A Nearest Neighbor Searches(NNS) Algorithm for Fast Registration of 3D Point Clouds based on GPGPU

Fangfang Wu[1, a], Fei Wang[1, b], Peilin Jiang[1, c], Chen Zhao[1,d], Jianhua Cheng[1, e]

[1]Xi'an Jiaotong University, Xi'an, Shaanxi Province, China

[a]email: w3112390009@stu.xjtu.edu.cn, [b]email: wfx@mail.xjtu.edu.cn,
[c]email: pljiang@mail.xjtu.edu.cn, [d]email: chenzhao@mail.xjtu.edu.cn,
[e]email: cjh24680@stu.xjtu.edu.cn,

Keywords: Registration; ICP; NNS; K-d tree; GPU

Abstract. For a large set of data points, registration is time-expensive. In order to deal with this problem, we propose an improved Nearest Neighbor Searches(NNS) algorithm for 3D point clouds registration based on Graphic Processing Unit(GPU). In order to maximize the storage of data points in the memory, we minimize the k-d tree element to decrease the consumption of memory. We use left-balanced median sort algorithm to compute root node and partition left and right sub-tree in the process of building k-d tree. During the search process, we use trim optimization to delete the sub-tree branch far away from the query node to avoid a huge amount of invalid computation. We use search stack to store search path, which decrease the computation. Experimental results indicate that an average processing speed for GPU is 15 times faster than that for Central Processing Unit(CPU) when the registration results are appropriate, and the acceleration ratio improves significantly while the number of point clouds increases.

Introduction

Nearest Neighbor Searches(NNS) algorithm aims to find the point closest to a query point among a set of n points. We measure distance in the Euclidean metric. The algorithm is commonly used in the domain of computer vision especially point cloud registration. However, registration algorithm with a huge amount of data consumes too long time.

Graphic Processing Unit(GPU) can be used for general purpose applications by using Compute Unified Device Architecture(CUDA). In CUDA programming, GPU is regarded as a parallel-processor that can execute a great number of threads simultaneously.

Traditional k-d tree-based NNS algorithm is difficult to be implemented on GPU because of its recursive nature. We present an improved NNS registration algorithm on GPU based on Arya's NNS algorithm. NNS algorithm plays an important role in computing correspondence of point clouds registration [1]. Iterative Closest Point(ICP) algorithm proposed by Besl and Mackay(1992) has been regarded as the most classic registration algorithm [2]. The algorithm iterates over three steps: (1)Compute correspondence by finding the nearest point. (2)Calculate optimal rigid transformation adopting Singular Value Decomposition (SVD) algorithm. (3)Transform the point sets. We implement step one and step three of ICP algorithm on GPU. Experiments indicate that the registration algorithm on GPU performs 15 times faster than that on CPU.

Related Work

The earliest GPU-based NNS algorithm was implemented by Brute Force(BF) method, in which the nearest point of each query point is calculated in parallel. There are many researchers who take advantage of high parallelism of BF method to accelerate NNS algorithm using CUDA. Ronzen et al(2008) partitioned 3D point clouds into cells and used a heap sort to search for the nearest point [3].

Compared with BF method, GPU-based NNS algorithm with advanced search structures works better. Singh et al(2007) presented a photon mapping framework to fit NNS in the Single

Instruction Multiple Data(SIMD) model, and used stack with k-d tree structure to search k nearest neighbor [4]. Zhou et al(2008) built a breadth-first GPU k-d tree for NNS and applied it to photon mapping and ray tracing [5].

GPU-based NNS algorithm is used to implement point clouds registration algorithm. Acceleration of ICP registration by GPU has made a great achievement. Qiu et al(2008) developed a GPU-accelerated nearest neighbor search algorithm for 3D point clouds registration and used a fixed length queue to search nearest point. If the queue is full, new candidate are discarded. Therefore, final result is an approximate nearest point not the true one [6]. Tamaki et al(2010) proposed CUDA-based implementations of Soft-assign and Expectation-Maximization (EM)-ICP for 3D point clouds registration. In order to overcome shortage of the memory on GPU, the EM-ICP algorithm adopted randomly down-sampling method, which reduces the accuracy of registration [7].

