A Method for Accelerating Bronchoscope Tracking Based on Image Registration by GPGPU

Takamasa Sugiura[1], Daisuke Deguchi[1], Takayuki Kitasaka[2], Kensaku Mori[1], and Yasuhito Suenaga[1]

[1] Graduate School of Information Science, Nagoya University, Furo-cho, Chikusa-ku, Nagoya, Aichi, 464-8603, Japan
tsugiura@suenaga.m.is.nagoya-u.ac.jp
[2] Aichi Institute of Technology, Toyota, Japan

Abstract. This paper presents an acceleration method for tracking a bronchoscope based on image registration. This method tracks a bronchoscope by image registration between real bronchoscopic images and virtual ones derived from CT images. However, since the computation cost of image registration, especially generating virtual bronchoscopic (VB) images, is quite expensive, it is difficult to track the bronchoscope in real-time. To solve this problem, we try to accelerate the process of image registration by utilizing GPU (Graphics Processing Unit) with CUDA language. Specifically, we accelerate two parts: (1) VB image generation by volume rendering, and (2) image similarity calculation between a real and a virtual bronchoscopic images. Additionally, to obtain the maximum performance of GPU as much as possible, we minimize (i) the amount of memory transfer between CPU and GPU, and (ii) the number of GPU function calls from CPU. We applied the proposed method to ten pairs of real bronchoscopic video and CT images. The experimental results showed that the proposed method could track the bronchoscope 16 times faster than the method using only the latest CPU.

1 Introduction

In recent years, a bronchoscope navigation system has been developed for assisting a bronchoscopist to operate a bronchoscope safely and efficiently [1–3]. This system gives a bronchoscopist much useful information, such as display of the current location of the bronchoscope's tip, the path to a target location, and anatomical structures beyond bronchi walls that cannot be observed through the bronchoscope.

To realize this system, it is indispensable to track the motion of the bronchoscope camera equipped on the tip of a bronchoscope in real-time. Several research groups reported tracking methods that estimate the motion by the image registration between real bronchoscopic (RB) and virtual bronchoscopic (VB) images [1–3]. Bricault et al. estimated the motion of an RB camera using structural information of bronchial branching patterns [1]. However, their method is not

applicable to locations where branching structures cannot be observed. Also, it cannot estimate camera location for the cases where tumors exist inside airways. Helferty et al. reported a registration method based on similarity measure using normalized mutual information between RB and VB images [2]. Also, Mori et al. proposed an image-registration based method that predicts the camera's motion by the Kalman filter [3]. Prediction results are used for determining initial locations of similarity maximization process to avoid falling into local minima. In the bronchoscope tracking method based on image-registration between RB and VB images, it is necessary to generate considerable number of VB images and to compare them with an RB image for obtaining a good registration result. However, since the computation cost for generating VB images is quite expensive, their methods were far from real-time tracking, for example 8.7 seconds for processing one frame in [3].Therefore, their methods cannot be applied to the bronchoscope navigation in an actual operation. For real-time tracking, we proposed an acceleration method for tracking the bronchoscope based on image registration.

GPU (Graphical Processing Unit), especially GPGPU with special language such as CUDA (NVIDIA Corp.), has great computational capabilities in floating point operation and it has become more than ten times faster than those of CPUs. Therefore, this paper uses computational power of GPU for accelerating the image-registration algorithm.

