PARALLEL 3D IMAGE SEGMENTATION BY GPU-AMENABLE LEVEL SET SOLUTION

A thesis submitted
to Kent State University in
partial fulfillment of the requirements
for the degree of Master of Science

by

Aaron Michael Hagan

August 2009
Thesis written by

Aaron Michael Hagan

B.S., Kent State University, 2004
M.S., Kent State University, 2009

Approved by

Dr. Ye Zhao, Advisor

Dr. Robert Walker, Chair, Department of Computer Science

Dr. John R.D. Stalvey, Dean, College of Arts and Sciences
# TABLE OF CONTENTS

LIST OF FIGURES ......................................................................................... v

LIST OF TABLES ............................................................................................ viii

Acknowledgements ........................................................................................ ix

1 Introduction ................................................................................................. 1
   1.1 Motivation .............................................................................................. 1
   1.2 Problem Statement and Objectives ........................................................ 2
   1.3 Approach and Contribution ................................................................... 3

2 Previous Work and Background ................................................................. 9
   2.1 Image Segmentation Techniques ........................................................... 9
   2.2 Level Set Equation ............................................................................... 10
   2.3 GPU Acceleration ............................................................................... 11
   2.4 Lattice Boltzmann Method ................................................................... 12

3 LBM for PDE Solution ................................................................................. 14
   3.1 LBM Computation ............................................................................... 15
   3.2 Extended LBM for PDEs ...................................................................... 18

4 Parallel Image Segmentation ................................................................. 20
   4.1 Level Set Method .................................................................................. 20
### LIST OF FIGURES

1. Image segmentation example from Sethian [1], using the level set equation.  
2. Sparse data storage that is used in Lefohn et al. [2, 3] application for a narrow band packed level set solver on the GPU. Only the active pages are stored in GPU memory (Physical Memory Space). After each step of the level set solver, active pages are loaded onto the GPU by the CPU (Virtual Memory Space). Image courtesy of Lefohn et al.  
3. LBM lattice structure. A typical D3Q7 lattice structure is shown with 6 links to its axial neighbors and a value for the lattice itself. \( f_i \) is the particle distribution function for a given lattice link. \( e_i \) is the vector value for the lattice link.  
4. Level Set Contour Evolution. Image courtesy of Sethian [1].  
5. Segmentation of an artificial 2D image. (a) A 2D box with rough boundary. (b)-(c) Segmentation results with different curvature-based smoothing coefficients: (b) 1.00 and (c) 5.125.  
6. Segmentation of a 3D foot with different curvature-based smoothing coefficients at the same number of segmentation steps.  
7. Segmentation of a 3D aneurism with different curvature-based smoothing coefficients. All images are shown at the same number of LBM steps (25 in this example).  
Image shown with the absence of ghost layer transfer between computational steps. Nodes solve the LBM streaming and collision process without the knowledge from neighboring nodes. The segmentation results are not accurate, showing the need for ghost layer transfers.

Data decomposition to cluster nodes for LBM computations. The master node is shown with the segmentation in a $4 \times 4$ in the X and Y direction. The blocks are sent to the individual nodes in the cluster for computation.

Uses ghost layer to transfer boundary data between neighboring nodes in the network. Extra layers are used to save data from the neighboring nodes. The LBM streaming and collision method is computed on the whole data including the ghost layer data.

Results of segmentation of a cross section in a CT brain. (a) Initial image with starting zero level set represented in red. (b) Segmented result of the object represented by a target isovalue of 228.

Results of segmentation of a cross section in a MRI brain. (a) Initial image with zero level set represented in red. (b) Segmented results of the object represented by a target isovalue of 167.

Results of segmenting a CT head with a target isovalue of 60. Data size is $128 \times 128 \times 128$. (s) Segmentation results using $\gamma = 20.125$; (b) $\gamma = 5.125$; (c) $\gamma = 1.125$; (d) Target isosurface generated by direct Marching Cubes.

Results of segmenting an Aneurism data with a target iso-value of 32. Data size is $512 \times 512 \times 512$. $\gamma = 1.5$. (a) Level set propagation after 3 steps; (b) After 10 Steps; (c) After 25 Steps; (d) After 50 Steps.
16 Results of segmenting a volumetric CT scan of a Bonsai tree with a target iso-value of 20. $\gamma = 5.125$. Data size is $1024 \times 1024 \times 308$. (a) Initial level set as a sphere; (b) After 25 Steps; (c) After 45 Steps; (d) Direct isosurface. 41

17 Results of segmenting a volumetric abdomen data with a target iso-value of 20. $\gamma = 1.125$. Data size is $512 \times 512 \times 174$. (a) After 3 Steps; (b) After 25 Steps; (c) After 45 Steps; (d) Direct isosurface. . . . . . . 41

18 Results of segmenting a volumetric Porsche data set with a target iso-value of 32. $\gamma = 5.125$. Data size is $509 \times 1023 \times 347$. The initial sphere of radius size 100 was used for the segmentation (a) After 3 Steps; (b) After 25 Steps; (c) After 45 Steps; (d) Direct isosurface. . . . . . . . . . . . . 42
LIST OF TABLES

1 Performance report: Per step (in seconds) average speed to perform LBM computation and ghost layer communication, the memory size per node, and the total ghost layer data size on the GPU cluster with 4 \times 4 configuration. ........................................ 42
Acknowledgements

I would like to thank the daytaOhio Wright Center of Innovation funded by the Ohio Department of Development (ODOD), which funded the cluster on which this research was performed.
CHAPTER 1

Introduction

1.1 Motivation

Large scale 3D images are becoming more popular in medical imaging, biology, industry and other areas. Similarly the accuracy of experiments and scanning devices are increasing rapidly. Often times in the acquisition of such images, inherent noise is introduced through the process. A need arises to explore noise removal techniques that preserve image properties such as smoothness and clearness. Another issue in studying medical images is the need to remove pieces or segments of the image that aren’t important to the study. Such examples of this include the study of tumors; in this case it may be desirable to remove the other features in the data set to get an unobstructed view of the tumor of interest.

Image processing techniques have been studied by many groups that explore how to efficiently and accurately perform this image processing task, including (not limited to) region growing, contour evolutions, and image thresholding. Computational computing hardware has also become more readily available which allows this problem to be greatly sped up by investigating true parallel methods for performing the image segmentation. Graphics Processor Units (GPU) are becoming affordable and prominent due to the ever increasing video game industry, which has made them more prevalent. The fascinating properties of these processors are that they are easy to program and can compute parallel operations in a SPMD environment.
Current methods for performing image segmentation on the GPU are not adept in handling large scale volume data sets and are designed to conserve memory by implementing sparse data management techniques that do not necessarily map to the parallel nature of the GPU. There is a need to have a segmentation approach that can operate on large scale data sets in a time efficient and accurate manner. The Lattice Boltzmann Method (LBM) solver for the level set partial differential equation (PDE), proposed in this thesis, is able to handle this criteria.

1.2 Problem Statement and Objectives

The main objective of re-examining image segmentation of 2D and 3D data sets is to explore how the problem can be mapped to modern graphics cards since their architecture and programming mechanisms have evolved over the recent years, and to investigate how using a different solver than traditional methods can produce the same accurate results with improved efficiency. This work also expands on developing a true parallel mechanism (based on the LBM schema) for performing image segmentation, compared to previous implementations and solvers. As data sets gain in size, a need to use multiple GPU’s is investigated as part of this work’s objectives.

