Analyzing Brain Tumor MRI to Demonstrate GPU-based Medical Image Segmentation*

Sajedul Talukder
Southern Illinois University
Carbondale, IL, USA
sajedul.talukder@siu.edu

Neil Noyes
Edinboro University
Edinboro, PA, USA
nn180872@scots.edinboro.edu

Abstract

Image processing, a computationally intensive task must be done quickly, efficiently, and painlessly in a variety of medical imaging applications to assure quality for both patients and clinicians. Graphics Processing Units (GPUs) have been increasingly used in medical image processing in recent years due to their high efficiency and parallel capabilities. In this paper, we compare the performance of several GPU-based image processing algorithms used in medical imaging against their CPU-based counterparts on a quantitative basis. To illustrate the possibilities of GPU segmentation on a brain MRI containing a significant brain tumor, we investigate an NVIDIA Clara-driven GPU segmentation extension within the program 3D Slicer.

*Copyright ©2021 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

One of the most essential forms of computing technology, both for consumer and corporate computing, is the graphics processing unit (GPU) [13]. The GPU was originally created to speed up the rendering of 3D graphics, but it is now utilized in a variety of applications, including graphics [13], digital automation [20, 11, 25] and video rendering. GPUs are becoming more popular for use in artificial intelligence (AI) [2], cybersecurity [22, 4, 23], and medical image segmentation [18, 24], despite their best-known capabilities in gaming and creative production [5]. Image segmentation, or the division of an image into disjoint sections based on image characteristics such as color detail, strength, or texture, is one of the most fundamental tasks in computer vision and image processing, and it is essential for many applications such as object detection, labeling, and recognition [14, 21]. Image segmentation is regarded the most important medical imaging procedure since it extracts the area of interest (ROI) using a semiautomatic or automated procedure. Picture segmentation is the process of dividing an image into ROIs based on a description. This is generally used for boundary detection, tumor detection/segmentation, and bulk detection by segmenting bodily tissue. Due to the enormous number of applications for image processing algorithms in the medical profession, there is a community committed to developing more efficient and speedier solutions.

The medical field is an excellent example of where rapid and efficient imaging techniques are required. Medical imaging is now used in a variety of clinical applications, ranging from medical scientific research to diagnoses and therapy planning. However, due to the huge three-dimensional medical datasets to analyse in real clinical applications, medical imaging techniques are typically computationally intensive. In comparison to traditional CPU-based computing frameworks, GPU computational speed has risen so quickly in recent years that it now serves as a better platform for considerable acceleration for many computationally complex activities.

Previous research in this field reveals that GPU-based picture segmentation and processing is far more efficient than CPU-based computing frameworks. We focus on the advantages that GPU-based image processing may provide when employed in a variety of medical applications, and we provide quantitative reasoning to propose the best strategy. We also implement an extension for the program 3D Slicer [1], a free, open-source, and multi-platform software package widely used for medical, biomedical, and related imaging research, to explore the capabilities of this computing framework by combining GPU-based computing with NVIDIA AI-Assisted Annotation (AIAA) [12].

Our Contributions. This paper presents the following contributions:

- **Survey Brain Image Segmentation**. Review the literature on im-

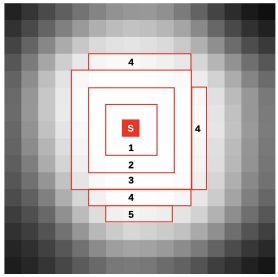

Figure 1: Parallel region growing with double buffering with S as the seed pixel. The numbers indicate at which iteration the pixels in the red regions are added to the final segmentation.

age segmentation within the medical field, considering the GPU as the primary computation brain.

- **Segmentation Approaches**. Provide a comprehensive overview of the low-level and high dimensional medical image segmentation approaches that involves medical image processing such as image registration, segmentation, denoising, filtering, interpolation, and reconstruction.

- **GPU Segmentation Implementation: NVIDIA AIAA**. Implement an extension for the program 3D Slicer with AI-assistive technology (NVIDIA AIAA) to demonstrate the capabilities of GPU segmentation on a brain MRI consisting of a large brain tumor.

The rest of the paper is organized as follows. Section II presents the literature review that closely relate and contribute to our research. Section III discusses the image segmentation approaches. Section IV presents our implementation of the GPU segmentation using NVIDIA AIAA. Finally, Section V concludes the paper with a highlight on the scope of future work.

