GPU Computing with Orientation Maps for Extracting Local Invariant Features

Naoyuki Ichimura
National Institute of Advanced Industrial Science and Technology (AIST)
1-1-1, Umezono, Tsukuba, Ibaraki 305-8568, Japan
nic@ni.aist.go.jp

Abstract

Local invariant features have been widely used as fundamental elements for image matching and object recognition. Although dense sampling of local features is useful in achieving an improved performance in image matching and object recognition, it results in increased computational costs for feature extraction. The purpose of this paper is to develop fast computational techniques for extracting local invariant features through the use of a graphics processing unit (GPU). In particular, we consider an algorithm that uses multiresolutional orientation maps to calculate local descriptors consisting of the histograms of gradient orientations. By using multiresolutional orientation maps and applying Gaussian filters to them, we can obtain voting values for the histograms for all the pixels in a scale space pyramid. We point out that the use of orientation maps has two advantages in GPU computing. First, it improves the efficiency of parallel computing by reducing the number of memory access conflicts in the overlaps among local regions, and secondly it utilizes a fast implementation of Gaussian filters that permits the use of shared memory for the many convolution operations required for orientation maps. We conclude with experimental results that demonstrate the usefulness of multiresolutional orientation maps for fast feature extraction.

1. Introduction

The use of local invariant features is regarded as a promising method for representing the contents of an image. Local invariant features can be extracted by the following two steps[21, 12, 15, 16]: (i) detecting local regions, (ii) calculating descriptors. Figure 1 shows an example of detecting local regions. Descriptors are calculated in the local regions shown by the squares. Local invariant features have two main advantages in describing scenes. The first is their robustness against occlusions; we can use the features of visible portions of the scene even if parts of the scene are occluded. The second advantage is, of course, their invariance. By introducing components such as scale space pyramids, local coordinate systems and norm normalization into feature extraction, we can make local features invariant to deformations and to changes in illumination. Because of these advantages, local invariant features have been widely used as fundamental elements for image matching and object recognition[21, 12, 15, 16, 4, 6, 8, 13, 18, 14, 24, 10].

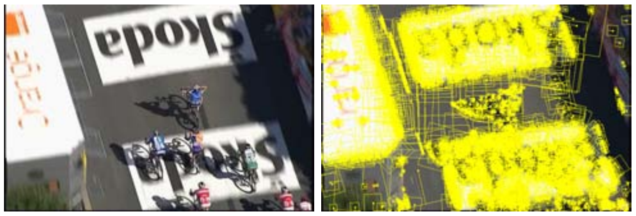


Figure 1. An example of detecting local regions. Left: an image obtained from a broadcast of the Tour de France[1]. Right: the result of detection. The squares represent local regions. We detect about 6000 regions based on both feature points and edges. Dense detection such as this is desirable in improving performance in image matching and object recognition[8, 13, 18, 14].

Feature point extraction has been used for local region detection in image matching to identify portions of images where the intensity changes. On the other hand, several sampling strategies have been adopted for *dense* local region detection in object recognition. Nowak et al.[18] have demonstrated that local regions sampled randomly from a scale space pyramid with regularly positioned grids show a better performance in object recognition compared to local regions obtained by feature point extraction. The main difference is in the number of local regions: feature point extraction cannot sample at a sufficiently high density to produce good results. Other investigators have also used dense sampling of local features in object recognition[6, 8, 13, 14]. Dense sampling can be used in image matching, because it is useful in improving the performance for scenes with heavy occlusions and few

