A GPU-based Implementation of Multimodal Deformable Image Registration Based on Mutual Information or Bhattacharyya Distance

Release 0.00

Yifei Lou[1], Xun Jia[2], Xuejun Gu[2] and Allen Tannenbaum[1]

May 24, 2011

[1]School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA
[2]Center for Advanced Radiotherapy Technologies and Department of Radiation Oncology, University of California, San Diego, CA, USA

Abstract

This paper describes a multimodal deformable image registration method on the GPU. It is a CUDA-based implementation of a paper by E. D'Agostino *et. al*, "A viscous fluid model for multimodal non-rigid image registration using mutual information" [4]. In addition, we incorporate an alternative metric as opposed to mutual information, called Bhattacharyya Distance, in the recent work of [9]. This paper is accompanied with the source code, input data, parameters and output data that the authors used for validating the algorithm.

Latest version available at the Insight Journal [http://hdl.handle.net/10380/1338]

Contents

Deformable image registration (DIR) is one of the major problems in medical image processing, such as dose calculation [10], treatment planning [19] and scatter removal of cone beam CT (CBCT) [14]. In many clinical scenarios, it is required to precisely establish a pixel-to-pixel correspondence between two images. For instance, registration of a CT image to MRI of a patient taken at different time can provide complementary diagnostic information. For applications as such, since the deformation of the patient anatomy cannot be represented by a rigid transform, DIR is almost the sole means to establish this mapping. In solving a DIR problem, a metric quantifying the similarity of two images is usually first defined. One then seeks for deformation vector fields so that one of the two images after deformation matches the other. DIR can be generally categorized into intra-modality and inter-modality, or multi-modality. While intra-modality DIR can be easily handled by conventional intensity-based methods [7, 16], multimodality DIR problems are still far from being satisfactory. Yet, since different imaging modalities usually provide their unique angles to reveal patient anatomy and delineate microscopic disease, multimodality registration plays a key role to combine the information from multiple modalities to facilitate diagnostics and treatment of a certain disease.

One straightforward approach to perform multimodal DIR is to first map the intensity of one image to the other and then apply intra-modality registration algorithms [6, 13]. Nonetheless, this approach is very difficult either, since it involves the precise intensity mapping between two modalities. In this paper, we adopt mutual information (MI) as the similarity metric. Such a metric can assess the similarity of images from the co-occurrence of intensities in both images as reflected by their joint histogram. It is widely used for multimodal image registration since the pioneer work of Voila and Wells [18] and independently by Maes *et. al* [11]. In addition, we consider another metric, Bhattacharyya Distance (BD) as proposed in [9], to solve the multimodal DIR problems.

The DIR algorithm is computationally expensive considering that the images to be registered are usually of 3 dimension. To meet the clinical requirement for the processing time, we have implemented our algorithm on a graphic processing uint (GPU) platform. Recently, GPUs have offered us a promising prospect of increasing efficiency for heavy duty tasks, such as deformable image registration [5], image reconstruction [8] and treatment plan optimization [12]. Though GPU has been used for multimodal image registration, majority of those available works focus on rigid and/or Demons-types of methods. To the best of our knowledge, the only two existing works on GPU-based multimodal DIR are [17, 15]. However, the former uses only 2D textures and is implemented with OpenGL and GLSL. The the latter is a B-spline based method and thus it is unable to handle large deformation.

The rest of the paper is organized as follows. A registration framework using the viscous fluid model [4] is reviewed in Section 1 with two choices of similarity metrics: MI and BD. Section 2 discusses the fine details of the GPU implementation, including the use of texture memory and how to efficiently compute the joint histogram and 3D convolution. Section 3 serves as a user's guide, followed by experiments in Section 4. The conclusion is then given in Section 5.

1 The Viscous Fluid Model

In this section, we briefly review the viscous fluid model [4] as a framework for multimodal DIR. The work of [4] is based on mutual information, while we include a new metric, the Bhattacharyya Distance, as proposed in [9].

Mutual information $\mathcal{M}$ between any two images $I_1(\vec{x})$ and $I_2(\vec{x})$ is defined as

$$\mathcal{M}(I_1, I_2) = \iint p(i_1, i_2) \log \frac{p(i_1, i_2)}{p(i_1)p(i_2)} \mathrm{d}i_1 \mathrm{d}i_2 \ , \tag{1}$$

where i_1 and i_2 are the image intensity values and $p(i_1,i_2)$ is the joint intensity distribution of I_1 and I_2 in the overlap region V. $p(i_1), p(i_2)$ are its marginal distributions, *e.g.*

$$p(i_1) = \int p(i_1,i_2)\mathrm{d}i_2 \ . \tag{2}$$

Based on two images $I_1(\vec{x})$ and $I_2(\vec{x})$, $p(i_1,i_2)$ can be obtained with the help of $\delta-$function as

$$\begin{aligned} p(i_1,i_2) &= \frac{1}{V}\int_V \delta(i_1 - I_1(\vec{x}), i_2 - I_2(\vec{x}))\mathrm{d}\vec{x} \ , \\ &\simeq \frac{1}{V}\int_V G_\sigma(i_1 - I_1(\vec{x}), i_2 - I_2(\vec{x}))\mathrm{d}\vec{x} \ , \end{aligned} \tag{3}$$

where G_σ is a Parzen windowing kernel taken to be a Gaussian function of a standard deviation σ.

