PARALLEL IMPLEMENTATION OF A MULTI-VIEW IMAGE SEGMENTATION ALGORITHM USING THE HOUGH TRANSFORM

Ye. V. Goshin [1, 2], A.P. Kotov [2]
[1] Image Processing Systems Institute of RAS – Branch of the FSRC “Crystallography and Photonics” RAS, Samara, Russia,
[2] Samara National Research University, Samara, Russia

Abstract

We report on the parallel implementation of a multi-view image segmentation algorithm via segmenting the corresponding three-dimensional scene. The algorithm includes the reconstruction of a three-dimensional scene model in the form of a point cloud, and the segmentation of the resulting point cloud in three-dimensional space using the Hough space. The developed parallel algorithm was implemented on graphics processing units using CUDA technology. Experiments were performed to evaluate the speedup and efficiency of the proposed algorithm. The developed parallel program was tested on modelled scenes.

Keywords: segmentation; three-dimensional model; Hough transform; CUDA.

Citation: Goshin YeV, Kotov AP. Parallel implementation of a multi-view image segmentation algorithm using the Hough transform. Computer Optics 2017; 41(4): 588-591. DOI: 10.18287/2412-6179-2017-41-4-588-591.

Acknowledgments: We thank our research supervisor professor Vladimir Fursov who provided insight and expertise that greatly assisted the research and for comments that greatly improved the manuscript. The work was partially funded by the Russian Foundation for Basic Research grants (#16-07-00729 a, #17-29-03112 ofi_m and # 16-29-09528 ofi_m).

Introduction

Image processing and analysis, as well as the task of image segmentation is one of the most important in various spheres of human activity. Many modern applications are based on information processing. In many cases the information to be processed is in the form of images obtained from cameras. However, some images do not contain enough information to perform a reliable segmentation. For example, this can be the case when the texture of scene objects consists of large regions of different colors. In this case, if there are several images, it is better to consider the three-dimensional structure of the scene rather than the intensity characteristics of individual images.

There is a considerable number of algorithms and methods for 3D scene model reconstruction from multi-view images [2, 3]. However, if the images were obtained from different views, the camera parameters are unknown, so it is necessary to determine these parameters. A number of papers [4, 5] were devoted to multi-view image matching in case when camera parameters are unknown.

One of the approaches to the three-dimensional scene segmentation consists in the detection of objects on this scene that have a certain similar characteristic [6, 7]. For example, in paper [8], the plane detection in a scene represented by point cloud is considered. It is devoted to the case when the point cloud was obtained by LIDaR surveying.

In this paper, the three-dimensional Hough transform is used to detect the planes. The purpose of the paper is to speed up the technology proposed in [9] through the parallel implementation of one of the stages of this technology, namely the detection of the most suitable planes using the Hough space. Experimental results demonstrating the speedup of the parallel implementation of the algorithm compared with the sequential implementation are given.

An overview of the technology

The main stages of the multi-view image segmentation technology using the three-dimensional Hough transform proposed in [9] are shown in Fig. 1.

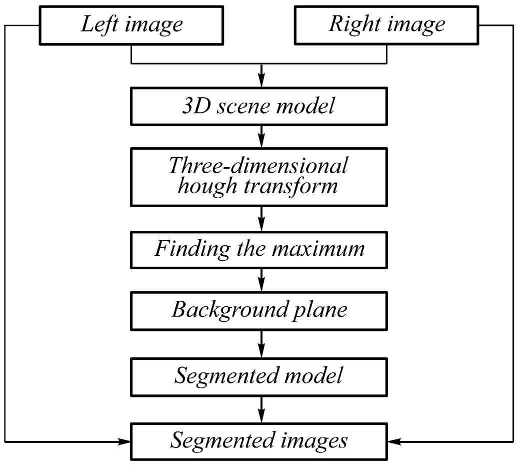

Fig. 1. The scheme of the technology

According to the scheme, first a three-dimensional scene model is constructed from two images. In this paper, we use the algorithm for camera parameters determination described in [10]. Using the Lucas–Kanade method [11], we form an optical flow which matches points between the first and second images. The point cloud based on the matches obtained is formed by triangulation [12]. Then, the Hough transform is applied to all points of the resulting three-dimensional scene. Among all the planes, the maximum in the accumulator space is detected using the transformation, by means of which the background plane is selected. Further, by calculating the distance from the points to the detected plane, we divide one model into two: one model consists of the background points, and the other contains the points of the objects. After these steps, it is possible to segment the initial images using the obtained segmented scene. The key stages of the technology will be considered hereafter.

