Multi-view Range Image Registration using CUDA

Sung-In Choi[1], Soon-Yong Park[1], Jun Kim[2], Yong-Woon Park[2]

[1] Department of Computer Engineering, Kyungpook National University
Daegu, 702-701, Korea
[2]Agency for Defense Development
Yuseong PO Box 35-1, Daejeon 305-600, Korea

Abstract: In this paper, we propose a real-time and on-line 3D registration system which acquires and registers multi-view range images simultaneously. The proposed system implements a 3D registration technique using GPU programming techniques. To register multi-view range images accurately in real-time, the repetitive parts of projection and transformation, which require a large computational overhead, are processed using CUDA. Using a portable range sensor with 320x240 resolution, the system shows about 5 frames per second of matching performance.

1. Introduction

Multi-view range image registration is a technique of bringing range images obtained from different camera coordinate systems to a common coordinate system. It is one of the important steps to generate complete 3D models of real objects from 3D range sensors. In recent years, many investigations have addressed multi-view range image registration[1]. This is due to the fact that many commercial range sensors become available nowadays. However, it is usually a time-expensive and difficult task whose accuracy affects the quality of final 3D models. Thus, conventional investigations usually focus on accurate implementation of registration refinement.

Registration refinement is also a 3D shape matching technique which finds correspondence between different 3D shapes. It usually needs considerable computation because that 3D data from a range sensor consists of hundreds of thousand points. In this reason, few systems have addressed real-time registration problem.

Recent graphics devices provide tremendous memory bandwidth and computational power and thus, computer vision researches employ GPU(graphics processing unit) for fast and real-time implementation of computer vision algorithms. Studipta N. Sinha in [2], proposes KLT feature tracker and SIFT using GPU, Alan Brunton [3] focused on belief propagation for stereo vision. In [4], James Fung introduced OpenVidia library which is accelerating the real-time based computer vision algorithm such as edge and corner detection or color object tracking on GPU.

Those cases, commonly, come from effort to reduce data processing time efficiently. If it is also possible to apply the computational power of GPU to the online 3D registration system using enormous range images, the speed problems will be solved.

In this paper, we propose a real-time and on-line 3D registration system which acquires and registers multi-view range images simultaneously. Continuously obtained range images from a hand-held range sensor are registered by using a geometric refinement technique.

To register range images in real-time, we implement a Point-to-Plane registration refinement technique using GPU. To use up-to-date techniques of GPU, we use the CUDA architecture which is available beyond GeForce 8000 series graphic boards from NVIDIA. Most linear algebra used in the refinement process is implemented by GPU programming except a least squares minimization. For experiments with real objects, a hand-held stereo camera is used to continuously obtain range images. Results in this extended abstract show the proposed system can registered the range images in very fast time.

2. Up-to-date GPGPU technique - CUDA

The GPU is a special-purposed processor that is originally designed to solve the bottleneck which is caused by graphics works on CPU. Because of the number of integrated transistors and the distinguished parallel processing power of the SIMD(single instruction multiple data) architecture, there has been a lot of researches to use GPU as a general processing unit[5][6]. In general, the technique which is named as GPGPU(general-purpose computation on GPUs) is especially used for medical image processing or video encoding and computer vision to speed up algorithms.

CUDA(compute unified driver architecture) is a new GPGPU technology that allows a programmer to use the C programming language for general-purpose computation on GPU. The driver and software development toolkit is provided by NVIDIA corporation. CUDA works with GeForce 8~9, higher series than Quadro NVS 130M and Tesla platform.

As a parallel programming model, CUDA follows traditional GPGPU concept. But the system architecture which is actually working is different. The previous GPGPU method consists of a combination of the OpenGL extensions and a shading language such as Cg or Renderman. Along graphics pipelines, users could use the GPU performance indirectly by programmed vertex shader and pixel shader. However, the memory model which could be accessed by the system is limited to a texture and the output address for writing is fixed by the rasterizer. Therefore, it is impossible to excute gathering and scattering operations simulataneously to any memory area.