As for another metric used to solve the multimodal DIR problem [9], Bhattacharyya Distance is defined as

$$\mathcal{B}(I_1,I_2) = \iint \sqrt{p(i_1,i_2)p(i_1)p(i_2)}\mathrm{d}i_1\mathrm{d}i_2 \ . \tag{4}$$

For the registration problem, we are seeking for a deformation field $\vec{u}(\vec{x})$, such that $I_1(\vec{x}-\vec{u}(\vec{x})) = I_2(\vec{x})$. As such, I_1, termed as a template or moving image, is deformed towards a target or fixed image I_2 by the deformation field $\vec{u}(\vec{x})$, *i.e.,* mapping a position $\vec{x}$ in I_2 onto the corresponding point $\vec{x}-\vec{u}$ in I_1. The viscous fluid model [4] assumes that the deformation is governed by the Navier-Stokes equation of viscous fluid motion. Mathematically it is expressed as

$$\nabla^2\vec{v} + \vec{\nabla}(\vec{\nabla}\cdot\vec{v}) + \vec{F}(\vec{x},\vec{u}) = 0 \ , \tag{5}$$

where $\vec{v}(\vec{x},t)$ is the deformation velocity, which is related to $\vec{u}$ by $\vec{v} = \frac{\mathrm{d}\vec{u}}{\mathrm{d}t}$ and $\vec{F}(\vec{x},\vec{u})$ is the force field that drives the deformation in the appropriate direction. Here the force is considered to maximize mutual information between $I_1(\vec{x}-\vec{u})$ and $I_2(\vec{x})$, which is equivalent to the gradient of $\mathcal{M}(I_1(\vec{x}-\vec{u}), I_2(\vec{x}))$ with respect to $\vec{u}$. The force $\vec{F}$ is formulated as

$$\vec{F}(\vec{x},\vec{u}) = \frac{\alpha}{V}\left[\frac{\partial G_\sigma}{\partial i_1} L(i_1,i_2;\vec{u})\right](I_1(\vec{x}-\vec{u}), I_2(\vec{x}))\nabla I_1(\vec{x}-\vec{u}) \ , \tag{6}$$

with a constant α and the term L corresponding to the mutual information is defined as

$$L_M(i_1,i_2;\vec{u}) = 1 + \log\frac{p(i_1,i_2;\vec{u})}{p(i_1;\vec{u})p(i_2)} \ . \tag{7}$$

As for the Bhattacharyya distance, the force is to minimize BD, and hence has the following expression,

$$L_B(i_1,i_2;\vec{u}) = -\sqrt{\frac{p(i_1;\vec{u})p(i_2)}{p(i_1,i_2;\vec{u})}} - \int\sqrt{\frac{p(i_2)p(i_1,i_2;\vec{u})}{p(i_1;\vec{u})}}\mathrm{d}i_2 \ . \tag{8}$$

To solve the Navier-Stokes equation (5), successive over relaxation (SOR) is applied in [3]. However, this approach is very computationally expensive. In this paper, we instead adopt the approach of convolution of the force field with 3D Gaussian kernel Φ_s as an approaximation to the solution [2, 4]. In summary, deformation field $\vec{u}$ at iteration k is given by

$$\vec{v}^k = \Phi_s\vec{F}^k \ , \tag{9}$$

$$\vec{R} = \vec{v}^k - \sum_{i=1}^{3} v_i^k\left[\frac{\partial u^k}{\partial x_i}\right] \ , \tag{10}$$

$$\vec{u}^{k+1} = \vec{u}^k + \delta_t\vec{R} \ , \tag{11}$$

where $\vec{x} = (x_1, x_2, x_3)^T, \vec{v} = (v_1, v_2, v_3)^T$ and the time step δ_t is chosen adaptively during iteration

$$\delta_t = \frac{\delta_u}{\max \|\vec{R}\|}, \tag{12}$$

with δ_u (in voxels) the maximal voxel displacement that is allowed in one iteration.

To preserve the topology of the deformed template, *re-gridding* is performed when there exists $\vec{x}$ such that the Jacobian of $\vec{x} - \vec{u}$ becomes less than a certain threshold, for instance, 0.5. Regridding is to set the current deformed image as a new template and the incremental displacement field to zero. In the end, the total deformation is the concatenation of the incremental deformation fields associated with each propagated template.

We have also employed a multi-resolution technique to increase computation efficiency and avoid local minima especially for large motion fields. It has been known that, the convergence rate of an iterative approach is usually deteriorated when a very fine grid size is used [1]. Moreover, fine grid also implies a large number of unknown variables, significantly increasing the size of the computation task. Therefore, a hierarchy of resolution is performed. In particular, we construct the image pyramid by downsampling the images by a factor of 2 in each dimension. We first perform registration at a low resolution grid. The deformation field obtained in the lower resolution is interpolated as the initial values for the higher resolution computation. At each resolution level, the registration algorithm consists of the following key steps:

1. warping of the input image according to the displacement field, *i.e.,* $I(\vec{x}) = I_1(\vec{x} - \vec{u})$;
2. computation of the joint histogram (3) of the warped image I and the fixed image I_2;
3. computation of the image gradient;
4. 3D Gaussian convolution;
5. updating the displacement field according to (9)-(11);
6. re-gridding if necessary.