As described above, image registration process requires to generate huge number of VB images for comparing them with an RB image. There are two methods for generating VB image: (a) surface rendering and (b) volume rendering method. However, when the RB camera is close to the bronchial wall, triangles forming the shape of the bronchus can be seen on VB images in the surface rendering [4]. Due to the artifacts of such triangles that do not appear on volume rendering images, tracking results become less-accurate. Therefore, this paper uses volume rendering in bronchoscope tracking. Since the computation cost for generating a VB image by volume rendering is quite expensive, this generation process is one of the biggest bottlenecks of image registration process. An acceleration of VB image generation can greatly improve the computation time of the image registration process. Therefore, the proposed method tries to accelerate VB image generation process by using CUDA, and uses acceleration technique proposed by Kruger et al.[5]. In the programming on CUDA, GPU is considered a processor that processes a large number of threads simultaneously. Each thread conducts the general-purpose computation by executing a single program (a kernel) of CUDA. In CUDA, there is a memory area where multiple threads can share data. Synchronization of threads on GPU is achieved without any intervention from CPU. Therefore, the data exchange among threads (= shader units) is much easier and cheaper to implement than that of common shader languages such as Cg or HLSL. However, CPU and GPU cannot share memories each other on a conventional PC platform. Data transfer between two types of memories would become a bottleneck of image registration, since memory transfer operations are quite slow than calculation operations on

GPU. Also, there is a small overhead in calling GPU function from CPU. To obtain the maximum performance of GPU as much as possible, we have to minimize (i) an amount of memory transfer between CPU and GPU, and (ii) the number of GPU functions called from CPU. With satisfying these requirements, the proposed method accelerates two parts of image registration: (1) VB image generation by volume rendering, and (2) image similarity calculation between an RB and a VB images by GPU. Since both parts are implemented on GPU, we can directly compute image similarity between RB and VB without giving any controls to CPU. Only a result of image similarity calculation is transferred from GPU to CPU. This greatly saves an amount of memory transfer between CPU and GPU.

2 Method

The GPU has been developed as Graphics Processor which execute graphics process. In graphics process, a lot of data are computed in parallel. Therefore, when GPU computes data weakly depending on each other with high parallelism, we can obtain the maximum performance of GPU. On the other hand, the CPU is good at sequential process which includes complex operations, such as branch instructions. Therefore, when using CPU and GPU, we must consider about CPU's and GPU's advantages. The image registration procedure presented in [4] is simplified as: (1) initialization of view point and view direction, (2) VB image generation, (3) image similarity computation, (4) convergence test and iteration, (5) output of view point and view direction (Fig. 1). Powell method is used for iterative minimization of the image similarity in Step (4) [6]. During volume rendering procedure, each pixels of VB image is calculated independently. Also, many summations during image similarity calculation can be accelerated by parallel computation. Therefore, this paper accelerates the image registration process by implementing the steps (2) and (3) on GPU. Since the steps (2) and (3) are executed in cascade on GPU, we can avoid penalty of data transfer.

2.1 Fast VB image generation by GPU

Volume rendering is one of the techniques for visualizing volumetric images. In prior to rendering, we need to define a transfer function of color and opacity values. Color of each pixel on a screen is calculated by casting a ray from the viewpoint. This ray casting process is performed by accumulating colors and opacities at sample points on the ray.

Volume rendering requires calculations of intensity value and normal vector at each sample point. Linear interpolation technique is usually used for calculating both intensity values and normal vectors. Here, by using linear interpolation technique, an intensity value $f(x, y, z)$ at (x, y, z) can be calculated as

$$f(x, y, z) = \big(1 - (z - \lfloor z \rfloor)\big)\, f(x, y, \lfloor z \rfloor) + (z - \lfloor z \rfloor)\, f(x, y, \lfloor z \rfloor + 1), \qquad (1)$$

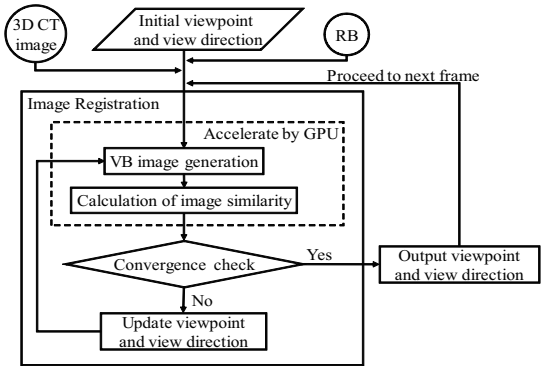

Fig. 1. Processing flow of the proposed method.