Computing correspondence consumes the majority time of ICP algorithm. We make a large improvement on the basis of Arya's NNS algorithm to compute correspondence of ICP algorithm and propose a GPU-based 3D point clouds fast registration algorithm. In this paper, we used a left-balanced median sort algorithm, a k-d tree with minimum element, an improved priority search method and the trim optimization approach. Experiments indicate that the efficiency of point clouds registration is improved to a large extent and the accuracy of registration is high.

GPU-Based Fast Registration

ICP algorithm aims to find an optimal rigid transformation consisting of a rotation R and a translation t, so that the shape point sets after transformation have minimum similarity measure with the model point sets. The registration with large point sets is computationally expensive. We present a GPU-based approach to accelerate ICP algorithm. See in Figure 1.

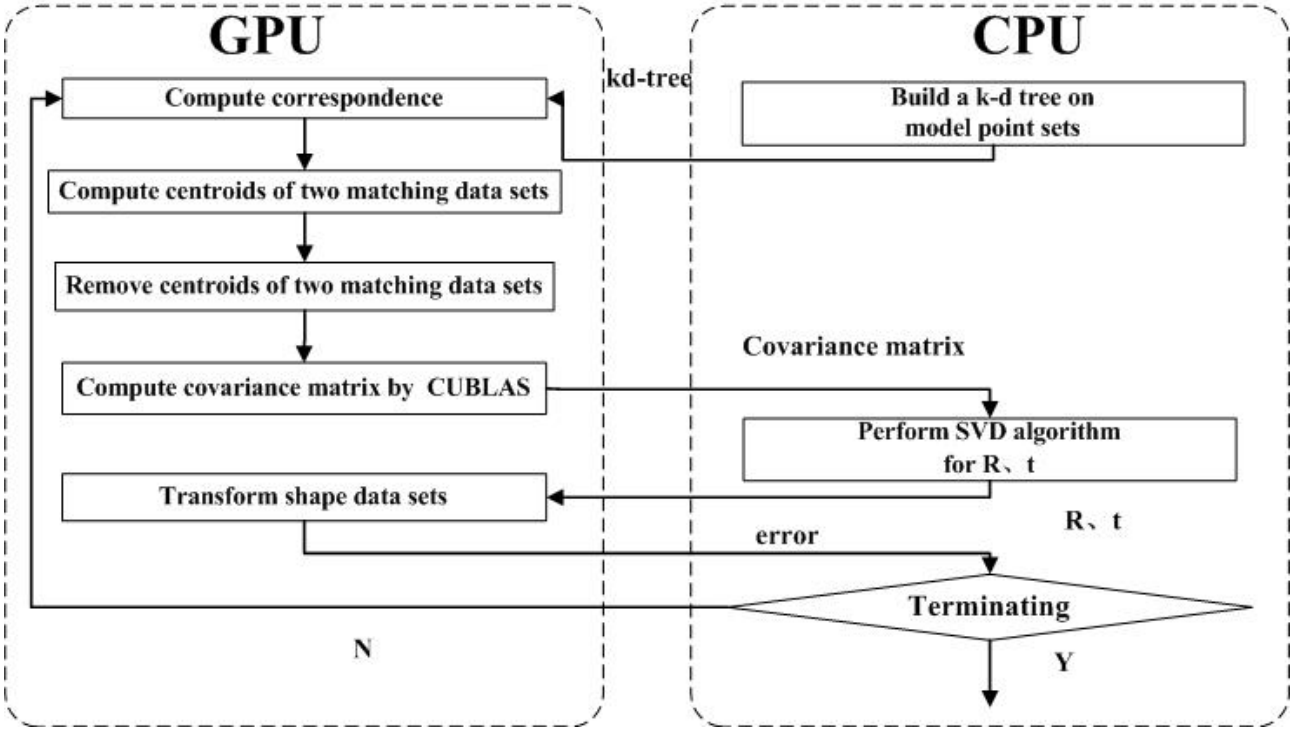


Fig.1. The GPU design scheme of ICP algorithm based on improved NNS

First, we construct k-d tree on the model point sets, then transfer it to the device terminal to compute correspondence in parallel. Second, we compute centroids of the two matching point sets by optimizing parallel reduction in CUDA and subtract their respective centroids. Then we compute covariance matrix by CUBLAS library and transfer it to the host terminal to calculate R and t by SVD algorithm. Third, we transfer the R and t to the device terminal to transform shape point sets. Finally, we transfer the Euclidean distance error to the host terminal to determine whether interrupt iteration. if the error is less than threshold, terminate the iteration. Otherwise recalculate the

correspondence in the device terminal.

GPU-based Improved NNS Algorithm

The correspondence computation consumes the majority time of a sequential CPU-based ICP algorithm. We improve Arya's NNS algorithm to compute correspondence of ICP algorithm and make it GPU-friendly. The major approaches include a left-balanced median sort algorithm, a k-d tree with minimum element, an improved priority search method and the trim optimization approach. These technologies optimize the process of building and searching k-d tree.

Building Array-based K-d Tree

