FOUNDATIONS OF COMPUTING AND DECISION SCIENCES
Vol. 42 (2017) No. 3
ISSN 0867-6356
DE GRUYTER OPEN DOI: 10.1515/fcds-2017-0010 e-ISSN 2300-3405

PARALLEL RANSAC FOR POINT CLOUD REGISTRATION

Daniel KOGUCIUK *

Abstract. In this paper, a project and implementation of the parallel RANSAC algorithm in CUDA architecture for point cloud registration are presented. At the beginning, a serial state of the art method with several heuristic improvements from the literature compared to basic RANSAC is introduced. Subsequently, its algorithmic parallelization and CUDA implementation details are discussed. The comparative test has proven a significant program execution acceleration. The result is finding of the local coordinate system of the object in the scene in the near real-time conditions. The source code is shared on the Internet as a part of the Heuros system.

Keywords: RGBD, point clouds, registration, CUDA

1. Introduction

One of the basic tasks in the perception of mobile robots is point cloud registration, which can be applied to target two principal tasks, wherein the first one corresponding the point clouds obtained from different views, resulting with one bigger and more accurate representation [10]. The second task related to registration is matching of a model into its instance on the scene is also called registration and it defines the metrical coordinate system of the instance giving the opportunity to this object manipulation. In the figure (1) is shown a result of model registration into the observed scene [11].

One of the most popular registration methods is a heuristic algorithm RANSAC. It is a widely used method which in more than thirty years after its publication revealed many derived methods used in different fields of science. The method is well-fit in the point cloud registration reality because it can find a valid solution even

*Faculty of Mechatronics, Warsaw University of Technology, Warsaw, Poland, d.koguciuk@mchtr.pw.edu.pl

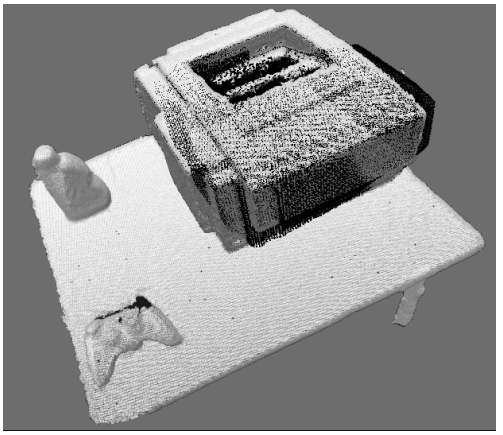

Figure 1. Result of registering of a printer model into the observed scene [11].

with a large number of outliers – points not fitting the model, because of sensor noise, scene occlusions etc. However, using this method of registering objects with tens of thousands of points is occupied by large processing time, even counted in seconds.

There was much effort put in scientific literature to make RANSAC more real-time fitting. For example, authors in [14] introduce a constant time budget policy. The algorithm is forced to propose the best solution after a specified amount of time. However, a big disadvantage is that it is possible that a better solution would be found after longer time of operation. Other works present parallel versions of RANSAC algorithm [14, 1], but the implementation strongly depends on the given mathematical model of the application and presented solutions were not directly applicable in the point cloud registering.

In this paper, the methodology and implementation details of RANSAC parallelization using the CUDA framework for point cloud registration are presented. Starting with a short description of basic serial implementation and discussion about newest improvements of the method in section 2, going through the design of the parallel approach in section 3 and details of its implementation in section 4 and ending up with comparative tests in section 5. The paper is summarized in the last section with the plans for the future work. The source code of the parallel RANSAC implementation is available under an open-source license as a part of the Heuros project [9].

2. RANSAC algorithm

2.1 Basic version

RANSAC (Random Sample Consensus) algorithm is, in general, an iterative method to estimate parameters of the mathematical model from data that can contain outliers

[6]. It is a non-deterministic algorithm producing a reasonable result only with a given probability, dependent on how many iterations were done. It is widely used in different fields of computer science, especially in artificial intelligence and computer vision. This work, however, is focused on point cloud registering.

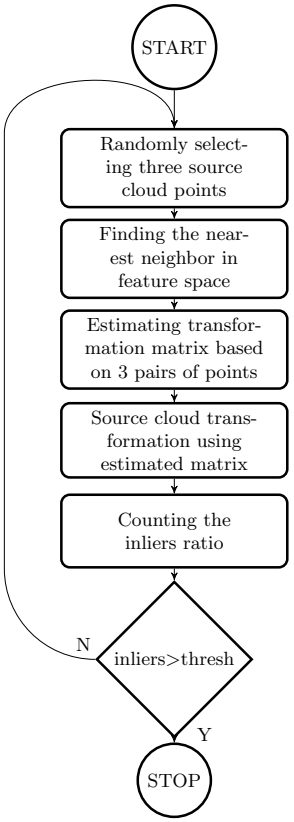

Figure 2. Flow chart of basic RANSAC algorithm for point cloud registration.