where $\lfloor a \rfloor$ is a floor operation, and it returns maximum integer value less than or equal to a. $f(x, y, \lfloor z \rfloor)$ and $f(x, y, \lfloor z \rfloor + 1)$ can be calculated as

$$\begin{aligned} f(x,y,\lfloor z\rfloor) &= w_1\, f(\lfloor x\rfloor,\lfloor y\rfloor,\lfloor z\rfloor) + w_2\, f(\lfloor x\rfloor+1,\lfloor y\rfloor,\lfloor z\rfloor) \\ &\quad + w_3\, f(\lfloor x\rfloor,\lfloor y\rfloor+1,\lfloor z\rfloor) + w_4\, f(\lfloor x\rfloor+1,\lfloor y\rfloor+1,\lfloor z\rfloor), \qquad (2) \\ f(x,y,\lfloor z\rfloor+1) &= w_1\, f(\lfloor x\rfloor,\lfloor y\rfloor,\lfloor z\rfloor+1) + w_2\, f(\lfloor x\rfloor+1,\lfloor y\rfloor,\lfloor z\rfloor+1) \\ &\quad + w_3\, f(\lfloor x\rfloor,\lfloor y\rfloor+1,\lfloor z\rfloor+1) + w_4\, f(\lfloor x\rfloor+1,\lfloor y\rfloor+1,\lfloor z\rfloor+1), \qquad (3) \\ w_1 &= (1-(y-\lfloor y\rfloor))(1-(x-\lfloor x\rfloor)), \\ w_2 &= (1-(y-\lfloor y\rfloor))(x-\lfloor x\rfloor), \\ w_3 &= (y-\lfloor y\rfloor)(1-(x-\lfloor x\rfloor)), \\ w_4 &= (y-\lfloor y\rfloor)(x-\lfloor x\rfloor). \end{aligned}$$

As shown in Eqs.(1), (2) and (3), $f(x, y, z)$ is calculated by referring intensity values of surrounding eight voxels. Also, a normal vector of each sampling point is calculated by referring intensity values of surrounding 32 voxels. Although the cost for each linear interpolation is quite small, numerous number of interpolation is required for generating VB images. Therefore, acceleration of linear interpolation will improve the performance of VB image generation. The proposed method tries to accelerate this by linear interpolation hardware of GPU.

To use this linear interpolation hardware that can be accessible from the CUDA language, the proposed method maps a 3D image coordinate $(x, y, \lfloor z \rfloor)$ onto a 2D texture coordinate (x', y') as

$$\begin{aligned} (x', y') &= \phi\,(x, y, \lfloor z \rfloor) \\ &= \left(x + W\Big(\lfloor z \rfloor \bmod N\Big), y + H\lfloor \frac{\lfloor z \rfloor}{N} \rfloor \right), \end{aligned} \qquad (4)$$

where N is the number of CT slices placed along the x-axis of 2D texture coordinate system, and W and H are the width and the height of each slice. Figure

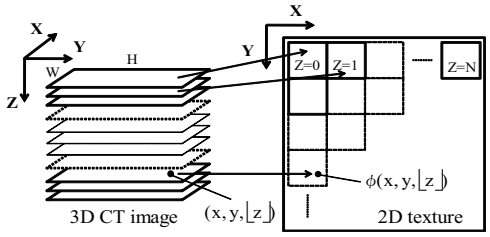

Fig. 2. Relationships between coordinate systems of 3D image and 2D texture.