2 Related Work

There are lots of resources on image segmentation within the medical field, even when considering the GPU as the primary computation brain. The following are a few key examples that closely relate and contribute to our research. Sharma et al. [15] proposed automated medical image segmentation techniques. This related work deals with the differences, benefits, and limitations among MR (magnetic resonance) vs CT (computed topography) imaging and how

they need to be effective for different types of application. Segmentation of a brain is much different from the segmentation of a thorax, so there needs to be clarity on which type of scan/segmentation method is used where and how the results will best reflect the desired outcome.

Taha and Hanbury [19] proposed metrics for evaluating 3D medical image segmentation. Since we are focusing on the efficiency of image segmentation algorithms, we need to clearly define ways in which to confirm quality. Comparing images to evaluate quality is essential to measuring progress within the space, and that is exactly what this related work shows us. The challenges in evaluating medical segmentation are: metric selection, the use in the literature of multiple definitions for certain metrics, inefficiency of the metric calculation implementations leading to difficulties with large volumes, and lack of support for fuzzy segmentation by existing metrics.

Di Salvo et al. [6] presented image and video processing on CUDA. The research proposed here argues for the use of CUDA for various applications of image and video processing. Since GPUs boast simple, data-parallel, deeply multi-threaded cores and high memory bandwidths, fields such as the medical industry can take advantage of the vast amount of efficiency that the platform offers. Using CUDA allows one to achieve very high-performance in time processing while also keeping the same performance in terms of accuracy.

Eklund et al. [7] presented a survey on the medical image processing on the GPU. This research delves into how graphics processing units (GPUs) are used in a wide range of applications. Since GPUs can dramatically accelerate parallel computing, are affordable as well as energy efficient, they can serve crucial for enabling practical use of computationally demanding algorithms used within the medical field (specifically with imaging). This review not only covers GPU acceleration on common image processing operations (such as filtering, interpolation, histogram estimation and distance transforms) but also goes over the most commonly used algorithms in medical imaging and how they are used toward individual scans.

Similarly, Shi et al. [17] presented a survey of GPU-based medical image computing techniques. The goal of this research was to provide a comprehensive reference source for the starters or researchers involved in GPU-based medical image processing. This work goes over the existing traditional applications within the three main areas of medical image processing: segmentation, registration, and visualization. As well as giving an overview of techniques used in application, we see a discussion of the advantages with GPU-based medical image visualization approaches.

Bankman [3] provided an introduction to segmentation. Within the medical field, imaging algorithms need to be able to handle variability. The data collected is from real people each with a unique condition. That is precisely what

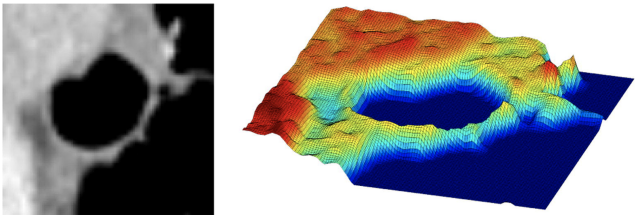

Figure 2: The watershed approach, where intensity is interpreted as height within the topological representation.

this piece of research addresses and reveals techniques used to combat data variation. Fuzzy clustering, the use of deformable models as well as volumetric data are discussed. Volumetric data deals specifically with 3D segmentation and material analysis which is useful for GPU-based imaging techniques.

Kafieh et al. [8] presented a review of algorithms for sgmentation of optical coherence tomography from retina. This research focuses on the different types of algorithms that can be used for the segmentation of optical coherence tomography. Optical coherence tomography (OCT) is a technique to describe different information about the internal structures of an object and to image certain aspects of the biological tissue. This paper reads of the history of the algorithms used within the field and shows how the medical imaging field has grown over the years relating to the precision, abilities, and accuracy of their results.

Smistad et al. [18] provided a comprehensive review of how the GPU can solve problems with segmentation of anatomical structures. Since segmentation of imaging from CT, MRI and ultrasound is key for diagnostics, planning and guidance, more efficient solutions become required as time goes on. This paper goes over the essentials of GPU computing, segmentation algorithms, and a proposal of the future for GPU hardware manufacturers.

