Performance Measurement of a General Multi-Scale Template Matching Method

Gábor Kertész, Sándor Szénási, Zoltán Vámossy
John von Neumann Faculty of Informatics
Óbuda University
kertesz.gabor@nik.uni-obuda.hu, szenasi.sandor@nik.uni-obuda.hu, vamossy.zoltan@nik.uni-obuda.hu

***Abstract*—Generally, template matching methods are very sensible to image orientation, rotation or the size of the template. However, there are known occasions, when they perform better then other keypoint-based techniques. This article demonstrates a massively parallelizable method to multi-scaled template matching. By implementing it sequentially on CPU and on GPU with a naive approach, the runtimes can be measured during a test with multiple scale sizes. With the use of the GPU, the runtimes are 12-times lower. These results indicate that there is place for further research, and to develop a GPU associated solution to handle real-time video processing.**

I. Introduction

Object detection is a key element in computer vision, and there are several approaches to find an object on a single image [1]. First of all, there is the color-based segmentation to find objects in a chosen color-region [2]. There are methods based on keypoints and descriptors: corners, edges and different features can be located on the object and on the reference image, and based on these keypoints matching could be done [3]. Haar-like features [4] are based on calculating the pixel intensity changes with summed area tables. This means, Haar-feature detection has less computational need compared to SIFT or SURF keypoint-detection [5].

And there is template matching, where each pixel of the searched image is matched with each pixel of the referral image. While keypoint-based methods are able to recognize scaled or rotated objects, template matching is very sensitive to these manipulations [6].

Even so, there are several problem types where template matching is the best or only option to find an object. Color-based segmentation can be easily ruled out, by defining a grayscale image or by searching for a multi-coloured object. Keypoint matching could fail on circle-based, highly symmetrical graphical elements or on smoothless artificial images [7], also keypoint matching on a noisy image could fail due to keypoint descriptor mismatches.

II. Template matching

Template matching [6] is a classic method to find the visual representation of an object (the template) on a reference image. A basic, pixel-level comparison of the reference pixels and the template is done. The matching process is comparing the pixels of the template image with the reference image, using a sliding window. The window is a template sized patch, which is moved over the reference image.

While moving the window over the reference image, the difference of the corresponding pixels is calculated. With the use of these differences, the matching value of each window position can be estimated. There are several methods to evaluate, the simple ways are to calculate the sum of squared differences, or the cross correlation [8]. However, the normalized versions of these methods usually provide better results.

Depending on the chosen method, the minimum or maximum value represents the best match on the image. Multiple matches can be handled as well, if a threshold is applied.

The sequentual algorithm (Algorithm 1) of template matching is based on two images, the template T and the reference image I. As a simplification, these images can be handled as two-dimensional arrays of integers, where the integers are intensity values fo each grayscaled pixel.

A two-dimensional array R is created for results, with the size of the difference of I and T row and column sizes. In each iteration, the result of the matching calculation will be stored here.

Algorithm 1 Template matching algorithm

```
function TemplateMatching(T, I)
    R ← new array[I.width − T.width, I.height − T.height]
    for i := 1 → I.width − T.width do
        for j := 1 → I.height − T.height do
            R[i, j] ← SumDiffs(T, I, i, i + T.width, j, j + T.height)
        end for
    end for
    return R
end function
```

After the process finishes, the best (minimal or maximal, depending on the method) value can be searched, and the row and column number will indicate the upper left corner of the found template on I.

The calculation of values in R can be done parallelly, because there is no dependency between the values, also I and T are both read-only.

III. Multi-scale implementation

As already mentioned, template matching is sensitive to rotational or scalable changes on the reference image.
A possible solution to solve the problems raised by different sizes is to create multiple images resizing the original template, match each of them on the reference, and finally select the best match of the results. Noteable, that this method could still fail detecting rotated or tilted objects.