The first step of each iteration $k < N$ (N is the maximum iterations number) is randomly selecting three points from source cloud S. The next one has to find three corresponding points in the target cloud T and it is done by calculating point descriptors for all points in the target cloud and finding the most similar ones to those selected from source cloud. According to [7] the best relation of descriptor uniqueness to required computation time has FPFH (Fast Point Feature Histograms) [15] and it will be used in this work.

The third step is the estimation of the transformation matrix with given three point pairs. It is a known and well-described linear algebra problem [8] and will not be discussed in this paper. Next, one should transform source cloud with estimated matrix and check how many of the target points have a point from source cloud in the surrounding region (usually it is defined as one and a half of point

density in the clouds). To find the closest point in the metrical space an octree structure is used within PCL (Point Cloud Library) library [16]. The algorithm stops with the iteration in which inliers ratio is greater than a certain threshold.

The RANSAC algorithm has both advantages and disadvantages strictly connected with its heuristic and non-deterministic nature. It can produce an acceptable solution even with a large number of outliers, which means it can overcome sensor noise, occlusions, segmentation errors etc.

A big asset of the algorithm is no initial pose knowledge needed for registered clouds. Constraining the conditions in which algorithm can be used makes the application area of this method really wide, along with the simplicity of implementation, explains why this is one of the most frequently used algorithms for point cloud registration.

On the other hand, there is no guarantee of achieving a reasonable solution and the time of finding it is far from real-time, which we desire in mobile robotics. The critical point here is checking the estimated transformation quality. It can be achieved by transforming the whole source cloud into the scene in each iteration.

2.2 Selected extended versions

RANSAC itself has become an algorithmic legend and there are several modifications in the literature aiming at making it faster and better. In [14], there are two types of featured improvements: non-uniform sample selecting in the very first step of each iteration and hypothesis validation optimization. The following part would contain the brief description of one representative of the first group and three of the second.

2.2.1 Lo-RANSAC

One of the assumptions of terminating the iterations in the basic version is an equal probability of all source cloud points having a corresponding point in the target cloud. This is obviously not true, because of the simple scene occlusions, some of the source cloud points would never have a corresponding point in the target cloud. That is why authors in [5] suggest an optimization where the constant hypotheses count (~ 20) is generated using the inliers in the previous step. Authors have shown such modification results in faster finding of terminal criteria of the algorithm.

2.2.2 Polygon test

It is one of the methods that allow rejecting of the hypothesis even before the transformation matrix estimation, introduced in [3]. It compares the polygons created with the source and the target cloud individually. If the ratio of the corresponding sides of the triangles is not in the acceptable interval, the hypothesis is rejected and the algorithm should proceed to the next iteration. This approach is based on the assumption that both source and target objects are rigid body type.

2.2.3 $T_{d,d}$ test

Before the transformation of the whole source cloud, authors of [4] suggest performing a check of a randomly picked subset of d points of a model cloud(where $d \ll |S|$). The inlier ratio of remaining points could be checked right after $T_{d,d}$ test - the probability of rejecting valid hypothesis is small, but gained time profit can be noticeable.

2.2.4 Visibility test

In the previous efforts [11], author of this paper was working on an early hypothesis rejecting the using of knowledge about the visible and hidden areas of the observed scene. If the target cloud is captured by RGBD sensors such as a Microsoft Kinect device, it is directly given which parts of the scene are empty, where are the object boundaries and where is the hidden area. Modification introduces different ratio of score counting: only points in the empty space between the observed object and the camera are treated as outliers because it is not known if the points in the hidden area are in- or outliers.

3. The project of parallel RANSAC

Considering the best design of the parallel algorithm, there is a need of making some assumptions about its application area. This work is directed towards matching the model into the real world scene and the problem of object segmentation is not considered here. Presented algorithm is not a complementary object recognition tool, it should rather be used as the last step of the recognition process, where other methods give information about objects expected in the scene and their estimated position. Consequently, the target is to find a metrical description of the object pose, for example for grasping purposes. This section describes both the idea and the implementation of the parallel RANSAC algorithm.