It can be assumed that modern scanning devices will continue to produce large scale images with more detail and accurate representation of the objects under study. This will take form, for example in the number of detectors in medical devices that will create smaller voxels, and microscopes that will increase in magnification resolution. It is necessary to have an approach that can handle the future needs for image segmentation.
1.3 Approach and Contribution

The method this thesis proposes for performing image segmentation is an inherently parallel scheme based on the level set equation for modeling contours in images. Solving the level set equation is performed by using an extended LBM which provides an alternative numerical solution for the equation. This method has several advantages in that it is very easy and straightforward to implement, implicitly includes the computation of curvatures, has a unique parameter that controls the smoothness of the results, and finally, is parallel which allows it to be mapped to low-cost graphics hardware (GPU) on a single system or in a cluster environment.

The level set method uses a PDE to model and track how fronts evolve in a discrete domain by maintaining and updating a distance field to the fronts. This scheme has been explored and expanded by many researchers to carry out a wide range of image processing techniques. Such work includes object recognition, image de-noising, and has widely been used in performing segmentation of 3D volume data sets. This is realized by creating a contour surface and having it propagate to target regions of interest. Meanwhile, a smoothing (diffusion) operation works to reduce noise and segmentation bleeding. Previous methods based on the level set formulation discretized the PDE with finite difference operators which lead to complex numerical computations. A state-of-the-art scheme, the narrow-band method, applies an adaptive strategy, where the level set computation is only performed on a narrow band around the propagating contour. This solution can be computationally expensive due to the special data structure (e.g. heap) maintenance, and does not necessarily map well to parallel computing. To expedite the narrow band
computing on the powerful graphics hardware, Lefohn et al [2, 3] proposed a successful GPU implementation with deliberated-designed band packing and virtual memory management that arranges CPU-GPU data communication. However, the approach cannot be easily extended to multiple GPUs, since keeping narrow band management working correctly and efficiently in such case imposes a vital challenge to cluster implementation. It is very hard to apply a global priority queue to manage sorted bands on multiple machines, to re-initialize the distance field every few steps, and to pack the bands on GPUs which evolve simultaneously. The difficulties illustrate an unclear future for current hardware-based narrow band method to process very large data sets, while the data sets cannot be accommodated in one machine due to the memory restriction.

With the increase in availability of large volume data sets it is important to promote the level set segmentation technology to handle such data sets, where distributed computing provides a good platform for this. In this paper, a new framework to solve the level set equation within an inherent parallel scheme, originated from a modified LBM solution is proposed. With simple, local and explicit operations, this method lends itself a good tool that is easily implemented on distributed machines, such as GPU clusters, with only minimal data management and communication through the network. Furthermore, the method has the ability to implicitly handle curvature flows with its explicit computation, leading to a controllable noise reduction effect in the segmentation results.

In detail, previous methods apply the narrow band with the corresponding priority data structure to adaptively propagate fronts to the target regions. A re-initialization of the narrow band is required as the band propagates for a few steps, to maintain the valid distance field. After re-initialization, the new narrow band is packed and reloaded to the
GPU. The proposed method is different in that it does not use a narrow band approach, and therefore, the distance field is always valid in the whole domain and no reload is needed. This leads to a good feature that no frequent CPU-GPU crosstalk is performed during segmentation. In other words, this method is computed totally on the GPU avoiding the computational bottleneck between the host and the GPU. It should be noted that abandoning the adaptive strategy increases the memory consumption. However, the benefit of easy operation, parallelizability and distributed computing validates the new scheme. Furthermore, there are also circumstances where performing the computation on the whole data is necessary and desirable. More important, this strategy is based on the rapidly increasing computational power, i.e. speed and memory size, on single machine and clusters, which is making adaptive operations unnecessary in many scientific applications. (For example, the GPU memory is evidently increasing in a fact that current graphics cards such as the Nvidia Quadro have been configured with 4 GB memory size on a single card, while just a few years ago, the maximum size of the GPU memory was only 256 MB.) Meanwhile, cluster systems that exploit many cards with huge memory capacity are becoming available and being used in many scientific applications. Thus, it can be anticipated that this trend will continue developing at the moment and in the near future. On the other hand, this method works well on an GPU cluster for segmenting large data sets with easy smoothing effect control, which is advantageous to a direct and non-adaptive upwinding level set solver. Further benefit comes from its particular easy-coding with under 100 lines of code of CPU and GPU implementation. A knowledgeable graduate student can implement the program in a very short period. In this implementation, a simple but suitable D3Q7 LBM lattice is used for the level set
solution which consumes little memory and achieves fast computing speed, in contrast
to a traditional D3Q19 lattice for LBM fluid solver.

This work modifies the lattice Boltzmann model to use it in solving the level set equa-
tions. The modified LBM scheme uses cellular-automata-style update rules that make it
inherently parallelizable. It implicitly includes curvature-based smoothness essential for
the level set formulation without extrapolation and gradient computations, which makes
it an attractive numerical solver. Typically, numerical methods to solve the implicit level
set equation imperially involve operations that involve computing conjugate gradients
and accuracy problems may be experienced due to the large time steps used during the
computation. Solving the linear systems in implicit equations also do not map well to
a parallel implementation. The explicit LBM solver is proposed to solve the level set
equation efficiently by taking advantage of it’s inherent parallel properties. By solving
the segmentation problem in a true parallel environment a greater speedup may occur
during the segmentation process.

In conclusion, this thesis contributes to the computer graphics and visualization re-
search by introducing a simple, explicit parallel implementation of solving the level set
method to perform image segmentation on large scale 3D data sets. The main advantages
of this method include:

(1) There are only a few steps that are required for an LBM simulation, therefore
to an experienced programmer it is rather trivial to implement the algorithm for the
computations with a small amount of code. This allows for the solution to be easy
plugged into existing visualization frameworks with little overhead. The implementation
of the GPU and CPU code for the LBM algorithm was written in C++ in approximately
100 lines of code. The concept is also easy to understand, allowing it to be implemented with simple data structures. Chapter 5 gives some pseudo code on how to implement the basic LBM constructs for solving the level set equation.

(2) One of the main features of the level set formulation is its ability to recover smooth surfaces of the target object. By extending the level set with the LBM solver, the LBM solver implicitly recovers the smoothing properties to preserve edges and surfaces of the target object. Propagating fronts with speeds dependent on curvature is foundational to the level set based segmentation and its computation is usually complicated and time consuming on conventional solutions.

(3) Flexible control of generating smooth segmentation edges. Producing smooth results that are noise free is a fundamentally important concept in image segmentation applications. The LBM based solver contains several free parameters that allow the user to control the smoothness of the results. There are a few simple parameters that need modified during the segmentation process to give different degrees of smoothness in the segmentation results. This thesis (Section 4.2.3.) shows the results of segmenting different 2D and 3D volume data sets and how a single free parameter can impact the smoothness of the results.

(4) Strong amenability to parallel computing, especially on the low-cost, powerful graphics hardware (GPU). The parallel nature of the LBM solver allows for a direct mapping of the computational algorithm to the graphics processor. The simple updating rules of the LBM algorithm allow each lattice site to be operated on independently of each other. This maps well to the single instruction multiple data (SIMD) paradigm, of the GPU. This can further be extended to a cluster environment where the lattice
domain can be split among many cards and updated in parallel.