Algorithm 2 Multi-scale approach

function MULTISCALETEMPLATEMATCHING(T, I)
  $T_S \leftarrow new\ array[numofscales]$
  $V_S \leftarrow new\ array[numofscales]$
  $L_S \leftarrow new\ array[numofscales]$
  for all $i := 1 \rightarrow I.x - T.x$ **do**
    $tmp \leftarrow$ TEMPLATEMATCHING($T_S[i], I$)
    $r, c \leftarrow$ BESTVALUE(tmp)
    $V - S[i] \leftarrow tmp[r, c]$
    $L_S[i] \leftarrow (r, c)$
  end for
  $idx \leftarrow$ BESTVALUE(V_S)
  return $L_S[idx], V_S[idx]$
end function

T_S is an array of the templates in different scale. T_S is created at the beginning, by resizing the default template. V_S and L_S are arrays to store the best matching values and the corresponding locations. With an iteration through the different template images, the best matches are selected, and stored in V_S and L_S (Algorithm 2). After the iteration, the best is selected from V_S, where the index indicates he corresponding L_S location, and the scale-rate itself.
There is a simple implementation to use parallel template matching in the core of this method: please note, that it is theoretically possible to parallelize the whole method, however this article aims on analyzing the results of this so-called naive implementation.

IV. Test results

A. Hardware and environment

The defined algorithm was implemented using the OpenCV 3.0 libraries [9], which contains functions for template matching. OpenCV contains a library, which supports CUDA driven GPU programming: as a precompiled binary, nVIDIA hardware can be used.
The following hardware configuration has been used during the tests:

- Processor: Intel Core i7-2600
- Architecture: Sandy Bridge
- Number of cores: 4
- Memory: 16 GB DDR2

During the testing of the GPU-accelerated implementation, we used the nVIDIA Tesla K40 Active videocard:

- Number of CUDA cores: 2880
- Memory: 6144 MB GDDR5
- Architecture: Kepler GK110

The host computer of the GPU was the same as mentioned above.

TABLE I
PROCESSING TIMES OF MATCHING MULTIPLE IMAGES

#	Step size	CPU runtime (ms)	GPU runtime (ms)	CPU time / GPU time
5	0.36	1498	296	5.06
10	0.18	2808	234	12
15	0.12	4337	359	12.08
20	0.09	5631	484	11.63
25	0.072	7113	593	11.99
30	0.06	8673	734	11.82
35	0.0514	9968	811	12.29
40	0.045	11232	936	12
45	0.04	12667	1045	12.12
50	0.036	14087	1170	12.04
55	0.03272	15351	1248	12.3
60	0.03	16864	1404	12.01

The implementation was made with C++ using the OpenCV libraries. A 4 MP (2560 x 1600) reference image, and a 1000 x 750 template image was used.
During template matching, normalized crosscorrelation was used to calculate the sum of differences, so when selecting the best match, we had to search for the maximum values.
The resizing was done scaling the original image from 0.1 to 1.9, using different number of templates. The step size of scaling was calculated as $1.8\ /\ <number\ of\ steps>$ (table I).
When running the tests, first the template images were created, and the multi-scale template matching algorithm was applied to detect the best matches, first on the CPU and after it finished, processing times was also measured on the GPU accelerated version.

B. Results

As expected, the GPU accelerated runtimes were significantly smaller than in the case of the CPU. However, we detected that OpenCV CUDA libraries compile the kernel on the first run, so the runtime of the very first template matching on the GPU is slower.
After all, without counting the first "warm-up" call, the GPU is around 12-times faster, than the sequential solution using the CPU. When using graphical accelerator, an interesting information is that how much time is wasted while transferring data from the host to the GPU, and backwards.
As it is visible on Table II, the variance of measured times are quite wide, however it is still visible, that the multiple GPU calls with different data is making the whole procedure slower.
Another interesting fact is, that how fast is template matching on each image size, both on the CPU and on the GPU. Less steps are needed to move the window if the window size is bigger, but there are more pixel differences to summarize.