In the traditional RANSAC algorithm, the nearest neighbor of every randomly selected point is being determined on an ongoing basis, because there is a possibility of finding the right transformation between source and target clouds in every single iteration. In the parallel version, there is no way of terminating the computations before all transformations are checked because all hypotheses are being processed at once (from the theoretical point of view). This is a two-edged side-effect of the parallelization: on the one hand there is the possibility of choosing the best solution from all generated hypotheses, but on the other hand, it is not possible to stop computations earlier.

It is worth pointing out that in the parallel version the number of generated hypotheses would be counted in thousands, maybe even in tens of thousands. This is why the same point from the model cloud can be selected in more than one hypothesis and thus finding the nearest neighbor in the feature space in every thread independently would be inefficient. The author suggests finding the nearest neighbors

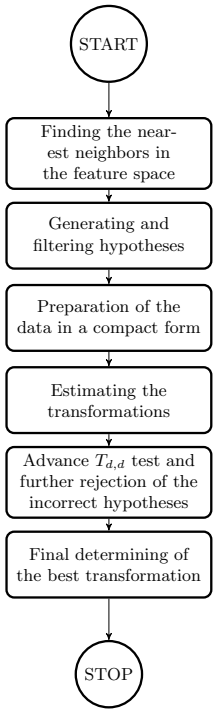

Figure 3. The flow chart of the parallel RANSAC algorithm for point cloud registration.

for all points from the model cloud once before the proper RANSAC algorithm is done. Like it is shown in the figure 3, finding the nearest neighbors in the feature space is the first step of the parallel version of the RANSAC algorithm.

With the available vector of indices of the nearest neighbors for every point in the source cloud, it is possible to move along with the proper RANSAC routine. The beginning looks just like the basic, serial version of the algorithm and consists of randomly selecting the 3 points of the source cloud and with the 3 according points from the target cloud, it is possible to perform early hypothesis rejection with the polygon (triangle to be exact) by similarity mechanism described in 2.2.2.

After the hypotheses filtering based on triangle similarity described in the previous section, the actual number of generated transformations is much smaller than N. In fact, about 99% of the hypotheses are filtered out here and further data processing without compacting the resulting vector would be an unjustified waste of CUDA computational power. This step compacts the vector of sparse data into a smaller, dense form.

It is worth to consider, that there is no need of estimating the transformation based on those 2 triples of points earlier because the vast majority of them would be filtered out. This is the time to do it.

At this point, there are many fewer hypotheses remaining. However, simply check-

ing all of them would be time-consuming, especially when it comes to the applying of the extended tests described in the last section. As described in 2.2, $T_{d,d}$ test seems to be simple and fast heuristics to the transformation evaluation based on only a few from thousands of source cloud points. After the $T_{d,d}$ test the remaining hypotheses should compact again.

In the last step, all remaining transformations are considered as valid solutions, but it is possible to choose the best one. This can be done using the same mechanism as in the basic serial RANSAC algorithm version.

4. Implementation

In this section, the implementation and the performance modeling and profiling would be discussed. Described mechanisms should work with most of modern CUDA-enabled GPUs, but probably would not achieve the best performance, because often not only the parameters but also the approach are strongly dependent on the particular GPU model or architecture. In the following section, the design of every part of the parallel RANSAC algorithm for GPU Kepler architecture is presented. The performance tests were carried out on a GTX660 model.

4.1 Finding the nearest neighbors in the feature space

Considering the CUDA programming model [13] and the evaluated amount of tens of thousand points in the clouds of household devices which we want to register, it seems a good idea to assign the task of finding nearest neighbors of one point of the source cloud to one block of CUDA threads. It is dictated by the necessity of communicating threads among themselves to find minimum distance value.

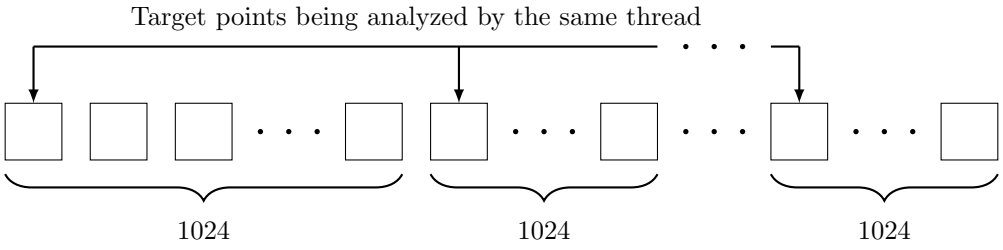

Figure 4. Finding the nearest neighbor scheme of one source cloud point in the target cloud containing more than 1024 points.