(5) Support large data sets in a cluster environment. As scanning devices become more advanced, the greater detail and size the data sets will become. It will be apparent that more computing power is necessary to handle these large data sets to perform the segmentation efficiently. The LBM based level set solver is extremely adaptable and is therefore easy to extend the solver across an array of GPUs. It is also becoming more prevalent to have mid to high range systems that have multiple GPUs, and this method is friendly to this type of setup. The results section (Chapter 6) shows segmentation results of "real-life" large scale images. As the size of GPU memory increases and the trend of having multiple GPUs in one machine continue, parallel algorithms such as the LBM will become a more viable option for handling large data sets.

This method is applied on various 2D images and 3D volume data sets on a single GPU and a GPU cluster, achieving excellent segmentation results with fast speed. This approach will provide an alternative to the implicit methods and provide a true parallel implementation for performing image segmentation. In the following, related work is shown in Chapter 2. Then, a brief introduction to the LBM method is provided in Chapter 3. Chapter 4 discusses the basic method using the scheme to solve the level set equation. Also provided is the GPU cluster implementation in Section 5.2. Finally in Chapter 6, examples illustrate the segmentation results of this method on single and cluster GPUs, with a report and discussion of the performance.
CHAPTER 2

Previous Work and Background

This chapter studies the previous work and background for the image segmentation algorithms. A comparison to the proposed method in this thesis and the previous implementations are given.

2.1 Image Segmentation Techniques

Image segmentation has been an active research domain in pattern recognition and image processing. Many approaches have been proposed. Cheng et al. [4] outlined methods for color image segmentation, which included the following techniques: histogram thresholding, characteristic feature clustering, edge detection, region based methods, fuzzy techniques, and neural networks. In particular for medical images, Pham et al. [5] presented a review of the segmentation approaches and applications. Wirjad [6] showed work based on 3D medical image segmentation, which is closely related to this research. This survey studies segmentation techniques for grey scale images. The concepts they cover include thresholding, region growing, deformable surfaces and level set methods, and probabilistic approaches for performing segmentation. The thresholding approach is performed by identifying the region of interest’s target density value and deleting all other areas of the image that do not fall in this target region. This is a simplified form of segmentation but does not address the noise that may be associated with the target area. This method may not give a clear view of the resulting segmentation because there
are no steps to find the borders of the region. Region growing algorithms work by identifying pixels or voxels belonging to an object area according to a derived function. This algorithm could be thought of as a neighborhood thresholding method, where groups of similar pixels are defined while the rest of the image is ignored. Similar to thresholding this algorithm does not preserve smooth boundaries or borders, which is undesirable.

2.2 Level Set Equation

The level set equation has been used in a wide variety of image processing operations such as noise removal, object detection, and modeling equations of motion. The level set equation can be used to perform the segmentation by creating an initial contour surface in the target image and having it evolve to regions of interest normally defined as target intensity values or gradients to attract the curve as shown by Malladi et al. [7]. A narrow band adaptive strategy is also used to improve the performance, where the level set computation is only performed on a narrow band around the propagating contour. However, the distance field around the narrow band needs to be re-initialized after each step. Implicit surfaces can easily change their topology, fuse together, or split up into several entities. Some of the challenges in using the level set framework in an application is its time consuming performance and the need to fine tune the free parameters to make the segmentation perform properly. Figure 1 shows an example of segmenting a data set of arteries. Here the zero level set is morphed to the outline of the arteries edges. Sethian’s [1] work shows technical and implementation issues of using the level set in segmentation applications.
2.3 GPU Acceleration

The level set formula is particularly useful in solving image segmentation and manipulation problems but often comes at a computationally expensive cost in performing the operation. It is also time consuming to find proper values for the free parameters that need modified during the deformation. It is apparent that an acceleration schema should be implemented to give faster results. Other groups have used the GPU in accelerating the level set based volume segmentation. For GPU acceleration, Lefohn et al. [2, 3] pioneered the acceleration of the level set method for medical image segmentation. Their method is implemented using a delicate narrow band packing technology and virtual memory management to arrange CPU-GPU data communication. The sparse adaptive data structures used in this method require a constant CPU-GPU exchange of data between each data step. The message passing schema is implemented using the automatic mipmap features to encode the messages. Figure 2 shows a description of this memory model where the narrow band and active level set are stored in pages on the GPU. The virtual memory space is maintained on the CPU and updated after each step of the level set computation. While achieving great success, there may be a problem for very large
Figure 2: Sparse data storage that is used in Lefohn et al. [2,3] application for a narrow band packed level set solver on the GPU. Only the active pages are stored in GPU memory (Physical Memory Space). After each step of the level set solver, active pages are loaded onto the GPU by the CPU (Virtual Memory Space). Image courtesy of Lefohn et al.

data sets, which have to be computed on multiple machines. Keeping the narrow band packing working well in such cases might be challenging.

Other GPU acceleration methods, by Klar [8], and Rumpf et al. [9] have successfully applied the level set equation for image segmentation by solving it on the GPU. A volume segmentation method proposed by Sherbondy et al. [10], based on the Perona-Malik model, is a region growing algorithm extended to the graphics card.

2.4 Lattice Boltzmann Method

LBM method has also been used in natural phenomena modeling in computer graphics and visualization as shown by Zhao et al. [11]. The LBM-based diffusion has been used in image processing by Jawerth et al. [12], where an anisotropic 2D image denoising is implemented on the CPU. Recently, Zhao [13] showed how the LBM scheme can be derived to solve volume smoothing, image fairing, and image editing applications. Tölke [14] described the parallel nature of the LBM scheme and how it can be mapped to the GPU through the CUDA library for modeling computational fluid dynamics. GPU
cluster computing has been a rapidly developing research area which is adopted in many scientific computing tasks. Fan et al. [15] used the classic LBM computing for fluid modeling as one of the examples to show their Zippy programming model for using GPU clusters.
CHAPTER 3

LBM for PDE Solution

The Lattice Boltzmann Method, in its initial design, has developed into a viable alternative to existing numerical methods for modeling and simulating fluids. Succi [16] shows how the LBM originates from the cellular automata scheme which models fictitious particles on a discrete grid where each point of the grid contains a lattice structure. The particles were initially used to simulate traditional fluid dynamics applications. Using a non-classical numerical computing process derived from microscopic statistical physics, the method can recover the Naiver-Stokes equations. The LBM works by constructing simplified kinetic models which encompass the same behavior as microscopic processes. The simulation method gives a clearer picture of the dynamics of fluid. This method also has the added advantage that it is fully parallel by allowing the particles to flow independently of each other during the simulation. The parallel nature of the LBM makes it a very attractive method for visualization and graphics because of the direct mapping to the GPU.

The independent variables in the LBM equation consist of particle distribution functions from each grid point to its neighbors. The distribution functions model the probability of a packet of particles streaming across one lattice link to its corresponding neighbor. Between the streaming computation of two consecutive steps, the function is modified by performing local relaxation that models particle collisions. This streaming-collision computational approach has the benefits that can easily simulate complex nonlinear equations.
Figure 3: LBM lattice structure. A typical D3Q7 lattice structure is shown with 6 links to its axial neighbors and a value for the lattice itself. $f_i$ is the particle distribution function for a given lattice link. $e_i$ is the vector value for the lattice link.

and model macroscopic behavior with explicit, local and simple updating rules. While the collision stage is completely local, only neighboring values are used for streaming at each time step. Consequently, the method lends itself to an excellent parallel computing scheme. Next, an outline of the basic LBM computational method for solving the Navier-Stokes equation, and the computational basis for level set solution is shown. A complete physical description of LBM is given by Succi [16], and its usage in visual simulations is shown by Zhao [11].