We construct a left-balanced k-d tree on the mode point sets and store it in an array. In order to satisfy the coalescing of memory in CUDA, we use the Structure of Array(SoA). Figure 2 shows the array-based left-balanced k-d tree. We use a left-balanced median sort algorithm to compute root node and partition left and right sub-tree in the process of building k-d tree. In order to reduce the memory consumption of parameters of each thread, we minimize the k-d-tree element by only storing node value and its index in the model point sets. The split axis is cyclically selected in dimension <x,y,z,x,y,z,…>. Child pointers are replaced by index relationship between array elements(2i,2i+1).

Left-balanced median sort algorithm: First, calculate the left-balanced median position LMpos of n points. Then pick the vale m of median position of n points to partition point sets. The points in the left side is less than m and the right side is greater than m. If the position of m is equal to LMpos, then m is regarded as root node and its left and right sub-sets were treated as left and right sub-tree of the root. If not equal, the algorithm iterates into the child point sets which contains the position LMpos.

$$LMpos = 2^{\left\lfloor \log_2^{(n+1)} \right\rfloor - 2} + \min\left(2^{\left\lfloor \log_2^{(n+1)} \right\rfloor - 2}, n - 2 \times 2^{\left\lfloor \log_2^{(n+1)} \right\rfloor - 2} + 1 \right) \quad (1)$$

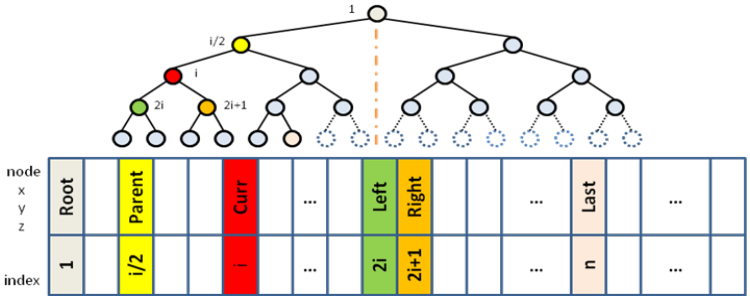


Fig.2. Array-based left-balanced k-d tree

GPU-based K-d Tree Searching

We take advantage of GPU's multi-threaded processing ability to accelerate NNS algorithm. We launch each point of shape point sets to a thread to compute nearest point in parallel.

The memory on GPU contains, from fastest to slowest, registers, shared memory, constant memory and global memory. In order to increase the speed of data transmission, we store k-d tree and shape point sets in the pinned memory. During the process of searching nearest point, k-d tree remains unchanged and is used only for comparison. So we store it in the constant memory. Because of the high accessing speed of the shared memory, we implement k-d tree searching in it. However, the shared memory is scarce resource, we only store query points, the value on the split axis of current k-d tree node compared with query point, and search stacks stored search path in the shared memory. Local variables are stored in the registers. The remaining variables all are stored in global memory, See figure 3.