This must carry the CUDA kernel threads number as a minimum of both: magnitude of the target cloud and a maximum number of threads per block (1024 for the Pascal CUDA microarchitecture): $min(|T|, 1024)$. If the scene contains more than 1024 points, each thread should compute the distance from the common source point

to several target points to cover every possibility. It is shown in figure 4. If the target is smaller than 1024 points, every thread computes the distance to only one target point.

After above-described kernel, there are $min(|T|, 1024)$ distances of target points to one source point. The operation of finding the minimum value from a set of values is a parallel operation primitive called *reduce*. It is a well-known element of CUDA programming [2] and will not be discussed here.

This is, without doubt, the weakest part of the algorithm. This is a computation-intensive, since the distance between every pair of a source-target points has to be calculated. The kernel uses 42 registers for each thread (up to 43008 registers for each block) resulting in only one active block per SM [13] leading to only 50% of occupancy. The author also tried to lower the registers count for each thread, but it leads to much grater computing load resulting in much longer execution time of the kernel.

4.2 Generating and filtering hypotheses

The first attempt to randomly choose source cloud points was made with the cuRAND library [13], but to generate 3 random source cloud indices it uses 38 registers for each thread, which leads to poor occupancy and performance. It randomly turns out, that choosing those indices using CPU and downloading them to the GPU is much faster.

Having the source cloud indices and corresponding target points, one can perform a polygon test where the output information is irrespective of hypothesis passed it or not. There are no a priori imposed kernel launch parameters so - minding overall good parallel programming practices, suggested values are maximum threads per block (1024) arranged in 16 blocks of threads. This is the assumptive value of $N = 16 \cdot 1024 = 16384$ initial generated hypotheses. One thread uses 28 registers on GTX660, which allows achieving 100% theoretical and 82.9% actual warp occupancy.

4.3 Preparation of the data in the compact form

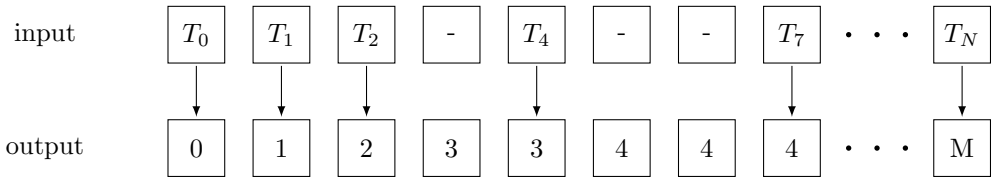

Figure 5. Exclusive scan operation: first valid transformation T_0 has been assigned address 0 in the output vector, second valid transformation T_1 has address 1 and so on up to M valid transformations.

This part of the algorithm is about reducing sparse vector into the compact form and it can be performed using two CUDA primitives. In the first step assigning of the addresses in the compact vector for every valid transformation (figure 5) can be done and it is called (*exclusive scan*) for vectors smaller than a maximum number of threads in the block and (*segmented exclusive scan*) for the greater ones. Second step is moving the proper data (source cloud indices) in the calculated position in the output, dense vector and it is called *compact* (figure 6). Both are well-known CUDA primitives, that can be found in the literature [17].

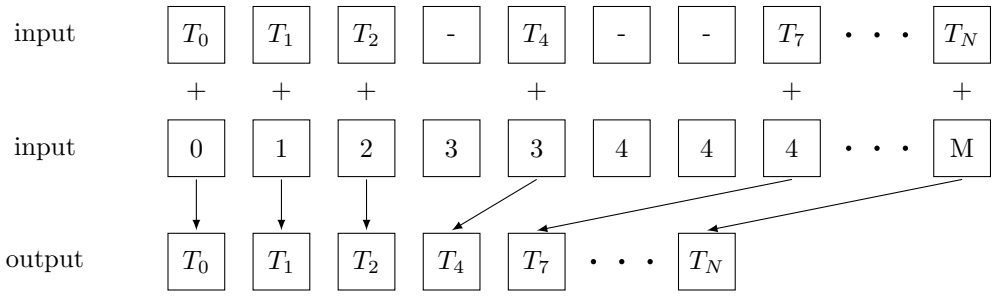

Figure 6. Compact operation scheme.

4.4 Estimating the transformations

The polygon similarity threshold value is selected to reject as much as possible of false positives, leaving the transformations number less than 1024. This is dictated by a maximum number of threads in one block to make this operation CUDA-optimal. Estimating is done using standard linear algebra approach [8].

