GPU-BASED IMAGE SEGMENTATION USING LEVEL SET METHOD WITH SCALING APPROACH

Zafer Güler[1]andAhmet Çınar[2]

[1]Department of Software Engineering, Firat University, Elazig, Turkey
zaferguler@firat.edu.tr
[2]Department of Computer Engineering, Firat University, Elazig, Turkey
acinar@firat.edu.tr

ABSTRACT

In recent years, with the development of graphics processors, graphics cards have been widely used to perform general-purpose calculations. Especially with release of CUDA C programming languages in 2007, most of the researchers have been used CUDA C programming language for the processes which needs high performance computing.

In this paper, a scaling approach for image segmentation using level sets is carried out by the GPU programming techniques. Approach to level sets is mainly based on the solution of partial differential equations. The proposed method does not require the solution of partial differential equation. Scaling approach, which uses basic geometric transformations, is used. Thus, the required computational cost reduces. The use of the CUDA programming on the GPU has taken advantage of classic programming as spending time and performance. Thereby results are obtained faster. The use of the GPU has provided to enable real-time processing. The developed application in this study is used to find tumor on MRI brain images.

KEYWORDS

CUDA, GPU Programming, Level Set Method, Image Segmentation, Scaling

1. INTRODUCTION

The first and most important step in image analysis is image segmentation. Segmentation is the process of dividing an image that created it, and the computer vision [1] and medical imaging [2] are among the most significant transactions. Another purpose of segmentation is to extract the boundaries of the object or objects in the images. For this purpose, level set method is used to extract object boundaries in images with highly successful results. The level set method is released by Osher and Sethian [3]. After it is released for the first time, it has become a favorite technique for finding, tracking or monitoring of moving surfaces. The overall objective of the level set method is to monitor moving surfaces and curves. Here, the moving surfaces can be traced by the speed term and the speed term depends on the current geometry. By means of this method, topological changes automatically manageable and an object can be divided into more than one object, or vice versa.

Sundarapandian et al. (Eds) : ICAITA, SAI, SEAS, CDKP, CMCA-2013
pp. 81–92, 2013. © CS & IT-CSCP 2013 DOI : 10.5121/csit.2013.3808

Level set method quite suitable for the method such as mud and water physically-based simulations [4], but the method require solving the partial differential equations (PDEs), so the solution requires a high computational cost. Many methods have been developed to deal with this problem. Two major approaches are narrow band [5] and sparse field technique [6]. With these methods, PDE improvements calculated only around the zero level set and the speed improvement has obtained. But enough acceleration could not achieve. Later studies focused on running parallel to the level set method. Thus the parallel algorithms have been developed [7].
Multiprocessor structures emerged in the last ten years, and today began to be used widely. These structures are used in both industrial and academic research, and the difficulties and problems can be solved with these structures. Nowadays, many problems can be implemented easily with this structure. Especially with the rapid advances in video card, running parallel to the level set method has become an important factor to the speed increase.

Graphics processor unit (GPU) to be used for general purpose applications is not an approach that emerged in recent years, but in 2007, with the development of the CUDA architecture, it is expanded rapidly. The CUDA architecture provide to execute general purpose application without knowledge of the graphics processor. GPU-based applications are used not only in the scientific field, but also in other fields that require high performance such as, image and video processing, fluid dynamics simulation [8].

Level set method began to run on the GPU in the early 2000s. The first GPU implementation of the level set method is implemented by Rumpf and Strzodka[18] in 2001. Researchers continued to study in this area until 2007[19, 11]. With the release of CUDA C programming language in 2007, high performance solutions have been obtained [12-15]. One of the important studies on the level set method was presented by Robert et al. [16] and Jalba et al [17]. Robert's method represents approach to implement narrow-band approaches on the GPU and Jalba's method represents approach to implement sparse approaches on the GPU.

In this paper, a novel GPU-based level set method is presented. The proposed method does not include PDE solution, so our algorithm has significantly increased the speed of calculation. We use basic geometric transformations for curve evolution. This paper is organized as flows. In Section 2 provides information on the new generation of GPU and memory management. In section 3 we explain the major details of developed GPU-based algorithm. Finally, results andtime measurements are presented.

2. CUDA ARCHITECTURE

Graphics processors are rapidly developed and become available for general purpose applications. Thus it has become to use in many areas to improve the speed of application. CUDA architecture developed by NVIDIA company and this architecture allows high increase in computing performance. In fact, this performance difference is due to the fact that, the GPU architecture designed for parallel operations. A simple CPU (central processor unit) and GPU architecture are illustrated in figure 1. As shown in the illustration GPU has a large number of arithmetic logic unit (ALU), but its cache memory is low [18].