3.1 LBM Computation

The first step in performing a LBM simulation is to discretize the simulation domain to a grid, and generate the lattice structure for each grid point. For LBM simulations each grid point has a variety of different links to its neighbors. Figure 3 shows a D3Q7 (three dimension with seven links) lattice configuration with 6 links to its axial neighbors and one to itself. Collision and streaming computations are performed during each step.
of the simulation which is mathematically described as

\[
\begin{align*}
collision & \Rightarrow f_i(\vec{x}, t^*) = f_i(\vec{x}, t) - \frac{1}{\tau}(f_i(\vec{x}, t) - f_i^{eq}(\vec{x}, t)), \\
streaming & \Rightarrow f_i(\vec{x} + \vec{e}_i, t + 1) = f_i(\vec{x}, t^*),
\end{align*}
\]

(1) (2)

The local equilibrium particle distribution, \(f_i^{eq}\), models collisions as a statistical redistribution of momentum. At a given time step \(t\), each particle distribution function, \(f_i\), along one link vector \(\vec{e}_i\) at a lattice point, \(\vec{x}\), is updated by a relaxation process with respect to \(f_i^{eq}\). The collision process is controlled by a relaxation parameter \(\tau\). \(\tau\) controls the rate at which the equation approaches the equilibrium state. After collision, the post-collision result is propagated to \(\vec{x} + \vec{e}_i\). Here, \(\vec{x} + \vec{e}_i\) locates a neighboring lattice point along the link \(i\). This provides the distribution function value at time step \(t+1\).

\(f_i^{eq}\) can be defined by the Bhatnagar, Gross, Krook (BGK) model as

\[
f_i^{eq}(\rho, \vec{u}) = \rho(A_i + B_i(\vec{e}_i \cdot \vec{u}) + C_i(\vec{e}_i \cdot \vec{u})^2 + D_i \vec{u}^2),
\]

(3)

where \(A_i\) to \(D_i\) are constant coefficients chosen via the geometry of the lattice links and \(\rho\) is the fluid density computed as the accumulation of particle distributions by

\[
\rho = \sum_i f_i. \tag{4}
\]

\[
0 = \sum_i f_i^k, k = 1 \text{ or } 2. \tag{5}
\]

The LBM can be easily extended to incorporate additional micro-physics, such as an external force \(\vec{F}\). This force affects the local particle distribution functions as follows:

\[
f_i \leftarrow f_i + \frac{(2\tau - 1)}{2\tau} B_i(\vec{F} \cdot \vec{e}_i). \tag{6}
\]
The Chapman-Enskog expansion is used to recover the Navier-stokes equation which can later be modified to show the level set formulation. The Chapman-Enskog analysis is essentially a formal multi-scaling expansion:

\[ f_i = f_i^0 + \varepsilon f_i^1 + \varepsilon^2 f_i^2 + \cdots, \]

\[ \frac{\partial}{\partial t} = \frac{\partial}{\partial t_0} + \varepsilon \frac{\partial}{\partial t_1} + \varepsilon^2 \frac{\partial}{\partial t_2} + \cdots, \]

\[ \frac{\partial}{\partial x} = \frac{\partial}{\partial x_0} + \varepsilon \frac{\partial}{\partial x_1} + \cdots, \]  

(7)

where the small expansion parameter \( \varepsilon \) is the Knudsen number, defined as the ratio of the molecular mean free path length to a representative macroscopic length scale. First, \( f_i(\vec{x} + \vec{e}_i, t + 1) \) in Equation 2 can be Taylor expanded on two variables \( t \) and \( \vec{x} \). Second, \( f_i(\vec{x}, t) \) is rewritten following the Chapman-Enskog expansion. Thus, comparing both sides of the resulting equation in the consecutive order of the parameter \( \varepsilon \), the following is realized:

\[ f_i^0 = f_i^{eq} : O(\varepsilon^0) \]  

\[ \left( \frac{\partial}{\partial t_0} + \mathbf{e}_i \cdot \nabla \right) f_i^0 = -\frac{1}{\tau} f_i^1 : O(\varepsilon^1) \]  

\[ \frac{\partial f_i^0}{\partial t_1} + \left( \frac{2\tau - 1}{2\tau} \right) \left( \frac{\partial}{\partial t_0} + \mathbf{e}_i \cdot \nabla \right) f_i^1 = -\frac{1}{\tau} f_i^2 : O(\varepsilon^2). \]  

(8)

(9)

(10)

Summing Equation 10 over \( i \) yields

\[ \sum_i \left( \frac{\partial f_i^0}{\partial t_1} \right) + \left( \frac{2\tau - 1}{2\tau} \right) \sum_i \left( \mathbf{e}_i \cdot \nabla \right) f_i^1 = 0, \]  

(11)

with the constraints of Equation 5. Combining Equation 11 and Equation 9, and using the constraints and Equation 16, finally Equation 12 is obtained

\[ \frac{\partial \rho}{\partial t} + (\nabla \alpha \cdot \Pi_{\alpha\beta} \cdot \nabla \beta) \rho = 0, \]  

(12)
where $\alpha$ and $\beta$ represent coordinate directions, and $\Pi_{\alpha\beta}$ is the diffusion tensor

$$\Pi_{\alpha\beta} = \sum_i ((1 - 2\tau^2) e_{i\alpha} e_{i\beta} A_i).$$  \hspace{1cm} (13)

By applying Chapman-Enskog analysis [17], the Navier-Stokes equation can be recovered from the equilibrium equation as

$$\nabla \cdot \vec{u} = 0, \hspace{1cm} \text{(14)}$$

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} = \nu \Delta \vec{u} + \vec{F}. \hspace{1cm} \text{(15)}$$

Here $\nabla$ defines the gradient operator ($\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}$) and $\Delta$ is the Laplacian $\Delta = \nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$.

### 3.2 Extended LBM for PDEs

Though initially designed for fluid dynamics, the LBM method can be modified for modeling typical diffusion computations. The equilibrium Equation 3 can be simplified to

$$f_i^{eq}(\rho) = A_i \rho, \hspace{1cm} \text{(16)}$$

which erases momentum terms and in effect removes the nonlinear advection term in the Navier-Stokes equation which is not needed for solving the level set equation. For the D3Q7 lattice structure when $i = 0$, $A_i = \frac{1}{3}$ and for $i = 1$ (the axil links), $A_i = \frac{1}{9}$. As shown by Zhao [13], the parabolic diffusion equation can be recovered by the Chapman-Enskog expansion:

$$\frac{\partial \rho}{\partial t} = \gamma \nabla \cdot \nabla \rho, \hspace{1cm} \text{(17)}$$
where \( \gamma \) is a diffusion coefficient defined for a D3Q7 lattice by the relaxation parameter \( \tau \) as

\[
\gamma = \frac{1}{6}(2\tau - 1).
\]  

(18)

In this case, the external force can be included in the same way as in Equation 6. And thus, the modified LBM computation can recover the following equation:

\[
\frac{\partial \rho}{\partial t} = \gamma \nabla \cdot \nabla \rho + \vec{F}.
\]  

(19)

Using this equation to compute the distance field (replace \( \rho \) by \( \phi \)), the level set equation can be recovered, where \( \vec{F} \) is used to accommodate the speed function and the first term relates to the curvature flow effects.
CHAPTER 4

Parallel Image Segmentation

4.1 Level Set Method

The level set PDE provides a good tool for tracking how curves or surfaces move in a discrete domain by maintaining and updating distance fields. Distance fields are numerical values which track how near or far any given point in a discrete domain is to its surface. Figure 4 shows a surface being modeled in the level set framework. This chapter examines how to use the level set method to perform parallel image segmentation using the LBM solver.