4.5 Advance $T_{d,d}$ test and further rejection of the incorrect hypotheses

The $T_{d,d}$ test should be applied to all remaining hypotheses at once and, having in mind the CUDA programming model, it can be concluded that the number of points taken into the test should be a multiple of the warp threads number: $d = k \cdot 32$. Here every thread is transforming source point cloud with estimated transformation matrix and every 32 threads are checking the same hypothesis ($k = 1$). Kernel launch parameters are the following: 1024 as the maximum number of threads in the block, each block checks 32 hypotheses so there should be $\lceil \frac{M}{d} \rceil = \lceil \frac{M}{32} \rceil$ blocks.

After the $T_{d,d}$ test, the result should be compacted again, as after the triangle test, however the input conditions are different. Earlier the vector was large (thousands of elements) and sparse, here we have only a few hundred of them, and only a few would pass the test. That is the reason why author decides to use simple *atomic add* rather than *scan* and *compact* operations [13].

4.6 Final determination of the best transformation and algorithm parameters discussion

Presented parallel RANSAC algorithm has many parameters which influence the quality and time of finding the final transformation. First of all the number of iterations has a different meaning for both versions: for serial, it is only the upper time limit of the algorithm and usually does not affect the quality of the result, however in the parallel version it directly impacts this quality. In the serial one, the first eligible transformation is found and the algorithm stops whereas in the parallel one the best hypothesis is being selected.

The Second important parameter is the polygon similarity threshold. The value of 0.2 (so only 20% of triangle dissimilarity is being accepted) results in about 99% of hypotheses being rejected, which is discussed more in the next section. Subsequently, the radius of the sphere, where points are considered as inliers, also affects the final result. Taking too big value could result in inaccuracy of the estimated transformation, too small value could lead to not finding a result at all. In the current implementation, the value of 7.5 *mm* as a sphere radius is set, which is one and a half leaf size of the voxel grid filter applied to the source and target clouds in the beginning (see the section 5 for more details).

Lastly, another significant parameter is the inlier ratio threshold. It is hard to estimate a universal value which would be appropriate in every scenario, so it should be purposefully chosen in every application. If 80% of target points are inliers, the estimated transformation is considered as a correct one.

It is worth pointing out that all of the discussed parameters used in the parallel implementation are also needed in the serial version of the RANSAC algorithm.

5. Comparative tests

Tests consist of a comparison between parallel and basic serial version of the RANSAC algorithm, using the public *3D Model-based Object Recognition and Segmentation in Cluttered Scenes* database presented in [12]. From the registration perspective, the most valuable feature of this database is containing both precise clouds of different test models and the real scenes with objects occluding each other viewed from only one perspective. One of the disadvantages of the database is that the data was acquired using a Minolta Vivid 910. This camera is registering only the shape of the objects without RBG information.

The original test scenes contain several objects affecting each other, but they were segmented out for registering purposes and used as an independent target cloud. The target here is registering the model cloud into the observed object on the scene and, from the RANSAC algorithm point of view, segmentation of the target cloud is not significant.

Tests were performed for two different objects. Originally model and target point clouds contain about one hundred of thousands of points and since in mobile robotics

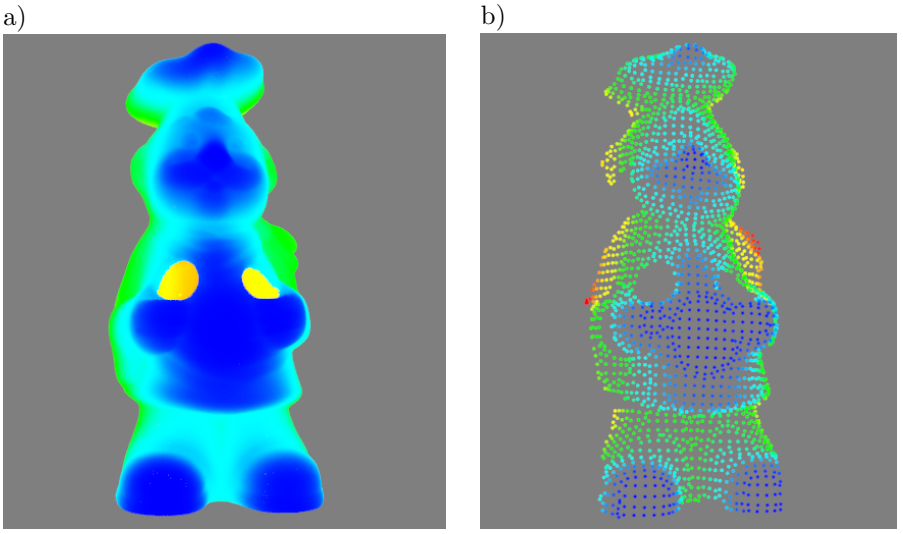

Figure 7. a) Example source and b) target cloud of one of the object used for testing the algorithms.