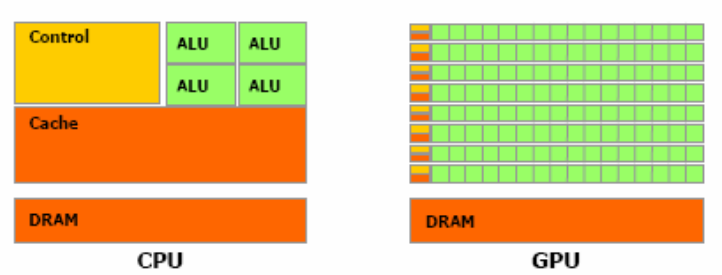


Figure 1.CPU and GPU architecture [18].

GPUs execute a large number of threads on a set of data at the same time. Therefore it is only appropriate for parallel data. Thus the successful result can be obtained. For example if a program contains many flow control, its calculation speed may be reduce rather than increase [7].

2.1. CUDA C Programming Language

A CUDA C programming language introduces a small number of extensions to C language and allows us to define C function called kernel. With kernel we can define and execute N function in parallel. When kernel calls, unique threadId is available. Thus, with the Id number we can determine which thread is currently running within the kernel. Threads are organized in blocks. In the same as thread, blocks have an identification number. So that which block and which thread actually running, can be easily determined in large data sets. But the number of thread in the block is limited. Current GPUs are supported up to 1024 thread. Threads and blocks can be one, two or three dimensional. Blocks are organized in grids. A general structure is illustrated in figure 2 [18].

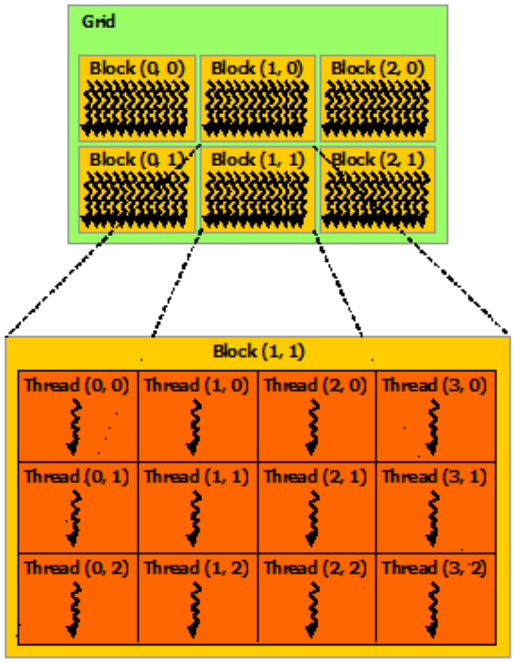


Figure 2.Grid of the blocks [18].

2.2. Memory Management

CUDA architecture supports many memory types. Threads, blocks and type of memory are illustrated in figure 3. Each thread access registers, local memory, shared memory, constant memory and global memory. The CPU part of the application can be access global memory and constant memory. The used memory types are listed below.

Registers: Registers are defined in the thread. These variables cannot be accessed from outside the thread. Generally, it is used to store local variable in the function. It is not require any extra programming extension [8, 9, 10]

Local Memory: Same as registers it is valid only in the thread. It is defined by __local__ keyword. İt is slower than register [8].
Shared Memory: All of the thread in the blocks can access to shared memory. The values in the blocks can be accessed by any thread. In general, it is fast as registers. İt can be defined by __shared__ keyword [8, 9].

Global memory: All application (CPU and GPU part) can access global memory. It is defined by __device__ keyword. Data transfer speed is very slow [18].

Constant Memory: Constant memory can only be read by the GPU. Each running thread can read from memory at the same time. Thus, we obtain very fast data transfer [8, 10].