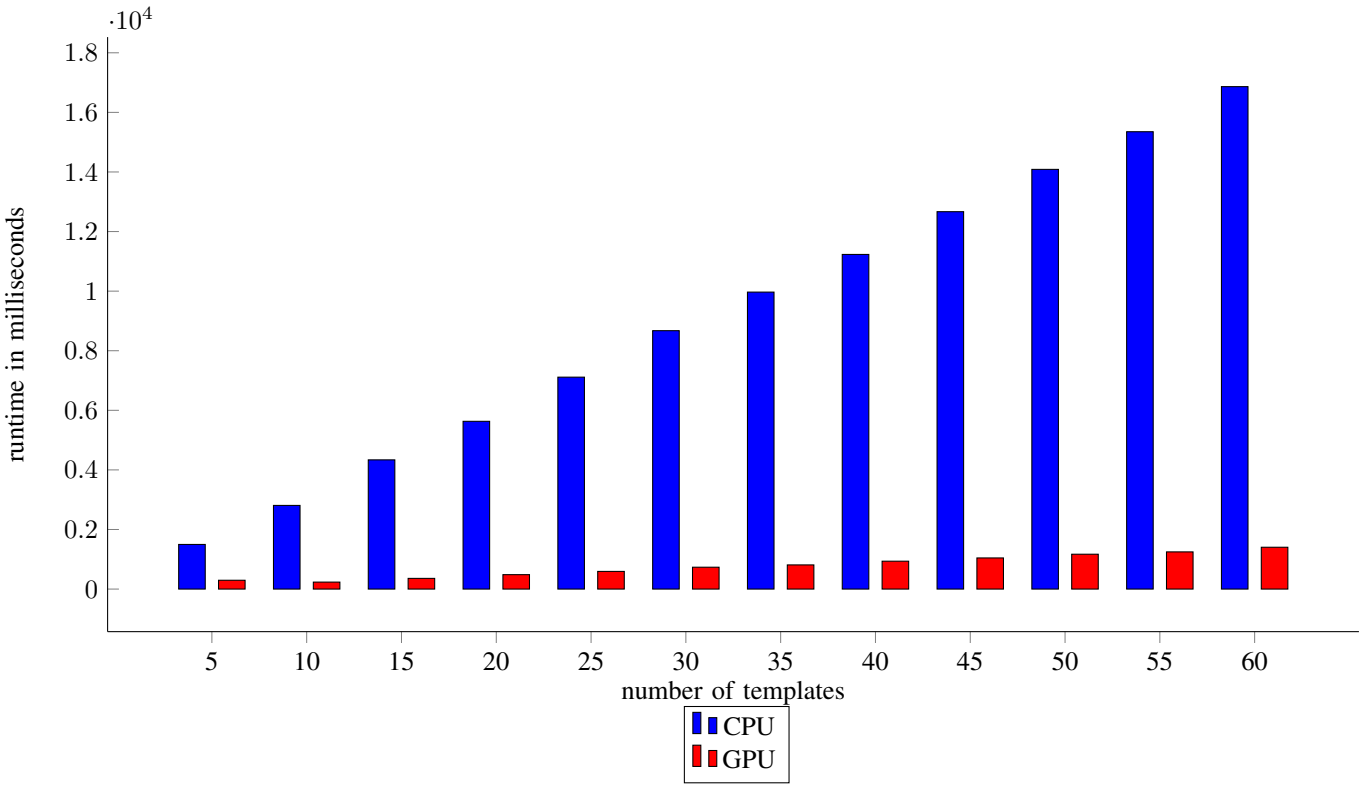

Fig. 1. Results for multi-scale template matching for different number of templates