2 illustrates this mapping function ϕ. As shown in Fig. 2, the proposed method places each slice of the 3D CT image as a 2D matrix. By using this function, we can obtain intensity values directly at arbitrary positions inside the 2D texture coordinate system by GPU hardware. An intensity value $f(x, y, \lfloor z \rfloor)$ can be obtained as

$$f\left(x, y, \lfloor z \rfloor\right) = F\big(\phi\left(x, y, \lfloor z \rfloor\right)\big), \tag{5}$$

where $F\left(\phi\left(x, y, \lfloor z \rfloor\right)\right)$ is an intensity value at a 2D texture coordinate $\phi\left(x, y, \lfloor z \rfloor\right)$. From Eqs.(4) and (5), Eq.(1) can be rewritten as

$$f(x, y, z) = \big(1-(z - \lfloor z \rfloor)\big)\cdot F\big(\phi\left(x, y, \lfloor z \rfloor\right)\big)+(z - \lfloor z \rfloor)\cdot F\big(\phi\left(x, y, \lfloor z \rfloor + 1\right)\big). \tag{6}$$

Since $F\big(\phi\left(x, y, \lfloor z \rfloor\right)\big)$ and $F\big(\phi\left(x, y, \lfloor z \rfloor + 1\right)\big)$ are directly obtained by GPU hardware, calculation of $f(x, y, z)$ is simplified to 1D linear interpolation. Therefore, $f(x, y, z)$ can be calculated by referring only two intensity values at the 2D texture. A normal vector of each sampling point can be calculated by same way.

2.2 Fast image similarity calculation by GPU

The proposed method uses mean-squared error as image similarity measure. Here, image similarity between an RB and a VB image is calculated as

$$MSE\left(\mathbf{B}, \mathbf{V}\right) = \sum_i \left(\left(\mathbf{B}_i - \overline{\mathbf{B}}\right) - \left(\mathbf{V}_i - \overline{\mathbf{V}}\right)\right)^2, \tag{7}$$

where $\mathbf{B}$ and $\mathbf{V}$ are an RB and a VB image. $\mathbf{B}_i$ and $\mathbf{V}_i$ are pixel values at i-th pixel, and $\overline{\mathbf{B}}$ and $\overline{\mathbf{V}}$ are mean pixel value of $\mathbf{B}$ and $\mathbf{V}$, respectively [4]. When we calculate this similarity, it requires two consecutive steps: (a) calculation of $\overline{\mathbf{B}}$ and $\overline{\mathbf{V}}$, and (b) calculation of Eq.(7). That is, two calls of GPU function are required for calculating Eq.(7). To reduce the overhead of GPU function calls from CPU, we rewrite Eq.(7) as

$$MSE\left(\mathbf{B}, \mathbf{V}\right) = \sum_i \left(\mathbf{B}_i - \mathbf{V}_i\right)^2 - |\mathbf{B}| \left(\overline{\mathbf{B}}^2 + \overline{\mathbf{V}}^2 - 2\overline{\mathbf{B}}\,\overline{\mathbf{V}}\right), \tag{8}$$

where $|\mathbf{B}|$ is the number of pixels inside $\mathbf{B}$. All terms in Eq.(8) can be calculated independently by GPU, and an image similarity is obtained by summing all

results on CPU. Eq.(8) requires only one GPU function call for calculating image similarity, and we can save an overhead of one GPU function call than Eq.(7). Eq.(8) is computed by the following steps.

[Image similarity calculation]

[**Step** 1] Thread i on GPU calculates

$$a_i = \left(\mathbf{B}_i - \mathbf{V}_i\right)^2, \quad b_i = \mathbf{B}_i, \quad c_i = \mathbf{V}_i.$$

[**Step** 2] Reduction algorithm [7] is used on GPU for summing a_i, b_i, c_i as

$$\alpha = \sum_i a_i, \quad \beta = \sum_i b_i, \quad \gamma = \sum_i c_i.$$

[**Step** 3] α, β and γ are transferred to CPU.