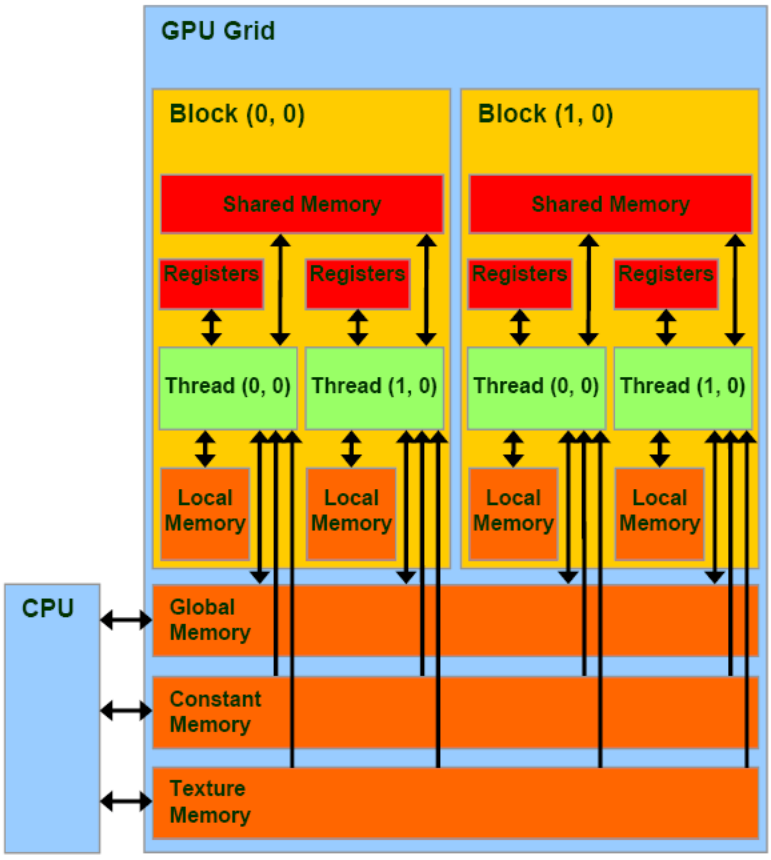


Figure 3.GPU hardware architecture and memory types [20].

3. DEVELOPED METHODS

In this paper, a novel level set method based on scaling approaches for extracting object on the images is proposed. In this section, developed GPU algorithm will be discussed. The application is implemented by using the Visual Studio 2012 environment with CUDA C language. The developed method consists of the following five stages. These stages and their working devices is presented in Figure-4.

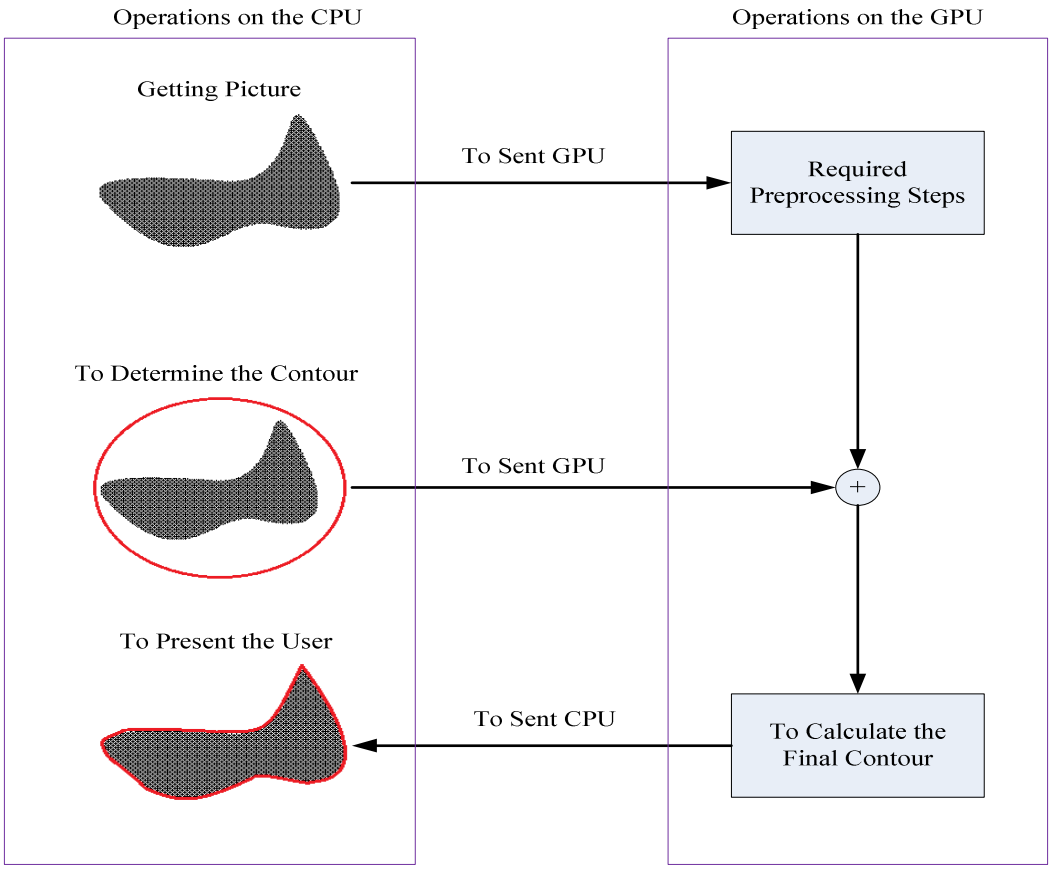


Figure 4.General algorithm of the developed method.

