set equal to the input volume image dimensions. The level set method is initialized by creating an initial surface envelope that either loosely surrounds the target structure or is contained inside it [10]. We use the user-painted surface envelope to construct the initial level set function ϕ_0. Specifically, to initialize the ϕ_0-grid, at each grid point we determine if it is inside, outside, or on the painted envelope boundary using the blended paint blobs' defining implicit function [12]. From Li et al. [10] the initial level set function, ϕ_0, is defined at each grid point of a 3D grid as:

$$\phi_0(x, y, z) = \begin{Bmatrix} -\rho & (x, y, z) \in \Omega_0 - \partial\Omega_0 \\ 0 & (x, y, z) \in \partial\Omega_0 \\ \rho & (x, y, z) \in \Omega - \Omega_0, \end{Bmatrix}, \tag{1}$$

where Ω is the volumetric domain, Ω_0 is a subset of the volumetric domain containing all points inside the painted envelope and $\partial\Omega_0$ is the set of all points exactly on the boundary of Ω_0 (i.e. the painted envelope boundary surface). Values inside the painted envelope are marked as $-\rho$ and outside are marked as $+\rho$, where ρ is a constant [10]. This initial level set function construction process is implemented in a separate compute shader, which accepts the array of paint blobs defining the painted envelope as input and writes out a 3D ϕ-grid.

4.2 Level Set Implementation

The level set segmentation algorithm uses edge image features to determine if the evolving level set surface has reached the boundary of the target structure. Input volume images are commonly convolved with a smoothing filter to remove noise before performing edge detection. We currently use two cascaded compute shaders - a Gaussian filter shader and an edge detection shader - to compute the edge detected image. The level set evolution is computed inside another compute shader, *updatephi.cs*. It takes an input ϕ-grid, along with the edge

detected intensity grid. It then outputs a grid containing updated values of the function ϕ referred to as ϕ-gridOut. As mentioned previously, we use buffer binding indices as hooks to interchange the input and output ϕ-grid buffers, which avoids copying or moving the grids. The final output of the level set segmentation algorithm, ϕ-gridOut, contains scalar values. This output grid is sent to a geometry shader that executes the Marching Cubes algorithm and generates a mesh of triangles representing the zero level set surface.

4.3 Compute Shader Implementation

Shader Buffer Objects containing 3D grids of scalar values are stored as a contiguous one-dimensional array on the GPU. In a GPU thread, we are able to use the thread identifier to compute a 3D grid point position. Individual threads are grouped in a local work-group or block. The local work-groups together make up the larger global work-group. Shader Buffer Objects are stored in the GPU's *L2* Cache and each thread looks up its required data from textures that are backed by these buffer objects.

In Kuo et al. [13], the authors mention several criteria for stopping the level set evolution, which is typically evaluated after each iteration of the segmentation algorithm. We use a simple but often effective stopping condition. The volume of the segmented 3D region is denoted V and the difference in the volume between the previous and current iteration is denoted ΔV. The evolution is stopped when $\Delta V/V$ falls below a small threshold value (e.g. 0.005). We have utilized atomic operations in the compute shader threads, supported as of OpenGL 4.3, to implement the simple stopping condition. Atomic operations write to (or read from) shared memory uninterrupted; if multiple threads attempt to access the same location simultaneously, they will be serialized. We use atomic operations that operate on a special stopping condition buffer that stores the previous and current computed volume of the segmented region, as well as the number of grid points that have been processed. When a thread begins executing at a current time step t (i.e. iteration), it checks the stopping condition ($\Delta V/V < 0.005$). If the condition is met, the thread returns. Otherwise the thread uses an atomic add operation to add 1 to the number of processed grid points. The thread then executes the level set evolution equation for its assigned grid point. If the ϕ function field value for this grid point is less than 0, we use the atomic add operation to add 1 to the current segmented region volume; that is, the number of voxels (i.e. grid points) inside the segmented region is used as a measure of the region's volume. When all grid points have been processed, we set the previous segmented region volume equal to the current volume and then reset the current volume and number of processed grid points to 0.