[**Step** 4] Image similarity of Eq.(8) is computed on CPU as

$$MSE\left(\mathbf{B}, \mathbf{V}\right) = \alpha - |\mathbf{B}| \left(\left(\frac{\beta}{|\mathbf{B}|}\right)^2 + \left(\frac{\gamma}{|\mathbf{B}|}\right)^2 - 2\frac{\beta}{|\mathbf{B}|}\frac{\gamma}{|\mathbf{B}|} \right).$$

3 Results and Discussion

We applied the proposed method to ten pairs of RB videos and CT images of the same patient. The specifications of CT images were: 512×512 pixelsC72 $\sim$ 361 slicesC2.0 $\sim$ 5.0 mm slice thicknessCand 1.0 $\sim$ 2.0 mm reconstruction pitchD RB videos were recorded onto digital video tapes and captured at 30 frames per second. All registration tasks were performed as off-line jobs. We used a Dell Precision workstation (Intel Quad Core Xeon 2.80 GHz×2CGeForce 8800 GTXC3.0 GB RAM, Windows XP), and implemented the proposed algorithm as GPU codes by using CUDA.

To evaluate the computation efficiency of the proposed method, we measured the average computation time for generating VB images by changing image size and observation parameters, such as camera position and orientation. Figure 4 shows the computation time for generating VB images on CPU and GPU. Also, we measured the computation time of the image registration process of the previous and the proposed methods. The results of both methods are shown in Table 1. Figure 3 shows the tracking results obtained by the previous and the proposed methods.

Figure 4 shows that the bigger image size is, the more remarkable the difference between computation time for VB image generation on CPU and GPU is. Also, VB image (128×128) generation on GPU was performed 22 times faster than on CPU. From this result, it is easily confirmed that GPU enabled us to generate VB images quite faster than CPU. As shown in Table 1 and Fig. 3, although the computational speed of the proposed method was quite faster than that of the previous method, we obtained almost the same tracking results from

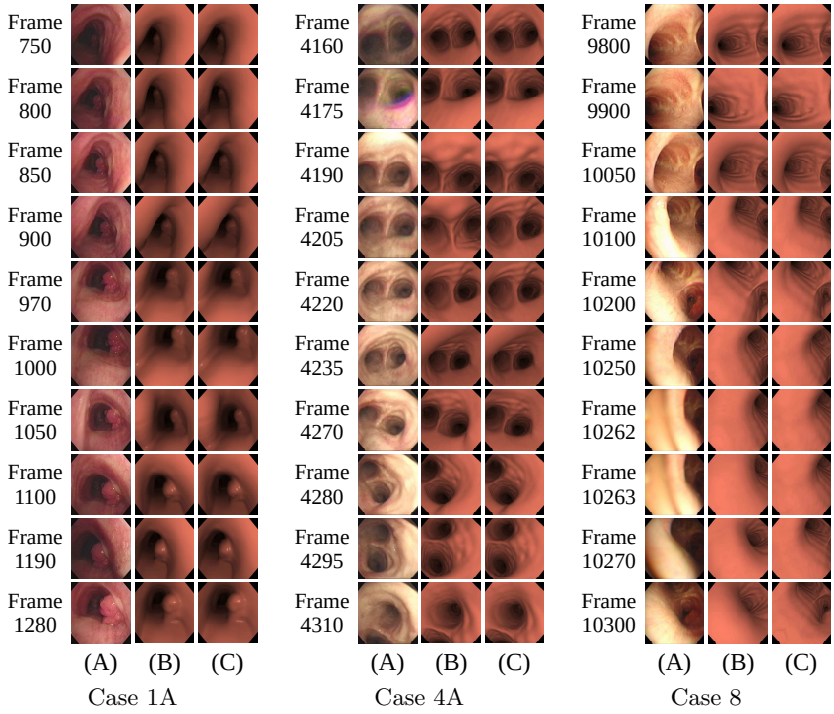

Fig. 3. Tracking results. (A) RB image, (B) results by the previous method, and (C) results by the proposed method.

the CPU-base and the GPU-base methods. However, there are small differences among the number of success frames tracked continuously in these methods. Since the accuracy of the linear interpolation hardware of GPU is not as accurate as CPU does (effective digits), VB images generated by GPU were slightly different from those of CPU. This caused different results despite of doing same procedures.