4.2 Parallel Level Set Segmentation

The applications of the Level Set Method were introduced in Section 2.2. This chapter gives an overview of how the level set equation is mathematically derived and describes an implicit function for representing surfaces of shapes. The contour evolution can be described as forces that pull the surface in certain directions to achieve a desired result. Figure 4 gives a view of a surface contour propagation in the normal direction. The contour propagation is performed by solving the level set equation, and in this application, is used for the segmentation. A distance field can typically be thought of with the surface representing the zero iso-value with points outside the surface having increasing values and values inside the surface containing negative values, with the biggest negative value being in the center of the surface. This representation is used to distinguish between
inside and outside the shape. The surface \( S \) is defined as distance field \( \phi : \mathbb{R}^3 \rightarrow \mathbb{R} \) for \( p \in \mathbb{R}^3 \).

\[
\phi(p) = \text{sgn}(p) \cdot \min \{|p - q| : q \in S\}
\] (20)

Equation 20 gives this distance field representation in a mathematical framework. In level set methods, the distance field is frequently modified with a particular speed function, to define an evolving zero level set representing the interface that propagates in the normal direction of the interface gradient. As shown in the following sections the speed function is derived as a function of the target iso value, that is the target for the segmentation results.

In image segmentation, the zero level set starts from an arbitrary starting shape (a simple 3D sphere is used in the examples) and evolves itself by a particular level set equation [1]:

\[
\frac{\partial \phi}{\partial t} = |\nabla \phi| [\alpha D(x) + \gamma \nabla \cdot \frac{\nabla \phi}{|\nabla \phi|}]
\] (21)

where \( \phi \) is the distance, \( D(x) \) is the speed function that performs as a driving force to move the evolving level set to target regions, with a user-controlling parameter \( \alpha \) (0.01 is used in the examples). The second term in the right side of the level set equation is a
smoothing term that represents curvature flow. $\gamma$ determines the level of curvature-based smoothness in the results. For a regular distance field, $|\nabla \phi| = 1$, which leads the last term to $\gamma \nabla \cdot \nabla \phi$. Please note that $|\nabla \phi| = 1$ in the framework at all steps, since an adaptive approach is not used and the distance field is valid in the whole domain. From this, Equation 21 is only a variational formula of Equation 19. It depicts that the modified LBM computation leads to a new solution to the level set equation, enabling the use of simple, explicit, parallel computational process for volume segmentation. In this way, this method also has the potential to be applied to other level set based applications. In implementation, a simple D3Q7 lattice (Figure 3) that uses less memory and improves the performance, comparing with a traditional fluid solver using D3Q19 (3D with 19 links to its neighbors) lattice is used. This is made possible since the level set solution does not need to solve the nonlinear advection term as in the Navier-Stokes equations, and the D3Q7 lattice can provide enough accuracy.

4.2.1 Driving Speed Function

Speed functions are designed to make the evolving front of the zero level set propagate to certain target regions. A popular approach is used from Sethian [1] and Lefohn et al. [18] where the speed function is defined by the difference between the target isovalue and the density value at each position:

$$D(I) = \epsilon - |I - T|, \quad (22)$$

where $I$ is the voxel/pixel value at a grid position, and $T$ represents the target density isovalue that the front should evolve to. As the front moves closer to the target region, the speed will converge to zero. The speed term also carries properties that allow the front
to propagate in either direction, based on the sign of the function. The propagating front will expand if $I$ falls in the $T-\epsilon$ or $T+\epsilon$ range, otherwise it will contract. The function $D(I)$ is easily applied to the LBM computation as a body force $\vec{F}$ in Equation 19. $D(I)$ can also be derived based on the gradient defining the object boundary or other user-specified rules.

4.2.2 Level Set Curvature Computation

As mentioned earlier the LBM scheme inherently contains properties that model curvature during the collision and streaming process. The benefit of the LBM method is that the curvature does not need to be computed explicitly, which is hidden in the microscopic LBM collision procedure.

From the LBM-solved diffusion Equation 17, substitute fluid density $\rho$ by the distance value $\phi$. And then by applying $|\nabla \phi| = 1$ for distance field, to get

$$\frac{\partial \phi}{\partial t} = \gamma \nabla \cdot \left( \frac{\nabla \phi}{|\nabla \phi|} \right) |\nabla \phi| = \gamma \kappa |\nabla \phi|, \quad (23)$$

where $\kappa$ represents the mean curvature:

$$\kappa = \nabla \cdot \left( \frac{\nabla \phi}{|\nabla \phi|} \right). \quad (24)$$

In summary, the modified LBM can implicitly provide the curvature-based smoothing effects. This gives great benefit comparing with conventional level set solutions that need to explicitly compute curvatures, as shown by Lefohn et al. [19]. Note that in computing curvatures, upwinding scheme and distance extrapolation may be needed in narrow band propagation, which makes curvature computation a key bottleneck for the method. In comparison, the LBM-based method has a very good feature by being released from such
computing, enabling very good performance and ease of programming.

4.2.3 Relaxation/Smoothing Coefficient

Curvature-based smoothing is essential for level set propagation, which reduces the image noise in the results, and gives a clear and unobstructed view of the target object. It is critical for a segmentation tool to provide user-interactions for controlling the degree of smoothness. The instrumental features of the LBM procedure pertain to its ability to easily control smoothness during the segmentation. This is implemented by utilizing different values of $\gamma$ in the diffusion Equation 17. In the LBM program, different $\gamma$ is easily computed by setting $\tau$ with Equation 18, which is the only parameter in the collision-streaming computations (Equations 1,2).

The effect in a 2D case on a box region with rough boundary is shown in Figure 5(a). Figure 5(b)-(c) show the segmentation result (in red) with different $\gamma$ values of (b) 1.00 and (c) 5.125. This artificial example shows that using a large $\gamma$ can overcome the rough boundary with smooth effects, while a small $\gamma$ can recover the boundary details.

Figure 6(a) shows a CT scan of a foot with size $128 \times 128 \times 128$ with artificial noise added to it. Figure 6(b)-(d) show results using different values for the smoothing coefficient $\gamma$ in Equation 23: (b) 1.00, (c) 5.125, and (d) 15.125. The larger the $\gamma$ coefficient, the more curvature-based smoothing effect is applied to the level set propagation, leading to different segmentation results.

Figure 7(a-d) shows a CT scan of an aneurism with size $512 \times 512 \times 512$. Figure 7(a) shows the results of segmenting the data with a low iso value ($\gamma = 1.125$). There is an appearance of noise in the results. Increasing the $\gamma$ values in images (c-d), the reduction
Figure 5: Segmentation of an artificial 2D image. (a) A 2D box with rough boundary. (b)-(c) Segmentation results with different curvature-based smoothing coefficients: (b) 1.00 and (c) 5.125.

of noise is evident, and preserves the smooth edges of the segmented results.