Fig. 1 summarizes our multimodal DIR algorithm.

2 GPU Implementation

Computer graphic cards, such as the NVIDIA GeForce series and the GTX series, are conventionally used for display purpose on desktop computers. It typically consists of 32-240 scalar processor units and 256 MB to 1 GB memory. Recently, NVIDIA introduces special GPUs solely dedicated for scientific computing, such as the Tesla C1060 card that is used in this paper. Such a GPU card has a total number of 240 processor cores (grouped into 30 multiprocessors with 8 cores each), each with a clock speed of 1.3 GHz. Though the clock speed for each processor is lower than a typical CPU, the overall computational power is higher due to the large amount of processors available on a single GPU. The card is also equipped with 4 GB DDR3 memory which is shared by all processor cores. Our DIR algorithm is coded under the Compute Unified Device Architecture (CUDA) platform developed by NVIDIA, which enables us to extend the C language to program an NVIDIA GPU [1].

[1] http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/CUDA_C_Programming_Guide.pdf

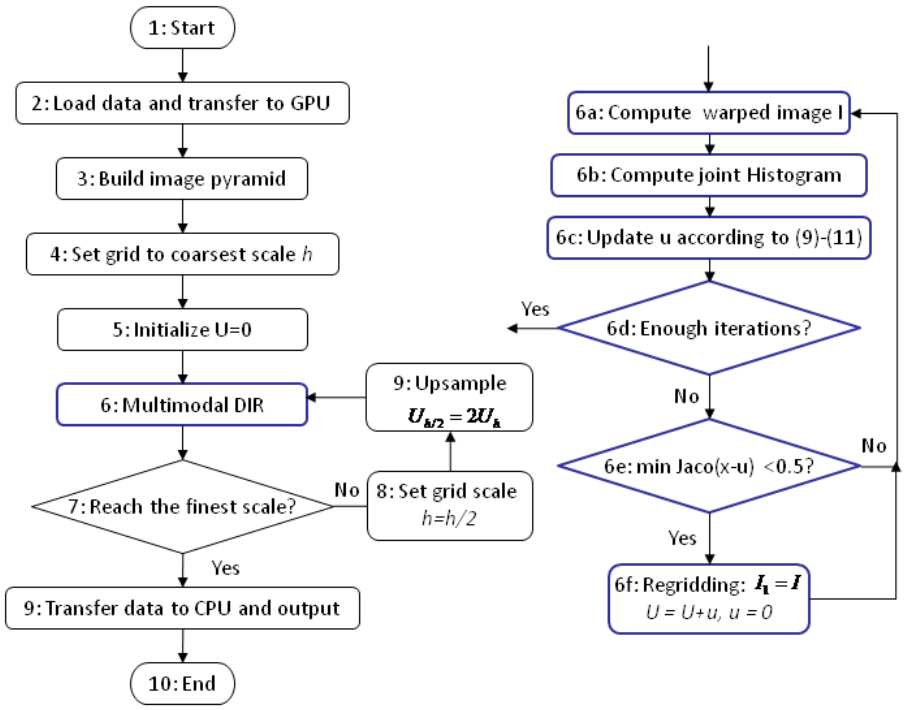

Figure 1: Flow chart of the multimodal DIR algorithm.

In fact, a number of computationally intensive tasks involved in our algorithm share a common feature, *i.e.* applying a single operation to different part of data elements. For this type of tasks, it is straightforward to accomplish them in a data-parallel fashion, namely having all GPU threads running the same operation, one for a given subset of the data. Such a parallel manner is particularly suitable for the SIMD (single instruction multiple data) structure of a GPU and high computation efficiency can be therefore achieved. In particular, Step 3 and step 5 are accomplished by simply launching one thread on GPU for each pixel to compute its gradient and to update its displacement field. Yet, other steps are not quite easily parallelizable, which will be addressed in detail in the following sections.

2.1 Using Texture Memory

Texture memory is a special type of memory space in GPU, which supports fast data access due to its associated memory cache. It also supports hardware linear interpolation in up to 3 dimensions, which greatly facilitates our computation. The application of texture memory in our algorithm is mainly on the following two contexts.

First, when warping the template image I_1 into a new image I according to the deformation field $\vec{u}$, *i.e.* computing $I(\vec{x}) = I_1(\vec{x} - \vec{u}(\vec{x}))$, new intensity values of I are found through linear interpolation. For this purpose, we first copy the data I_1 into an opaque CUDA memory of type CUDA Array and binding it with appropriate texture memory references. To fetch the data $I_1(\vec{x} - \vec{u}(\vec{x}))$ and assign it to $I(\vec{x})$, where $\vec{x} - \vec{u}(\vec{x})$ is not necessarily on a regular grid point $\vec{x}$, we invoke the built-in CUDA function **tex3D**, which returns the interpolated image intensity on the desired coordinates $\vec{x} - \vec{u}(\vec{x})$.

