GPU Implementation for 3D GVF Force Field

Quan Wang, Yu Wang
ECSE, Rensselaer Polytechnic Institute, Troy, NY, USA
November 2010

Abstract: Gradient Vector Flow (GVF) snake is one kind of active contours - curves that can move within images to find the boundaries of objects. 3D active contours are also known as deformable models. GVF snake begins with calculating the GVF force field over the image domain, which will force the snake into concave regions of original images. Since the calculation of GVF force field usually takes a long time, in this project we implement the GVF algorithm with GPU, which will accelerate the algorithm to a great extent.

1 Introduction

Active contours, or snakes, have wide applications in image processing, especially in detecting boundaries of objects. However, there are two major problems with active contours. First, we must ensure that the initial contour is close enough to the boundary. Second, the active contours will not converge to concave regions. The force balance equation of active contours consists of two parts, the internal force and the external force. By substituting the external force with GVF force, these two problems will be well solved.

In 2D scenario, we first define the *edge map* $f(x,y)$ of an image $I(x,y)$. $f(x,y)$ will be larger near the image edges. We can define the edge map in one of the following forms:

$$f(x,y) = |\nabla I(x,y)|^2$$

$$f(x,y) = |\nabla (G_\sigma(x,y) * I(x,y))|^2$$

where ∇ is the gradient operator and $G_\sigma(x,y)$ is a 2D Gaussian function with standard deviation σ.

The vector in field ∇f will point to the edges of the image. However, the capture range is small and in regions where $I(x,y)$ is constant, ∇f will be zero, thus providing no information about neighbouring edges.

The 2D *gradient vector flow* (GVF) field is defined as the vector field $\mathbf{v}(x,y)=(u(x,y),v(x,y))$ which minimizes the energy function

$$\varepsilon = \iint \mu\left(u_x^2 + u_y^2 + v_x^2 + v_y^2\right) + |\nabla f|^2 |v - \nabla f|^2 dxdy$$

We can see that in regions where ∇f is large, the energy function is dominated by the second term, and will be minimized when $\mathbf{v} = \nabla f$. In regions where ∇f is small, the energy function will be dominated by the first term, the partial derivatives of the vector field. Thus minimizing the energy function will yield a smooth vector field.

The energy function minimizing task is equivalent with solving for the following Euler equations:

$$\mu\nabla^2 u - (u - f_x)(f_x^2 + f_y^2) = 0$$

$$\mu\nabla^2 v - (v - f_y)(f_x^2 + f_y^2) = 0$$

where ∇^2 is the Laplacian operator. With the Euler equations, we can treat *u* and *v* as functions of time and solve for them in an iterative way:

$$u(x,y,t+\delta t) = u(x,y,t) + \mu\nabla^2 u(x,y,t) - (u(x,y,t) - f_x(x,y))(f_x(x,y)^2 + f_y(x,y)^2)$$

$$v(x,y,t+\delta t) = v(x,y,t) + \mu\nabla^2 v(x,y,t) - (v(x,y,t) - f_y(x,y))(f_x(x,y)^2 + f_y(x,y)^2)$$

In 3D scenario, we similarly have the GVF vector field $\mathbf{v}(x,y,z)=(u(x,y,z),v(x,y,z),o(x,y,z))$, and the iterative solution is

$$u(t+\delta t) = u(t) + \mu\nabla^2 u(t) - (u(t) - f_x)(f_x^2 + f_y^2 + f_z^2)$$

$$v(t+\delta t) = v(t) + \mu\nabla^2 v(t) - (v(t) - f_y)(f_x^2 + f_y^2 + f_z^2)$$

$$o(t+\delta t) = o(t) + \mu\nabla^2 o(t) - (o(t) - f_z)(f_x^2 + f_y^2 + f_z^2)$$

However, the CPU implementation of this algorithm can be extremely time-consuming, especially when the 3D image is large. In this project, we present a GPU implementation of the iterative algorithm for GVF force field.

2 Implementation

In this project, we implement a framework to compute the 3D GVF force field with ITK and CUDA. ITK (Insight Toolkit) is an open-source framework for image segmentation and image registration, while CUDA (Compute Unified Device Architecture) is a parallel computing architecture developed by NVIDIA.

2.1 Framework

In this project, we mainly use ITK for reading and writing images, and use CUDA to implement the GVF iterative algorithm. The input of this framework is the edge map image, the regularization parameter μ and the number of iterations of the iterative algorithm. The framework will generate three *.mhd* and three *.raw* files as its output, containing the GVF force field information in three directions.