4.3 Computational Procedure

Initially the original volume is copied to the GPU memory where all computational steps are completed. The steps for the computation are outlined as follows:

Figure 6: Segmentation of a 3D foot with different curvature-based smoothing coefficients at the same number of segmentation steps.
1. Generate initial distance field $\phi$ from a starting circular or spherical shape representing zero level set. This is trivial because an analytic solution exists for a sphere. Using a simple initial shape has the advantage that it could be created on the worker nodes in a cluster environment.

2. Initialize LBM distribution functions of lattice links for each grid point $x$, i.e., compute initial $f_i$ from Equation 16.

3. Define preferred curvature parameter $\gamma$ and then use it to set relaxation coefficient $\tau$;

4. Compute driving speed function $D(x)$ at each grid point $x$ by Equation 22; Set external force $F$ by $D(x)$ in LBM by Equation 6.

5. Perform streaming-collisions as described by Equation 1 and Equation 2;

6. Accumulate the $f_i$ values at each grid point by Equation 4, which generates an updated distance value at each point $x$.

7. If level set converges to the target region, output zero level set. Segmentation is complete.

8. Otherwise, go back to Step 5 to perform streaming-collision again.

These steps are also valid for cluster computing. After data decomposition, only the streaming operation (Equation 2) needs to transfer $f_i$s to neighbors, which requires only neighboring operations. While some neighboring information are on other nodes in the cluster, ghost layers are applied to handle the boundary cases, which is described in next
(a) $\gamma = 1.125$  (b) $\gamma = 5.125$  (c) $\gamma = 10.125$  (d) $\gamma = 15.125$

Figure 7: Segmentation of a 3D aneurism with different curvature-based smoothing coefficients. All images are shown at the same number of LBM steps (25 in this example).

chapter. All the other LBM computations are performed locally on each machine, which implies the advantage of this parallel scheme.
CHAPTER 5

Hardware Accelerated Segmentation

The graphics processing units (GPU) have gained much exposure in recent years as they are crucial in rendering and visualizing advanced realistic scenes and data in modern games, CAD, medical applications, etc. The advantage of modern graphics cards lies in its ability to be extended by the end user to execute custom written code. The architecture of the GPU is SIMD, where the programs can operate on multiple data in parallel. These GPUs have renewed much interest on deriving and solving algorithms that can be extended to run in parallel. The Lattice Boltzmann Method fits very well to be solved by the GPU, which is why it was chosen for implementation of the algorithm. New libraries are also being developed by the GPU manufacturers that simplify the programming steps that have to occur to code and run kernels.

GPUs in general are a set of multiprocessors in a single hardware device. Each multiprocessor is a 32-bit processor with an SIMD architecture. One set of code (kernel) is streamed over multiple data inputs. At each clock cycle a single multiprocessor executes the same instruction on a group of threads. The architecture of the G80 is a new unified shader model which breaks away from the traditional vertex, geometry, pixel shader pipeline that has traditionally been used to program graphics processors. The G80 architecture allows developers to view the device as a multiprocessor based system as its primary feature and its rendering capabilities as a secondary feature if necessary. A programmer will be able to treat G80 as a hugely parallel data processing engine.
Figure 8: GPU Device Architecture. Courtesy of Nvidia
Applications that require massively parallel computational power will see huge speed up when running on G80 as compared to the CPU. This includes financial analysis, matrix manipulation, physics processing, and all manner of scientific computations.

CUDA is a "C" based syntactical language that allows programmers without a graphics background to easily take advantage of the hardware environment. The CUDA Programming Guide [20] outlines the library for developers and instructs how to implement and develop programs. Much of the traditional graphics pipeline, where users were required to program the hardware via kernel programs for the vertex and fragment shaders, and loaded via graphic based api libraries such as DirectX and OpenGL, is abstracted away and embedded in the CUDA libraries, allowing the programmers to just spend time on the algorithm programming. CUDA also has enhanced debugging features compared to the previous methods of programming GPUs, that supports breakpoints and single stepping through code. Ryoo et al. [21] outlines optimization principals that can greatly improve the performance of CUDA-based code run on Nvidia processors. The architecture of the GPU consist of multiple streaming processors that are able to share device memory. GPU memory is increasing very rapidly and user systems can be configured with multiple GPUs. This trend should continue in the future, giving more rise to parallel algorithms being solved on GPU(s).

5.1 Single GPU

An overview of how to implement and design the LBM based level set solver using the CUDA language will be explained, based upon the overview in Section 4.3. The solver can be implemented using three main structures on the GPU to store the following data:
• amData - The volume data set; allocated on the GPU as a single dimensional array of size $X \times Y \times Z$ (size of the data set).

• amDistanceField - The distance field that will track the zero level (segmentation results) stored as a linear floating point array of size $X \times Y \times Z$ (same size of initial data).

• amFqi - The LBM lattice structure (D3Q7). A structure containing 7 floating points is defined. A linear array using the lattice structure is created so that each voxel will have lattice structure assigned to it.

After the volume data set and initial distance field (sphere in the examples) are copied to the GPU, the $amFqi$ lattice data is initialized. To initialize $amFqi$, the initial $f_{eq}^i$ is calculated using the starting distance field. The programmer can easily specify how many threads are used to execute the kernel on the GPU. Listing 5.1 shows how to call the cuda kernel to initialize the $amFqi$ structure. By specifying the number of threads of $(X, 1, 1)$ on a grid size of $grid(Y, Z)$, every time the kernel runs, each voxel being processed will run on its own thread. Listing 5.2 shows how to access the current voxel inside the kernel code. In this listing threadIdx and blockIdx are cuda variables that specify the current thread and block.
Listing 5.1: Shows the thread and grid configuration of the kernel being called. $X \times Y \times Z$ represents the size of the volume data.

```c
dim3 threads(X, 1, 1);
dim3 grid(Y, Z);

// Initialize the LBM lattice structure
IntiLbm3D_k<<<grid, threads>>>(X, Y, Z, amFqi, amDistanceField);
```

Listing 5.2: Shows how to index the voxels inside of the kernel.

```c
// Indices
int tx = threadIdx.x;
int bx = blockIdx.x;
int by = blockIdx.y;

// Local X, Y, Z
int x = tx; int y = bx; int z = by;
```

Coding listing 5.3 shows the a sample CUDA kernel used to implement the diffusion streaming and collision process with neighboring voxels for a D3Q7 lattice structure. The code shows the calculations for updating the $amFqi$ data structure which is essentially the implementation for Equations 1 and 2. After the collision and streaming steps are finished the distance field is updated by accumulating the values along the lattice links.
Listing 5.3: Calling Diffusion Stream and Collision Kernel

```c
// Indexs
int tx = threadIdx.x;
int bx = blockIdx.x;
int by = blockIdx.y;

// Local X, Y, Z
int x = tx;  int y = bx;  int z = by;

// Stream and Collision along the lattice links
for(int nDirection = 0; nDirection < 7; nDirection++)
{
    index = Index(x, y, z);  // Current 3D to 1D voxel mapping

    // Return the neighboring voxels
    if(GetNeighbour(nDirection, &nNeighbourXYZ))
    {
        mFqi = amFqi[index].Data[nDirection];
        mEqu = DiffusionEquilibriumFqi(nDirection, amDistanceField[index]);
        mTemp = mFqi - (mFqi-mEqu) / relaxTime;

        speed = (amTargetObject[index] - isoValue);
        if(speed < 0)
            direction = 1;
        else
            direction = -1;

        amFqi[NeighbourXYZ].Links[nDirection] = mTemp + (direction * fabs(speed) * alpha);
    }
}
```

5.2 GPU Cluster

Using the LBM as a numerical solver is straightforwardly mapped to a single GPU implementation, as shown by many LBM-based simulation applications. To handle large data sets, the algorithm can be extended to multiple GPUs organized in a cluster environment. Initially, the large volumetric data sets are distributed to the cluster nodes (with the initial distance field) by a designated master node. The role of the master node is to be responsible for partitioning the initial data, and after the nodes are done with their computations, assembling the data for visualization. If simple shapes are used as the distance fields, work could be extended to allow the worker nodes to generate the
initial distance field.