3.1. Preparation Stage

In this stage, user can select any picture by means of the developed user interface. After getting the image from a user, it sends to the graphics processor. OpenGL library was used to display the picture from the user. This process will be performed by the CPU.

3.2. Pre-processing Stage

In this stage some preprocessing techniques are performed on the images. The first technique is noise reduction. We use Gaussian smoothing for this purpose [21]. The Gaussian smoothing operator is a 2-D convolution operator that is used to remove detail and noise. We use 5x5 Gaussian filter. The second technique is to convert color to grayscale. The developed method applies on gray level images so gray level transformation is made in this section for color images. The final technique is Sobel operator. This operator is used in image processing, particularly edge

detection algorithm. This technique is used to help "EDGE" algorithm which is mentioned in section 3.4. These three techniques executes on the GPU.

3.3. Determination of Contour and Center

Sometimes the initial contour is very important to find object boundaries quickly. The developed application has tested on MRI brain tumors images. Here, we have expected to identify tumor area sketchy by expert. Determined initial contour curve is drawn as a circle. Then this curve is sent to the graphic processor to provide curve evolution.

The center of the object is the main element of the curve evolution algorithm. But we have no idea about the center of the object. Fortunately, it can be adjusted by the system automatically or by an expert. In this paper, we prefer to determine the center by an expert. Thus, we provide a more accurate center. Also, we allow selecting more than one center, so that the concave shapes is successfully determined. An example is illustrated in figure 5. U-shaped object having a concave region at the top of the figure. Figure 1a shows initial condition, figure 1b shows the result of boundaries with one center, and figure 1c shows the result of boundaries with three centers. As shown in the figure, using more than one center is more accurate.

(a) Initial Condition **(b)** Result with one center **(c)** Result with three centers

Figure 5.Comparison of the use of a single center and multi-center on U-shaped object.

3.4. Contour Evolution

In level set method, curve evolution is based on the solution of partial differential equation. This causes the algorithm to run slowly. The developed algorithm for this paper use different approach for curve evolution.

The developed curve evolution method uses two-dimensional geometric transformations such as scaling and it uses the logic of binary search. Binary search technique is applied on series of numbers and with this techniques search time significantly reduces. In developed algorithm, the evolution of next point on the curve is determined for each point. Normally, binary search is performed on numbers but in our method, we use pixels for binary search operation. Rather than a set of number of binary search, we define a set of pixels and half of the cluster is eliminated at each cycle approximately. This process continues until the edge is found or until there are no elements in the cluster. Elements of the cluster are pixels between a point on the contour and shape center.

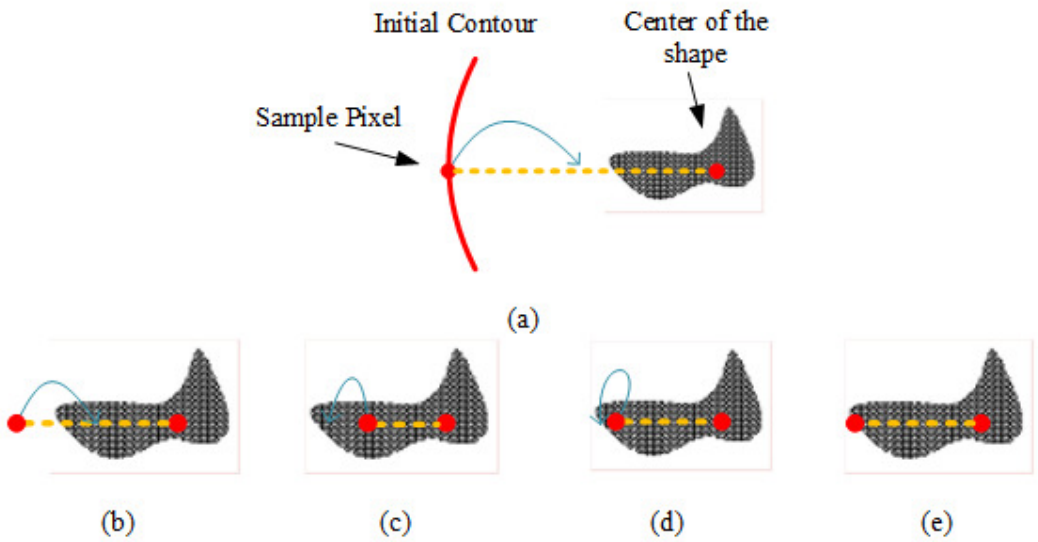


Figure 6.The movement of a pixel on the curve. (a) Initial state. (b) After first step. (c) After secondstep. (d) After third step. (e) After fourth step – Edge found.