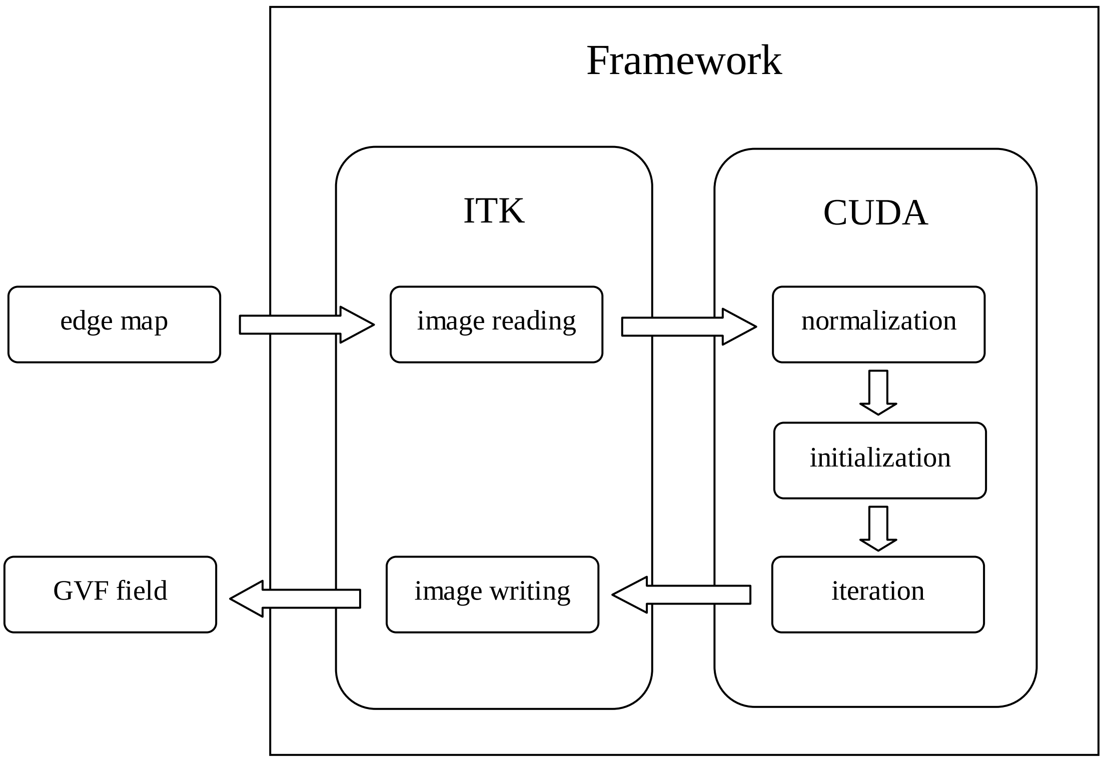

Figure 2.1.1 Framework of the project.

2.2 Image Reading and Writing

The ITK module requires the input image must be a 3D image with pixel type *unsigned char*. The input image will first be loaded as an *itkImage*. Then we allocate a memory section for the edge map pixel values. We also allocate three memory sections for the resulting GVF vector field, which should have *float* pixel type. Then we pass the address of the four memory sections, the size of the edge map image (height, width and frames), the regularization parameter μ, the number of iterations, the minimum pixel value of the edge map, and the range of pixel value of the edge map (maximum pixel value minus minimum pixel value) to the CUDA module. The CUDA module will implement the iterative algorithm and the GVF vector field will be saved in the three corresponding memory sections.

After the computation, the result will be saved in three vector images, corresponding to three different directions. Each vector image is saved in a *.mhd* file and a *.raw* file, with the *.mhd* file containing the image information and the *.raw* file containing the raw data. There are lots of open-source toolkits online for reading such vector images in MATLAB.

The ITK module is also a good example showing how to access to the CUDA module. If we have other forms of edge map images, or we do not need to generate vector image files but want to use the GVF vector field directly, we can write other programs like the ITK module of this framework, and call the CUDA module.

2.3 Parallel Computing

In our CUDA module, all operations based on pixels are performed in a parallel manner. The edge map is first normalized to pixel value range of [-1,1]. Then we initialize the *u*, *v*, *o* for *t*=0 by setting $u(x,y,z,0) = dx(x,y,z)$, $v(x,y,z,0) = dy(x,y,z)$, and $o(x,y,z,0) = dz(x,y,z)$.