3 Image Segmentation Approaches

GPU's, although originally created for rendering graphics, have become popular for lots of high-performance computation due to the more recent programmability and low cost to performance. When compared with a CPU-based computing framework "...the difference in theoretical performance can differ by a factor ten in favor of the GPU" [7]. While both modern CPU and GPU can manage thousands of threads, the GPU running a suitable algorithm may progress over thousands of threads concurrently, while the CPU progresses

over less than a dozen (generously). One of the fields of which capitalized upon this was the medical field within their medical image processing/segmentation. Since there is a strong nature of parallelism among GPU computation, performance often is tied to the implementation of parallel adapted algorithms. This bears fortunate since image processing algorithms are often tremendously parallel by nature, which makes it easier to implement on a GPU [7]. Medical image processing runs on the backbone of a patient's available data, which is steadily increasing for each case, resulting in the need for fast algorithms that excel with accuracy as well as speed. There are many different types of algorithms involved medical image processing such as image registration, segmentation, denoising, filtering, interpolation, and reconstruction [18]. This research will focus mainly on the algorithms involved within image segmentation specifically. The first GPU-based medical image segmentation technique was an attempt to exploit the high performance of graphics cards for different numerical computations by formulating level set segmentation as a sequence of graphics operators of image blending [17]. Another technique, which used the then recent programmable shader technology, used the enhanced flexibility to enable 3D image segmentation using curvature regularization to favor smooth iso surfaces. Among other methods, including ones implemented with CUDA, we can generalize with two types of segmentation approaches: low-level and high-level. Low-level approaches, such as GPU-based implementations of watershed and region growing methods, require no statistical information about the types of objects within the image being segmented and directly manipulate the voxel/pixel information to then form connected regions of interest (ROI's). This is contrast to high-level segmentation frameworks, which do not directly manipulate the pixel/voxel information from the image. There has been work proposed in this area using geodesic active contours as well as contours with gradient vector flow [9] as well as CUDA powered approaches. Since CUDA was released, there has been high improved performance in terms of speedup with respect to the sequential version of various image segmentation algorithms using CUDA and CUDA-enabled GPUs [6]. Since the medical field needs real time, high dimension segmentation approaches, CUDA seems to be the go-to implementation going forward to deal with the very large data sets the medical imaging scans provide. CUDA does not operate proficiently without optimized algorithms though, so one needs to make the right decision with respect to choosing the most fitting algorithm to for the application.

3.1 Interacting Seeded Region Growing

The seeded region approach for image segmentation was designed to be robust, rapid and have no tuning parameters. Instead of the tuning parameters, the algorithm requires the input of a number of seeds (as either pixel or voxel

regions) that control the formation of regions that the image will be segmented. A gray-scale volume image is usually a digital cubic grid $G = (V, E)$ where the vertices V are valued by a function $g : V \to [h_{\min}, \ldots, h_{\max}]$ with $V \subseteq \mathbb{Z}^3$ the domain of the image and $h_{\min}$ and $h_{\max}$ the minimum and the maximum gray-value. A digital grid is a special kind of graph $G = (V, E)$ defined by a set V of vertices and a set $E \subseteq V \times V$ of pairs defining the connectivity. If there is a pair $e = (p, q) \in E$ we call p and q neighbors, or we say p and q are adjacent. The set of neighbors $N(p)$ of a vertex p is called the neighborhood of p. Usual choices are the 6-Connectivity, where each vertex has edges to its horizontal, vertical, front and back neighbors, or the 26-Connectivity,where a point is connected to all its direct neighbors. The vertices of a cubic digital grid are called voxels. Once the seeds are set, either using a GUI or automatically with prior knowledge, we use those seeds as start nodes to then grow our region of interest based on if neighboring pixels fit the predefined criteria. These criteria compare the current pixel to the seed or the already included pixels (once region has grown from seeds) using attributes such as intensity, gradient, or color. The region will grow as long as there are these neighboring pixels that satisfy such condition. The algorithm behaves comparably to a breadth first search algorithm.

As one could imagine, the number of threads grows as the border of the region of interest expands. On CPU-based computing frameworks, changing the number of threads usually is done by restarting the kernel, which makes it so all values from global memory need to be read again. We can, however, use the GPU by having one thread for each pixel in the entire image in each iteration. But doing this adds more computational work due to branch divergence as well as increasing memory usage [18]. Since the traditional CPU-based implementation has too high of a computational cost when images scale larger, CUDA and the GPU have been used to accelerate the seeded region growing approach. One notable approach, respective to this method, proposed to exploit sophisticated graphics hardware functionality (such as floating-point precision, render to texture, computational masking, and fragment programs) to allow the users to interactively paint growing seeds by drawing on the sectional views of the volume. In addition to visualization and interaction, the algorithm leverages the parallel processing capabilities of the GPU to run significantly faster than previous methods [16]. For this approach, it is crucial to draw as many seeded points as possible to make full use of the parallel performances of the GPU.