The machine used for this research is a Linux-operated cluster consisting of seventeen nodes, each having a dual core or a quad core AMD Opteron processor, and equipped with an Nvidia 8800 GTX graphics card with 768 MB memory. A 3D volume data set is divided into 16 blocks to 16 worker nodes, with a $4 \times 4$ organization of the nodes. The data could be segmented differently, but to keep the number of ghost layer transfers to a minimal amount, the data was just partitioned in the X and Y direction. The master node assembles separate results with correct coordinates transformation and indexing, and uses a Marching Cubes method to render the segmented features on the distance field. Figure 10 shows the cluster configuration described in this manner. The data distribution is performed in parallel and dispatched to different nodes using OpenMP on the master node, which is used to accelerate the data transfer in the cluster environment. During computing, necessary boundary data and loading data on the worker nodes is transferred with the MPI (Message Passing Interface) library.

Between consecutive LBM steps, it is necessary for the working nodes to share the LBM data (i.e., $f_i$ values) residing on the boundaries between each pair of neighboring blocks. A ghost layer method is used to handle this problem. Figure 9 shows the result of the segmentation without the use of the ghost layer transfer between computation steps. The nodes in this example are solving the LBM streaming and computation with out the knowledge of the lattice data on the other nodes, hence the appearance of seams. This is why it is necessary to use the ghost layers. With ghost layers, each data block contains an extra layer of data, the ghost layer, to communicate with each of its neighbors, which is shown in Figure 11. For example, one node $A$ performs computation on data layer $A_1$ to
Figure 9: Image shown with the absence of ghost layer transfer between computational steps. Nodes solve the LBM streaming and collision process without the knowledge from neighboring nodes. The segmentation results are not accurate, showing the need for ghost layer transfers.

After each step, data in $A_1$ is transferred to the ghost layer $B_{n+1}$ of its neighbor node $B$. Meanwhile, $B$’s $B_n$ layer will be transferred to the ghost layer $A_0$ of $A$. Next step, $A$ will use $A_0$ to implement streaming operation and $B$ will use $B_{n+1}$ as well. The data transfer only involves the boundary layers with a very small amount of data compared with the total data size. With an infiniBand network equipped in the cluster, data can be transferred with a speed at an order of gigabits per second.
Figure 10: Data decomposition to cluster nodes for LBM computations. The master node is shown with the segmentation in a $4 \times 4$ in the X and Y direction. The blocks are sent to the individual nodes in the cluster for computation.

Figure 11: Uses ghost layer to transfer boundary data between neighboring nodes in the network. Extra layers are used to save data from the neighboring nodes. The LBM streaming and collision method is computed on the whole data including the ghost layer data.
CHAPTER 6

Results and Performance

The results of the level set segmentation solved by the LBM method are given in this chapter. The method is applied to several volume data sets, in both 2D and 3D domain. To visualize the evolving interface, a simple Marching Cubes method is used to generate a mesh representation of the target distance field for the 3D case. The system used for the tests is an AMD Athlon X2 4600+ 2.41 GHz with 2 gigabytes of RAM.

Since the LBM is an explicit local method it can easily be discretized across multiple parallel processors and the target data can be solved independently in blocks and finally combined on a single GPU to view the results. A cluster of 17 GPUs in a Linux cluster environment was used for the multiple GPU segmentation process. After computation, a Marching Cubes method is used on the master node to visualize the resultant zero level set. Further improvement to the visualization algorithm could be implemented by applying the Marching Cubes on each node simultaneously and sending the triangle list back to the master node, paralleling the visualization process. To visualize the triangle mesh, it is rendered with OpenGL.

6.1 2D Slices on Single GPU

First the LBM segmentation results are shown by visualizing 2D medical imaging slices in Figure 12 and 13, where two regions of a CT and a MRI brain are segmented, respectively. Both slices have a dimension of $512 \times 512$ and are able to complete the
Figure 12: Results of segmentation of a cross section in a CT brain. (a) Initial image with starting zero level set represented in red. (b) Segmented result of the object represented by a target isovalue of 228.

computations in 6 LBM computational steps with each step taking less than a second on one Nvidia 9800 GPU. In comparison, the same results are accomplished in around 5 seconds per step on an AMD 2.4 GHz processor. Please note that the total steps used for the front to reach final result are determined by the position and shape of the roughly-initialized circle. The red circle in Figures 12(a) and 13(a) represent the initial fronts, while Figure 12(b) and 13(b) are the segmented results. The target iso-level in Figure 12 is a grey scale pixel value of 228. In Figure 13, a value of 167 is used.

6.2 3D Images on Single GPU

The LBM-based 3D segmentation result running on a single GPU is shown in Figure 14, on a 3D CT head with a data size of $128 \times 128 \times 128$. Figure 14(a)-(c) depict different segmentation results using different curvature controlling coefficients (by useing different $\gamma$ values). The results are shown by applying a Marching Cubes computation to generate the zero level set from the final distance field. In Figure 14(c), a preferred segmentation
result is achieved using $\gamma = 1.125$. It can be found that the segmented results are smoother than the direct isosurface shape (Figure 14(d)), removing and smoothing noises. The simulation is performed in about 0.8 seconds per LBM computation, while in contrast it takes 15 seconds per step for the CPU implementation. The total number of LBM steps required for the front to approach the final result is controlled by how users initiates the initial sphere. However, this method can achieve very fast front propagation speed (in a few steps) even for an awkward input of the initial shape. Furthermore, using larger $\gamma$ values, the segmentation results may show unfavorable curvature flow effects that impair the level set front propagation behavior, as in Figure 14(a-b). This system provides easy control on examining different coefficients. The GPU-accelerated simulation can give users very fast segmentation results, so that they can interactively adjust the coefficient to finalize a perfect result.
Figure 14: Results of segmenting a CT head with a target iso-value of 60. Data size is $128 \times 128 \times 128$. (a) Segmentation results using $\gamma = 20.125$; (b) $\gamma = 5.125$; (c) $\gamma = 1.125$; (d) Target isosurface generated by direct Marching Cubes.

Figure 15: Results of segmenting an Aneurism data with a target iso-value of 32. Data size is $512 \times 512 \times 512$. $\gamma = 1.5$. (a) Level set propagation after 3 steps; (b) After 10 Steps; (c) After 25 Steps; (d) After 50 Steps.
Figure 16: Results of segmenting a volumetric CT scan of a Bonsai tree with a target iso-value of 20. $\gamma = 5.125$. Data size is $1024 \times 1024 \times 308$. (a) Initial level set as a sphere; (b) After 25 Steps; (c) After 45 Steps; (d) Direct isosurface.