The iterative algorithm in *x* direction can be written as

$$u(t+1) = u(t) + \mu\nabla^2 u(t) - (u(t) - f_x)(f_x^2 + f_y^2 + f_z^2).$$

Setting $g = f_x^2 + f_y^2 + f_z^2$, the equation above is equivalent to

$$u(t+1) = (1-g)u(t) + \mu\nabla^2 u(t) + f_x g.$$

This equation implies that in our iterative process, we can save $(1-g)$, $f_x g$, $f_y g$ and $f_z g$ as four different constants instead of the original g, f_x, f_y and f_z. The parallel computing process for *u* can be represented by Figure 2.3.1.

At the beginning of each iteration, we use mirror mapping to handle the boundary pixels of *u*, *v*, and *o*. That is to say, we set the pixel values of the outermost pixels of the 3D vector image cube the same as the third outermost pixels of the cube. For example, we set *u*(*i*,*j*,0)=*u*(*i*,*j*,2).

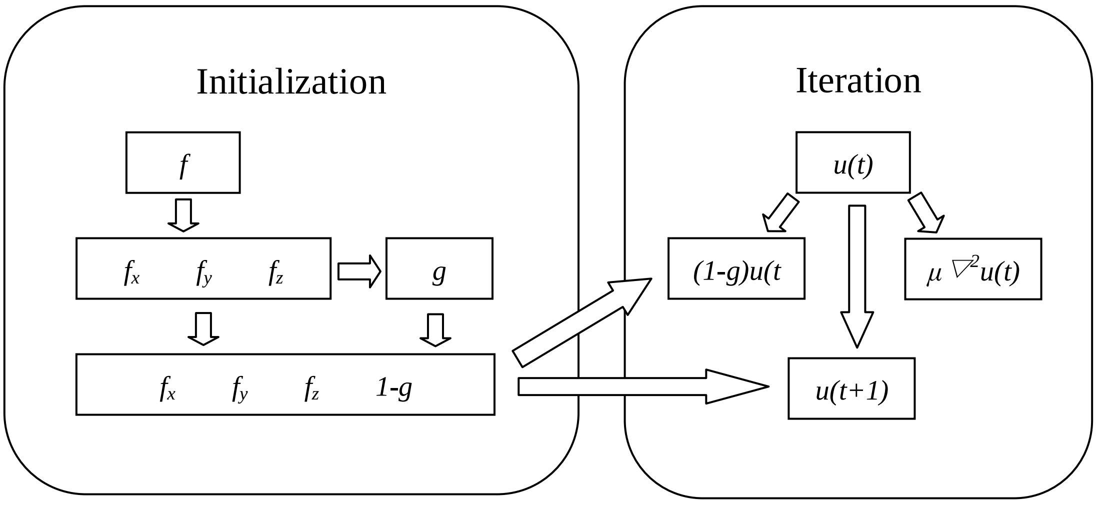

Figure 2.3.1 Parallel computing process for *u*.

3 Experimental Results

3.1 Testing Images

We use some of the data images from the 2010 DIADEM Grand Challenge Qualifiers Round, which is held by Howard Hughes Medical Institute (HHMI). The first image is one of the Neocortical Layer 6 Axons images, which consists of 34 separate mouse neocortical layer 6 axons all contained within the same 6 image stacks. The second image we use is one of the Neuromuscular Projection Fibers images, which consists of 12 separate mouse neuromuscular axonal projection fibers all contained within the same 152 image stacks. Both images are cut to size of $512\times512\times60$ pixels.