4.4 Editing a Labeled 3D Region

The user uses the painting interface to erase or add labels to the labeled region outputted from the segmentation algorithm in the form of the ϕ-grid. Edit compute shaders accept the ϕ-grid as input as well as the array of paint blobs defining

the edit region. In Museth et al. [14], the authors define editing operators based on level sets, which provide advantages such as avoiding boundary surface self-intersection and coping with topological genus changes. In this paper, we use a simple version of the Constructive Solid Geometry (CSG) operations mentioned in Museth et al. [14]. A remove or erase operation is analogous to the difference operation in CSG and an add operation is equivalent to a union operation. Both the erase and add operations are parallelized on the ϕ-grid points. For the add operation, at each ϕ-grid point, we read its field value and store it at a corresponding output grid point. Using the array of painted blobs and the blended blobs' inside-outside function, we then check if this output grid point is inside the painted region. If so, we overwrite the field value from the ϕ-grid, with a $-\rho$ value, making it a part of the labeled region. For the erase operation, at each ϕ-grid point, we read its field value and set the corresponding output grid point to the same value. We then check if this output grid point is inside the painted region and if its value is less than 0 (i.e. indicating it is currently part of the labeled region). If so, it is overwritten with a $+\rho$ value, removing it from the labeled region.

5 Experimental Results

The paper is concerned with the modular integration of volume image processing algorithms into an existing volume image rendering pipeline, and with the algorithms' compute shader implementation. Consequently, in this section we present experiments to demonstrate a working compute shader implementation of the level set segmentation algorithm and some rough measurement of its performance, rather than on a formal analysis of segmentation accuracy and efficiency. Level set segmentation has been heavily researched over the years and its accuracy measured numerous times. The reader is referred to [15] as a representative example. In this paper, we currently use the system's 3D slice plane capability and visually inspect slices containing the target anatomical structures as well as the segmentation "paint" to assess segmentation accuracy. In addition, our level set segmentation implementation currently utilizes simple Gaussian smoothed gradient magnitude edges to stop the level set evolution. More accurate edges may lead to improved segmentation performance. We are currently investigating GPU-based median filtering [5] combined with more sophisticated edge detectors [6] and we plan to collect quantitative data of segmentation accuracy in the immediate future. Finally, all experiments were performed on a machine with an NVIDIA GTX 570M graphics card containing 1.5 GB GDDR5 of random access memory (RAM) and 7 streaming multiprocessors, each with 48 cores.

5.1 Segmenting Synthetic Data Sets

We use two synthetic "cloverleaf" volume images, with voxel values inside the cloverleaf smoothly graded and background voxels set to 0, and with dimensions $128 \times 128 \times 128$ and $256 \times 256 \times 256$, respectively. We painted an initial envelope

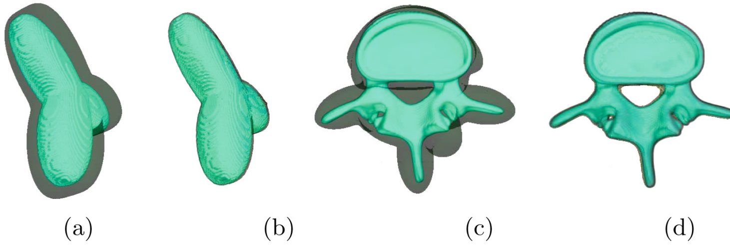


Fig. 4. Segmenting a $256 \times 256 \times 256$ synthetic cloverleaf data set. (a) Initial painted level set surface. (b) Final segmentation result. (c)(d) Segmenting a $168 \times 160 \times 92$ CT image of a human vertebra phantom. (c) Initial painted level set surface. (d) Final segmentation result.

surrounding the cloverleaf directly in 3D using three paint brush strokes (Fig. 4a). Figure 4b shows the segmentation after 100 iterations. The time required for the segmentation was 39 s. The result is visually very accurate. For the smaller data set (i.e. $128 \times 128 \times 128$ voxels) only 50 iterations (2.4 s) were required to generate an accurate result. The parameter settings for all tests, real and synthetic were $\mu = 0.02$, $\gamma = 5$, $\lambda = 5$, $\epsilon = 1.5$, $\tau = 5$. See Li et al. [10] for parameter descriptions.

In a second test, we use a $168 \times 160 \times 92$ CT volume image of a topologically complex human vertebra phantom (Fig. 4c and d) to further validate a working compute shader implementation of the segmentation algorithm. We painted an initial envelope without holes, directly in 3D, that surrounds the vertebra. The level set segmentation algorithm correctly captures the topology of the vertebra. The segmentation ran for 200 iterations. The approximate time required for the segmentation was 11 s and the result again is visually very accurate.