The goal of the 3D scene model segmentation is to separate the objects from the background of the scene. To detect the planes (background and objects in the scene), the three-dimensional Hough transform is performed. The Hough transform is a way of parametric objects detection, which is commonly used to detect lines and circles, and other shapes in the image. For example, in paper [13] the generalized Hough transform is used for detection of a variety of two-dimensional objects with the reference contour.

When performing the Hough transform, for all given points in the initial space the assumption is made whether they belong to the desired object or not. Thus, for this purpose the equation for each point of the scene is solved to determine certain parameters that represent the Hough space. At the final step the maximum values are determined in the Hough space. Thus, we obtain the parameters for the equation of the desired object, whether it is a line, a circle, or some other figure.

There are also several modifications of the Hough transform: probabilistic, random, hierarchical, phase space blur, the use of the image gradient, and others.

As the input values we use a set of points from three-dimensional real space. The plane can be represented using the normal vector **n** to this plane and the distance ρ from the origin to the plane. Then, for each point **p** on the plane the following equation is satisfied:

$$\rho = \mathbf{p}\cdot\mathbf{n} = p_x n_x + p_y n_y + p_z n_z .$$

After substituting expressions for the angles between the normal vector and the selected coordinate system, the plane equation can be written as follows:

$$p_x \cdot \cos\theta \cdot \sin\varphi + p_y \cdot \sin\varphi \cdot \sin\theta + p_z \cdot \cos\varphi = \rho , \qquad (1)$$

where θ and φ are the angles defining the normal vector. The coordinates φ, θ and ρ form such three-dimensional Hough space, that for each point in this space there is a corresponding plane in real three-dimensional space. In turn, for each point (x_0, y_0, z_0) of a real three-dimensional space there is a corresponding surface in the Hough space, so that each point of this surface (φ, θ, ρ) characterizes a certain plane passing through the required point (x_0, y_0, z_0).

In this paper, we solve the problem of determining the background plane containing the greatest number of points from the formed point cloud. For all the points from the initial cloud, after determining the parameters $(\hat{\varphi}, \hat{\theta}, \hat{\rho})$ of the background plane, it is determined whether this point belongs to the plane or not. To find this out, the coordinates of the point are substituted into the plane equation. Next, we obtain some value that we compare with a certain threshold:

$$\begin{aligned} & p_x \cdot \cos\hat{\theta} \cdot \sin\hat{\varphi} + p_y \cdot \sin\hat{\varphi} \cdot \sin\hat{\theta} + \\ & + p_z \cdot \cos\hat{\varphi} - \rho < \Delta . \end{aligned} \qquad (2)$$

All the points satisfying this inequality belong to the plane, the others are considered objects of the scene.

The results of the model segmentation can be used for the initial image segmentation, since there is a one-to-one correspondence between the pixels of the images and the reconstructed points of the three-dimensional model.

Sequential implementation of the three-dimensional Hough transform algorithm

Consider the algorithm that is used for the three-dimensional Hough transform realization in this paper. A three-dimensional array of integer values is used as an accumulator array. For each element in this space there is a corresponding plane with the parameters that are specified by using the coordinates of this element.

Since the exact mapping is impossible due to the discreteness of the array elements, then for each point from the point cloud the algorithm increments the value of those elements of the accumulator array that correspond to the planes passing through the given point or in its neighbourhood.