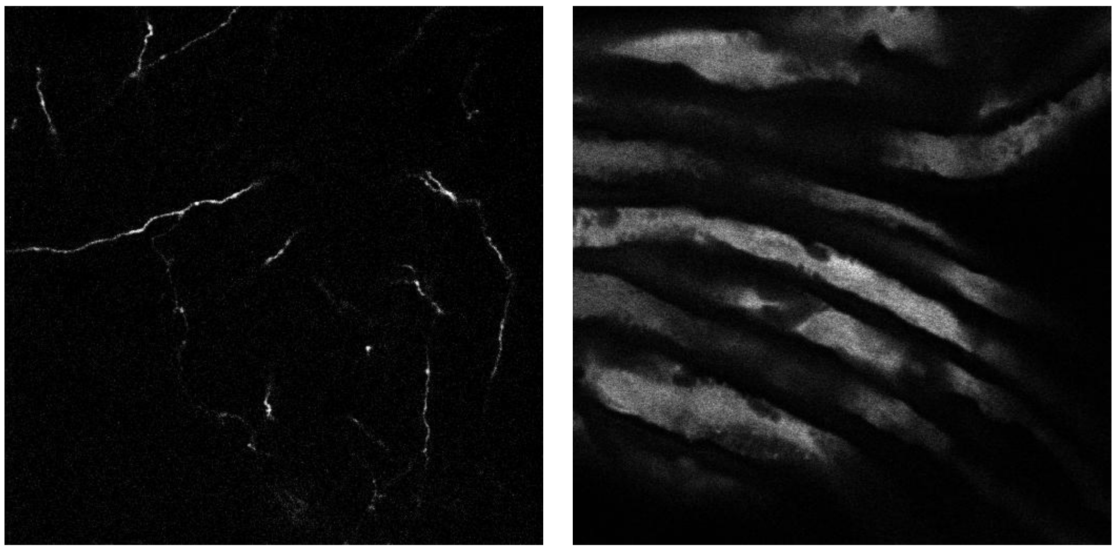

Figure 3.1.1 (a) The 30th frame of Neocortical Layer 6 Axons image; (b) The 30th frame of Neuromuscular Projection Fibers image.

3.2 GVF Force Field

We load the GVF Force Field output files in MATLAB, and plot the vector field in *x* direction and *y* direction of one frame together with the original image of that frame. We zoom into a smaller region of the image to see more details of the GVF force field.