Algorithm is illustrated for a pixel in Figure 6. Thus, when we move contour pixel by pixel, the running time of the edge detection process will be O(n) , but with this algorithm the running time of the edge detection process will be O(logn) (n is the number of pixels between initial point and center point). This process is repeated for all points on the curve, by this way the development of the initial curve is provided.

In algorithm, we use scaling in order to ensure the movement of pixels. At the beginning of the evolution, there are two variables defined for scaling which are called “sr1” and “sr2” and respectively they are set 1 and 2. “sr1” variable refers to scaling factor which is applied to the previous cycle and “sr2” variable refers to current scaling factor. Variables are calculated at the end of each cycle by the following formula according to the movement of each pixel.

$$sr1 = sr2 \tag{1}$$

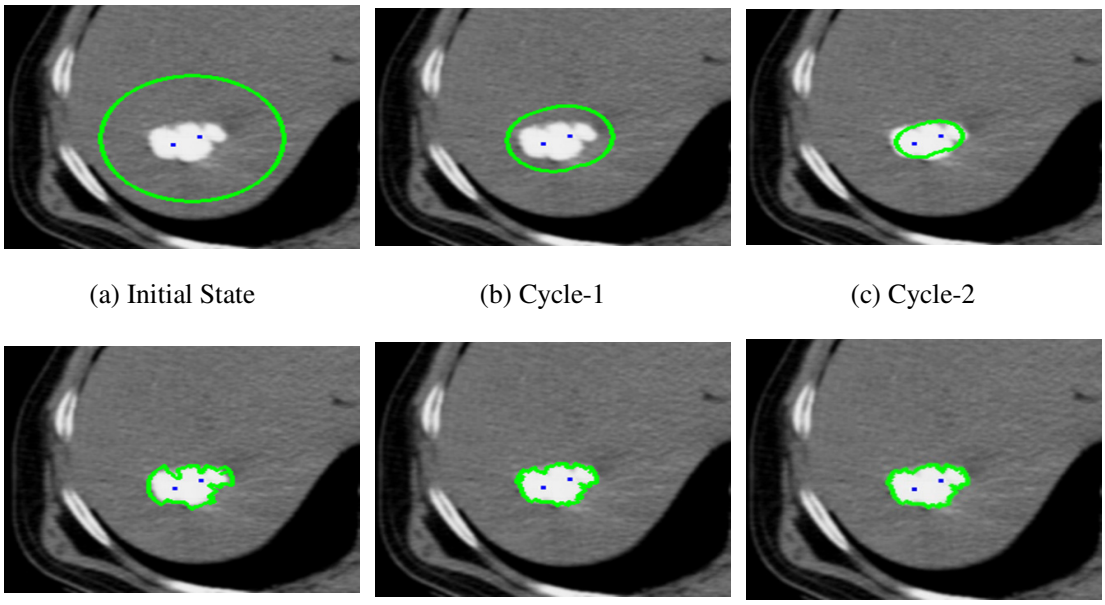
$$sr2 = \begin{cases} sr2 + \dfrac{|sr1 - sr2|}{2} & If\ pixel\ inside\ of\ the\ shape \\ sr2 - \dfrac{|sr1 - sr2|}{2} & If\ pixel\ outside\ of\ the\ shape \end{cases} \tag{2}$$

Developed GPU algorithm is given in Figure 7. Algorithm uses two functions. The first function is “EDGE” function. This function provides information about whether or not a pixel closes to the edge. The other function is “INSIDE” function. This function is used to determine a pixel is inside or outside the shape.

```
Giris: img   {resim},
       Tid   {kanal id},
       C     { ilgili kanalda islenecek kontur piksel},
       M     {C'ye en yakın merkez noktası}

Cıkıs: s    {sonuc konturu}

Y ← C
mid_x ← C_x + M_x
mid_y ← C_y + M_y

while (Y != mid || M != mid)

     if KENAR(img, mid) then
          mid değişkenini s listesine ekle  {Artık sabit }
          break

     else if ICINDE(img, mid) then     { Nokta nesnenin içinde mi?}
          M = mid

     else
          Y = mid
```

Figure 7.GPU algorithm of the algorithm.

"EDGE" function uses the calculated edge information which is calculated in preprocessing section and return in the form of true or false. In the same way "INSIDE" function returns result as true or false. "INSIDE" function uses centers values, initial contour and the average density of contour area to decide. Every pixels on the contour execute on a CUDA thread, so application accelerated. Curve evolution algorithm is applied an MRI liver image and illustrated in figure 8. The image size is 512x512 and as shown in the illustration contour evolution is very fast.



