Fast segmentation, conversion and rendering of volumetric data using GPU

Ciril Bohak, Anže Sodja, Matija Marolt
Faculty of Computer and Information Science,
University of Ljubljana,
Tržaška 25, 1000, Ljubljana,
Slovenia.
Email: {ciril.bohak, matija.marolt}@fri.uni-lj.si

Uroš Mitrović, Franjo Pernuš
Faculty of Electrical Engineering,
University of Ljubljana,
Tržaška 25, 1000, Ljubljana,
Slovenia.
Email: {uros.mitrovic, franjo.pernus}@fe.uni-lj.si

***Abstract*—In this paper we present a proof-of-concept implementation of fast volumetric data segmentation, conversion to polygonal mesh geometry and rendering. All parts of the method are implemented on the graphical processing unit, which allows high degree of parallelization. Implementations of presented algorithms are done in the OpenCL framework and are integrated in blood vessel visualisation software Neck Veins. This paper presents where and to what degree parts of method can be parallelized. In results we also show to what degree we can speed-up the implementation by using parallel computing power of the graphical processing units.**

***Keywords*—volume data segmentation, GPU computation, medical visualisation, 3D visualisation.**

I. Introduction

With development of imaging techniques more data is available for supporting physicians decisions with diagnosis as well as for support during the medical procedures such as surgeries. A very popular and useful method is also 3D imaging which allows physicians to get a volumetric scope to the inner structure of the human body. The volumetric CT scanning [1], [2] was introduced in 1991 and is widely used now as support in making diagnoses. In the following years many improvements of the volumetric CT scanning techniques were presented that allow marking the specific tissue, and thus extracting it from the rest. This also allows use of minimally invasive endovascular image-guided interventions for treating different cerebrovascular conditions, where detailed knowledge of the structure of the target and surrounding tissue is needed for achieving the expected results. While the 3D imaging can not be used for real-time monitoring during the surgical procedures the surgeons have to obtain and memorise the structure of the tissue, in our case the blood vessels, as precisely as possible.

Modern technology also allows us better visualisation. Even though we have 3D data available we still had to use 2D displays for presenting the data to the user. This still means that users loses some part of spatial information. To get better idea on how an object is placed in the space a 3D visualization devices such as stereoscopic displays that give better representation of 3D models with depth feeling.

There are many aspects that one has to take into account when visualising the 3D data. Since the original data is volumetric we have to decide what method to use for rendering the final image on screen. There are different rendering techniques one can use, such as volumetric rendering [3], [4], or we can first create the geometric model from the volume data and use conventional real-time rendering techniques such as scanline rendering [5] or ray tracing [6]. To obtain the geometry for rendering with conventional techniques the data needs to be converted from volumetric form into geometric - polygonal form. This is more thoroughly discussed in following section.

Goal of our method is to show that fast conversion from volumetric data to mesh geometry and rendering are possible with use of computational power of modern graphics cards. We have created an example realisation using simple techniques, which could later be replaced with more complex ones for obtaining better final result as well as performance.

In next section we present the related work and it's connection with presented approach, next we present our method for fast background extraction and volume to mesh conversion in Section III. At the end we give the future work guidelines and conclusions.

II. Related Work

Creating fast segmentation, polygonization and visualisation of the volumetric data is computationally hard problem. For bigger volumes it can not be done on everyday hardware. To speed-up the process one can use the high computational power of today's graphical processing units (GPUs) [7]. Modern GPUs are capable of performing general purpose computation as presented in [8]. Use of GPUs where high degree of parallelization is possible, can result in great increase of speed. To use GPU's capabilities one has to program in specific way. There are several possible ways for developers to use GPU's computational power. While some frameworks are dedicated for specific hardware (e.g. Nvidia Cuda), we use Open Computing Language (OpenCL) [9], which is supported by most common graphics card manufacturers. This is an important feature in development of platform independent software.

First step in our approach is segmenting the vascular tissue from the background in the volumetric data. There are many segmentation methods that give good results for such problems. Since we are only creating proof of concept implementation we have selected a simple method for background extraction - thresholding. One example of such method is Otsu's method presented in [10], where the threshold is automatically selected by the discriminant criterion, to maximize the separability of the final classes in gray levels. The method allows high level of parallelism, thus it is appropriate for implementation on GPU.

Next in line is conversion of the volumetric data into the polygonal geometry which is used in rendering. We are again using a simple method called the Marching cubes [11]. A good thing about the Marching cubes algorithm is that it allows high level of parallelism and is thus good for implementation on GPU. There are many other methods that could be used, such as methods described in [12]–[14]. We have used the simple method which still gives good results on our data.

