Real-Time Intensity-Based Rigid 2D-3D Medical Image Registration Using RapidMind Multi-Core Development Platform

Lin Xu and Justin W.L. Wan

***Abstract*— In this paper, we present an efficient intensity-based rigid 2D-3D image registration method. We implement the algorithm using the RapidMind Multi-core Development Platform[1] to exploit the highly parallel multi-core architecture of graphics processing units (GPUs). We use a ray casting algorithm to generate the digitally reconstructed radiographs (DRRs) on GPUs and efficiently reduce the complexity of DRR construction. The registration optimization problem is solved by the Gauss-Newton method. To fully exploit the multi-core parallelism, we implement almost the entire registration process in parallel by RapidMind. We also discuss the RapidMind implementation of the major computation steps. Numerical results are presented to demonstrate the efficiency of our method.**

I. Introduction

In image-guided procedures such as radiation therapies [1] and computer-assisted surgeries [2], intensity-based rigid 2D-3D medical image registration is often used. Although aims are different depending on different applications, the 2D-3D registration processes are similar. For instance, 3D datasets (e.g. CT Volume) are often collected for pre-operative planning. During surgeries, 2D portal image slices (e.g. X-Ray image) are collected in real time and aligned with DRRs generated from the pre-acquired 3D datasets. By registering the 3D volume with the 2D portal slice, more useful information can be obtained.

Although acquiring 2D portal images can be efficient, DRR generation and optimization processes are very computationally intensive and still require significant amount of time. In this paper, we employ multi-core processing to accelerate the entire registration process in order to achieve real-time performance.

Hardware acceleration methods using hundreds or even thousands of processors have been studied and employed for image registration (e.g. [3]). Though it has been reported with success in many cases, the physical size of large parallel computers limits their use in clinical applications. Multi-core processing has become an attractive alternative in the past few years. The multi-core architectures exploit hardware parallelism on the chipsets while maintaining small form factor. Ohara et al. [4] implement mutual information-based multiresolution algorithms on the Cell Broadband Engines. However, it is only for 3D-3D registration, and random sampling strategy is used due to the memory limit of Cell Broadband Engines. Chisu [5] investigates both GPU and CPU acceleration techniques for 2D-3D rigid image registration, applies GPU for solving the optimization registration problem, but still uses CPU for the DRR computation. Based on our experiments, the CPU time for DRR computation is a significant part in the registration process; see Section IV. Hence, an efficient GPU computation for DRR becomes crucial for optimal real-time performance.

The contribution of this paper is to develop a real-time rigid image registration by using the RapidMind Multi-Core Development Platform (RapidMind) on GPUs. In contrast with the previous work [4], [5], our real-time image registration performs almost the entire 2D-3D registration process using GPUs to fully exploit the multi-core parallel efficiency.

The paper is structured as follows. In Section II, we give an overview of 2D-3D medical image registration, and then discuss how the DRRs are constructed and how the optimization problem of image registration is solved. In Section III, we explain how the registration method is implemented using RapidMind for efficient processing on GPUs. Numerical results as well as comparisons between RapidMind and regular C++ code are presented in Section IV. Finally, concluding remarks are given in Section V.

II. Intensity-based Rigid 2D-3D Registration

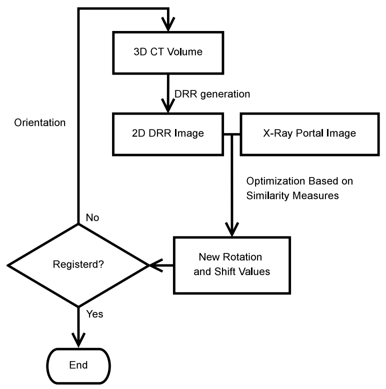

Fig. 1. Iteration scheme of 2D-3D medical image registration

Fig. 1 shows a schematic of 2D-3D medical image registration procedure. The goal is to rotate and translate the 3D image volume so that its projection to a 2D plane will be the same (or very close) to the given portal image. More precisely, let $\mathbf{s} = (\theta_x, \theta_y, \theta_z, \tau_x, \tau_y, \tau_z)^T$ be the parameters

This work was supported by Natural Sciences and Engineering Research Council of Canada.