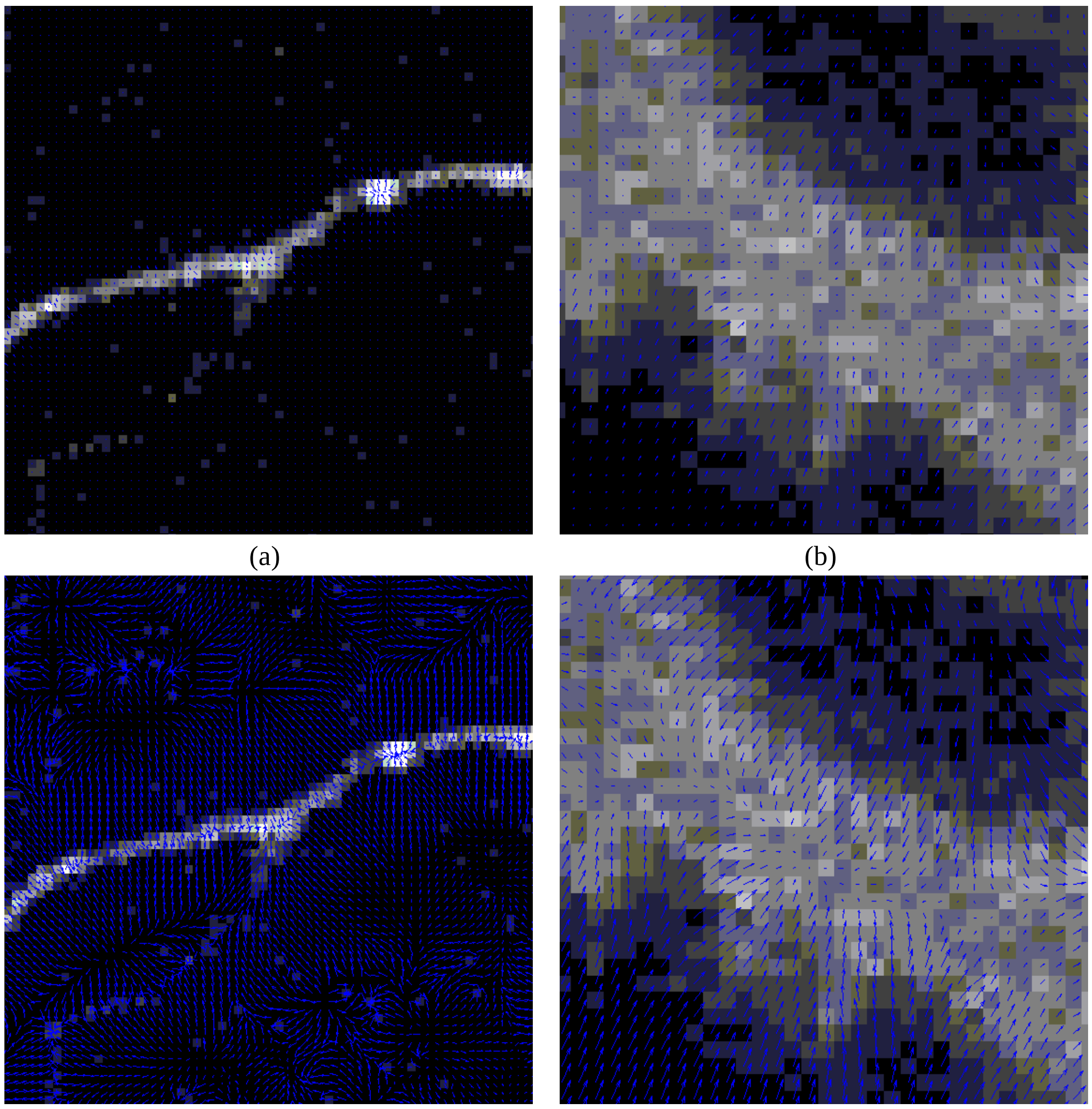

Figure 3.2.1 Part of the 30th frame of the 3D image and the GVF force field. (a) Neocortical Layer 6 Axons image; (b) Neuromuscular Projection Fibers image; (c) Neocortical Layer 6 Axons image and normalized GVF force field; (d) Neuromuscular Projection Fibers image and normalized GVF force field.

We can see that the GVF force field points to the center of axons even in background regions which might be far away from objects.

3.3 Computational Performance

To compare with the computational performance of our GPU implementation of GVF force field, we also implemented a CPU version of this algorithm, which is written in C++ and ITK. We define the time cost of this algorithm as the time period beginning from before the normalization of the edge map image to the end of the last iteration. We run the GPU version and the CPU version on the Neocortical Layer 6 Axons image and the Neuromuscular Projection Fibers image for five times, with the regularization parameter $\mu = 0.2$ for 50 iterations. All operations are performed on the same computer under same conditions. The results are as following.

Neocortical Layer 6 Axons image		Neuromuscular Projection Fibers image	
GPU	CPU	GPU	CPU
12.000	538.968	11.891	451.078
11.875	467.172	11.906	450.656
11.953	465.157	11.922	451.047
12.172	465.890	11.875	450.703
12.172	465.219	11.890	450.844

Form 3.3.1 Computational Performance of GPU and CPU implementation

In average, the GPU implementation is 39.93 times faster than CPU implementation on Neocortical Layer 6 Axons image, and 37.90 times faster on Neuromuscular Projection Fibers image. The superiority of GPU implementation will be more obvious as the size of image increases, but that would require larger device memory of the GPU for enough space to perform the parallel computing.

4 Conclusion and Future Work

In this project, we have developed a framework to compute the GVF force field of a 3D image in a parallel manner. The GVF force field is obtained by an iterative algorithm, which is implemented on GPU with CUDA. We tested our programs on two 3D images from 2010 DIADEM Challenge, and got satisfying results. Our GPU implementation is about 38 ~ 40 times faster than CPU implementation on these two images, and promises to be more efficient on larger images.

The framework of this project consists of two modules, the ITK module and the CUDA module. The CUDA module is the kernel of the algorithm, and the ITK module calls the CUDA module. However, the ITK module only load one edge map image, perform the GVF algorithm, and save the GVF force field information in six files. Since writing GVF files, which takes much more time than the iterative algorithm, might be unnecessary in most applications, more interfaces can be implemented by calling the CUDA module, and these interfaces can be integrated to other programs.

References

[1] C. Xu and J.L. Prince, "Gradient Vector Flow: A New External Force for Snakes," *Proc. IEEE Conf. on Comp. Vis. Patt. Recog. (CVPR)*, Los Alamitos: Comp. Soc. Press, pp. 66-71, June 1997.
[2] C. Xu and J. L. Prince, "Snakes, Shapes, and Gradient Vector Flow,"*IEEE Transactions on Image Processing*, 7(3), pp. 359-369, March 1998.
[3] The DIADEM Challenge: http://www.diademchallenge.org/.
[4] The Insight Toolkit: http://www.itk.org/.
[5] "The ITK Software Guide," Insight Software Consortium.
[6] "NVIDIA CUDA C Programming Guide," NVIDIA Corporation.