After the conversion we also have to find the connected components of our model. Main reason for this is that we do not want to show the components that are too small or are too far from the main model and are not connected to it. This step could be done even before conversion of models with methods such as [15]. In our case we are excluding isolated components after conversion process with own implementation of discovering such components.

In next section we are presenting the implementation details of our method, we show the obtained results and present the application with integrated implementation of method for computations on GPU.

III. Method

Our method for fast segmentation and rendering of the volumetric data was designed as proof of concept that such method can be developed with the use of current graphics hardware and implementation of parallel algorithms for individual parts of the process. The method consists of several steps presented in following subsections; segmentation of the data, conversion from the volumetric data to the polygonal geometry and search for the connected components.

A. Segmentation

The volumetric data, in our case data of blood vessels, consists of two parts. The vessels and the background. Since we only want to display the vessels we first have to segment them from the background. In volumetric data the vessel tissue has a greater value than the surroundings. For obtaining only the vessel tissue we use a simple thresholding approach - Otsu's method [10] for selection of the optimal threshold in the grayscale images. The method can be described with the following steps:

1) calculation of model's histogram;
2) calculate the frequency of each histogram value;
3) calculate the distribution function and average value of both classes for each histogram value;
4) search for the histogram value that maximizes the dispersion between classes of background and foreground, the obtained value is our threshold value;
5) binarization of the data according to the threshold into values 0 and 1.

We leave out the last step (5) of presented method - binarization, due to later use of Marching cubes algorithm which also takes into account the values of individual voxels in the volumetric data for better approximation of the produced mesh.

In parallelisation we have to take into account that some steps have to be done with splitting the data into separate parts, calculating the step on part of the data and back propagate the results to the previous stages repeating the calculation at the higher level and calculating the result at top level. An example of such problem is searching for the maximum in the data. Same is true for searching the appropriate threshold, where only parts can be well parallelized.

As part of the segmentation step we also apply Gauss smoothing. Mostly we are used to use Gauss smoothing in one or two dimensions. In our case we use three dimensional Gauss function for smoothing the volumetric data before conversion into the polygonal mesh geometry.

Due to good separability of the Gauss function, it can be well parallelized and thus well implemented on GPU. We can split the calculation of smoothing into next steps:

1) calculation of window for single dimension for selected value;
2) averaging over data values in first dimension;
3) averaging over data values in second dimension;
4) averaging over data values in third dimension.

The calculation of the smoothing could be performed even faster with use of FFT implementation of the Gauss filter function.

B. Volume to geometry conversion

Conversion from the volumetric to the polygonal mesh geometry was done using Marching cubes algorithm [11]. The algorithm takes into account values of individual and neighbouring voxels.

The Marching cube algorithm determines for each voxel how to approximate it with polygonal mesh. For each voxel there are 256 different possibilities of geometry according to the neighbouring values. This possibility can be presented with 14 different examples discarding rotations as shown in Figure 1.

For our purposes we have implemented an array of of 256 possible output geometries of algorithm. Next we calculate which output geometry is used for individual voxel. For better precision of geometry we use the values in individual vertices in linear approximation of surface, thus obtaining smoother polygonal output geometry.

The Marching cube algorithm is used for calculating the polygonal geometry as well as the surface normals used for accurate rendering. We calculate the normals for every vertex

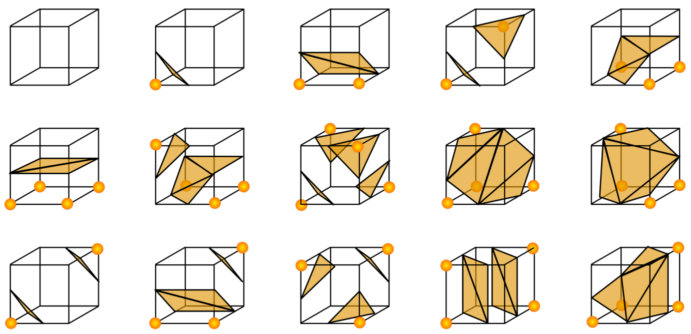

Fig. 1. Figure shows all possible examples of polygonal approximation of voxel values discarding rotations (image source: Wikipedia).

of geometry using the gradient approach with neighbouring vertices, thus obtaining smoother interpolation.

The algorithm can be well parallelised and thus allows good implementation on GPU. We can use the degree of parallelisation for each individual voxel for best utilisation of GPU.

C. Finding connected components

Finding the connected geometry presents an important role in finding the model that represents the main vessel tissue and parts of tissue that are not directly connected with the main model. There are different approaches that can be used for finding such components. First approach could be used on the volumetric data, however we have decided to use the approach after conversion to the polygonal geometry.