On the other hand, CUDA provides environment that can be considered as an independent platform to graphics hardware. A user don't have to understand the graphics pipeline any more and is free from restriction about input/output. Anyone who has experiences in C programming language and understanding CUDA architecture is able to use parallel computing using GPU.

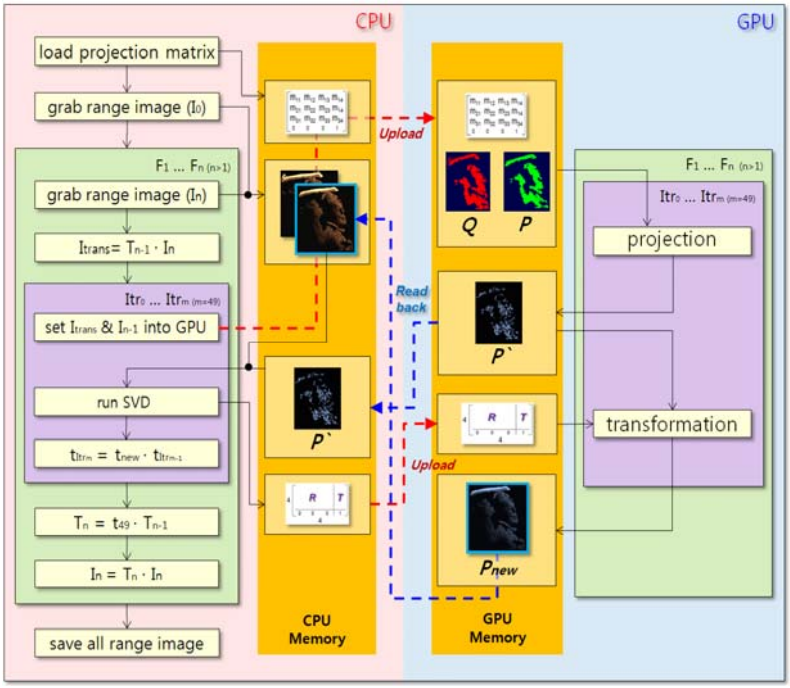


Figure 1. the proposed scheme of 3D registration

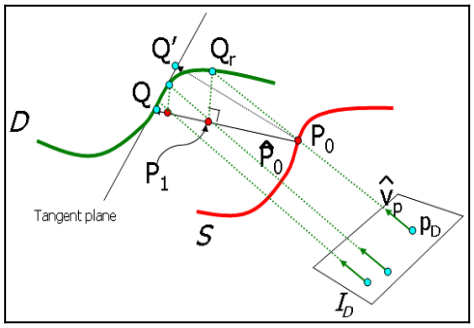


Figure 2. searching matching point using IPP

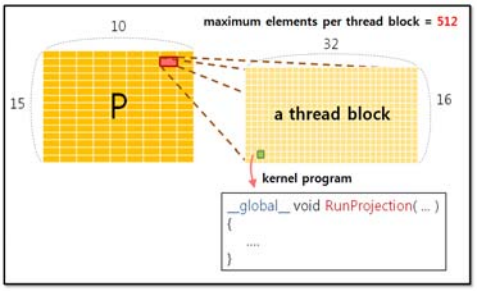


Figure 3. thread execution model

3. GPU-based realtime 3-D registraion system

The proposed system uses a 3D registration technique which is named IPP(iterative projection point) using iteration of the projection and the transformation. In this section, we first describe IPP algorithm for understanding 3D registration. Second, we describe the programming techniques of CUDA which is applied in our system. Finally, the implementation details of our system is described.