(a) Initial State (b) Cycle-1 (c) Cycle-2

(d) Cycle-3 (e) Cycle-5 (f) Cycle-7

Figure 8.An example of curve evolution algorithm.

3.5. Presentation

At this stage, the entire calculation has been completed. Final contour and preprocessed image copy from GPU to CPU memory. Then, contour points and image are combined and presented to the user.

4. RESULTS

In this section we present experimental results obtained by the proposed methods. All experiments were performed on a machine equipped with an Intel Core i7-3770 CPU at 8GB RAM and a GeForge GTX 660 Ti GPU. GPU has 7 streaming multiprocessor (SM) and each SM has 192 CUDA processor. This means 1344 computing core per chip. The number of register per multiprocessor is 65536, the total amount of constant memory is 64KB. The amount of shared memory per multiprocessor is 48KB, organized into 32 banks. The 2GB amount of global memory is reached through a GGDR5 interface. The architecture supports the double precision floating point arithmetic.

In this study, we use MRI brain tumor images. The resolution of the image is512x512. The results are illustrated in figure 9. As shown in the figure the tumor has found with high success rate.

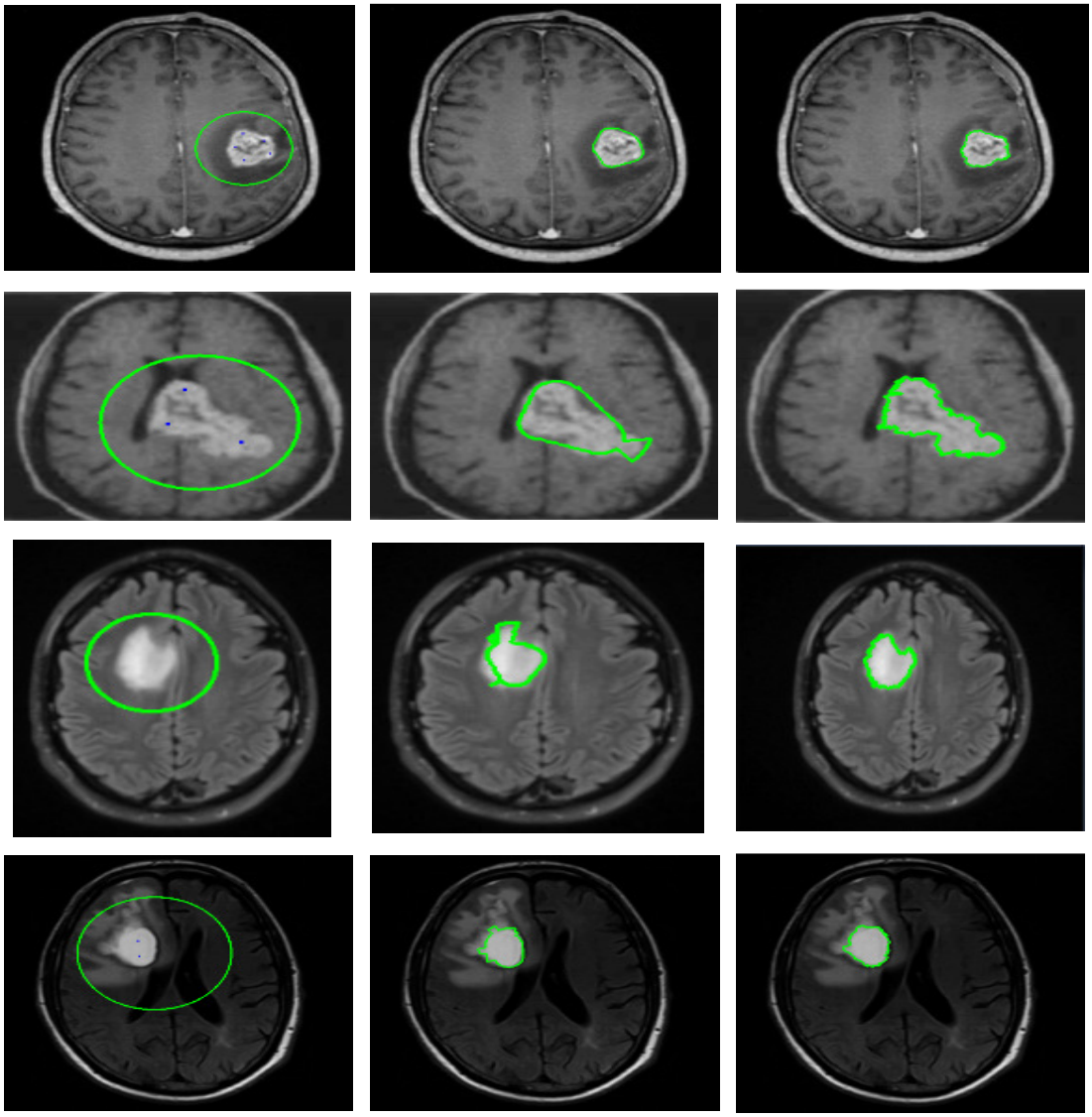


Figure 9.The results of the proposed method.