In our case we first merge the obtained geometry that touches each other by finding vertices that are used in definition of several triangles. Such triangles are representing same connected geometry. In such way we can obtain the components and their sizes in number of triangles. With this we can draw only the components of desired size.

It is not the best solution since we do not check whether components could present same vessel, but it is good way to get rid of the noise in the data.

IV. Implementation

All steps of the described method were implemented two times. First in Java programming language and later in OpenCL. The Java implementation was intended for testing whether implementation is done right. For fast usage we have done implementation in OpenCL for fast computation with use of GPU.

The basic GPU implementation did not run fast, but with improvements and reimplementation of individual steps the method of segmentation, conversion and rendering could be performed in real-time on high-performance graphics cards (i.e. Nvidia Quadro series). There were many issues with creating hardware independent OpenCL code, due to many differences between different products.

The method was integrated into the Neck Veins application. The idea of the application is to create the visualization platform for displaying 3D medical data - in our case the data represents blood vessel in head of the patient, which would be used for search of aneurysms. In first step the developed application allowed the visualization of precomputed patients data where 3D images were already segmented and only model of patient's head vessel was imported and displayed. Later on many extensions of original application were developed where different functionalities were implemented such as support for 3D stereoscopic display of 3D data and 3D mouse integration for more natural interaction with the model of vessels.

The application is developed under open source licence and is hosted in online repository GitHub under name Neck Veins[1]. The application can be downloaded and tested, however the data is not available due to patient confidentiality. We also invite new developers to contribute their share in extending and improving the applications. One can see the application in Figure 2.

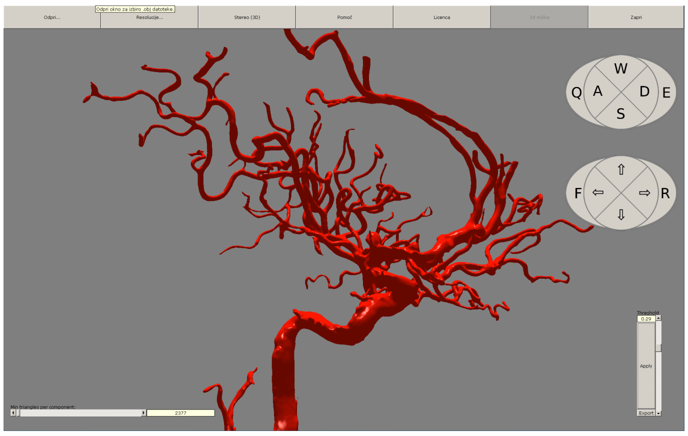

Fig. 2. Figure shows screenshot of application Neck veins.

V. Experiments and Results

Part of our work was also to compare the speed of different implementation of methods. As described above we have implemented the method in Java as well as in OpenCL. Testing system was Intel Core 2 Duo T6600 2.20GHz, 4GB RAM, Nvidia GT240M 1GB, Windows 8. We have compared the time needed for completing computation on the CPU and the GPU. In Tables I - III we present the times needed for computation as well as speedup with use of GPU for the individual part of presented method.

TABLE I
TIME NEEDED FOR CALCULATING GAUSS SMOOTHING.

Volume size	CPE (Java)	GPU (OpenCL)	Speedup
512 × 512 × 390	81.652s	2.997s	27.24×
256 × 256 × 195	5.462s	0.595s	9.18×
128 × 128 × 97	0.673s	0.268s	2.51×

One can see that the speedup factor is different for each individual part of presented method. While speedup factor is actually less than 1 in case of Otsu's method due to high

[1]https://github.com/asodja/Neck_Veins

TABLE II
TIME NEEDED FOR CALCULATING OTSU'S METHOD.

Volume size	CPE (Java)	GPU (OpenCL)	Speedup
512 × 512 × 390	0.695s	1.422s	0.49×
256 × 256 × 195	0.111s	0.236s	0.47×
128 × 128 × 97	0.034s	0.127s	0.27×

TABLE III
TIME NEEDED FOR CALCULATING MARCHING CUBES.

Volume size	CPE (Java)	GPU (OpenCL)	Speedup
512 × 512 × 390	22.186s	1.969s	11.27×
256 × 256 × 195	2.295s	0.439s	5.23×
128 × 128 × 97	0.454s	0.158s	2.87×

overhead with allocating memory on GPU, sending data to GPU and getting data back from it, in most cases where there is more need of computation we get as high speedup as 27-times in case of Gauss smoothing, which could be implemented even faster. In Table IV we show the times needed for completing the whole method. In final state we get as high as 30-times increase in the calculation speed which is very promising.

TABLE IV
TIME NEEDED FOR CALCULATING COMPLETE METHOD.