Lin Xu is with David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada l8xu@uwaterloo.ca

Justin W.L. Wan is with David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada jwlwan@uwaterloo.ca

[1]http://www.rapidmind.com

of the 3D rigid body motion where θ_x, θ_y, and θ_z are the rotation angles in the three axes, and τ_x, τ_y, and τ_z are the translations in the x-, y-, and z- directions, respectively. The rotated and translated 3D volume is projected onto a plane to obtain a 2D image $f = (f_{i,j})$, where the subscript (i, j) denotes the pixel location. This 2D image is usually known as the DRR. In general, $f = f(\mathbf{s})$ depends on the rigid body motion $\mathbf{s}$ of the 3D volume. Let $g = (g_{i,j})$ be the given portal image. Then the 2D-3D registration problem can be formulated as an optimization problem:

$$\min_{\mathbf{s}} \sum_{i,j} (f(\mathbf{s})_{i,j} - g_{i,j})^2. \tag{1}$$

Hence, we want to match the DRR with the portal image by appropriately rotating and translating the 3D volume. Besides sum of squared differences similarity measure, other common choices include sum of absolute differences, correlation coefficients, and mutual information [6].

There are two major steps in the process: 1) the construction of DRR and 2) the solution of the optimization problem, which also needs step 1) in this procedure. They are described in the following sections.

A. Digitally Reconstructed Radiographs

The DRR is obtained from perspective projecting the 3D image volume onto a given plane relative to a ray source; see Fig. 2. In all of our experiments, the 3D image volume, the image plane, and the X-Ray source are aligned in the z-direction for simplicity. Moreover, we assume the image plane is perpendicular to z-direction, and the center of the 3D volume data set is located in the origin. The positions of the ray source and the projection plane are chosen to be $2.5 \times L_z$ and $1.5 \times L_z$, respectively, where L_z is the length of the 3D image volume in the z direction. These positions can be changed by the users.

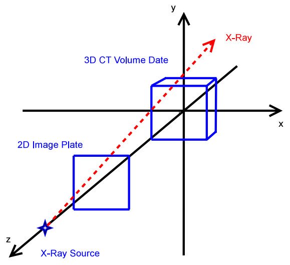

Fig. 2. Projection of the 3D image volume onto a 2D plane

Volume rendering techniques such as ray casting [7] and shear warp factorization [8] have been used. However, shear warp factorization normally makes artifacts in the resulting DRRs and thus is not suitable for accurate image registration. In comparison, ray casting is simple and amenable to highly parallel architecture of GPUs. Hence, we choose a ray casting method for volume rendering.

In ray casting as shown in Fig. 3, a light ray is passing from the source through one of the pixels on the DRR image. This ray will eventually hit the 3D image volume. In principle, if we neglect the effect of attenuation for simplicity, the pixel value on the DRR image should be given by the accumulated intensities of the 3D image volume along the ray. However, this would be quite expensive to determine the length of the ray segment inside each voxel. Instead, we only select m sample points on the ray where m is chosen proportional to the size of the 3D volume data; e.g. $m = 3 \times L_z$ seems to work well for relatively small rotations and translations. The locations of the m samples are the intercepts of the ray with m concentric spheres with different radii covering the 3D volume; see Fig. 3. The same spheres are used for all the rays. Thus, the sample points are more or less uniformly distributed inside the 3D volume. The intensity values at the sample points are computed by tri-linear interpolation. Finally, the pixel values on the DRR are the sum of the m intensity values along each ray.

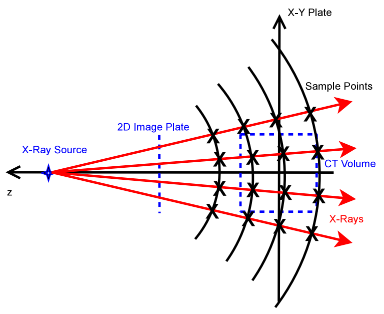

Fig. 3. The volume rendering is based on rays from the source through the pixels of the 2D image. The samples along each ray are determined by concentric spheres of different radii covering the 3D volume.

The DRR construction is very straightforward and is completely parallel for each ray. We remark that, by computing the sum recursively, we can further reduce the complexity from $O(m)$ to $O(\log m)$.

B. Solving the registration optimization problem

We use the Gauss-Newton method to solve the nonlinear least squares problem (1). Let $\mathbf{s}^n$ be the previously computed 3D rigid body motion and $\mathbf{s}^{n+1} = \mathbf{s}^n + \Delta\mathbf{s}$ be the current approximation. We want to determine the update $\Delta\mathbf{s}$ so that the resulting rigid body motion $\mathbf{s}^{n+1}$ will lead to a better DRR image. By Taylor series expansion,

$$f(\mathbf{s}^{n+1})_{i,j} = f(\mathbf{s}^n)_{i,j} + \nabla f(\mathbf{s}^n)_{i,j}^T \cdot \Delta\mathbf{s} + \cdots ,$$