Figure 17: Results of segmenting a volumetric abdomen data with a target iso-value of 20. $\gamma = 1.125$. Data size is $512 \times 512 \times 174$. (a) After 3 Steps; (b) After 25 Steps; (c) After 45 Steps; (d) Direct isosurface.
Figure 18: Results of segmenting a volumetric Porsche data set with a target iso-value of 32. $\gamma = 5.125$. Data size is $509 \times 1023 \times 347$. The initial sphere of radius size 100 was used for the segmentation (a) After 3 Steps; (b) After 25 Steps; (c) After 45 Steps; (d) Direct isosurface.

Table 1: Performance report: Per step (in seconds) average speed to perform LBM computation and ghost layer communication, the memory size per node, and the total ghost layer data size on the GPU cluster with $4 \times 4$ configuration.

<table>
<thead>
<tr>
<th>Data Model</th>
<th>Data Size</th>
<th>LBM Speed Per Step (sec)</th>
<th>Ghost Layer Transfer Per Step (sec)</th>
<th>Total Speed Per Step (sec)</th>
<th>GPU Memory Size Per Node (MB)</th>
<th>Total Ghost Layers Data Size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CT Head</td>
<td>$128 \times 128 \times 128$</td>
<td>0.01</td>
<td>0.04</td>
<td>0.05</td>
<td>4.3</td>
<td>5.2</td>
</tr>
<tr>
<td>MRI Head</td>
<td>$256 \times 256 \times 256$</td>
<td>0.08</td>
<td>0.21</td>
<td>0.29</td>
<td>36</td>
<td>21</td>
</tr>
<tr>
<td>CT Abdomen</td>
<td>$512 \times 512 \times 174$</td>
<td>0.22</td>
<td>0.51</td>
<td>0.73</td>
<td>97.9</td>
<td>28.5</td>
</tr>
<tr>
<td>CT Colon Phantom</td>
<td>$512 \times 512 \times 412$</td>
<td>0.57</td>
<td>1.61</td>
<td>2.18</td>
<td>248.6</td>
<td>72.3</td>
</tr>
<tr>
<td>Aneurism</td>
<td>$512 \times 512 \times 512$</td>
<td>0.66</td>
<td>1.96</td>
<td>2.62</td>
<td>288</td>
<td>84</td>
</tr>
<tr>
<td>Porche</td>
<td>$509 \times 1023 \times 347$</td>
<td>0.96</td>
<td>2.57</td>
<td>3.53</td>
<td>427</td>
<td>88</td>
</tr>
<tr>
<td>Bonsai</td>
<td>$1024 \times 1024 \times 308$</td>
<td>1.57</td>
<td>5.21</td>
<td>6.81</td>
<td>653</td>
<td>101.1</td>
</tr>
</tbody>
</table>

6.3 3D Images on GPU Cluster

Figure 15 shows the segmentation results on the GPU cluster as described in Section 5.2. A 3D Aneurism data is used with a data size of $512 \times 512 \times 512$. The propagating zero level set at different steps are displayed. The image sequence demonstrates the process of the level set propagation from 3 steps (Figure 15(a)) to 50 steps (Figure 15(d)), at which the final surface covering the target region is achieved. In between, Figure 15(b)-(c) are the intermediate results at step 10 and 25, respectively. Please note that the total steps used for the level set to reach final results are determined by the position and
shape of the initial starting level set, while a simple sphere is used in all the 3D example computations.

In Figure 16, a CT scan of a Bonsai tree is used, with a volume size of $1024 \times 1024 \times 308$. A $\gamma = 5.125$ coefficient is applied. The initial level set in Fig. 16(a) is shown, which is started as a sphere. Figure 16(b) is the level set propagation result after 25 steps, where the rough shape of tree is recovered. After 45 steps, Figure 16(c) is the result when the level set evolving stops. It is compared with a direct isosurface rendering in Figure 16(d), where a smoother effect is discovered. The effect may be improved by using a different controlling parameter, $\gamma$.

Figure 17 shows another example with a CT abdomen data set, whose data size is $512 \times 512 \times 174$. Figure 17(a) is the segmentation results with a $\gamma = 1.125$ and after 25 steps. In comparison, Figure 17(b) is the direct isosurface visualization. Results of segmenting a volumetric Porsche data set is shown in Figure 18. The level set equation has the properties of being able to split and merge the zero level set contour to map to non-contiguous pieces of data in the data set, which is shown with this example. The iso value of interest is 32 in this example and the segmentation is shown after 3, 25, and 45 steps (a-c).

6.4 Performance and Discussion

Several large volumetric data sets with various data sizes are run on the GPU cluster. The computation is completely GPU-based. The $4 \times 4$ configuration is used to decompose and distribute the data to 16 nodes, as discussed in Section 5.2. Table 1 outlines the performance results. The average speed per step, which is composed of two parts: LBM
computation and ghost layer handling is computed. Also reported is the GPU memory consumption on each node, and the total size of all ghost layers that determines the network traffic speed. In Table 1, it clearly shows that the method achieves very good performance to segment very large data. For the largest Bonsai data, it uses 6.81 seconds on average per step. The segmentation of the Bonsai completes at around 45 steps in Figure 16, leading to a total processing time at around 306 seconds. The segmentation usually uses tens of total steps for a large data. With an averaging per step speed at a few seconds, the whole process can generally be accomplished in tens to hundreds of seconds depending on the data sets.

In detail, the LBM level set computation is very fast even for a very large data set. It uses 1.57 seconds for the Bonsai data, which runs on one $256 \times 256 \times 77$ volume per node due to the data division scheme. The computation for the ghost layers handling is a little slower, which includes (1) data readback from GPUs, (2) network transfer, and (3) data write to GPUs. For the Bonsai data, it costs 5.24 seconds. The total ghost layer data size (on all the nodes) reaches 101.1 MB. Although such data size does not impose a challenge on the infiniBand network, the GPU readback may consume a little more time than the LBM computation, which is a known bottleneck of GPU computing. The performance could be improved by further optimization of the ghost layer processing. Also a different cluster configuration for data decomposition could be implemented. For the memory occupation, the Bonsai data uses 693 MB on each GPU node. Although each node has a limited 765 MB GPU memory (which will be improved soon), with a large-sized GPU memory (currently 4GB is available), the method can be directly applied for an even larger volume data. Anticipation of using the parallel segmentation technique to many
applications, which are in dire need for adopting distributed computing to segment very large data sets is expected.
CHAPTER 7

Conclusion and Future Work

Common segmentation techniques such as isovalue thresholding are not adequate enough to handle complex 3D images generated by medical or other scanning devices. It proves necessary to implement advanced techniques which have the power to give clearer segmentation results by solving level set equations. Popular level set approaches on a single GPU are not easily extended to very large volume data sets, which however, are prevalent in practical applications. This thesis has proposed an inherent parallel method to solve the segmentation problem flexibly and efficiently on single and multiple GPUs. Based on an extended LBM method, this method lends itself as a good segmentation tool with easy implementation, implicit curvature handling, and thus controllable smoothness of the segmented data. With its parallel scheme, only minimal data processing is required for implementing the method in a GPU cluster, comparing with the previous single GPU approaches. This thesis reported the good performance achieved on multiple data sets on the cluster. In summary, the scheme provides a viable solution for large-scale 3D image segmentation in adoption of distributed computing technology. It has great potential to be applied in various applications. In the future, work could be done on combining parallel visualization techniques with the segmentation, to further augment the ability of the method. Parallel visualization techniques could include, volume rendering on the nodes and combining the results in a single image. A marching cubes method could also be used where the nodes in the cluster generate the triangles of the mesh and send those
to master node instead of the distance field.
BIBLIOGRAPHY