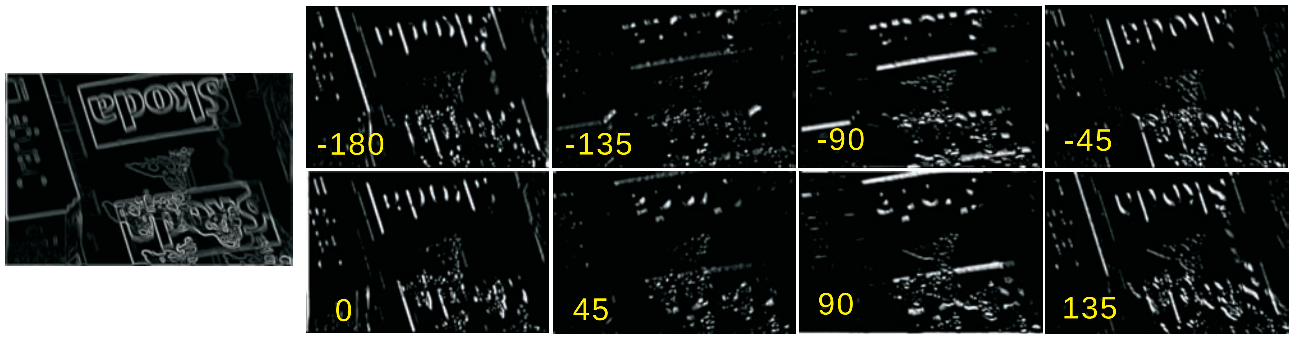


Figure 2. Examples of orientation maps. Left: an edge detection result for the image shown in Fig.1. Right: the orientation maps corresponding to the eight gradient orientations obtained from the edge image on the left. Each of the maps contains gradient magnitudes for a quantized orientation.

textures. Dense detection of local regions is, therefore, desirable in both image matching and object recognition.

Because dense sampling increases the computational costs of feature extraction, we examine fast computational methods for extracting local invariant features. In particular, we consider an algorithm for calculating local descriptors consisting of the histograms of gradient orientations. We can explain the importance of fast computation of descriptors by decomposing the computational costs of feature extraction into the two steps shown above. The cost for detecting local regions is determined mainly by the resolution of the image, because convolution operations of filters such as Gaussian, Laplacian and gradient filters are dominant. In calculating descriptors, the number of pixels to be processed is normally much larger than the resolution of an image because of presence of the overlaps among local regions that are detected at a high density. Thus the fast computation of descriptors is a significant factor in feature extraction.

The computational cost for calculating descriptors can be reduced by detecting the overlaps among local regions, followed by the elimination of multiple computations. However, it is difficult to detect the overlaps because the sizes and locations of local regions vary from scene to scene. We adopt a strategy of applying the processes required to compose descriptors for all the pixels in a scale space pyramid. Using this strategy, we can compute descriptors merely by looking up the results of the processes for all the pixels and we thereby avoid the need for overlap detection. An algorithm for the computation of descriptors based on this strategy has been proposed by Tola et al.[24]. To calculate the descriptors consisting of the histograms of gradient orientations, the algorithm uses *orientation maps* containing the gradient magnitudes for quantized gradient orientations. Figure 2 shows some examples of orientation maps. Gaussian filters with various scales are applied to orientation maps to aggregate the gradient magnitudes locally. The convolution operations performed by the Gaussian filters are equivalent to the computation of the voting values for the histograms of gradient orientations. We can, therefore, obtain the descriptors merely by looking up convolved orientation maps, which greatly reduces the number of multiple computations in the overlaps among local regions.

Because the strategy explained above involves exchanging the costs of computing descriptors in the overlaps among local regions for the costs of convolutions of Gaussian filters for orientation maps, we need to pay attention to the balance between these two sets of costs. In the wide-baseline stereo matching technique of Tola et al.[24], the computational cost for the Gaussian filters is smaller than that for calculating the descriptors without using orientation maps, because the descriptors of all the pixels in an image are required. This is not the case for local invariant features used in image matching and object recognition, where multiresolutional analysis using a scale space pyramid is performed to cope with large changes in scale. Because we generate orientation maps from the edge detection result of every scale, a large number of convolution operations need to be performed on many orientation maps. In addition, descriptors for all the pixels in a scale space pyramid are not always necessary in image matching and object recognition. It is not, therefore, clear that the use of orientation maps is efficient in extracting local invariant features.

We present a method for feature extraction based on the use of a graphics processing unit (GPU) to show the effectiveness of orientation maps. Although several investigators have adopted a GPU to implement algorithms for local invariant features, such as SIFT[9, 22], SURF[23, 3] and HOG[20], algorithms using orientation maps have not been considered. We point out that the use of orientation maps has two advantages in GPU computing. First, it improves the efficiency of parallel computing by reducing the number of memory access conflicts in the overlaps among local regions, and secondly it utilizes a fast implementation of Gaussian filters that permits the use of shared memory for the many convolution operations required for orientation maps.