5.2 Segmenting a Real Data Set

We also ran our segmentation algorithm on a $240 \times 240 \times 192$ MRI brain volume to segment the lateral ventricle. Segmentation of structures in MRI scans is challenging due to noise, voxel intensity inhomogeneity and the similar voxel intensities of neighboring structures. We used the system's 3D slice painting facility and painted an envelope on several slices containing the lateral ventricle. Figure 5a–c shows the initial level set envelope, the segmentation result and an expert manually segmented guide. The 3D slice painting facility allows the user to quickly paint a highly accurate initial level set (Fig. 5a) - an important factor for segmentation algorithm robustness. The segmentation ran for 75 iterations and required approximately 14 s. Figure 5d–f shows several 3D slice views to demonstrate segmentation accuracy.

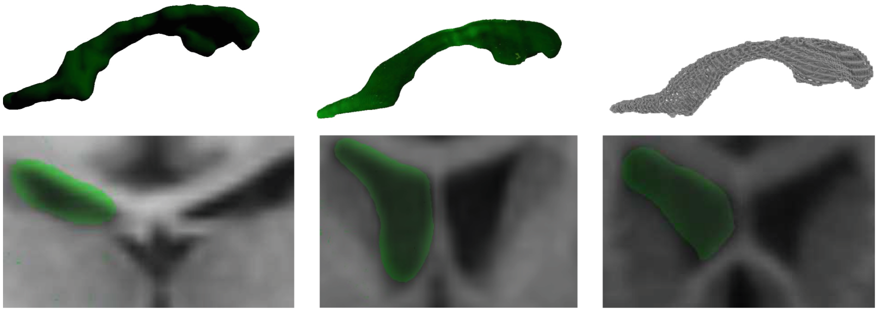
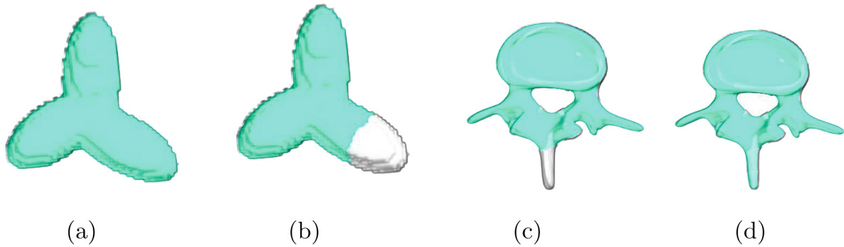
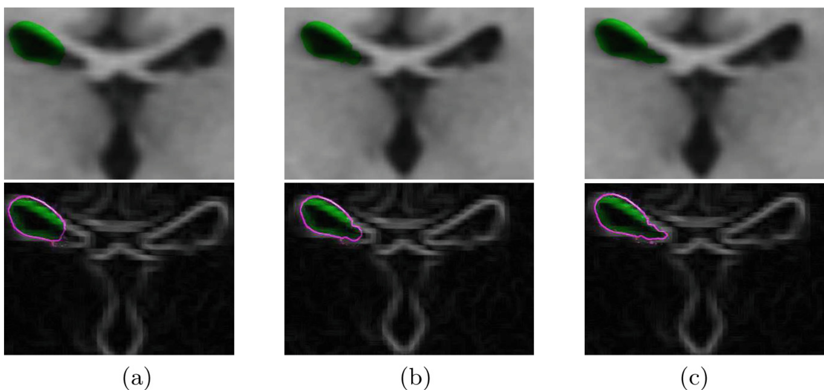


Fig. 5. Segmenting the lateral ventricle in a $240 \times 240 \times 192$ MRI brain image. Top row left: initial 3D slice painted level set surface, middle: final segmentation result, right: volume rendering of manually segmented ventricle. Bottom row: example image slices showing segmentation result.



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

Fig. 6. (a)(b) Example of erasing paint on a synthetic clover leaf and, (c)(d) adding paint to the vertebrae segmentation.



(a) (b) (c)

Fig. 7. Editing a 3D image slice of an MR brain volume to correct the lateral ventricle segmentation by adding paint. The bottom row shows corresponding edge detected images.

5.3 Editing

In this section we demonstrate the editing capability. Figure 6 shows a simple example of direct 3D editing of the cloverleaf (erasing the segmentation) and the vertebra phantom (adding edit paint). In Fig. 7 we demonstrate editing of the lateral ventricle segmentation result on a 3D image slice of the MR brain volume. The ventricle is under-segmented on this slice and is corrected by adding paint using a small paint brush tip. In this example, a circular brush tip is used. However, the superquadric brush tip can be shaped to a flattened ellipsoid or rectangular solid to better match a desired boundary curvature.