3.2 Watershed

The watershed approach to image segmentation is based on viewing an image as a 3D object. The height is determined from the intensity of the pixel/voxel. A path $\pi = (v_0, v_1, \ldots, v_l)$ on a grid G from voxel p to voxel

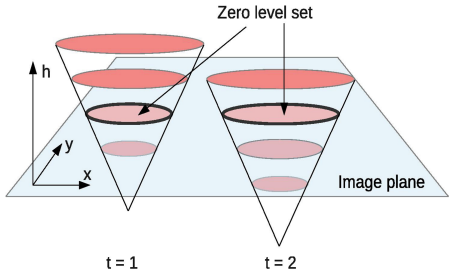

Figure 3: The level-set approach showing the movement through the image plane, which shows a circle that will grow over time.

q is a sequence of vertices where $v_0 = p, v_l = q$ and $(v_i, v_{i+1}) \in E$ with $i \in [0, \ldots, l)$. The length length (π) of a path π is defined as length$(\pi) = l$. The geodesic distance $d_G(p, q)$ between two voxels p and q is defined as the length of the shortest path $\pi_{\min}$ between p and q, with $\forall v \in \pi_{\min} : v \in G$. The geodesic distance between a vertex p and a subset of vertices Q is defined by $d_G(p, Q) = \min_{q \in Q} (d_G(p, q))$ [26].

Once the generated landscape is complete, there are three types of points, categorized by how a drop of water would behave at the specific point. These three categories include scenarios in which a drop of water would:

- stay at the point (local minima)
- move downward to into one local minimum
- move downward into > 1 local minimum

The points that are of type 2 are called watersheds or catchment basins and points of type 3 are called watershed lines or divide lines since they split into more than one local minimum. The distinction between types of points is important because the main purpose of this algorithm is to find watershed lines. This then begs the question; how does one find watershed lines? To answer this question, we must look to analogy based on the topological representation. Suppose there are holes in all the places where a point of type 1 (local minimum) occurs. Therefore, water would flow through these holes. The watersheds within the topological representation would then be flooded at a constant rate. So, when two watersheds are about to merge together, something called a dam (which is exactly the same to a dam you think of holding back a large body of water) is built between them, with an increasing height as the water level rises. This is continued until the height of the dam is the same as the highest

represented point within the topological landscape. These dams that were created are the watershed lines.

As far as acceleration with parallelism on the GPU, we see an obstacle of constructing good multi-threaded algorithms. The watershed approach has an inherent sequential nature. However, attempts have been successful in this regard by either transforming the topological landscape into a graph, subdividing the image, or flooding each local minimum in parallel. Overall, these speedups are reported to be only within the range of 2-7x faster. One notable attempt, by Kauffmann and Piche in 2008 [10], presented the watershed segmentation implementation using the cellular automation approach. They defined their algorithm (see Algorithm 1) by using a weighted graph, $G = (V, E, w)$, or by a valued graph, $G = (V, E, f)$, where respectively $w : E \to \Re$ is a weight function defined on the edges, and $f : V \to \Re$ is a valued function defined on the vertices. They used the Bellman-Ford algorithm to calculate a weighted cost of the shortest path from all pixels to each local minimum. Thus, the shortest path will then always lead downwards. By using this approach, they established a method that can process all of the pixels/voxels in an image in parallel using the same instruction, boasting speedups of 2.5x.

Algorithm 1 Watershed Automata evolution rule

for all minima: s_i with $i \in [1, k]$ **do**

$\lambda(s_i) \leftarrow 0$ and for all $v \neq s_i, \lambda(v) \leftarrow \infty$

$\text{label}(s_i) \leftarrow i$ and for all $v \neq s_i, \text{label}(v) \leftarrow 0$

for all $p \in V$ **do**

$U^t = \min_{q \in N_p} \{\lambda^t(q) + f(p)\}$

$\lambda^{t+1}(p) = \min[\lambda^t(p), U^t]$

$\text{label}^{t+1}(p) = \text{label}\{\min[\lambda^t(p), U^t]\}$

end for

end for

Wagner et al. [26] presented a parallel watershed-transformation algorithm that takes a gray-value gradient image $g : \Omega \to [h_{\min}, \ldots, h_{\max}]$ as input and computes a label image $l : \Omega \to \mathbb{N}$ which contains the segmentation result. For each gray-level h, starting at the global minimum $h_{\min}$, new basins are created according to local minima of the current level h. Already existing basins, are expanded if they have adjoining pixels of the gray-value h. The procedure stops when the maximum gray-value $h_{\max}$ is reached. This implementation was 5-7x faster than serial implementation on 3D images [16].