where $\nabla f(\mathbf{s}^n)_{i,j} = (\frac{\partial f(\mathbf{s}^n)_{i,j}}{\partial \theta_x}, \frac{\partial f(\mathbf{s}^n)_{i,j}}{\partial \theta_y}, \frac{\partial f(\mathbf{s}^n)_{i,j}}{\partial \theta_z}, \frac{\partial f(\mathbf{s}^n)_{i,j}}{\partial \tau_x}, \frac{\partial f(\mathbf{s}^n)_{i,j}}{\partial \tau_y}, \frac{\partial f(\mathbf{s}^n)_{i,j}}{\partial \tau_z})^T$ is the gradient of $f(\mathbf{s}^n)_{i,j}$. Then, the optimization problem (1) can be approximated by

$$\min_{\Delta\mathbf{s}} \sum_{i,j} (f(\mathbf{s}^n)_{i,j} - g_{i,j} + \nabla f(\mathbf{s}^n)_{i,j}^T \cdot \Delta\mathbf{s})^2.$$

This is a linear least squares problem which can be rewritten in the matrix form as:

$$\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|_2^2, \tag{2}$$

where $A = \nabla f(\mathbf{s}^n) = N^2 \times 6$ matrix, $\mathbf{x} = \Delta\mathbf{s}$, $\mathbf{b} = g - f(\mathbf{s}^n) = N^2 \times 1$ vector, assuming the resolution of the DRR is $N \times N$. The linear least squares problem is solved by the normal equations

$$A^T A\mathbf{x} = A^T \mathbf{b}$$

using Gaussian elimination on the 6×6 linear system. Once $\mathbf{x} = \Delta\mathbf{s}$ is determined, it can be used to update the position of the 3D volume. The procedure is repeated until $\{\mathbf{s^n}\}$ have converged to an accurate solution. In our experience, Gauss-Newton method is quite effective, and usually converges in around 10 iterations. More importantly, the entire process can be easily parallelized.

III. RapidMind Development Platform

The computation of the 2D-3D registration in general is very intensive. We exploit the high parallelism of the multi-core architecture of GPUs for efficient computation of image registration. The cell processor used in [4] has 9 cores, but a GPU can have as many as over 100 cores. General programming on multi-core architectures, however, can be quite challenging. For instance, sophisticated caching schemes are developed in [4] for fast computations. On the other hand, to take full advantages of the GPU cores would often require substantial knowledge of GPU architectures such as vertex shader, rasterizer, texture maps, etc. A number of software tools have been developed for general GPU programming, such as OpenGL [9], Brook for GPUs [10] and Cg [11]. We have implemented our 2D-3D registration algorithm using RapidMind. RapidMind provides a software layer between the multi-core hardware and the application. Thus, it abstracts the hardware complexity from the users and makes GPU programming quite easy. The basic data structures are arrays. Parallel processing is achieved by performing operations on all the elements in the arrays at the same time (SIMD model). Our implementation essentially converts the registration computations into array operations.

The RapidMind implementation of the construction of DRR is as follows. The interpolation of the intensity values at the sample points is done in parallel on the 3D array of the sample points. The sum of the m intensities along each ray is also done in parallel.

For the optimization problem (2), in each iteration, the gradient vector at each pixel (i, j) is approximated by finite difference. For example,

$$\frac{\partial f(\mathbf{s}^n)_{i,j}}{\partial \theta_x} \approx \frac{f(\mathbf{s}^n + \Delta\mathbf{s}_{\theta_x})_{i,j} - f(\mathbf{s}^n)_{i,j}}{\Delta\theta_x},$$

where $\Delta\mathbf{s}_{\theta_x} = (\Delta\theta_x, 0, 0, 0, 0, 0)^T$. $f(\mathbf{s}^n + \Delta\mathbf{s}_{\theta_x})$ is obtained from DRR by changing the rotation angle θ_x^n to $\theta_x^n + \Delta\theta_x$ but keeping the other parameters fixed. The parallel DRR construction has been explained above. Afterwards, subtraction and division can be computed easily in parallel. Other derivatives are computed similarly. At the end, we have six 2D arrays of $\frac{\partial f}{\partial \theta_x}, \cdots, \frac{\partial f}{\partial \tau_z}$ which correspond to the six columns of A. To compute an entry of $A^T A$ in the normal equations, we take two derivative arrays, multiply the corresponding elements in parallel, and then use a reduction operation to compute the global sum. The right-hand side $\mathbf{b}$ is just the subtraction of the two arrays f and g. However, solving the normal equations using Gaussian elimination in parallel is not that obvious. Since it is only a small 6×6 linear system, we transfer the data and computation from GPU to CPU, and the computational time is typically very minimal. This is the only part in the registration procedure that parallel algorithms on the GPU are not used.