Similarly, when it comes to regridding, we also create 3D textures for deformation fields as well, since we need to compute the new vector field according to the following expression,

$$\vec{U}^{new} = \vec{U}^{old}(\vec{x} - \vec{u}) \tag{13}$$

where $\vec{u}$ is the deformation from the target to the current template and $\vec{U}$ is to the original one. Apparently, trilinear interpolation is necessary in this update, and using the texture memory can greatly simply coding while ensuring efficiency.

2.2 Histogram Computation

Computing the joint histogram is another frequently encountered task during the registration process. Consider two images $I_1(\vec{x})$ and $I_2(\vec{x})$, whose intensities ranges in a known interval $[a,b]$, for instance $[0,255]$. Let us partition the interval $[a,b]$ into a set of N bins of equal size. A joint histogram is simply a two dimensional array of size $N \times N$ defined according to Eq. (3). In a CPU implementation, it is straightforward to compute the joint histogram according to Eq. (3), where we sequentially loop over all the coordinate $\vec{x}$ and increase $p(i_1,i_2)$ by unity at $i_1 = I_1(\vec{x})$ and $i_2 = I_2(\vec{x})$ followed by a Parzen window smoothing. However, it is difficult to parallelize this sequential process on GPU. If we were simply having each GPU thread responsible for one coordinate $\vec{x}$ and update $p(i_1,i_2)$, we would encounter a memory conflict problem due to the potential simultaneous update of a same memory address holding $p(i_1,i_2)$ from different GPU threads. To ensure the correctness, one thread will have to wait for another thread whenever this conflict occurs, thus significantly reducing the computational efficiency.

To overcome this difficulty, we compute the joint histogram with the help of a GPU library called THRUST [2]. Let us first lable the 2D histogram bins (i_1,i_2) by a 1D index $i = i_1 * N + i_2$ and create a vector $P =$

[2] http://code.google.com/p/thrust/wiki/QuickStartGuide

$(I_1(\vec{x}), I_2(\vec{x})), \forall \vec{x}$ of dimension $2 \times N_p$, where N_p is the number of voxels $\vec{x}$ on two images. A thrust function is then used to transform each pair in P to the corresponding histogram indices, yielding a vector P_I of length N_p with entries

$$\text{histogram index}(\vec{x}) = \text{ bin index of } I_1(\vec{x}) \times \#\text{ of histogram bins } + \text{ bin index of } I_2(\vec{x}) \ .$$

We then sort the histogram index vector P_I using a sorting function *thrust::sort* from THRUST, so that P_I is in an ascending order. Finally, we compute a histogram $p(i)$ by using a THRUST function to count the number of entries in P_I that equal to i. Note $p(i)$ is exactly the joint histogram to be computed after transforming the linear index i to its equivalent row and column subscripts i_1 and i_2. During this process, the key operations invoking THRUST functions are executed on GPU in parallel, ensuring the computational efficiency.

Once a joint histogram $p(i_1, i_2)$ is obtained, the marginal distributions can be computed by summing $p(i_1, i_2)$ over i_1 or i_2 coordinate. To perform this computation on GPU, we employ a technique of parallel reduction [3]. Its computational complexity is $O(\log_2 N)$, as opposed to $O(N)$ in a sequential implementation of summing over each coordinate i_1 or i_2. This also improves the computation efficiency considerably.

2.3 3D Convolution

Convolution is another key operation in the registration problem. It is often taken for granted to use FFT for convolution. However, it is not optimal in the case of smoothing kernel of size much smaller than the image size. For Gaussian type of kernels, which can be separated to be the product of two Gaussian kernels along two directions, 2D convolution can be carried out by the combination of the one-dimensional convolution of the image vertically and then horizontally afterwards. It's tested that using *convolutionSeparable* [4] outperforms twice a purely texture-based implementation of the same algorithm running on the same hardware. We adopt the same technique of *convolutionSeparable* for computing the convolution of a 3D separable Gaussian kernel.

3 User's Guide

The user needs to have CUDA toolkit properly installed. Please refer to the following page for details

http://developer.nvidia.com/cuda-toolkit-32-downloads

It is assumed that the users are familiar with NVIDIA GPU development environment. They should have background on how to compile and execute CUDA code on an NVIDIA platform. For more information on this category, users are suggested to consult NVIDIA development documents such as http://www.nvidia.com/object/cuda-home-new.html.

After downloading the package, there should be the following files under the target directory,

```
*.cu            ->  CUDA source codes

global.h        ->  C source code defining variables etc

Makefile        -> Sample makefile for compiling and linking
```

[3] http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/reduction/doc/reduction.pdf
[4] http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/convolutionSeparable/doc/convolutionSep

To run the registration code, please follow these steps:

1. Configure Makefile according to the computer architecture. Then run "make" to compile and link this source code. You may also use NVIDIA NVCC to do so.