Below, we explain the descriptor, the orientation maps, and the implementation on a GPU. Then we conclude with

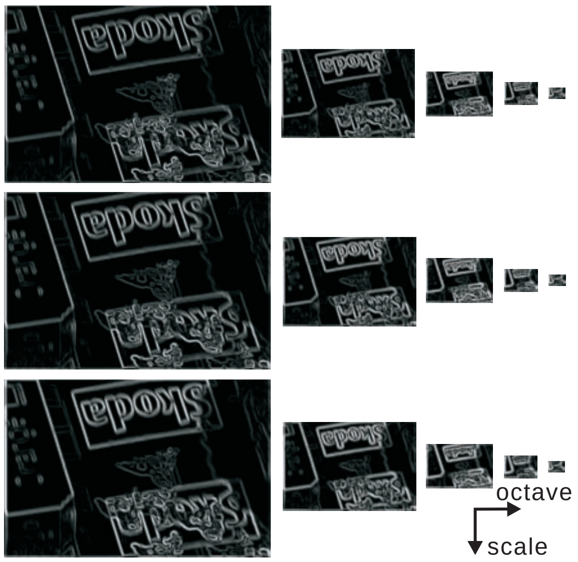


Figure 3. Examples of multiresolutional edge images. Computing the orientation maps for each edge image as shown in Fig.2, we can obtain multiresolutional orientation maps. Because there are five octaves and three scales in an octave in this scale space pyramid, the number of multiresolutional orientation maps is 120 for the eight gradient orientations. Gaussian filters with various scales should be applied to all the maps to calculate descriptors.

a description of our experimental results.

2. Review of Calculating Descriptors

In this section, we review algorithms commonly used for calculating descriptors, such as SIFT[12], GLOH[15], HOG[6] and DAISY[24], that involve the histograms of gradient orientations.

Multiresolutional analysis by a scale space pyramid is performed for scale invariance. Laplacian of Gaussian (LoG) filters with various scales, together with down sampling of an image, are used to detect local regions by extracting feature points in a scale space pyramid. Scale space extrema are identified to determine the locations and characteristic scales of feature points[11]. The locations of the local regions are the same as those of the feature points. The size of each local region is determined by multiplying the characteristic scale by a certain factor. Another scale space pyramid is generated for the intensity gradients required to calculate the descriptors. Gaussian filters, gradient filters and down sampling are combined to obtain multiresolutional edges. Examples of multiresolutional edge images are shown in Fig.3. Multiresolutional edges can also be used for local region detection[13, 17]. For example, local regions in Fig.1 are detected using both feature points and edges. The descriptors are calculated in local regions using the intensity gradients obtained from the edge images corresponding to the characteristic scales.

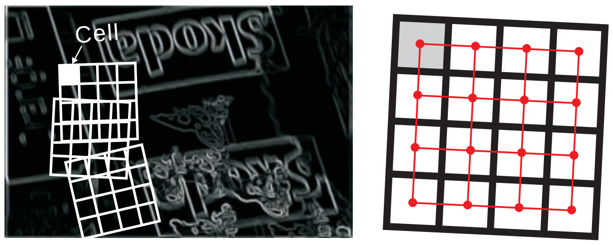


Figure 4. Left: local regions with overlaps. Right: the cells in a local region. If we use convolved orientation maps, we need only look up the values at the centers of the cells shown as small circles. All the positions in the cells should be accessed without the use of orientation maps. This increases the number of memory access conflicts in the overlaps among local regions and reduces the efficiency of parallel computing.

We now explain how the descriptors are calculated. A local region is divided into cells as shown in Fig.4. The histogram of the gradient orientations of the pixels in each cell is computed. The voting values for the histogram are the gradient magnitudes of the pixels weighted by a Gaussian function located at the center of the local region or cell. The histogram of the cell is shifted for rotation invariance based on the direction of a local coordinate system attached to the local region. The local coordinate system can be set using the dominant orientation proposed in SIFT[12]. A vector is then constructed by concatenating the histograms of the cells. A descriptor is obtained by normalizing the vector for illumination invariance. Since gradient orientations are related to the directions of edges, the descriptor represents the shape information in the cells. The usefulness of this representation in image matching and object recognition has been confirmed[12, 15, 16, 4, 6, 8, 13, 18, 14, 24, 10].

3. Parallel Computing by GPU

Before presenting the method using orientation maps for calculating the descriptors explained in Section 2, we outline the process of parallel computing by a GPU to clarify the necessity for orientation maps. CUDA(Compute Unified Device Architecture), which was developed by NVIDIA[5], is applied to feature extraction. A layer is used in the hardware model of CUDA: there are streaming multiprocessors (SMs), and each SM has multiple cores. The data to be processed by the GPU are represented by a layer in accordance with the hardware model. The data are divided into blocks associated with threads, and the blocks are assigned to SMs. The threads for the data in a block are executed in parallel by the cores. As a result, the GPU performs parallel computing. For image filtering and local region detection in feature extraction, an image is divided into rectangular blocks, and parallel computing is performed with the cores carrying out computation for the pixels in one block. For the descriptors, we can group multiple local regions into a block and perform the computations for the local regions in parallel.