6 Conclusions

The paper demonstrates the potential of compute shaders for creating a flexible and modular software system that tightly integrates interactive 3D medical image visualization and processing. The prototype system described in this paper uses a single intuitive painting interface for all selection, initialization and editing interactions with the data, and supports the processing of noisy volume images by integrating a 3D slice plane view directly with the volume rendered view. Further improvements can be made to the framework. The GPU level set implementation can be further optimized by implementing the narrow band algorithm using stream compaction as in Roberts et al. [8] to obtain a subset of the current narrow band as it evolves. Slice-by-slice editing can be labor intensive if many slices require editing. We intend to experiment with interactively painted "barriers" that reinforce target structure boundaries in regions with no edge features. The idea is to allow the user to paint thin 3D regions that are then used to modify the edge detected image used by the level set segmentation algorithm.

References

1. Shreiner, D., Sellers, G., Kessenich, J.M., Licea-Kane, B.M.: OpenGL Programming Guide, Version 4.3, 8th edn. Addison-Wesley Professional (2013)
2. Faynshteyn, L., McInerney, T.: Context-preserving volumetric data set exploration using a 3D painting metaphor. In: Bebis, G., Boyle, R., Parvin, B., Koracin, D., Fowlkes, C., Wang, S., Choi, M.-H., Mantler, S., Schulze, J., Acevedo, D., Mueller, K., Papka, M. (eds.) ISVC 2012, Part I. LNCS, vol. 7431, pp. 336–347. Springer, Heidelberg (2012)
3. Fogal, T., Kruger, J.: Tuvok, an architecture for large scale volume rendering. In: Proceedings of the 15th International Workshop on Vision, Modeling, and Visualization (2010)
4. Nickolls, J., Buck, I., Garland, M., Skadron, K.: Scalable parallel programming with CUDA. Queue **6**, 40–53 (2008)
5. Perrot, G., Domas, S., Couturier, R.: Fine-tuned high-speed implementation of a gpu-based median filter. J. Signal Process. Syst. **75**, 185–190 (2014)
6. Luo, Y., Duraiswami, R.: Canny edge detection on nvidia cuda. In: Computer Vision and Pattern Recognition Workshops, CVPRW 2008, pp. 1–8 (2008)

7. Schwarzkopf, A., Kalbe, T., Bajaj, C., Kuijper, A., Goesele, M.: Volumetric nonlinear anisotropic diffusion on GPUs. In: Bruckstein, A.M., ter Haar Romeny, B.M., Bronstein, A.M., Bronstein, M.M. (eds.) SSVM 2011. LNCS, vol. 6667, pp. 62–73. Springer, Heidelberg (2012)
8. Roberts, M., Packer, J., Sousa, M.C., Mitchell, J.R.: A work-efficient GPU algorithm for level set segmentation. In: Proceedings of the Conference on High Performance Graphics, HPG 2010, pp. 123–132 (2010)
9. Eklund, A., Dufort, P., Forsberg, D., LaConte, S.M.: Medical image processing on the GPU - past, present and future. MIA **17**, 1073–1094 (2013)
10. Li, C., Xu, C., Gui, C., Fox, M.D.: Level set evolution without re-initialization: a new variational formulation. In: Computer Vision and Pattern Recognition, CVPR 2005, pp. 430–436 (2005)
11. Nisar, I.: Vis3D+: a tightly integrated GPU-accelerated computation and rendering framework for interactive 3D image visualization. Master's thesis, Department of Computer Science, Ryerson University, Toronto, ON, Canada (2015)
12. Faynshteyn, L.: Context-preserving volumetric data set exploration using a 3D painting metaphor. Master's thesis, Department of Computer Science, Ryerson University, Toronto, ON, Canada (2012)
13. Kuo, H.C., Giger, M.L., Reiser, I.S., Boone, J.M., Lindfors, K.K., Yang, K., Edwards, A.: Level set segmentation of breast masses in contrast-enhanced dedicated breast ct and evaluation of stopping criteria. J. Digital Imaging **27**, 237–247 (2014)
14. Museth, K., Breen, D.E., Whitaker, R.T., Barr, A.H.: Level set surface editing operators. ACM Trans. Graph. **21**, 330–338 (2002)
15. Li, C., Huang, R., Ding, Z., Gatenby, C., Metaxas, D.N., Gore, J.C.: A level set method for image segmentation in the presence of intensity inhomogeneities with application to MRI. IEEE Trans. Image Process. **20**, 2007–2016 (2011)