3.3 Level set Approaches/Methods

The level set approaches to segmentation are based on spreading a contour within the input images. We define a scalar function with positive values inside the current segmentation and having negative values outside, thus defining the segmentation boundary by the function's zero level set. This level set function is one dimension higher than the contour, which means that we can start with 2D images, define a boundary at the start and then propagate along the z axis, almost creating a 3D object out of slices, much like a 3D printer works. The level set function in 2D segmentation, the $z = \phi(x, y, t)$, is defined as a function which returns the height z from the position x,y in the image plane to the level set surface at time t. The contour is defined implicitly as the zero-level set, which is where the height from the plane to the surface is zero. At level-set zero, the image plane and the surface intersect. To propagate the contour in the x, y plane, the level set surface is moved in the z direction. How fast and in which direction a specific part of the contour moves, is determined by how the level set surface bends and curves. The closer the surface is to being parallel with the image plane, the faster it propagates. When the level set surface is orthogonal to the image plane, the contour does not propagate at all. Assuming that each point on the contour moves in a direction normal to the contour with speed F, the contour can be evolved using the following PDE:

$\frac{\partial \phi(x,y,t)}{\partial t} = F(x, y, I)|\nabla \phi(x, y, t)|$

The speed function F varies for different areas of the image I and can be designed to force the contour towards areas of interest and avoid other areas. In image segmentation, the speed function is usually determined by the intensity or gradient of the pixels, and the curvature of the level set function. A negative F makes the contour contract, while a positive F makes it expand.

The level set method starts by setting an initial contour on the object of interest. This is done either manually or automatically using prior knowledge. Next, the level set function is initialized to the signed distance transform of the initial contour. Finally, the contour is updated until convergence. Smistad et al. [18] proposed a parallel level sets algorithm as shown in Algorithm 2. The main issue with the level-set approach comes down to the extremely high computational cost when processing large images. A set of numerical simulations must be performed on every voxel of an image, so the limited bandwidth and limited number of execution cores on a CPU are not ideal.

3.4 Active Contours

Active contours, also known as snakes, were introduced by Kass et al. [9]. These contours move in an image while trying to minimize their energy, as

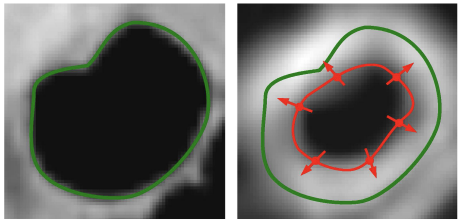

Figure 4: The active contour approach where the left image is the input, and the right image shows the gradient magnitude. The red line superimposed on the right image is the active contour, which is driven towards the high gradient parts of that image, corresponding to the edges in the original image. The green line superimposed on both images show the contour of the lumen.

shown in Fig 4. Active contour segmentation is very similar to that of the level set approach. The major difference between them is that the contour is represented explicitly by a large number of nodes, rather than a mathematical function. They are defined parametrically as $v(s) = [x(s), y(s)]$, where $x(s)$ and $y(s)$ are the coordinates for part s of the contour.

This method can be divided into two different, but still data parallel, operations. The first operation is calculating the external energy. The energy of a contour depends on the shape and is mathematically related to the tension and rigidity of the contour. The energy E of the contour is composed of an internal E_{int} and external energy E_{ext} : $E = \int_0^1 E_{\text{int}}(v,s) + E_{\text{ext}}(v(s))ds$ The internal energy depends on the shape of the contour and can, for example, be defined as:

$E_{\text{int}}(v,s) = \frac{1}{2}\left(\alpha \left|v'(s)\right|^2 + \beta \left|v''(s)\right|^2\right)$ where α and β are parameters that control the tension and rigidity of the contour.

The second is then evolving the contour itself. The contour can be driven towards interesting features in the image, by having an external energy with low values at the interesting features and high elsewhere. There are several different choices of external energy. A popular choice is the negative magnitude of the image gradient, i.e. $E_{\text{ext}}(\vec{x}) = -\left|\nabla\left[G_\sigma * I(\vec{x})\right]\right|^2$, where $G_\sigma *$ is convolution with a Gaussian lowpass filter. The convolution and gradient calculation can be executed in parallel for each pixel, and optimized using texture or shared memory.