To achieve fast parallel computation, it is important to use the appropriate type of memory in the GPU. Although shared memory has the shortest latency, its capacity is limited, e.g., 16KB. Global memory, on the other hand, has a much larger capacity, e.g., 1GB, but its latency is much greater. In image filtering, we can put all the pixels in a block into shared memory by adjusting the size of the block. Since neighboring regions for image filtering overlap, each pixel in a block needs to be accessed many times by various threads. Therefore, reuse of data in shared memory is very useful for achieving fast computation by avoiding access to global memory. Furthermore, latency arising from memory access conflicts in the overlaps is alleviated by the use of shared memory. In contrast with the case of image filtering, it is intractable to reuse data through the use of shared memory for computing descriptors. Because the overlaps among local regions depend on scenes, the overlaps are not organized unlike those among neighboring regions in image filtering. As it is difficult to determine the sizes and shapes of the overlaps among local regions, it is not easy to reuse the data in the overlaps using shared memory of a limited capacity. We, therefore, calculate the descriptors in global memory. Although the memory latency is tried to hide by activating a large number of threads in CUDA, the effectiveness of this scheme is reduced as a result of latency caused by memory access conflicts in the overlaps among local regions. These memory access conflicts decrease the efficiency of parallel computing by the GPU. In the next section, we show that the computational efficiency can be improved by the use of orientation maps.

4. Calculating Descriptors Using Orientation Maps

The gradient orientations and gradient magnitudes of all the pixels in a scale space pyramid are obtained by gradient filters. Using two dimensional arrays corresponding to the bins of the histogram of gradient orientations, we can separate the gradient magnitudes based on the gradient orientations. The two dimensional arrays containing the gradient magnitudes are called *orientation maps*[24]. Examples of orientation maps are shown in Fig.2.

Applying a Gaussian filter to orientation maps, we can aggregate weighted gradient magnitudes locally. The extent of aggregation is determined by the scale of the Gaussian filter. If we choose the scale so that the extent of aggregation fits the size of a cell for the descriptors, the convolution operations performed by the Gaussian filter are equivalent to the computation of the voting values for the histograms of gradient orientations. This allows us to calculate the descriptors merely by looking up the locations of convolved orientation maps corresponding to the centers of cells, as exemplified by the red points in the right-hand illustration

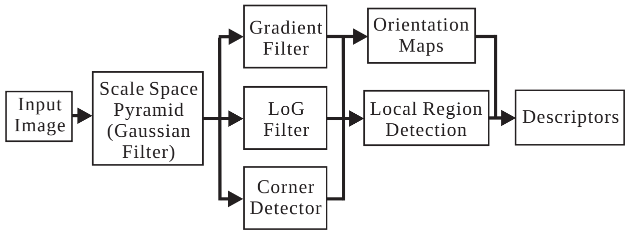


Figure 5. The flowchart of the feature extraction algorithm that we implement. Local regions are detected based on both feature points and edges. The local descriptors consisting of the histograms of gradient orientations are calculated using orientation maps.

in Fig.4. Using orientation maps, therefore, greatly reduces the number of conflicts in memory access among local regions, because no exhaustive access to the pixels in cells is required.

To extract local invariant features, we generate *multiresolutional orientation maps* from multiresolutional edge images for scale invariance. The number of multiresolutional orientation maps is determined by the number of bins in the histograms of gradient orientations and by the numbers of octaves and scales in a scale space pyramid. For example, when we use the gradient orientations and scale space pyramid shown in Fig.2 and 3, the number of orientation maps is 120, as there are eight bins, five octaves and three scales. Because many convolution operations need to be performed on large numbers of orientation maps as in this example, it is important to examine whether the computational cost of Gaussian filters for multiresolutional orientation maps can actually be exchanged for that of descriptors calculated without the use of orientation maps.

Since Gaussian filters can be efficiently implemented by utilizing shared memory, as we have explained in Sec.3, the introduction of multiresolutional orientation maps and the GPU could be useful for fast computation of the descriptors. We implement a feature extraction algorithm on the GPU to confirm the usefulness of this approach; this is discussed in the next section.

5. Implementation on GPU

The data to be processed by the GPU are divided into blocks associated with threads in CUDA. The numbers of blocks and threads are called the *execution configuration* (EC). The number of blocks can be specified by the width and/or the height of a block.