most common are Kinect-like sensors, which have much fewer points, both clouds need to be down-sampled with voxel grid filter with a leaf value of 5 mm. This reduces the number of points to values of about five thousand and two and a half thousand respectively. For both clouds, the FPFH features would be calculated for the 25 mm radius (about 70 points). $T_{d,d}$ and final transformation tests require finding the nearest neighbors in 3D space and to make this possibly fast, an octree would be built for the target cloud. All of the above auxiliary methods were implemented with PCL [16] functionality (data input, clouds filtering and RANSAC algorithm in serial version with the additional *poly* test). All tests were performed using i5-4570 processor and GTX 660 GPU. The comparison was made against the serial program running on a single CPU (there is no open implementation of multi-core CPU RANSAC algorithm neither in PCL nor anywhere else according to the author's knowledge).

Table 1. The comparison between serial (SR) and parallel (PR) implementation for chef model registering in three different test scenarios for 10 iterations.

	chef_1	chef_2	chef_3
Inlier percentage number (SR)	72.6 ± 0.1	70.9 ± 0.1	93.8 ± 0.1
Inlier percentage number (PR)	$\mathbf{88.1 \pm 8.4}$	$\mathbf{90.0 \pm 6.8}$	$\mathbf{95 \pm 2.3}$
Processing time (SR) [ms]	3242 ± 45	3691 ± 32	3504 ± 31
Processing time (PR) [ms]	$\mathbf{12.9 \pm 4.1}$	$\mathbf{12.4 \pm 3.2}$	$\mathbf{9.2 \pm 4.1}$
Time of finding nearest neighbors (PR) [ms]	$\mathbf{77.1 \pm 2.3}$	$\mathbf{77.0 \pm 2.1}$	$\mathbf{76.3 \pm 1.7}$
Maximum number of hypotheses (SR/PR)	16384	16384	16384
Number of hypotheses after *poly* test (IR)	566 ± 32	539 ± 13	449 ± 22
Number of hypotheses after $T_{d,d}$ test	4.3 ± 1.9	4.4 ± 1.4	3.1 ± 2.1

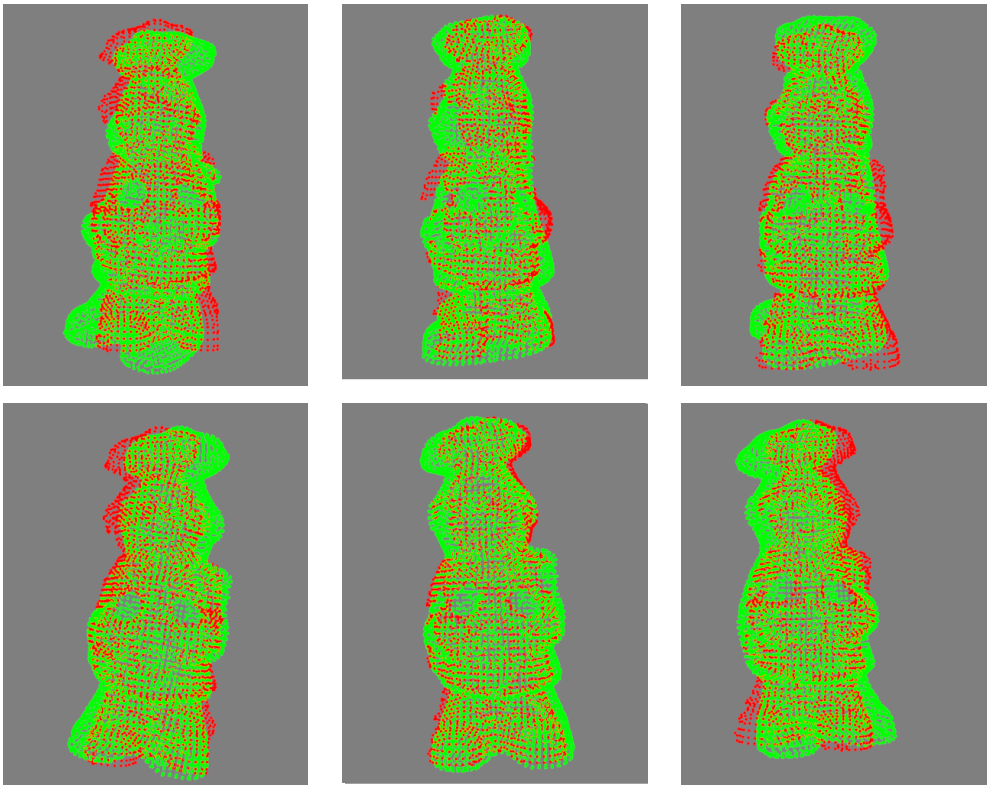

Figure 8. Registered chef model cloud (green) into the target cloud (red) for three different test scenarios using serial (top) and parallel (bottom) RANSAC implementation,

In figure 8, three different test scenarios with the chef figure are presented for both serial (top) and parallel (bottom) version of RANSAC algorithm. For every scenario a test of 10 iterations was made, the results were averaged and collected in table 1. In all test scenes, the result found with the parallel implementation is a few percent better than in the serial one. This could be caused by the different natures of maximum iteration number for both versions - the parallel one is choosing the best result from all passing the final test.