A numerical solution to find a contour that minimize the energy E can be found by making the contour dynamic over time $v(s,t)$ $\alpha v''(s,t) - \beta v^{(4)}(s,t) - \nabla E_{\text{ext}} = 0$. The thread count is equal to the number of sample points on the contour, which is much lower than the number of pixels in the image. To evolve

the contour, each point s has to be extracted from the image using interpolation. Thus, active contours may benefit from using the texture memory, which can perform interpolation efficiently.

As shown by Xu and Prince [27], some different formulations of the external force field ∇E_{ext} may get stuck in local minima, especially if boundary concavities are present. They introduced a new external force field, gradient vector flow (GVF), which addressed this problem. The GVF field is defined as the vector field $\vec{V}$, that minimizes the energy function E : $E(\vec{V}) = \int \mu |\nabla \vec{V}(\vec{x})|^2 + \left|\vec{V}(\vec{x}) - \vec{V}_0(\vec{x})\right|^2 \left|\vec{V}_0(\vec{x})\right|^2 d\vec{x}$ where $\vec{V}_0$ is the initial vector field and μ is an application dependent constant. This approach differs from other choices of external energy, which are generally not iterative. GVF is thus more time consuming as many iterations are needed to reach convergence.

Algorithm 2 Parallel level sets

Initial segmentation and input image I Segmentation result S Initialize ϕ to signed distance transform from the initial segmentation

while a number of iterations or until convergence **do**

  for all voxels $\vec{x}$ in parallel **do**

    Calculate first order derivatives

    Calculate second order derivatives

    Calculate gradient $\nabla\phi(\vec{x})$

    Calculate curvature

    Calculate speed term $F(\vec{x}, I)$ $\phi'(\vec{x}) \leftarrow \phi(\vec{x}) + \Delta t F(\vec{x}, I)|\nabla\phi(\vec{x})|$

  end for

  $\phi = \phi'$

end while

for all voxels $\vec{x}$ in parallel **do** $\phi'(\vec{x}) \leq \phi(\vec{x})$ $S(\vec{x}) \leftarrow 1$

  $S(\vec{x}) \leftarrow 0$

end for

return

3.5 GPU Segmentation Implementation: NVIDIA AIAA

There are many tools developed for image segmentation in the medical field, and I would like to conclude my GPU-based image segmentation research by

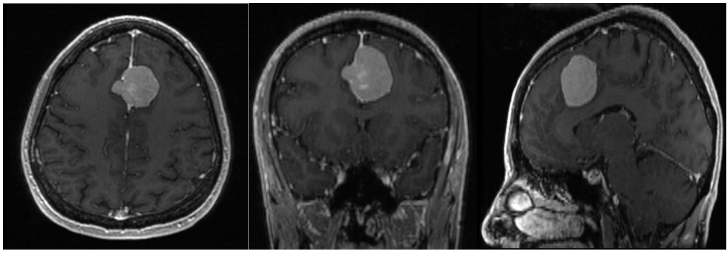

Figure 5: Three views from a contrast-enhanced brain MRI.

implementing a popular and free extension called "NVIDIA AI-Assisted Annotation (AIAA) for 3D Slicer" to demonstrate the capabilities of image segmentation based on the GPU. This extension uses artificial intelligence to train models based on human clinical data sets. Once a scan is initiated, the input scans are sent to a server running linux with an NVIDIA GPU for processing and returns a 3D segmentation result to slicer. For example, this may include a 3D tumor or mass of some kind.

3D Slicer is an open-source software platform for medical image informatics, image processing, and three-dimensional visualization. Mainly written in C++ and based on the NA-MIC kit, 3D Slicer relies on a variety of libraries: VTK, ITK, CTK, CMake, Qt and Python. 3D Slicer consists of both an application core, as well as having modules that offer specific functionality. The core implements the UI, data I/O, and visualization, while exposing developer interfaces for the use of extensions with new modules.

Using this extension, we will walk through an example of segmenting a brain tumor. We are provided a pre-trained ai model for segmenting tumors on contrast-enhanced brain MRI scans, we will be using this for our segmentation.

Step 1. The first step is to load the dataset into 3D Slicer, and once that is completed, we end up having three views from a contrast-enhanced brain MRI as shown in figure 5.