IV. Numerical Results

We demonstrate the efficiency of the proposed RapidMind 2D-3D image registration algorithm by two experiments. Particularly, we compare the CPU times using regular C++ and RapidMind. All computations are performed on a standard PC running with Ubuntu 7.04 Linux operating system, Intel processor with 3GHz, 1GB memory, and NVIDIA GeForce 8800 GTX.

The first experiment is an artificial example where the 3D image volume is a white cube. An advantage of using an artificial data set is that we can generate image volume with different resolutions without affecting the image quality. We synthesize an X-ray 2D portal image by using the DRR obtained from rotating and translating the 3D image volume with known parameters. For consistency, we scale the size of the cube to $[0, 1] \times [0, 1] \times [0, 1]$ for all the resolutions. As an example, Fig. 4 (left) shows the projected 2D image with no rotation and translation. Fig. 4 (right) shows the projected 2D image with rotations of $(5°, 5°, 5°)$ along each axis and translations of $(0.1, 0.1, 0.1)$ in the x-, y-, and z- directions.

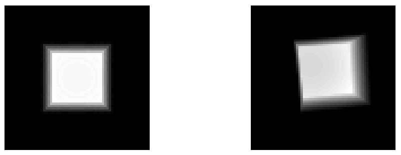

Fig. 4. The 2D projected images with (left) no rotation and translation and (right) rotations of $(5°, 5°, 5°)$ along each axis and translations of $(0.1, 0.1, 0.1)$ pixels in the x-, y-, and z- directions.

We use the 3D volume data and the simulated X-Ray portal image as the raw data for registration. We implement our registration algorithm using both regular C++ and RapidMind. The regular C++ code is to test the CPU efficiency, and the RapidMind code is to test the GPU efficiency.

Table I shows the times with different resolutions. The 3D volume resolution varies from $16 \times 16 \times 16$ to $128 \times 128 \times 128$. Each total CPU time is subdivided into 2 CPU times corresponding to the time for DRR computation (DRR), which is included in both DRR construction and linearized optimization problem, and the time for other non-DRR computation (Non-DRR). Note that the times are per iteration. The stopping criteria are $\|\Delta\mathbf{s}\|_2 < 0.1$. The total numbers of iterations are also shown.

The portal images are synthesized by changing the orientation of the 3D volume with the rotations of $(5°, 5°, 5°)$ and the translations of $(0.1, 0.1, 0.1)$ for all resolution datasets. Meanwhile, we use the angles of $(0°, 0°, 0°)$ and the translations of $(0, 0, 0)$ to construct the initial DRRs. We have

TABLE I

Average CPU times for each iteration of the image registration process.

Resolution	Time (sec)	Regurlar C++	RapidMind
$16 \times 16 \times 16$	DRR	0.012	0.029
	Non-DRR	0.003	0.109
	Total	0.015	0.138
	Iteration	6	6
Resolution	Time (sec)	Regular C++	RapidMind
$32 \times 32 \times 32$	DRR	0.140	0.047
	Non-DRR	0.060	0.139
	Total	0.200	0.186
	Iteration	5	5
Resolution	Time (sec)	Regular C++	RapidMind
$64 \times 64 \times 64$	DRR	3.990	0.091
	Non-DRR	1.318	0.171
	Total	5.308	0.262
	Iteration	5	5
Resolution	Time (sec)	Regular C++	RapidMind
$128 \times 128 \times 128$	DRR	40.331	0.152
	Non-DRR	16.388	0.248
	Total	56.720	0.400
	Iteration	6	6

also tried other parameters for constructing portal images and initial DRRs.

As the 3D resolution increases, the CPU time for the regular C++ code increases with a factor of around 10. While the per iteration time is less than 1 second for small dataset, it increases rapidly to over 50 seconds for the higher resolution dataset. The CPU time for the RapidMind code increases much slower; the per iteration time is less than 1 second for all datasets. For the large dataset $(128 \times 128 \times 128)$, the total registration time for RapidMind is around 2 seconds, whereas for regular C++, it is almost 6 minutes.

By comparing the DRR time and non-DRR time in the column of regular C++, we note that the former is around 2-4 times longer than the latter. Since DRR computing dominates the CPU time in the entire registration process, it is important to use efficient parallel algorithms for DRR computation to achieve real-time performance. Also, it is easy to implement the algorithms using RapidMind on GPUs.