2. Execute the executable file. For example if the executable file is named tfr, you may run ./tfr This will perform the registration on the case specified in the sample case. Information during the registration will be displayed on screen, and simulation results will be written to an output files, which is raw data of float type with no headers.

The format of the input data is *float*. The input parameters are specified in *global.h*. Note, the code should be recompiled each time the input parameters are changed. Specifically you need to adjust the following parameters in the first portion of global.h. It is not recommended to change other parameters unlisted here unless you are familiar with the mathematical model.

```
NBLOCKX
            --- the leading dimension of the 2d GPU thread grid
NTHREAD_PER_BLOCK
            --- number of GPU threads per block
DEVICENUMBER
            --- GPU device number to be used
NX0, NY0, NZ0
            --- 3D image dimension
NSCALE
            --- total number of scales for mutli-resolution

MAX_ITER
            --- max number of iterations
METHOD
            --- 1 for Bhattacharyya
            --- 2 for mutual information
```

4 Experiments

The nVidia Tesla C1060 GPU is used in the present work, which contains 4GB off-chip device memory and 16KB of on-chip shared memory. The GPU supports a maximum of 512 threads per thread block.

We first demonstrate our algorithm on a real patient case of registering a CT image to cone beam CT (CBCT). Both CT and CBCT are of size $256 \times 256 \times 68$, while we only visualize one 2D slice in Figure 2. The registration results are illustrated in Figure 3, which shows that MI and BD have the same performance. It takes about 2.76 seconds to do 30 iterations on two multiresolution levels to get reasonable results.

It is also tested on a sample dataset that we provide in this submission. It consists of a T2 image and a DTI baseline image as shown in Fig. 4. The data files are from the NA-MIC site [5] with two preprocessing steps: (1) rigid registration of DTI to T2 and (2) cut the region of interest to be $256 \times 256 \times 64$. BD seems to

[5] http://www.na-mic.org/Wiki/index.php/Projects:RegistrationLibrary:RegLib_C29

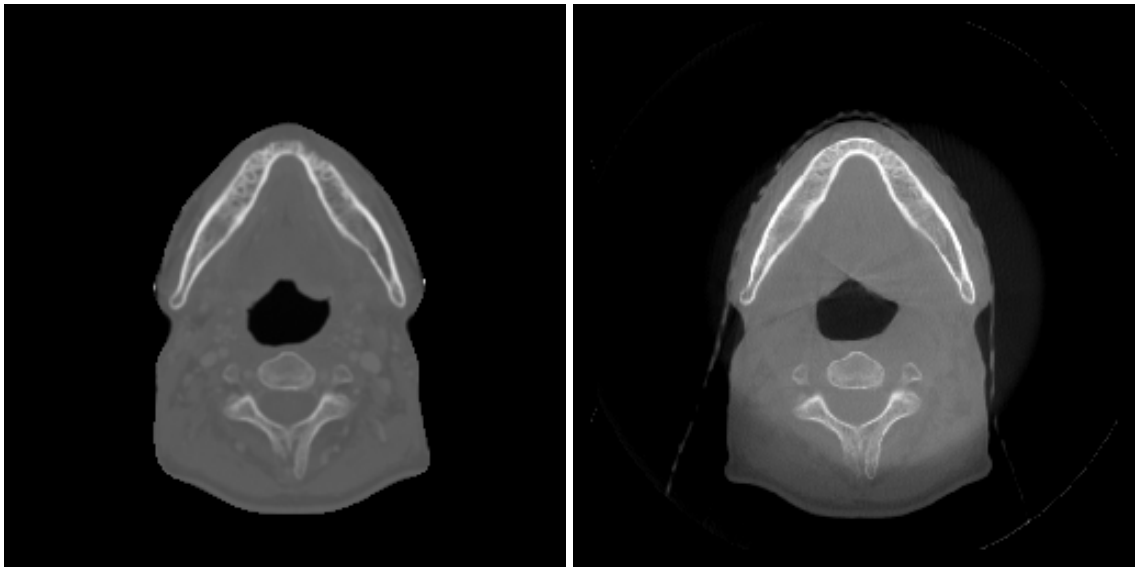

Figure 2: Left: CT (template) and right: CBCT (target).