3. 1 Registration Refinement

The IPP technique combines Point to Plane and Point to Projection methods to match range images accurately[7][8]. Point to Projection method uses the projection-relation between a 2.5-D range image and a 3-D surface instead of using a fast search algorithm. The IPP technique matches range images more accurately than Point to Projection method. Description of IPP technique is shown in Fig. 2. The purpose of matching is to find a matching point Q of point P_0, which is on S surface, on D surface. Let us evaluate coordinate of p with projecting P_0 on 2-D image of D surface. Point Q on D surface, relevant to coordinate p, can be evaluated using range images. P_1 with projecting this point on a normal of P_0 is computed again, and we can get Q by repeating the process above until P_t converges. Given a sufficient number of a homogeneous set $\{P\}$ and $\{Q\}$, we can calculate an equation (1) for transformation matrix, which minimize matching error ϵ between two point sets with using a SVD(singular value decomposition).

$$\epsilon = \sum_i \| P_i - TQ_i \| \quad (1)$$

3. 2. CUDA Programming.

CUDA provides the method of allocating, copying, and canceling memory through functions such as cudaMalloc(), cudaMemcpy(), cudaFree(), and so on, in similar way to C language. Here the memory relevant to GPU is DRAM existing on graphic card, and uses remaining area after allocating for display. However, because present CUDA does not support memory paging, it must always consider spare area when allocating memory. To use efficient system resources, we use the method that up-loads data, performs algorithm, and then reads results back after allocating memory depth fixed at the beginning of program, without using the memory relevant to the whole frame sequence. Memory uses two float3*320*240 size of input arrangements and one output arrangement, and additionally uses one float*16 size of arrangement to input projection and transformation matrices.

In GPU, projection and transformation are performed with an execution model as shown in Fig. 3. In CUDA, GPU is regarded as a parallel-processor that can execute a great number of threads simultaneously. This thread is a similar concept to the thread that generally we can see in an operating system. Multi-processor of GPU performs the program called as kernel by each thread. And threads are performed at the same time with being tied in one block. The kernel becomes a substantive kernel of CUDA programming, and is made out by user. A CUDA program embodied by us also is composed of projection and transformation kernels.

When performing CUDA program, user must assign the size of one thread block and a grid composed of these blocks. Assigning a grid and the size of a thread block depends on user. But, a user must guarantee that arrangement and thread are executed by 1:1 mapping for

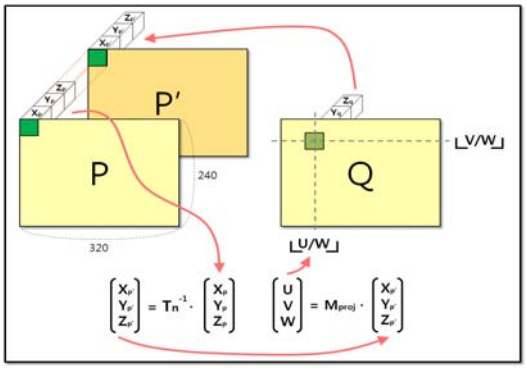


Figure 4. projection

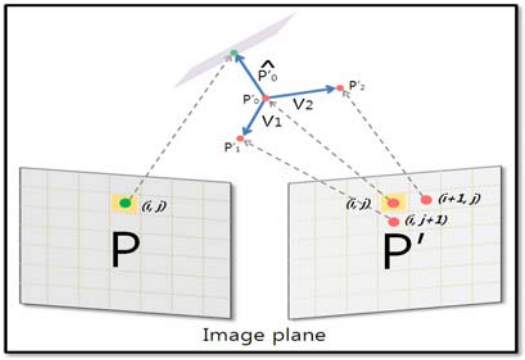


Figure 5. sampling of P' using tangent plane

efficient parallel-performance, and have an eye on kernel coding carefully about the fact that communication between threads is possible only in a thread block. We used total 76800(10*32*15*16 = 320*240) threads for 320x240 image with using 10*15 grids and 32*16 thread blocks. This numerical value was assigned for 512 threads to be tied in each block, and it didn't consider communication between threads when assigning the size of grid, because an adjacent thread didn't have to refer to each other within an arrangement in this system. For your guidance CUDA allows maximum 512 threads per thread block on G80 GPU.