Volume size	CPE (Java)	GPU (OpenCL)	Speedup
512 × 512 × 390	104.476s	3.561s	29.34×
256 × 256 × 195	7.973s	0.612s	13.02×
128 × 128 × 97	0.907s	0.275s	3.30×

VI. CONCLUSIONS AND FUTURE WORK

There are many possibilities of improving the presented method as well as speeding it up even more. Since this is proof-of-concept method, we have selected the basic algorithms as well as their implementations for testing the method on GPU. Many parts of the method could be even more optimised. The Gauss smoothing could be performed with FFT which would increase the speed substantially. Other possible optimisations could be done in the Marching cube algorithm implementation where we should take into account the properties of individual graphics card, since parallelisation could be used even to higher degree if we would use all available units on graphic card.

In the future we are planning on implementing more state-of-the-art methods for the segmentation as well as the conversion for achieving better final results. We are also planning of implementing the volumetric rendering which would allow us to skip the conversion part of the method and render the volumetric data directly. We are also planning on optimising current approach for real-time use on mid-range graphics cards.

During the implementation of presented methods we had several problems. One of biggest was that we had to be very careful with memory consumption. The volumetric data takes quite a lot of space which is not a problem when computing on CPU since allocating up to 2 or more GB of memory is not problematic, it is much different on GPU, where usually we do not have more than 1 GB of memory available. The use of high-performance GPUs would make implementation much easier. The fact that almost every model of graphics card has it's own structure of working groups and cores, it is very hard to write hardware independant OpenCL code, resulting in need for debugging on several machines.

We are satisfied with obtained results which can be tested in an open source software Neck Veins.

ACKNOWLEDGMENT

The product is result of several students as part of their work on Bachelor thesis. The software will be further developed with help of new students in the coming years. We would also like to thank the Laboratory of Imaging Technologies from Faculty of Electrical Engineering at University of Ljubljana. Software was developed as part of collaboration with University Medical Centre Ljubljana.

REFERENCES

[1] C. R. Crawford and K. F. King, "Computed tomography scanning with simultaneous patient translation," *Medical Physics*, no. 17, pp. 967 – 982, 1990.

[2] W. A. Kalender, W. Seissler, E. Klotz, and P. Vock, "Spiral volumetric CT with single-breath-hold technique, continuous transport and continuous scanner rotation," *Radiology*, no. 176, pp. 181 – 183, 1990.

[3] R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," *SIGGRAPH Comput. Graph.*, vol. 22, no. 4, pp. 65–74, Jun. 1988.

[4] M. Levoy, "Display of surfaces from volume data," *Computer Graphics and Applications, IEEE*, vol. 8, no. 3, pp. 29–37, 1988.

[5] W. J. Bouknight, "A procedure for generation of three-dimensional half-toned computer graphics presentations," *Commun. ACM*, vol. 13, no. 9, pp. 527–536, Sep. 1970.

[6] A. Appel, "Some techniques for shading machine renderings of solids," in *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, ser. AFIPS '68 (Spring). New York, NY, USA: ACM, 1968, pp. 37–45.

[7] A. Sodja, "Segmentation of medical 3D volumes on a GPU," 2013.

[8] D. Luebke, M. Harris, N. Govindaraju, A. Lefohn, M. Houston, J. Owens, M. Segal, M. Papakipos, and I. Buck, "Gpgpu: General-purpose computation on graphics hardware," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, ser. SC '06. New York, NY, USA: ACM, 2006.

[9] J. Stone, D. Gohara, and G. Shi, "Opencl: A parallel programming standard for heterogeneous computing systems," *Computing in Science Engineering*, vol. 12, no. 3, pp. 66–73, 2010.

[10] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 9, no. 1, pp. 62–66, Jan. 1979.

[11] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *SIGGRAPH Comput. Graph.*, vol. 21, no. 4, pp. 163–169, Aug. 1987.

[12] C. Kirbas and F. Quek, "A review of vessel extraction techniques and algorithms," *ACM Comput. Surv.*, vol. 36, no. 2, pp. 81–121, Jun. 2004.

[13] D. Lesage, E. D. Angelini, I. Bloch, and G. Funka-Lea, "A review of 3d vessel lumen segmentation techniques: Models, features and extraction schemes," *Medical Image Analysis*, vol. 13, no. 6, pp. 819 – 845, 2009.

[14] M. T. Dehkordi, S. Sadri, and A. Doosthoseini, "A review of coronary vessel segmentation algorithms." *J Med Signals Sens*, vol. 1, no. 1, pp. 49–54, 2011.

[15] V. M. A. Olivera and R. A. Lotufo, "A study on connected components labeling algorithms using gpus," *Graphics, Patterns and Images (SIBGRAPI)*, 2010.