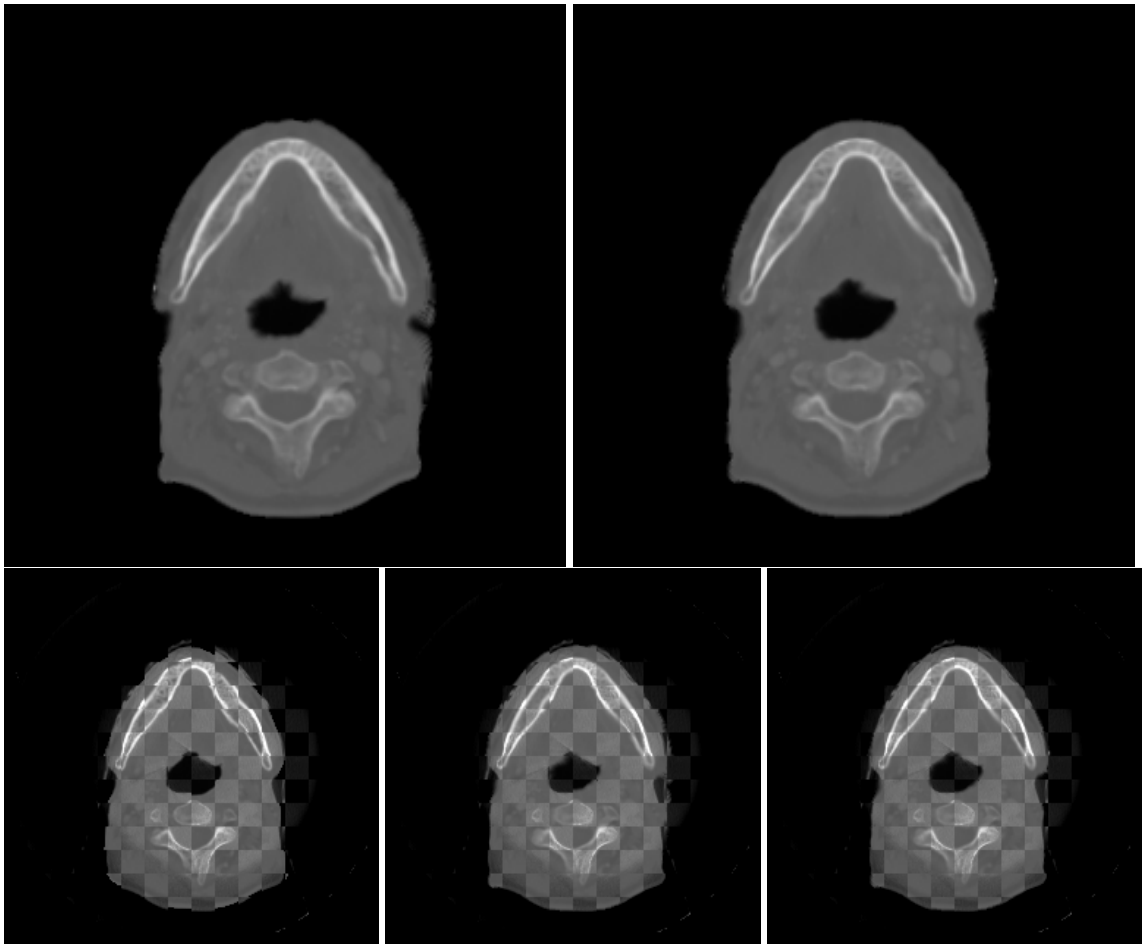

Figure 3: Top are the registered CT produced by BD(left) and MI (right). The checkerboard comparison is on the bottom with CT+CBCT (left), registered CT by BD + CBCT (middle) and registered CT by MI + CBCT.

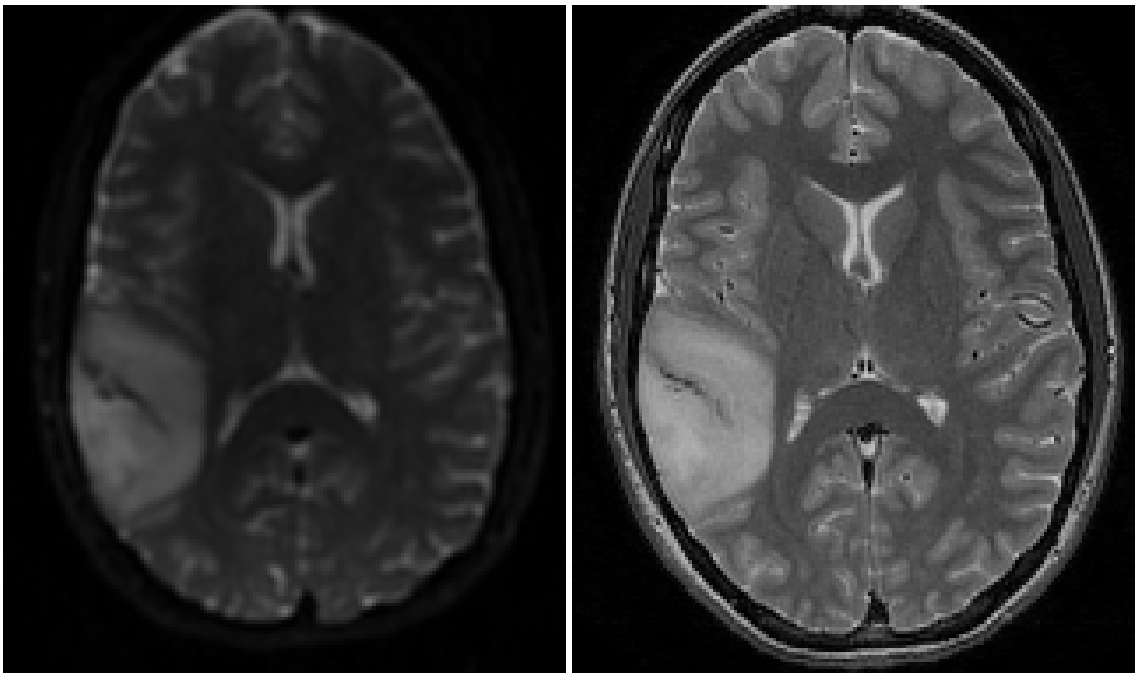

Figure 4: Left: DTI baseline image (template) and right: T2 image (target).

produce artifacts on the top part of the registered image as shown in Figure 5. It is mathematically correct since it strives to match the boundary of DTI baseline image to the outer white boundary in the T2 image. It can be improved by adding local constraints to the force term, which is beyond the scope of this paper.