During the searching process, we use the trim optimization to delete the sub-tree branch far away

from the query node. Each k-d tree node is associated with a rectangle in real d dimensional space and a hyper-plane orthogonal to the split axis. If the distance from query node to the current node's partitioning hyper-plane is greater than the current minimum distance, then delete the sub-tree branch of current node, otherwise push it into the search stack. Arya use a breadth-first search algorithm to search the nearest point and store search path in the priority search queue. He uses the heap sort algorithm in the priority search queue when new node is pushed into the queue, which is computationally expensive. We use search stacks to store the search path and adopt a depth-first search algorithm to query the nearest point. The depth-first search algorithm first explores the node smaller to query point while storing the remaining one in the search stack. The minimum distance decreases as we traverse k-d tree. So we can delete node far away from query node by the trim optimization during the backtracking process, which avoids a huge amount of invalid computation.

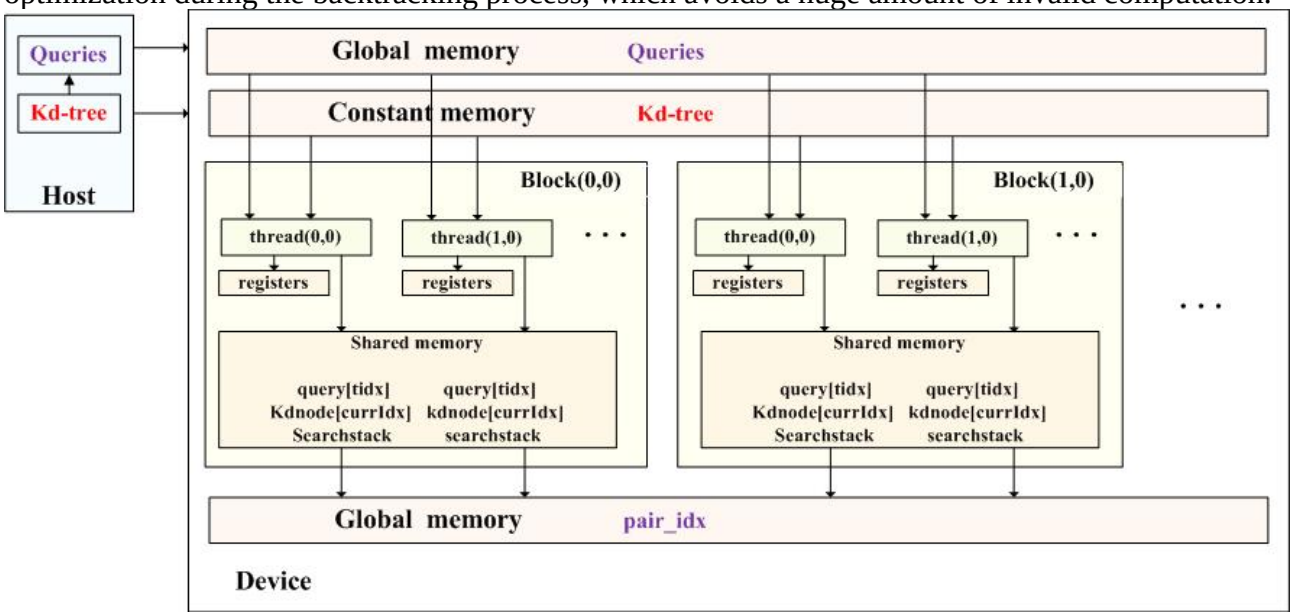


Fig.3.The memory distribution of GPU-based NNS algorithm

Experiments

All experiments we performed on a quad-core processor. The configuration is shown in table 1.

Table 1: Configuration of processor

CPU	GPU
Intel(R) Xeon(R) E5-1603	NVIDIA GeForce GTX660
2.8 GHz	1.12 GHz
8 GB	2 GB
Visual Studio 2010(64 bit)	CUDA 5.5

The experimental data is four groups of Stanford bunny point sets. See in table 2.