The time of processing is very important – about 90 ms for the parallel implementation and about 3500 ms for the serial one. This is a real performance boost, allowing to register clouds in almost unnoticeable time – at least for human. The same tests were performed for the chicken model (figure 9) with three different target scenes for both serial (top) and parallel (bottom) implementation.

In this case, both implementations find the result with almost the same score (about 75%). Time difference between serial and parallel version are compared to the previous scenario.

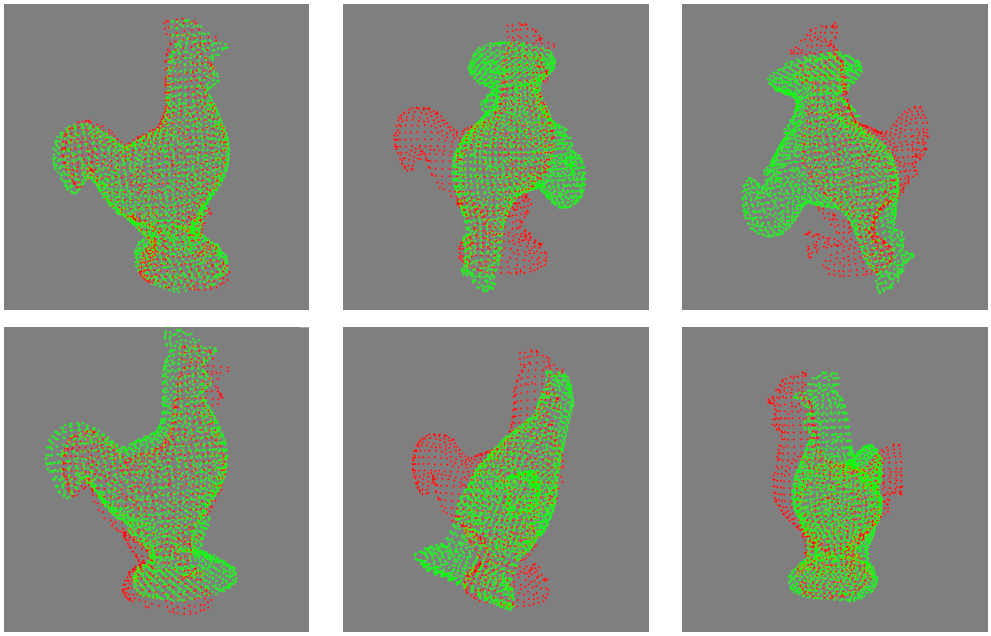

Figure 9. Registered chicken model cloud (green) into the target cloud (red) for three different test scenarios using serial (top) and parallel (bottom) RANSAC implementation.

It is worth pointing out that the number of hypotheses after each step in the parallel implementation is drastically falling. After the *poly* test, about 97% of the hypotheses are rejected, leaving about few hundred to be checked with $T_{d,d}$ test, where only a few are considered as a valid solution.

6. Conclusions

In this paper the parallel implementation of the RANSAC algorithm for point cloud registration using CUDA was presented. It is a very important task for everyday object manipulation purposes by social robots – it allows to set a reference between the object and the global coordinate system.

The design is trying to fit CUDA architecture best practices to exploit the best possible performance using CUDA warps, operation primitives, memory hierarchy, etc. The implementation was confronted with the serial version of the algorithm available in PCL using public "3D Model-based Object Recognition and Segmentation in Cluttered Scenes" database.