5 Conclusion

In this paper, we have present a GPU-based implementation of 3D multimodal deformable image registration. It takes 2.76 seconds for a pair of images with size $256 \times 256 \times 68$ to do 20 iterations on two multiresolution levles. It meets clinical requirements in terms of both time and accuracy. We have provide the users with source codes and sample dataset to test our algorithm. The future work is to visualize the registration process during iteration in order to allow experts to quickly check the plausibility of the registration and stop properly.

Acknowledgments

This work was supported in part by grants from National Science Foundation (NSF), AFOSR, ARO as well as by a grant from NIH (NAC P41 RR-13218) through Brigham and Women's Hospital. This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149. Information on the National Centers for Biomedical Computing can be obtained from `http://nihroadmap.nih.gov/bioinformatics`. We would like to thank NVIDIA for providing GPU cards for this project.

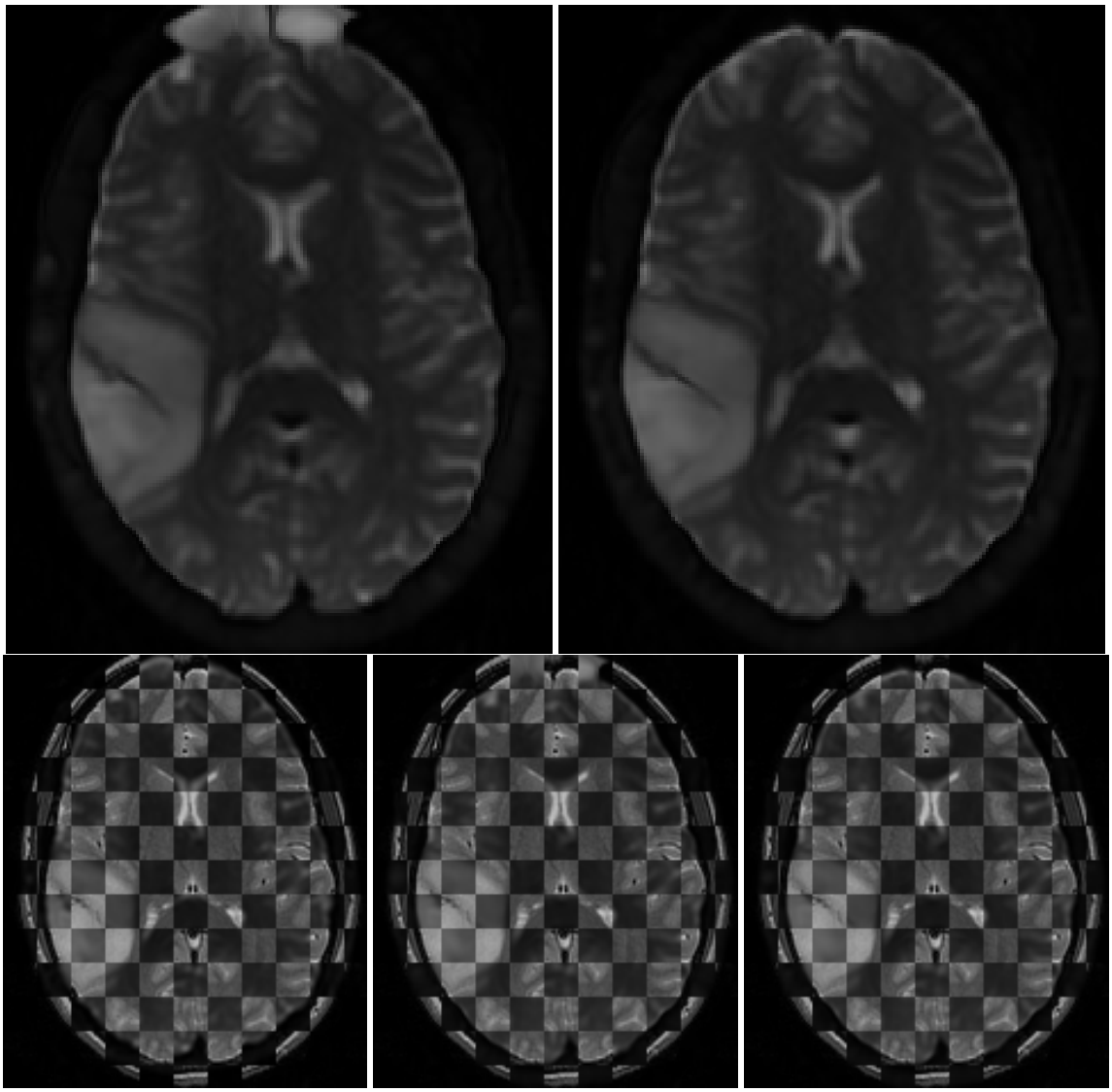

Figure 5: Top: deformed DTI via BD (left) and MI (right). Bottom is the comparison using checkerboard. From left to right: DTI + T2, deformed DTI by BD + T2 and deformed DTI by MI + T2.