This importing process gives the user a scrollable three panel window displaying the scans. Each of these three views are fully interactable. If you are familiar with MRI scans, we can scroll through them to see all the layers through the scan. This means you can scroll from the very edge of the skull, all the way into the head to get to the meat of the mass we are trying to segment. This is our region of interest.

Step 2. Once this process is complete, we need to go to what Slicer has coined the Segment Editor. The segment editor is where the user adds segmentations to the imaging scans. Since we are using the NVIDIA AIAA extension,

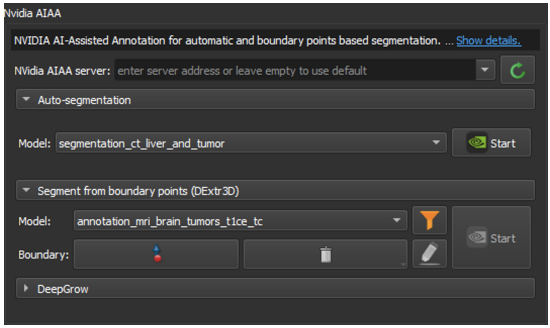

Figure 6: NVIDIA AIAA Slicer's UI.

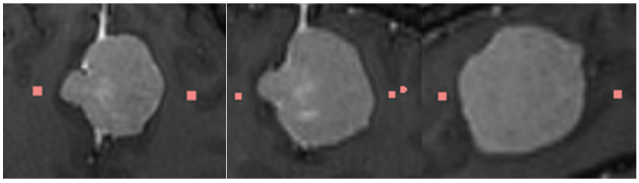

Figure 7: Setting two boundary points on the left and right side of the ROI.

we need to configure the segmentation accordingly. Once inside the segment editor, we add a new segment using the add button in Slicer's UI and add the NVIDIA AIAA effect. Once the effect is added to the segmentation, we get more options in the Slicer UI (see figure 6). We want to configure a segment from boundary points that uses the model "annotation_mri_brain_tumors_-t1ce_tc." This model is trained to segment tumors on contrast enhanced brain MRIs.

Step 3. In each image, we set two boundary points on the left and right side of the ROI, so the algorithm has a base of where to start segmenting, shown in figure 7 as pink squares when placed.

Step 4. After the boundary points have been set on each view of the scan, we are then ready to hit the now ungrayed out NVIDIA start button (when compared with figure 6) to send our data to their server for processing on their NVIDIA GPU powered server. After a few seconds, thanks to the power and capabilities of GPU computing, we get returned to Slicer a fully manipulatable 3D representation of the tumor, as shown in figure 8.

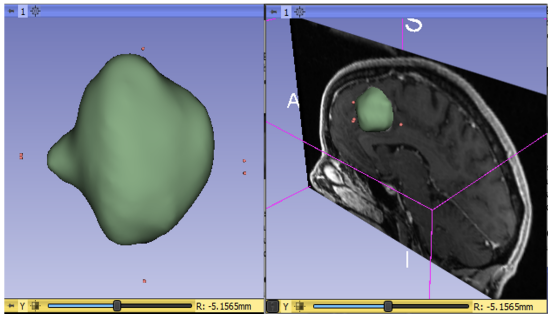

Figure 8: A fully manipulatable 3D representation of the tumor.

4 Conclusions

In terms of medical imaging, we've demonstrated how useful it is to use GPU-based processing. There are some applications where GPU-optimized algorithms fail to yield significant improvements, which is to be anticipated since the algorithms were designed with CPU architecture in mind. This is a small percentage of algorithms, as the great majority are parallel by nature and so thrive on a GPU-based computing platform. We've seen how great the gains are for specific algorithms designed for usage on the GPU in all of the trials included in this study, and the results speak for themselves. With speedups ranging from 2 to 13 times, we can conclude that GPU-based picture segmentation is the way to go at the moment and in the future. Since GPU technology has improved dramatically in the previous decade, we've seen a slew of new picture segmentation implementations emerge, resulting in nothing but better outcomes that will continue to develop and progress.

References

[1] *3D Slicer*. `https://www.slicer.org/`. Accessed: 2021-01-06. 2021.

[2] Toru Baji. "Evolution of the GPU Device widely used in AI and Massive Parallel Processing". In: *2018 IEEE 2nd Electron Devices Technology and Manufacturing Conference (EDTM)*. IEEE. 2018, pp. 7–9.

[3] Isaac N Bankman. *Introduction to segmentation*. 2000.