3. 3 Matching by using CUDA

Main purpose of using GPU in 3D matching is to improve the whole run-time by performing the repetitive part of projection and transformation, which is a kernel of algorithm and relevant to a large computational overhead part. A whole scheme of a GPU-based real-time 3-D matching system is shown in Fig. 1.

Proposed system performs pair-wise matching with a previous frame on the basis of a present frame. Therefore it tries to match just after acquiring the data from the second frame, and projection and transformation are repeated between two frames. Iteration is performed 50 times in this paper. If the iteration is bigger, they can be matched more precisely, but the performance in terms of time is decreased.

Whole flow of the system is as follows. Firstly if a range image I_n of present frame is acquired, I_{trans} is created by multiplying transformation matrix T_{n-1}, which is from matching process in the previous stage, to I_n with previous processing. This process is to move newly acquired 3-D model to the location of previous frame approximately, and ultimately plays a role of reducing the error of multi-view range images. When previous processing is finished, I_{trans} and I_{n-1} are newly defined as a matching point set P and Q respectively in Fig. 2, and up-loaded in GPU memory.

When P and Q are up-loaded in memory, it passes to the next stage, a projection stage. Fig. 4 shows the data structure of a projection algorithm performed in GPU. The projection stage is the process of evaluating a set of points projected from P to Q, and accomplished by the following 3 steps.

1) Multiply T_{n-1}^{-1} by (X_p, Y_p, Z_p), relevant to [i, j] (i<320 j<240) index of P, and then evaluate (u, v, w) by multiplying projection matrix M_{proj} once more.
2) After dividing (u, v, w) with w, round off its result, and then make [u', v'].
3) Substitute (X_q, Y_q, Z_q), relevant to [u', v'] index of Q, for [i, j] index of P'.

After passing through projection stage, finally P', which the projected result of P for Q is saved, is created. This P' is led back to CPU memory, and used as an input value to evaluate the transformation matrix.

The transformation matrix is evaluated by using P and P' as follows.

1) Linear-sampled range image P and P' with the same index, then extract about 300～500 3-D point clouds. To prevent sampling for wrong point, if all xyz values of 3D point are 0 or vector magnitude between extracted P and P' point is 10 or above 10, except these case from sampling. xyz values of P' use the point found by projecting P on tangent surface of P' as shown in Fig. 5.
2) Calculate centroid C_p and $C_{p'}$ between two sampled range images.
3) Evaluate the remainder between each point cloud of P and P' and centroid, and compose H.
4) Perform SVD of H.
5) Evaluate transformation matrix with using U and V component.

When the transformation matrix is evaluated, we accumulate it to t_n. And after up-loading this transformation matrix in GPU memory, we create P_{new} by performing transformation of range image P. Created P_{new} will be read back to CPU memory, used as a new input value of P in the next repetition.

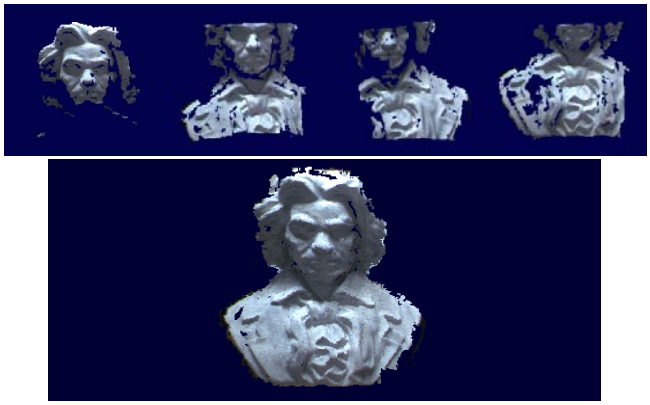
When sufficient iterations of projection and transformation are finished, accumulated t_n has the final transformation matrix to match the present range image to the previous frame. Accumulating the final transformation matrix to T_n, and multiplying this by range image I_n of the present frame, matching process for one frame is finished.