Table 2: The experimental point sets

Type	The number of points			
bunny	10503	20417	30331	40256

1）The running time comparison of CPU-based ICP and GPU-based ICP：

For each group of point sets, we respectively implement the CPU-based ICP and GPU-based ICP algorithm and compare their running time.

Table 3: Running time of bunny point sets

	10503		20417		30331		40256	
	iter	time(s)	iter	time(s)	iter	time(s)	iter	time(s)
CPU-ICP	13	0.195000	16	0.456673	14	0.568258	15	0.944537
GPU-ICP	13	0.017295	16	0.030512	15	0.037422	19	0.049913

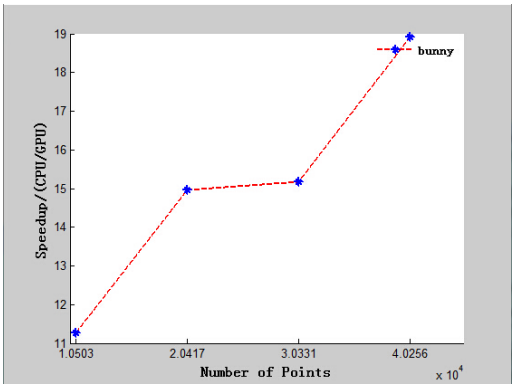


Fig.4.The speedup of ICP algorithm

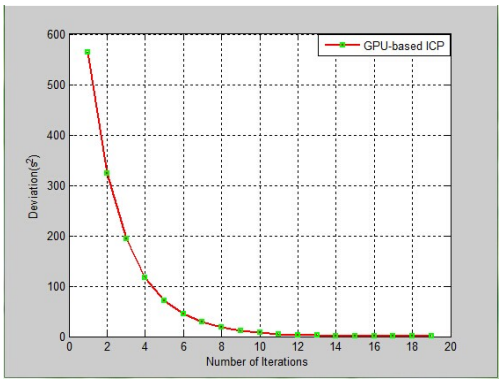


Fig.5. Convergence tests of ICP algorithm

Figure 4 shows the registration speedup(CPU/GPU) ratio of ICP algorithm. The improved ICP algorithm runs faster on GPU than CPU. The algorithm's speedup ratio increase as the data size grows(the speedup is the ratio between the running time of CPU-based ICP and that of GPU-based ICP). Because the data transferring between CPU and GPU consumes much time when the number of points is small. It weaken the acceleration effect. For a large number of point sets, the algorithm can take full advantage of multi-thread resources to acquire a better acceleration.

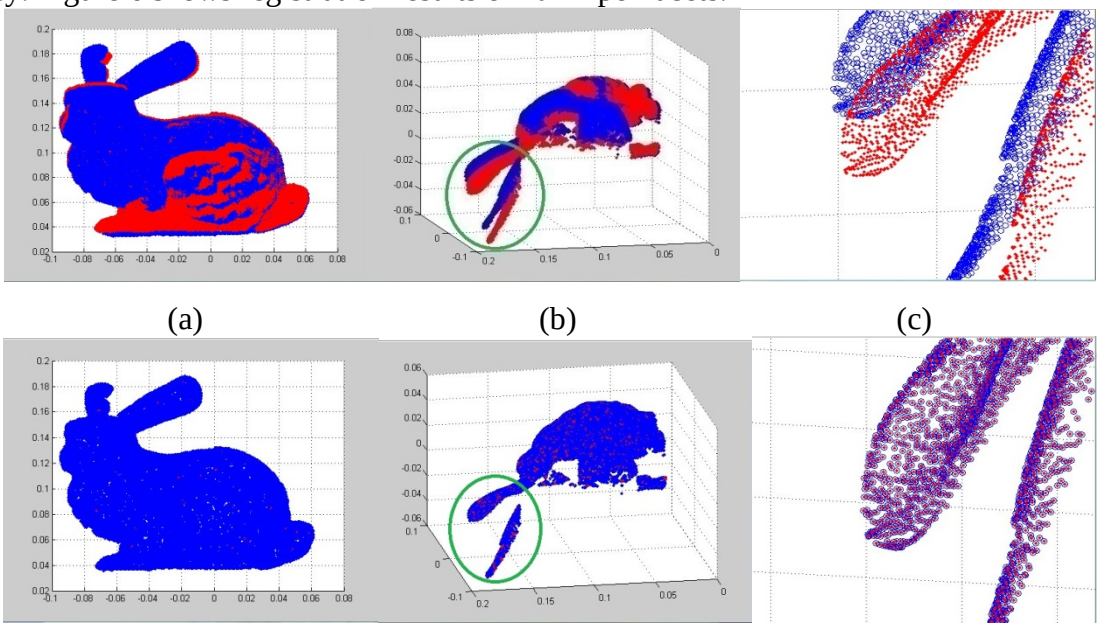
2）The registration error comparison of CPU-based ICP and GPU-based ICP：