The developed application has been tested with different settings on different images with different resolution. Firstly, the preprocessing operation is tested with different settings. We test our algorithm with three different parameters. These are block number, thread number and image resolution. The results are shown in Table 1. Table 1 also shows that execution time generally depends on the image resolution but block number and thread number are also very important for performance optimization. As shown in second and last row of the table 1, thread number is selected very low, so the execution speed increases.

Table 1. Execution time of the preprocessing stage.

Image Resolution	Block Number	Thread Number	Execution Time (ms)
256x256	16x16	16x16 (256)	0,36
256x256	32x32	8x8 (64)	0,70
256x256	8x8	32x32 (1024)	0,36
256x256	16x8	16x32 (512)	0,36
512x512	16x16	32x32 (1024)	1,34
512x512	32x32	16x16 (256)	1,33
512x512	16x32	32x16 (512)	1,32
512x512	32x64	16x8 (128)	1,33
1024x1024	64x64	16x16 (256)	5,14
1024x1024	64x32	16x32 (512)	5,19
1024x1024	32x32	32x32 (1024)	5,19
1024x1024	128x64	8x16 (128)	5,36

Secondly, curve evolution algorithm is tested with different settings. We test our algorithm with four different parameters. These are block number, thread number, image resolution and number of contour points. The results are shown in Table 2. Considering that, curve evolution algorithm is a circular structure, so we only give one cycle's execution time. In Table 2, execution time is given in milliseconds and indicates that, the slow-running cycle. As shown in table 2, the execution time is not change with the image resolution, because we use image only for information. In other word, we do not use image information primarily, so threads will access image data with global memory when it is necessary.. We use contour points as primary data, but as shown in the figure 2, it is generally no effect for execution time. There are two reasons for this result. Firstly, the complexity of the algorithm is O(1). Each thread executes their code and this code does not include a circular part. Second and main reason, the video card capacity is not used in this process. As shown in table 2, threads and blocks number are very limited. Here, the important factor for execution time is to select a sufficient number of threads. For example, the second, fourth and sixth row of the table 2, we select 4x4 thread number. But there is no effect for performance. On the other hand, when we increase the number of contour points, 4x4 thread number is not enough and this leads to increase execution time. For example, in seventh row in table 2, we use 4096 contour points and 4x4 thread number is not enough.

A key decision for performance optimization in CUDA programming is not only the choice of the size of the block and thread number, but also the choice of the memory type. With an only small change to the type of memory used, can be achieved great acceleration. We use shared and constant memory in this study. The main aim is to maximize memory bandwidth. For this,in

preprocessing stage, we use shared memory for image convolution. Also the constant memory is used in all GPU-based function.

Table 2. Execution time of the curve evolution algorithm. (For 1 cycle)

Image Resolution	Block Number	Thread Number	Contour Point Number	Execution Time (ms)
256x256	4x4	8x8	1024	0,058
256x256	8x8	4x4	1024	0,055
256x256	4x2	4x8	256	0,057
256x256	4x4	4x4	256	0,054
512x512	4x4	8x8	1024	0.058
512x512	8x8	4x4	1024	0.056
512x512	16x16	4x4	4096	0.090
512x512	8x8	8x8	4096	0.059
1024x1024	4x8	16x8	4096	0.060
1024x1024	4x4	16x16	4096	0.060

5. CONCLUSIONS

In this paper, a novel GPU-based level set method has presented with scaling approach.We followed the method of heterogeneous programming technique for this study. Some of the steps were made on the CPU side and some of the steps were made on the GPU side. Minor procedures and user inputs performed on the CPU side. Operations that require high computing such as curve evolution and preprocessing performed on the GPU side. Thus, we achieved very fast execution time.

In developed method, the curve evolution algorithm used basic geometric transformation like scaling, so that the algorithm has been run much faster. As a result, all operations were completed in about 2 milliseconds with all memory operations. Thus, it is possible to show that with the developed method, we can perform real-time operation.

REFERENCES

[1] S. Osher, and N. Paragios, (2003)“Geometric Level Set Methods in Imaging, Vision and Graphics”, Springer-Verlag New York, Secaucus, NJ, USA.