4. Experimental Results

The experiment was performed wth the assumption that anyone can scan a portable range sensor to generate 3-D models of some cultural assets or general scene. The range sensor is a stereo camera manufactured by PointGrey research in Canada. We obtaine range images from the stereo camera with the size of 320x240. Also Intel Core2 Duo E6750 and NVIDIA GeForce 8800 GTS are used respectively as a graphics device and a CPU unit for proposed registration system.

Fig. 6(a) is a result of an experiment performed at a distance of 0.5 meters from a statue of Beethoven with 40 range images. As shown in the figure, all range images aggregated together to form point clouds, which show consistency in color textures of real objects. Even though some noises are seen at the border of the statue, the reconstructed 3-D model looks successful.

The second and third results are obtained from indoor scenes. Fig. 6(b), 6(c) show the results. The distance between the subject and the camera is about 2 meter. From the left to right direction, we took 50 and 40 range images, respectively. From the results, we can find that all range images are registered as point clouds. In these figures, there are some holes, but those are due to the limitation of the sensor, rather than registration error. Table 1 shows registration time of the experiments.



(a) statue of Beethoven

(b) indoor scene 1

(c) indoor scene 2

Figure 6. the results of 3-D registration

Table 1. Registration Time

Subject	# of frames	# of total points	Registration time (sec / frame)
Beethoven	40	1,421,143	0.203
Scene 1	50	2,238,481	0.219
Scene 2	40	1,726,171	0.211

5. Conclusion

In this paper, we introduce a CUDA implementation of multi-view range image registration. By implmenting repeative algorithms in a 3D registration techqniue, we can use a hand-held 3D scanner to model real scenes. CUDA programming yields very fast 3D registration, 4~5 frames per second. To make more fast implementation, as described in introduction, fast transaction algorithm is needed. In the future study, we will improve the system to get registration performance of more than 10~15 frames per second. To do this, we plan to improve SVD, which is now performed in CPU, to executed in GPU level. Because the frequencies of upload and lead-back, which are occurred between CPU memory and GPU memory, processing time can be reduced if the SVD is executed in GPU.

Acknowledgement

This research has been supported by funding from the Agency for Defense Development in Korea.

References

[1] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," *Proc. 3D Digital Imaging and Modeling*, 145-152, 2001.

[2] Sudipta N Sinha, Jan-Michael Frahm, Marc Pollefeys and Yakup Genc, "GPU-Based Video Feature Tracking and Matching," *EDGE 2006, workshop on Edge Computing Using New Commodity Architectures*, Chapel Hill, May 2006.

[3] Qingxiong Yang, Liang Wang, Ruigang Yang, Shengnan Wang, Miao Liao and David Nister, "Real-time Global Stereo Matching Using Hierarchical Belief Propagation," *BMVC06*

[4] James Fung, Steve Mann, Chris Aimone, "OpenVIDIA: Parallel GPU Computer Vision," *Proceedings of the ACM Multimedia 2005* , Singapore, Nov. 6-11, 2005, pages 849-852

[5] Mark J. Harris, Greg Coombe, Thorsten Scheuermann, Anselmo Lastra, "Physically-Based Visual Simulation on Graphics Hardware," *Proc. SIGGRAPH / Eurographics Workshop on Graphics Hardware*, 2002.

[6] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüuger, Aaron E. Lefohn, and Timothy J. Purcell, "A Survey of General-Purpose Computation on Graphics Hardware," *In Eurographics 2005*, August 2005, pp. 21-51.

[7] Soon-Yong Park and Murali Subbarao, "An Accurate and Fast Point-to-Plane Registration Technique," *Pattern Recognition Letter*, 24 (16), pp. 2967-2976, Dec 2003

[8] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," *Proc. 3D Digital Imaging and Modeling, 145-152, 2001.*