Using the pseudocode, the above algorithm can be written as follows:

```
Sequential implementation
Input data: Point cloud
Output data: Accumulator array

For each point $(x_0, y_0, z_0)$ in point cloud
    For each angle $\theta$ from 0 to $\pi$ with step $\pi/180$
        For each angle $\varphi$ from 0 to $\pi$ with step $\pi/360$
            Calculate $\rho$ according to (1)
            Cast to integer type $\rho$
            If $\rho < \Delta$
               Increment operation: $A(\theta, \phi, \rho) = A(\theta, \phi, \rho) + 1$
        End loop $\varphi$
    End loop $\theta$
End loop $(x_0, y_0, z_0)$
Find maximum $A(\theta, \phi, \rho)$
```

As a result of this algorithm implementation, each element of the resulting array is assigned a number defined as the number of points from the initial point cloud, where the points are located in the neighbourhood of the plane specified by this element. The element of the array with the maximum value is the required point specifying the background plane.

Parallel implementation of the proposed algorithm

The Hough transform is computationally complex due to the irregular access to the memory during the increment operation of the accumulator array. The use of CUDA (Compute Unified Device Architecture) technology enables us to decompose this operation. However, due to the aforementioned irregular and unpredictable memory access, the effective implementation of the Hough transform algorithm on a graphics processing unit is nontrivial [14].

The architecture of NVIDIA GPU (Graphical Processing Unit) is based on streaming multiprocessors (SMs), scalable by the number of threads. Each GPU multiprocessor executes a thousand threads at a time. When the CUDA program on the host CPU calls the GPU kernel grid, the thread blocks that form the grid are distributed among the streaming multiprocessors (SMs). The GPU kernel grid is the part of the CUDA program code running on the GPU. The threads do not necessarily execute the same program (the GPU kernel) simultaneously. At the same time, threads combined in one block of threads are executed. The threads inside the block of threads are located in warps, and each warp contains 32 threads. Each thread in a warp performs the same instruction per one clock period [15].

The proposed three-dimensional Hough transform algorithm is implemented as a CUDA program. In CUDA program, a part of the code is executed either on the CPU (host) or on the GPU (device). The algorithm of the implemented program consists of five successive steps which are given below. The device performing the procedures at this step is indicated parentheses (host or device).

The main steps of the CUDA program:

1. allocation of memory for input and output data in the global memory of GPU (host);
2. copying the input data from RAM into the global memory of the GPU (host);
3. performing GPU kernel grid and saving the calculated values of the accumulator array in the global memory of the GPU (device);
4. copying the results from the GPU global memory to the RAM (host);
5. release the global memory (host).

After the accumulator array formation, the task of determining the parameters of the required plane becomes trivial.

For the above-mentioned scheme of the CUDA program, two implementations differing in the third step were considered. These implementations differ in the number of parallel processes (threads) and the computational complexity of each of these processes.

In the case of the first parallel implementation, each thread calculates values ρ for all angles θ, φ for a certain point in the three-dimensional space. In the case of the second implementation each thread calculates values ρ for all points for a certain pair of angles θ, φ. The drawback of the second implementation consists in multiple calls to the global memory of the GPU to read the coordinates of the three-dimensional point. However, for both implementations it is difficult to estimate the collisions that arise when the content of the same memory cell needs to be changed for the execution of a transaction of different threads.

As it can be seen from the pseudocodes of the parallel implementations, each thread executes loops with different parameters and different number of operations. The size of the grid also varies.

The speedup of parallel implementations in comparison with the sequential one was calculated by the following formula:

$$s=\frac{t_{CPU}}{t_{HtoD}+t_{\text{kernel}}+t_{DtoH}}, \qquad (3)$$

where t_{CPU} – execution time of the sequential algorithm; t_{HtoD} – transfer time of the input data from RAM of CPU to global memory of GPU (host-to-device); t_{kernel} – time of CUDA kernel execution; t_{DtoH} – transfer time of the resulting data from global memory of GPU to CPU RAM (device-to-host).