From Table 1, the average computation time was 0.068 sec. in the proposed method, while 1.085 sec. in the previous method. This means that the proposed method is 16 times faster than the previous method. From these results, we can confirm that GPU is quite helpful for accelerating bronchoscope tracking based on image registration. However, the proposed method may be more accelerated by modifying the process of the image registration for GPU. For example, by changing optimization problem to parallel algorithm, such as [8], we may accelerate the process on GPU.

In general, the tracking speed should be faster than video rate (30 frames/sec.) for real-time guidance. However, the proposed method could only track the bronchoscope at 15 frames per second. We compare the tracking performances of two methods: (a) execution of image registration for every frame and (b) for every

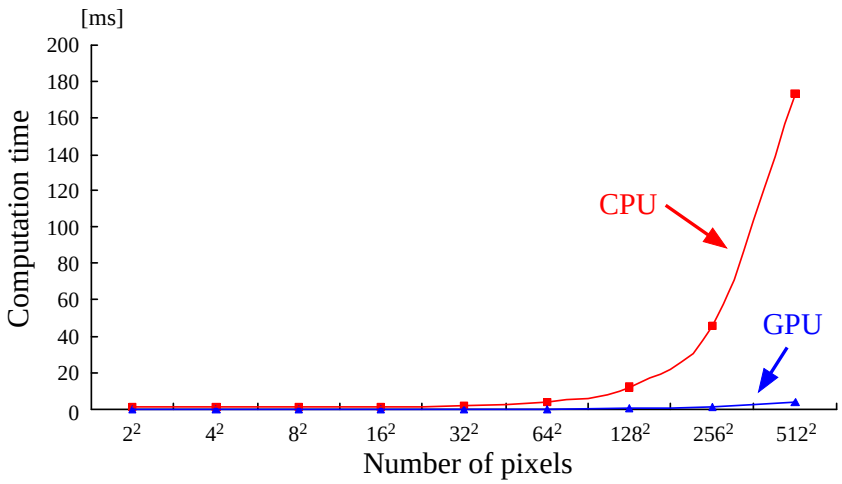

Fig. 4. The computation time for generating VB images when image size changes.

two frames. The results showed that the method (b) could track frames is 5724 successive frames in total. Although the tracking performance decreased to 89 % of the method (a), this results demonstrate the possibility of the method (b) for real-time tracking. On the other hand, since this paper uses single initial guess for image registration, tracking performance of the proposed method was poor than that of [3]. To improve tracking performance, we should try to implement powerful image similarity on GPU. This is also our future work.

4 Conclusions

This paper presented a method for accelerating bronchoscope tracking based on image registration by GPU. To accelerate the process of bronchoscope tracking, we implemented the bottleneck procedures of image registration on GPU, such as generation of VB images by volume rendering and image similarity calculation. In the proposed method, volume rendering can be accelerated efficiently by fast linear interpolation of GPU hardware, and the process of bronchoscope tracking is also accelerated by the reduction of the traffic between CPU and GPU. The experimental results showed that the computational speed of the proposed method was 15 frames per second on average.

Future work includes: (a) investigation of task assignment between CPU and GPU, (b) accuracy improvement of the image registration process.

References

1. I. Bricault, G. Ferretti and P. Cinquin, “Registration of Real and CT-Derived Virtual Bronchoscopic Images to Assist Transbronchial Biopsy,” IEEE Transactions on

Table 1. Results of bronchoscope tracking. The numbers show consecutive frames successfully (average computation time [sec.]). The number of successive frames were counted with our visual inspection.