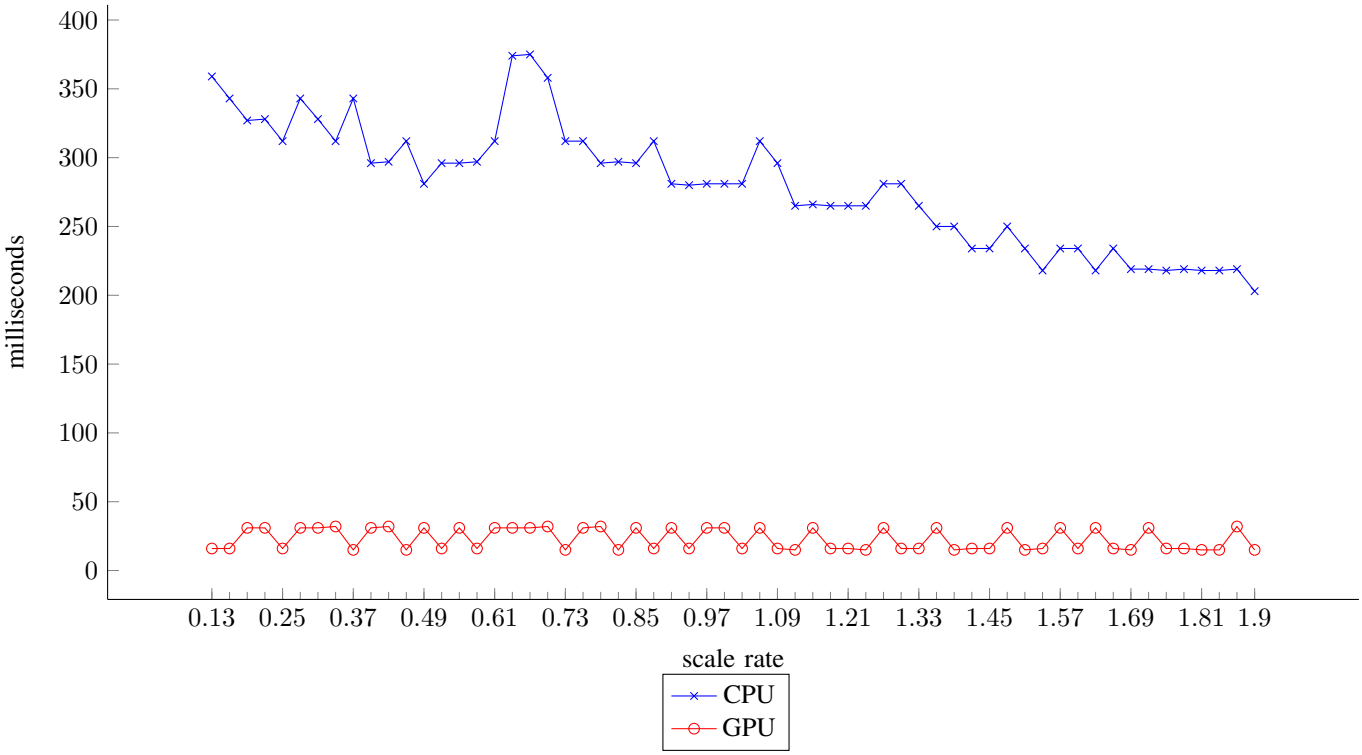

Fig. 2. Processing times of template matching different sized template images on CPU and GPU

TABLE II
MEMORY TRANSFER TIME WHEN USING THE GPU

#	Step size	GPU time (ms)	Memory transfer time (ms)
5	0.36	296	0
10	0.18	234	47
15	0.12	359	48
20	0.09	484	96
25	0.072	593	63
30	0.06	734	108
35	0.0514	811	78
40	0.045	936	140
45	0.04	1045	136
50	0.036	1170	155
55	0.0327	1248	233
60	0.03	1404	141

Data is collected from the last measurement, where 60 different template images were matched to the reference image.

As the test results show on Table III and on Figure IV-B, the bigger the template image is, the less time is needed to calculate the match. However, this data is indifferent when using the GPU: since there are more than 2 thousand processing units (2880 in the Tesla architectured TITAN), each and every processing unit could be used when calculating the matrix of matching values.

V. CONCLUSION

This article defined a naive solution to multi-scaled template matching, and by implementing and testing the results on both CPU and GPU, the difference of processing times confirms, that multi-scale template matching can be efficient with the use of graphical processing units.

The results indicate, that the time elapsed while transferring data between the CPU memory and the GPU memory is not considerable, however by reducing the number of memory transfer operations results could get better.

Results also show, that while there is a measurable difference on matching different sized images using the CPU, these differences are immeasurable when using the GPU.

Further plans are to create a robust multi-scale method, where each scaled templates matching procedure is on different thread, and the threads can use the shared memory of the GPU. In theory, multi-scaled template matching can be done using GPU with less kernel calls, using the computational capabilities of the processor arrays, real-time performance could be achieved. A massively parallel solution needs to be created, where there are less GPU calls, and each call is reaching more processors from the array, the threads share their results. If a real-time solution is planned, template image resizing should also be done parallelly.