The parallel approach to the RANSAC algorithm not only decreases the processing time of about 30 times, but also presents a qualitative improvement by proposing multiple good hypotheses. Using parallel computing on high-end GPGPU brings new

Table 2. The comparison between serial (SR) and parallel (PR) implementation for chicken model registering in three different test scenarios for 10 iterations.

	chicken_1	chicken_2	chicken_3
Inlier percentage number (SR)	74.9 ± 0.1	77.7 ± 0.1	77.0 ± 0.1
Inlier percentage number (PR)	$\mathbf{90.1 \pm 2.9}$	$\mathbf{87.7 \pm 9.5}$	$\mathbf{84.5 \pm 7.4}$
Processing time (SR) [ms]	2139 ± 15	1030 ± 11	1462 ± 16
Processing time (PR) [ms]	$\mathbf{8.2 \pm 4.8}$	$\mathbf{6.4 \pm 0.6}$	$\mathbf{7.0 \pm 1.0}$
Time of finding nearest neighbors (PR) [ms]	$\mathbf{27.0 \pm 0.1}$	$\mathbf{27.1 \pm 0.3}$	$\mathbf{27.0 \pm 0.1}$
Maximum number of hypotheses (SR/PR)	16384	16384	16384
Number of hypotheses after *poly* test (IR)	192 ± 21	241 ± 10	227 ± 15
Number of hypotheses after $T_{d,d}$ test	5.0 ± 2.2	5.4 ± 1.1	6.3 ± 2.0

possibilities of point clouds registering in almost real-time conditions.

GPGPUs are also the disadvantage of the presented method. It means a necessity of mounting high power processing units on the mobile robot platforms, which drastically decreases the operational time of agents. However, progressing technological development will finally meet those expectations with embedded mobile platforms.

References

[1] Alehdaghi M., Esfahani M. A., and Harati A. Parallel ransac: Speeding up plane extraction in rgbd image sequences using gpu. In *Computer and Knowledge Engineering (ICCKE), 2015 5th International Conference on*, pages 295–300, Oct 2015.

[2] Blelloch G. E. *Vector Models for Data-parallel Computing*. MIT Press, Cambridge, MA, USA, 1990.

[3] Buch A., Kraft D., Kamarainen J.-K., Petersen H., and Kruger N. Pose estimation using local structure-specific shape and appearance context. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2080–2087, May 2013.

[4] Chum O. and Matas J. Randomized ransac with t d,d test. In *IMAGE AND VISION COMPUTING*, pages 448–457, 2002.

[5] Chum O., Matas J., and Kittler J. Locally optimized ransac. In Michaelis B. and Krell G., editors, *Pattern Recognition*, volume 2781 of *Lecture Notes in Computer Science*, pages 236–243. Springer Berlin Heidelberg, 2003.

[6] Fischler M. A. and Bolles R. C. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[7] Hänsch R., Weber T., and Hellwich O. Comparison of 3D interest point detectors and descriptors for point cloud fusion. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 57–64, Aug. 2014.

[8] Hartley R. and Zisserman A. *Multiple view geometry in computer vision*. Cambridge University Press, Cambridge, 2003. Choix de documents en appendice.

[9] Heuros 3D object recognition system. https://bitbucket.org/rrgwut/heuros. Accessed: 04-05-2015.

[10] Izadi S., Kim D., Hilliges O., Molyneaux D., Newcombe R., Kohli P., Shotton J., Hodges S., Freeman D., Davison A., and Fitzgibbon A. Kinectfusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, pages 559–568, New York, NY, USA, 2011. ACM.

[11] Koguciuk D. and Harasymowicz-Boggio B. Wykorzystanie obszarów nieznanych w dopasowywaniu chmur punktów. *Prace Naukowe Politechniki Warszawskiej. Elektronika*, pages 267 – 276, 2014.

[12] Mian A., Bennamoun M., and Owens R. Three-dimensional model-based object recognition and segmentation in cluttered scenes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(10):1584–1601, Oct 2006.

[13] NVIDIA Corporation. NVIDIA CUDA C programming guide, 2015. Version 7.0.

[14] Raguram R., Frahm J.-M., and Pollefeys M. A comparative analysis of ransac techniques leading to adaptive real-time random sample consensus. In Forsyth D., Torr P., and Zisserman A., editors, *Computer Vision - ECCV 2008*, volume 5303 of *Lecture Notes in Computer Science*, pages 500–513. Springer Berlin Heidelberg, 2008.

[15] Rusu R., Blodow N., and Beetz M. Fast point feature histograms (fpfh) for 3d registration. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 3212–3217, May 2009.

[16] Rusu R. B. and Cousins S. 3d is here: Point cloud library (pcl). In *International Conference on Robotics and Automation*, Shanghai, China, 2011 2011.

[17] Sengupta S., Harris M., Zhang Y., and Owens J. D. Scan primitives for gpu computing. In *Graphics Hardware 2007*, pages 97–106. ACM, Aug. 2007.

This is an extended version of the paper presented at the 14th National Conference on Robotics (KKR 2016), Polanica Zdrój, Poland, September 14-18, 2016