Case	Path	Num. of frames	Results		Improvement ratio
			Previous	Proposed	
1	A	993	575 (0.795)	572 (0.078)	0.99 (10.2)
	B	700	154 (1.223)	154 (0.081)	1.00 (15.1)
	C	1120	45 (1.586)	24 (0.077)	0.53 (20.6)
	D	379	70 (0.892)	70 (0.057)	1.00 (15.6)
	E	279	93 (0.828)	94 (0.079)	1.01 (10.5)
2	A	429	14 (1.318)	14 (0.058)	1.00 (22.7)
3	A	1500	119 (1.327)	119 (0.064)	1.00 (20.7)
	B	1100	359 (1.085)	361 (0.050)	1.01 (21.7)
	C	1000	279 (1.203)	136 (0.050)	0.49 (24.1)
	D	202	32 (1.850)	32 (0.071)	1.00 (26.1)
	E	250	111 (1.607)	111 (0.057)	1.00 (28.2)
4	A	840	161 (1.518)	161 (0.108)	1.00 (14.1)
	B	204	204 (1.044)	204 (0.067)	1.00 (15.6)
5	A	500	22 (1.748)	22 (0.085)	1.00 (20.1)
6	A	1000	1000 (1.062)	718 (0.084)	0.72 (12.6)
	B	800	44 (1.034)	44 (0.061)	1.00 (17.0)
	C	1000	13 (0.941)	41 (0.075)	3.15 (12.5)
7	A	500	89 (1.151)	89 (0.083)	1.00 (13.9)
	B	530	140 (1.119)	140 (0.071)	1.00 (15.8)
8	A	1600	462 (0.621)	536 (0.056)	1.16 (11.1)
9	A	450	125 (0.870)	125 (0.061)	1.00 (14.3)
10	A	900	194 (0.690)	315 (0.074)	1.62 (9.3)
	B	1400	558 (0.433)	695 (0.055)	1.25 (7.9)
	C	2000	1183 (0.578)	1182 (0.044)	1.00 (13.1)
	D	820	499 (0.611)	499 (0.047)	1.00 (13.0)
Total (ave.)			6545 (1.085)	6458 (0.068)	0.99 (16.2)

Medical Imaging, Vol. 17, No. 5, pp. 703-714, 1998.
2. J. P. Helferty and W. E. Higgins, "Technique for Registering 3D Virtual CT Images to Endoscopic Video," Proceedings of ICIP (International Conference on Image Processing), pp. 893-896, 2001.
3. K. Mori, D. Deguchi, T. Kitasaka et al, "Bronchoscope Tracking Based on Image Registration Using Multiple Initial Starting Points Estimated by Motion Prediction," Proc. of MICCAI 2006, LNCS, Vol. 4191, pp. 645-652, 2006.
4. D. Deguchi, K. Mori, J. Hasegawa et al, "Camera motion tracking of real bronchoscope using epipolar geometry analysis and CT derived bronchoscopic images," Proceeding of SPIE Vol. 4683, pp. 30-41, 2002.
5. J.Kruger and R. Westermann, "Acceleration Techniques for GPU-based Volume Rendering," IEEE Visualization, pp. 287-292. 2003.
6. W. H. Press, S. A. Teukolsky, W. T. Vetterling et al, "Numerical Recipes in C," The Art of Scientific Computing Second Edition, CAMBRIDGE UNIVERSITY PRESS, pp. 321-336, 1999.

7. Mark Harris, "Optimizing Parallel Reduction in CUDA," CUDA Advanced Topics, CUDA ZONE,
"http://www.nvidia.com/object/cuda_sample_advanced_topics.html".
8. F. V. Berghen, H. Bersini, "CONDOR, a new parallel, constrained extension of Powell's UOBYQA algorithm: Experimental results and comparison with the DFO algorithm," Journal of Computational and Applied Mathematics, Elsevier, Volume 181, Issue 1, September 2005, pp. 157-175.