Another interesting future plan is to match the results of different cameras, which could be used in multi-view camera network solutions [10], or even at distributed camera networks [11], [12] . Each camera has its own processing unit, and these units should perform a matching on their views, and

TABLE III
PROCESSING TIMES ON DIFFERENT SIZED TEMPLATES

#	Scale rate	CPU time (ms)	GPU time (ms)
1	0,13	359	16
2	0,16	343	16
3	0,19	327	31
4	0,22	328	31
5	0,25	312	16
6	0,28	343	31
7	0,31	328	31
8	0,34	312	32
9	0,37	343	15
10	0,4	296	31
11	0,43	297	32
12	0,46	312	15
13	0,49	281	31
14	0,52	296	16
15	0,55	296	31
16	0,58	297	16
17	0,61	312	31
18	0,64	374	31
19	0,67	375	31
20	0,7	358	32
21	0,73	312	15
22	0,76	312	31
23	0,79	296	32
24	0,82	297	15
25	0,85	296	31
26	0,88	312	16
27	0,91	281	31
28	0,94	280	16
29	0,97	281	31
30	1	281	31
31	1,03	281	16
32	1,06	312	31
33	1,09	296	16
34	1,12	265	15
35	1,15	266	31
36	1,18	265	16
37	1,21	265	16
38	1,24	265	15
39	1,27	281	31
40	1,3	281	16
41	1,33	265	16
42	1,36	250	31
43	1,39	250	15
44	1,42	234	16
45	1,45	234	16
46	1,48	250	31
47	1,51	234	15
48	1,54	218	16
49	1,57	234	31
50	1,6	234	16
51	1,63	218	31
52	1,66	234	16
53	1,69	219	15
54	1,72	219	31
55	1,75	218	16
56	1,78	219	16
57	1,81	218	15
58	1,84	218	15
59	1,87	219	32
60	1,9	203	15

the results could be analysed mutually.

A. Acknowledgements

The authors would like to thank NVIDIA Corporation for providing graphics hardware for the GPU benchmarks through the CUDA Teaching Center program. The authors also wish to thank the members of the CUDA Teaching Center at Óbuda University for their constructive comments and suggestions.

REFERENCES

[1] S. Sergyán, "Recent advances in image, audio and signal processing," *WSEAS Press, 2013*, vol. 28, p. 243, 2013.

[2] A. E. Abdel-Hakim and A. A. Farag, "Color segmentation using an eigen color representation," *Information Fusion, 2005 8th International Conference on*, vol. 2, 2005.

[3] J. C. Dunn, "Group averaged linear transforms that detect corners and edges," *Computers, IEEE Transactions on*, vol. C-24, pp. 1191–1201, 1975.

[4] K.-Y. Park and S.-Y. Hwang, "An improved Haar-like feature for efficient object detection," *Pattern Recognition Letters*, vol. 42, pp. 148–153, 2014.

[5] R. Rios-Cabrera, T. Tuytelaars, and L. V. Gool, "Efficient multi-camera vehicle detection, tracking, and identificationin a tunnel surveillance application," *Computer Vision and Image Understanding*, vol. 116, pp. 742–753, 2012.

[6] R. Brunelli, *Template Matching Techniques in Computer Vision: Theory and Practice*. Wiley, 2009.

[7] G. Windisch and M. Kozlovszky, "Image sharpness metrics for digital microscopy," *SAMI 2015 • IEEE 13th International Symposium on Applied Machine Intelligence and Informatics*, pp. 273–276, 2015.

[8] S. A. Bilings, *Nonlinear System Identification: NARMAX Methods in the Time, Frequency and Spatio-Temporal Domains*. Wiley, 2013.

[9] I. Corporation and Itseez. (2015) Opencv 3.0 online documentation. [Online]. Available: http://docs.opencv.org

[10] R. Pflugfelder and H. Bischof, "People tracking across two distant self-calibrated cameras," *IEEE Conference on Advanced Video and Signal Based Surveillance*, pp. 393–398, 2007.

[11] Y. Yao, C.-H. Chen, A. Koschan, and M. Abidi, "Adaptive online camera coordination for multi-camera multi-target surveillance," *Computer Vision and Image Understanding*, vol. 114, pp. 463–474, 2010.

[12] D. Stojcsics, "Autonomous waypoint-based guidance methods for small size unmanned aerial vehicles," *Acta Polytechnica Hungarica*, vol. 11, pp. 215–233, 2014.