Parallel implementation 1

Input data: Point cloud

Output data: Values of the accumulator array for each pair of angles θ and φ for a single point

```
Calculate the thread index $id$
$id = blockIdx.x * blockDim.x + threadIdx.x$
Read the 3D coordinates from global memory
For each angle $\theta$ from 0 to $\pi$ with increment $\pi/180$
    For each angle $\varphi$ from 0 to $\pi$ with increment $\pi/360$
        Compute $\rho$
        Cast $\rho$ to integer
        If $\rho < \Delta$
            Atomic increment: $A(\theta, \phi, \rho)$
        End loop $\varphi$
End loop $\theta$
```

Parallel implementation 2

Input data: Point cloud

Output data: Values of the accumulator array for all points for a single pair of angles θ and φ

```
Calculate the thread index $id$
$id = blockIdx.x * blockDim.x + threadIdx.x$
Calculate $\theta$, $\varphi$: $\theta = id/360$ and $\varphi = id/360$
For each point $(x_0, y_0, z_0)$
    Read the 3D coordinates from global memory
        Compute $\rho$:
        Cast $\rho$ to integer
        If $\rho < \Delta$
            Atomic increment: $A(\theta, \phi, \rho)$
End loop $(x_0, y_0, z_0)$
```

5. Experimental results

To test the efficiency of CUDA implementations of the parallel algorithm, the following experiments were carried out. A point cloud of 158877 points was used as input data. The experiments were carried out using the following equipment: CPU: Intel Core i7-6700K, 4 GHz, GPU: GeForce GTX 750 Ti. The results of comparative studies of the execution time of the algorithm are shown in Fig. 2, Fig. 3 and Table 1.

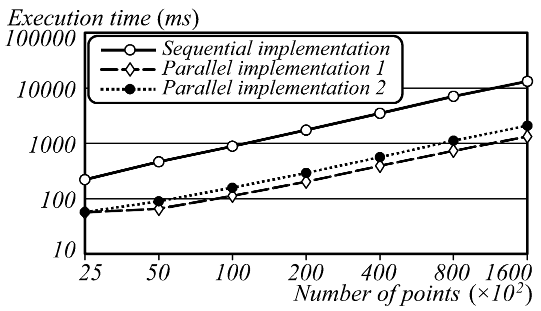

Fig. 2. Dependence of the execution time (ms) for each implementation on the number of points

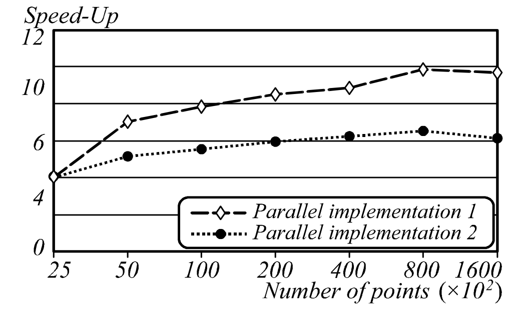

Fig. 3. Dependence of the speed-up of each implementation on the number of points

Table 1. Execution time and speedup for 158877 points

	Execution time (milliseconds)	Speedup
Sequential implementation	13098	–
Parallel implementation 1	1353	9.7
Parallel implementation 2	2139	6.1

Fig. 2 illustrates the dependency of the implementation time of sequential and parallel implementations on the number of points. Fig. 3 shows the dependence of the speed-up of the parallel implementations on the number of points. Both parallel implementations demonstrate the same time of 2500 points. However, when the number of points is greater, the first implementation is executed 1.5 times faster than the second one.

The sequential implementation of the Hough algorithm was carried out in 13 seconds. For parallel implementations 1 and 2, the execution time was 1353 and 2139 milliseconds, respectively. A feature of the parallel implementations is the occurrence of situations when different threads simultaneously perform an increment operation on the same variable. For atomic access of each thread to a specific area of memory, a special operation *atomicAdd*() was used to ensure atomicity. The smallest execution time was registered for parallel implementation 1, for which parallelism was implemented at the level of decomposition by cloud point data.

Conclusion

The proposed algorithm was implemented as a C++ program using CUDA technology. Experimental studies of achievable values of accuracy and reliability were carried out. During the experimental studies of the technology, its operability was demonstrated and a comparative study of the efficiency of various parallel program implementations of the proposed algorithm was carried out. The greatest speedup (by a factor of 9.7) was obtained for the parallel realization 1.