In the second experiment, we use a real 3D CT volume data, which shows a tripod fracture of a skull[2]. The resolution of the image is $256 \times 256 \times 203$. Each voxel dimension is $0.7mm \times 0.7mm \times 2mm$. This 3D volume is quite large, so we only use the central chuck in the first 100 slices. The resulting resolution is $128 \times 128 \times 100$ for our numerical experiment. The 2D portal images are also synthesized similar to the first experiment, but we use different parameters, which are shown in the first column of Table II. The parameters are still $\mathbf{0}$ for constructing the initial DRR. Table II shows the CPU times for the entire image registration process. The stopping criteria is $\|\Delta \mathbf{s}\|_2 < 0.8$ in this case. The resulting relative errors are less than 0.012 for all of our experiments. Similar to the previous example, the RapidMind implementation accelerates the registration process by a factor of over 100 for the medical images.

[2]http://www.radiology.uiowa.edu/downloads/

TABLE II

Total CPU times for the entire image registration process.

Portal image parameters	Time (sec)	Regular C++	RapidMind
Rotations: $(2^\circ, 2^\circ, 2^\circ)$ Translations: $(2mm, 2mm, 2mm)$	DRR	123.27	0.62
	Non-DRR	44.89	1.13
	Total	168.16	1.75
	Iteration	5	5
Portal image parameters	Time (sec)	Regular C++	RapidMind
Rotations: $(4^\circ, 4^\circ, 4^\circ)$ Translations: $(4mm, 4mm, 4mm)$	DRR	191.94	0.98
	Non-DRR	71.36	1.72
	Total	263.30	2.70
	Iteration	8	8
Portal image parameters	Time (sec)	Regular C++	RapidMind
Rotations: $(6^\circ, 6^\circ, 6^\circ)$ Translations: $(6mm, 6mm, 6mm)$	DRR	267.96	1.32
	Non-DRR	103.43	2.33
	Total	371.39	3.65
	Iteration	11	11

V. Conclusions

We have developed an efficient 2D-3D rigid registration method which is amenable for GPU processing. We have implemented the algorithm using RapidMind to exploit the highly parallelism of GPUs. Numerical results show that our method is much faster than regular CPU processing. Moreover, for real image datasets, it only takes around 3 seconds for performing 2D-3D image registration.

References

[1] W. Wein, "Intensity based rigid 2D-3D registration algorithms for radiation therapy (http://campar.in.tum.de/twiki/pub/Main/WolfgangWein /thesis.pdf)," Master's thesis, Technische Universität München, Fakultät für Informatik, München, 2003.

[2] R. Smolikova, M. P. Wachowiak, and M. Drangova, "Registration of fast cine cardiac MR slices to 3D preprocedural images: Toward real time registration for MRI-guided procedures," in *Medical Imaging 2004: Image Processing*, J. Fitzpatrick and M. Sonka, Eds. SPIE, 2004, pp. 1195–1205.

[3] N. Chrisochoides, A. Fedorov, A. Kot, N. Archip, P. Black, O. Clatz, A. Golby, R. Kikinis, and S. K. Warfield, "Toward real-time image guided neurosurgery using distributed and grid computing," in *ACM/IEEE SC 2006 Conference*, 2006, p. 37.

[4] M. Ohara, H. Yeo, F. Savino, G. Iyengar, L. Gong, H. Inoue, H. Komatsu, V. Sheinin, S. Daijavad, and B. Erickson, "Real-time mutual-information-based linear registration on the cell broadband engine processor," in *4th IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, 2007, pp. 33–36.

[5] R. Chisu, "Techniques for accelerating intensity-based rigid image registration (http://campar.in.tum.de/twiki/pub/Main/WolfgangWein/DA-Chisu.pdf)," Master's thesis, Technische Universität München, Fakultät für Informatik, München, 2005.

[6] J. P. W. Pluim, J. A. Maintz, and M. A. Viergever, "Mutual-information-based registration of medical images: A survey," *IEEE Transactions on Medical Imaging*, vol. 22, pp. 986–1004, 2003.

[7] H. Ray, H. Pfister, D. Silver, and T. Cook, "Ray casting architectures for volume visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, pp. 210–223, 1999.

[8] P. Lacroute and M. Levoy, "Fast volume rendering using a shear-warp factorization of the viewing transformation," *Computer Graphics*, vol. 28, pp. 451–458, 1994.

[9] R. Rost, J. Kessenich, and B. Lichtenbelt, *OpenGL Shading Language*. Boston: Addison-Wesley, 2004.

[10] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for GPUs: Stream computing on graphics hardware," *ACM Trans. on Graphics*, vol. 23, pp. 777–786, 2004.

[11] NVIDIA Corporation, "Cg toolkit release notes," 2003.