[4] Sweta Bhattacharya et al. "A novel PCA-firefly based XGBoost classification model for intrusion detection in networks using GPU". In: *Electronics* 9.2 (2020), p. 219.

[5] Hao Chen et al. “Learned Resolution Scaling Powered Gaming-as-a-Service at Scale”. In: *IEEE Transactions on Multimedia* (2020).

[6] Roberto Di Salvo and Carmelo Pino. “Image and video processing on CUDA: state of the art and future directions”. In: *MACMESE* 11 (2011), pp. 60–66.

[7] Anders Eklund et al. “Medical image processing on the GPU–Past, present and future”. In: *Medical image analysis* 17.8 (2013), pp. 1073–1094.

[8] Raheleh Kafieh, Hossein Rabbani, and Saeed Kermani. “A review of algorithms for segmentation of optical coherence tomography from retina”. In: *Journal of medical signals and sensors* 3.1 (2013), p. 45.

[9] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. “Snakes: Active contour models”. In: *International journal of computer vision* 1.4 (1988), pp. 321–331.

[10] Claude Kauffmann and Nicolas Piche. “Cellular automaton for ultra-fast watershed transform on GPU”. In: *2008 19th International Conference on Pattern Recognition*. IEEE. 2008, pp. 1–4.

[11] Chiyoung Lee, Se-Won Kim, and Chuck Yoo. “VADI: GPU virtualization for an automotive platform”. In: *IEEE Transactions on Industrial Informatics* 12.1 (2015), pp. 277–290.

[12] *NVIDIA AI-Assisted Annotation (AIAA)*. `https://docs.nvidia.com/clara/tlt-mi/aiaa/index.html`. Accessed: 2021-01-06. 2021.

[13] John D Owens et al. “GPU computing”. In: *Proceedings of the IEEE* 96.5 (2008), pp. 879–899.

[14] Linda G Shapiro and G Stockman. “Computer vision prentice hall”. In: *Inc., New Jersey* (2001).

[15] Neeraj Sharma and Lalit M Aggarwal. “Automated medical image segmentation techniques”. In: *Journal of medical physics/Association of Medical Physicists of India* 35.1 (2010), p. 3.

[16] Anthony Sherbondy, Michael Houston, and Sandy Napel. “Fast volume segmentation with simultaneous visualization using programmable graphics hardware”. In: *IEEE Visualization, 2003. VIS 2003.* IEEE. 2003, pp. 171–176.

[17] Lin Shi et al. “A survey of GPU-based medical image computing techniques”. In: *Quantitative imaging in medicine and surgery* 2.3 (2012), p. 188.

[18] Erik Smistad et al. “Medical image segmentation on GPUs–A comprehensive review”. In: *Medical image analysis* 20.1 (2015), pp. 1–18.

[19] Abdel Aziz Taha and Allan Hanbury. "Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool". In: *BMC medical imaging* 15.1 (2015), pp. 1–28.

[20] Sajedul Talukder. "Tools and Techniques for Malware Detection and Analysis". In: *arXiv preprint arXiv:2002.06819* (2020).

[21] Sajedul Talukder and Bogdan Carbunar. "AbuSniff: Automatic Detection and Defenses Against Abusive Facebook Friends". In: *Twelfth International AAAI Conference on Web and Social Media*. 2018.

[22] Sajedul Talukder and Zahidur Talukder. "A Survey on Malware Detection and Analysis Tools". In: *International Journal of Network Security & Its Applications* 12.2 (2020).

[23] Sajedul Talukder et al. "Attacks and defenses in mobile ip: Modeling with stochastic game petri net". In: *2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC)*. 2017, pp. 18–23.

[24] Sajedul Talukder et al. "Mobile Technology in Healthcare Environment: Security Vulnerabilities and Countermeasures". In: *arXiv:1807.11086* (2018).

[25] Sajedul Talukder et al. "Usensewer: Ultrasonic sensor and gsm-arduino based automated sewerage management". In: *2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC)*. IEEE. 2017, pp. 12–17.

[26] Björn Wagner, Paul Müller, and Gundolf Haase. "A parallel watershed-transformation algorithm for the GPU". In: *Workshop on Applications of Discrete Geometry and Mathematical Morphology*. 2010, pp. 111–115.

[27] Chenyang Xu and Jerry L Prince. "Snakes, shapes, and gradient vector flow". In: *IEEE Transactions on image processing* 7.3 (1998), pp. 359–369.