References

[1] A. Brandt. Multiscale scientific computation: review 2001. In T. J. Barth, T. F. Chan, and R. Haimes, editors, *Multiscale and multiresolution Methods: thoery and applications*, pages 1–96. Springer Verlag, Heidelberg, 2001. 1

[2] M. Bro-Nielsen and C. Gramkow. Fast fluid registration of medical images. In *Proceedings of the 4th International Conference on Visualization in Biomedical Computing*, pages 267–276, 1996. 1

[3] G. E. Christensen, R. D. Rabbitt, and M. I. Miller. deformable templates using large deformation kinetics. *IEEE Trans. on Image Process.*, 5(10):1435–1447, 1996. 1

[4] E. D'Agostino, F. Maes, D. Vandermeulen, and P. Suetens. A viscous fluid model for multimodal non-rigid image registration using mutual information. In *Proceedings of the 5th International Conference on Medical Image Computing and Computer-Assisted Intervention-Part II*, MICCAI, pages 541–548, 2002. (document), 1, 1, 1

[5] X. Gu, H. Pan, Y. Liang, R. Castillo, D. Yang, D. J. Choi, E. Castillo, A. Majumdar, T. Guerrero, and S. B. Jiang. Implementation and evaluation of various demons deformable image registration algorithms on a GPU. *Phys. Med. Biol.*, 55(1):207–219, 2010. (document)

[6] A. Guimond, A. Roche, N. Ayache, and J. Meunier. Three-dimensional multimodal brain warping using the demons algorithm and adaptive intensity corrections. *IEEE Trans. on Med. Imaging*, 20(1):58–69, 2001. (document)

[7] B. K. P. Horn and B. G. Schunck. Determining optical flow. *ARTIFICAL INTELLIGENCE*, 17:185–203, 1981. (document)

[8] X. Jia, Y. Lou, R. Li, W. Y. Song, and S. B. Jiang. GPU-based fast cone beam CT reconstruction from undersampled and noisy projection data via total variation. *Med. Phys.*, 37(4):1757–1760, 2010. (document)

[9] Y. Lou and A. Tannenbaum. Multimodal deformable image registration via the bhattacharyya distance. *submitted to IEEE Trans. on Image Process.*, 2011. (document), 1, 1

[10] M. Dirkx M. Van Zijtvetd and B. Heijmen. Correction of conebeam ct values using a planning ct for derivation of the "dose of the day". *Radiother Oncol.*, 85(2):195–200, 2007. (document)

[11] F. Maes, A. Collignon, D. Vandermeulen, G. Marchal, and P. Suetens. Multimodality image registration by maximization of mutual information. *IEEE trans. on Med. Imaging*, 16:187–198, 1997. (document)

[12] C. Men, X. Jia, and S. B. Jiang. GPU-based ultra-fast direct aperture optimization for online adaptive radiation therapy. *Phys. Med. Biol.*, 55(15):4309–4319, 2010. (document)

[13] S. Nithiananthan, S. Schafer, A. Uneri, D. J. Mirota, J. W. Stayman, W. Zbijewski, K. K. Brock, M.J. Daly, H. Chan, J. C. Irish, and J.H. Siewerdsen. Demons deformable registration of ct and cone-beam ct using an iterative intensity matching approach. *Med. Phys.*, 38(4):1785–1798, 2011. (document)

[14] T.Y. Niu, M.S. Sun, J. Star-Lack, H.W. Gao, Q.Y. Fan, and L. Zhu. Shading correction for on-board cone-beam ct in radiation therapy using planning mdct images. *Med. Phys.* (document)

[15] V. Saxena and J. Rohrerand L. Gong. A parallel GPU algorithm for mutual information based 3d nonrigid image registration. In *Proceedings of the 16th international Euro-Par conference on Parallel processing: Part II*, Euro-Par'10, pages 223–234, 2010. (document)

[16] J. P. Thirion. Image matching as a diffusion process: an analogy with Maxwell demons. *Med. Image Anal.*, 2:243–260, 1998. (document)

[17] G. Vetter, C. Guetter anc C. Xu, and R. Westermann. Non-rigid multi-modal registration on the GPU. In *Proceedings of SPIE*, volume 6512, 2007. (document)

[18] P. Viola and W. M. Wells III. Alignment by maximization of mutual information. *Int. J. Comput. Vision*, 24:137–154, September 1997. (document)

[19] Q.J. Wu, D. Thongphiew, Z. Wang, B. Mathayomchan, V. Chankong, S. Yoo, W.R. Lee, and F.F. Yin. On-line re-optimization of prostate IMRT plans for adaptive radiation therapy. *Phys. Med. Biol.* (document)