References

[1] Pollefeys M, Nistér D, Frahm J-M, Akbarzadeh A, Mordohai P, Clipp B, Engels C, Gallup D, Kim S-J, Merrell P, Salmi C, Sinha S, Talton B, Wang L, Yang Q, Stewénius H, Yang R, Welch G, Towles H. Detailed real-time urban 3D reconstruction from video. International Journal of Computer Vision 2008; 78(2-3): 143-167. DOI: 10.1007/s11263-007-0086-4.

[2] Baillard C, Maître H. 3-D reconstruction of urban scenes from aerial stereo imagery: A focusing strategy. Computer Vision and Image Understanding 1999; 76(3): 244-258. DOI: 10.1006/cviu.1999.0793.

[3] Pollefeys M, Koch R, Van Gool L. Self-calibration and metric reconstruction in spite of varying and unknown intrinsic camera parameters. International Journal of Computer Vision 1999; 32(1): 7-25. DOI: 10.1023/A:1008109111715.

[4] Eisert P, Steinbach E, Girod B. Automatic reconstruction of stationary 3-D objects from multiple uncalibrated camera views. IEEE Transactions on Circuits and Systems for Video Technology 2000; 10(2): 261-277. DOI: 10.1109/76.825726.

[5] Reitberger J, Schnörr C, Krzystek P, Stilla U. 3D segmentation of single trees exploiting full waveform LIDAR data. ISPRS Journal of Photogrammetry and Remote Sensing 2009; 64(6): 561-574. DOI: 10.1016/j.isprsjprs.2009.04.002.

[6] Tarsha-Kurdi F, Landes T, Grussenmeyer P. Hough-transform and extended RANSAC algorithms for automatic detection of 3D building roof planes from lidar data. Proceedings of the ISPRS Workshop on Laser Scanning 2007; 36(3): 407-412.

[7] Zhang J, Lin X, Ning X. SVM-based classification of segmented airborne LiDAR point clouds in urban areas. Remote Sensing 2013; 5(8): 3749-3775. DOI: 10.3390/rs5083749.

[8] Borrmann D, Elseberg J, Lingemann K, Nüchter A. The 3D Hough Transform for plane detection in point clouds: A review and a new accumulator design. 3D Research 2011; 2(2): 02003. DOI: 10.1007/3DRes.02(2011)3.

[9] Goshin YeV, Loshkareva GE. Segmentation of stereo images with the use of the 3D Hough transform. CEUR Workshop Proceedings 2016; 1638: 340-347. DOI: 10.18287/1613-0073-2016-1638-340-347.

[10] Goshin, YeV, Fursov VA. 3D scene reconstruction from stereo images with unknown extrinsic parameters. Computer Optics 2015; 39(5): 770-776. DOI: 10.18287/0134-2452-2015-39-5-770-776.

[11] Lucas BD, Kanade T. An iterative image registration technique with an application to stereo vision. IJCAI 1981; 81: 674-679.

[12] Hartley RI, Sturm P. Triangulation. Computer Vision and Image Understanding 1997; 68(2): 146-157.

[13] Fursov VA, Bibikov SA, Yakimov PYu. Localization of objects contours with different scales in images using Hough transform. Computer Optics 2013, 37(4): 496-502.

[14] Van Den Braak G-J, Nugteren C, Mesman B, Corporaal H. GPU-vote: A framework for accelerating voting algorithms on GPU. Euro-Par 2012 Parallel Processing 2012; 945-956. DOI: 10.1007/978-3-642-32820-6_92.

[15] NVIDIA Corporation. NVIDIA CUDA C Programming Guide: Version 8.0; January 2017. Source: ⟨http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf⟩.

Authors' information

The information about author **Yegor Vyacheslavovich Goshin** you can find on page 563 of this issue.

Anton Petrovich Kotov, Master of Applied Mathematics and Computer Science. Currently studies at Samara University. Research interests are image processing, recognition algorithms, 3D-scene reconstruction, parallel computations. E-mail: antonykotov@gmail.com .

Code of State Categories Scientific and Technical Information (in Russian – GRNTI)): 28.23.15, 50.41.25.
Received June 30, 2017. The final version – August 19, 2017.