For each group of point sets, we respectively implement the CPU-based ICP and GPU-based ICP algorithm and compare their error. The total error is the summation of the Euclidean distance error of each point .

Table 4: Total error of bunny point sets

error	bunny			
	10503	20417	30331	40256
CPU-ICP	0.000001	0.000000	0.005457	0.006248
GPU-ICP	0.000001	0.000000	0.005480	0.006248

Table 4 shows the matching accuracy of ICP algorithm based the improved NNS on GPU is equal to that on CPU. Figure 5 shows the registration iteration error curve of 40256 point sets. We can see the error monotonically decreases in the iterative process, which means good matching accuracy. Figure 6 shows registration results of 20417 point sets.



(a) (b) (c)

(d) (e) (f)

Fig.6. The registration results of GPU-based ICP algorithm. (a), (b) and (c) show the figure before registration. (d), (e) and (f) show the figure after registration. (a) and (d) show the front view of registration figure. (b) and (e) show the side view of registration figure. (c) and (f) show the detailed registration figure.

Conclusion

In this paper, we introduce a NNS-based fast point clouds registration algorithm on GPU. Our implementation on NVIDIA GeForce GTX660 systems achieve a speedup of about 15 times compared with Intel(R) Xeon(R) E5-1603 on average. In order to enhance the efficiency of search, we propose an improved priority search method, which uses a search stack. We also adopt the trim optimization approach for the efficient search. In addition, we introduce a left-balanced median sort algorithm to accelerate building k-d tree. By minimizing the k-d tree element, we decrease the consumption of memory.

Acknowledgement

In this paper, the research was supported by National High Technology Research and Development Program of China (Project No. 2013AA014601) and National Science and Technology Pillar Program (Project No. 2013BAH62F03-3).

References

[1] Arya S., Mount M. Algorithms for Fast Vector Quantization [C], IEEE Proceedings of Data Compression Conference, IEEE Computer Society Press, 1993.381-390.

[2] Paul J. Besl, Neil D. McKay, A Method for Registration of 3-D Shapes [J], IEEE Transactions on Pattern Analisys and Machine Intelligence, 1992. 239-256.

[3] Rozen T, Boryczko Alda W., GPU Bucket Sort Algorithm with Applications to Nearest-Neighbor Search [C], Journal of the 16th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, 2008.

[4] Singh S., Faloutsos P., SIMD Packet Techniques for Photon Mapping [C], In: Proc.of the IEEE/EG Symposium on Interactive Ray Tracing, 2007. 87–94.

[5] Zhou, K., Hou, Q., Wang, R., Guo B., Real-time KD-Tree Construction on Graphics Hardware [P], ACM SIGGRAPH Asia 2008, 2008.10.

[6] Qiu D., May S., Nuchter. A., GPU-accelerated Nearest Neighbor Search for 3D Registration [C], 7th International Conference on Computer Vision Systems [C], 2009.194-203

[7] Tamaki T, Abe M, Kandea K. Softassign EM-ICP on GPU [C]. Proceedings of the 2010 1st International Conference on Networking and Computing(IEEE2010), 2010.179-183.