[2] D. W.Shattuck, G. Prasad, M. Mirza, K. L. Narr, and A. W. Toga, (2009)“Online resource for validation of brain segmentation methods”, NeuroImaging, 45, 431-439.

[3] S. Osher, and J. Sethian, (1998)“Fronts propagating with curvature-dependent speed: Algorithims based on Hamilton-Jakobi formulation”, J. Comput. Phys., 79-1, 12-49.

[4] R. Fetkiw, (2002)“Simulating Natural Phenomena for Computer Graphics, Geometric Level Set in Imaging”, Vision and Graphics, 461-479.

[5] D. Adalsteinsson, and J. Sethian, (1995)“A Fast Level Set Method for Propagating Interfaces”, Journal of Computational Physics, 118-2, 269-277.

[6] R. Whitaker, (1998)“A Level-Set Approach to 3D Reconstruction from Range Data”, International Journal of Computer Vision, 29-3, 203-232.

[7] S. P. Awate, and R. T. Whitaker, (2004)"An Interactive Parallel Multiprocessor Levet-Set Solver with Dynamic Load Balancing", Scientific Computing and Imaging Institute Technical Report, UUCS-05-002, University of Utah, Salt Lake City, UT, USA.
[8] J. Sanders, and E. Kandrot, (2011)"CUDA By Example: An Introduction to General-Purpose GPU Programming", Addison-Wesley, NVIDIA Cooporation.
[9] M. Rumpf, and R. Strzodka, (2001) "Level Set Segmentation in Graphics Hardware", IEEE International Conference on Image Processing (ICIP'01), Thessaloniki, Greece, October 7-10, pp. 1103-1106.
[10] A. Lefohn, and R. Whitaker,(2002)"A GPU-Based Three-Dimensional Level Sey Solver with Curvature Flow", Scientific Computing and Imaging Institute Technical Report, UUCS-02-017, University of Utah, Salt Lake City, UT, USA.
[11] A. Lefohn, J.M. Kniss, C. D. Hansen, and R. T. Whitaker, (2004)"A Streaming Narrow-Band Algorithm: Interactive Computation and Visualization of Level Sets", IEEETransactions on Visualization and Computer Graphics, 10, 422-433.
[12] O. Klar, (2007)"Interactive GPU-based Segmentation of Large Medical Volume Data with Level-Sets", 11th Central European Seminar on Computer Graphics (CESCG'07), Budmerice Castle, Slovakia, April 23 - 25.
[13] H. Mostofi, and K. Colege, (2009)"Fast level Set Segmentation of Biomedical Images using Graphics Processing Units", Final Year Project, University of Oxford, Department of Engineering Science Oxford.
[14] A. Hagan, and Y. Zhao, (2009)"Parallel 3D Image Segmentation of Large Data Sets on a GPU Cluster", 5th International Symposium on Visual Computing, Las Vegas, Nevada, USA, November 30 - December 2, pp. 960-969.
[15] G. J. Tornai, and G. Cserey, (2010)"2D and 3D level-set Algorithm on GPU", 12th International Workshop on Cellular Nanoscale Network and Their Applications (CNNA), Berkeley, CA, USA, February 3-5, pp. 1-5.
[16] M. Roberts, J. Packer, M. C. Sousa, and J. R. Mitchell, (2010)"A Work-Efficient GPU Algorithm for Level Set Segmentation", High Performance Graphics (HGP '10), Saarbrucken, Germany, June 25-27, pp. 123-132.
[17] A. C. Jalba, W. J. Van Der Laan, and J. B. T. M. Roerdink, (2013)."Fast Sparse Level Sets on Graphics Hardware", Visualization and Computer Graphics, 19-1, 30-44.
[18] NVIDIA., (2012) "CUDA C Programming Guide v5.0". http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf, 15 Mayıs 2013.
[19] S. Cook, (2012)"Cuda Programming: A Developer's Guide to Paralel Computing with GPUs", Elsevier, Morgan Kaufmann.
[20] J. Ooster,"Cuda Memory Model", http://3dgep.com/?p=2012, 14 Mayıs 2013.
[21] "Gaussian Smoothing", http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm, 14 Mayıs 2013.

Authors

ZaferGüler is a research assistant in the Department of Software Engineering at the Firat University of Elazig where he has been a faculty member since 2010. He completed his master at Firat University Computer Engineering department. His research interests are GPU programming, level set methods.image segmentation.

AhmetÇınar was born in Elazig (1972). He received the PhD degree in Electric-Electronics Engineering in 2003 from Firat University. He has graduated in 1993 BSc. He has been working on Firat Univ. Department of Computer Engineering, (Assistant Professor). His research is interested in development and improvement of mesh generation methods, and applications of virtual reality, augmented reality, artificial intelligence and game programming.