Figure5 is the flowchart of the feature extraction algorithm that we implement. First, a scale space pyramid is generated from an input image by Gaussian filters with various scales. A gradient filter, approximated LoG filters and corner detectors are then applied to the scale space pyramid for local regions and orientation maps to calculate the descriptors consisting of the histograms of gradient orientations. Blow, we present details of feature extraction including additional processes that are not shown in Fig.5. Because the appropriate choice of ECs is an important factor in achieving fast computation, we show their specific values.

Image transfer: An input image is transferred from the host (CPU) to the GPU.

Y Component: An intensity image is computed if the input image has colors. The global memory is used. The size of a block is 16×32. Threads are assigned to all the pixels in a block.

Down sampling: Down sampling for a scale space pyramid is performed in the global memory. The EC is the same as that for the **Y Component**.

Gaussian filter: We generate a scale space pyramid using Gaussian filters. The initial scale of the pyramid is 1.6 and the interval of scales is $2^{1/s}$, where s is the number of scales in an octave. The sizes of the Gaussian filters are determined by truncating the domain using the threshold of 10^{-3} for the range . We place the coefficients of the Gaussian filters into constant memory that has short latency and is read-only in execution of the program. The image data in a block are copied to the shared memory for reuse. The separability of a Gaussian function is exploited. The width of a block in filtering for the rows of the image is 128, and threads are assigned to all the pixels in a block. The size of a block in filtering for the columns is 48×16. Since the maximum number of threads in a block is limited, we divide a block into sub-blocks with the height of eight. We then assign threads to all the pixels in a sub-block. Filtering for the pixels in the remaining sub-blocks is carried out sequentially by the assigned threads.

Gradient filter: The intensity gradients of the scale space pyramid are computed by the 5×5 gradient operator[2]. The use of memory and the EC are the same as those for filtering for the columns in the **Gaussian filter**, except that the size of a block is 16×16.

ALoG-CD filter: Two filters are applied to the scale space pyramid to detect local regions through feature points. First, we apply an approximated LoG (ALoG) filter shown in Fig.6 to all the scales. We reduce the computational cost for contrast detection based on the LoG filter using only the pixel values at the center and on the circle. Secondly, fast corner detectors (CD)[25] with various scales are used to remove feature points on edges that may cause aperture problems. The corner detector also requires only the pixel values at the center and on the circle. The memory usage and the EC are the same as those for the **Gradient filter**.

Feature point: Feature points and their characteristic scales[11] are obtained by identifying 3×3×3 extrema in the scale space pyramid generated by the **ALoG-CD filter**. We search for extrema in each octave. Feature points are selected by thresholding for the responses of the approximated LoG filters and the corner detectors. The thresh-

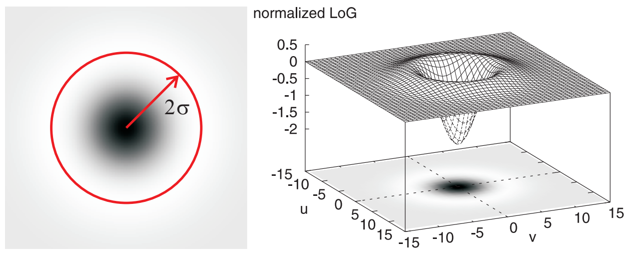


Figure 6. An approximated LoG filter. The left-hand illustration represents the values of an LoG operator in the right-hand illustration as an intensity. We approximate the response of the LoG filter by the difference between the pixel value at the center and the mean of the pixel values on a circle with a radius of 2σ, where σ is the scale of a Gaussian filter applied to an image in a scale space pyramid. The position of the circle corresponds to the local maximum of the LoG function.

olds are 10 and 100, respectively. All the processes are performed in the global memory. The size of a block is 16×32 and threads are assigned to all the pixels in a block. The positions and scales of feature points are the corresponding values for local regions. We organize the positions and scales as a feature list. Since no synchronization of threads over blocks is allowed, parallel processing to generate the feature list cannot be realized because the number of features is present as a common variable in all the threads. Consequently, we use the CPU to generate the feature list which entails data transfer between the GPU and the CPU.

Edge sampling: We sample the pixels in the scale space pyramid generated by the gradient filter by extracting spatial 3×3 extrema of gradient magnitudes. A threshold of 10 is used to select the extrema. The global memory is used, and the EC is the same as that for the **Feature point**. The positions and scales of the extrema are added to the feature list using the CPU.

Dominant orientation: The dominant orientations[12] of the local regions in the feature list are computed for rotation invariance. The size of each local region is obtained by multiplying its characteristic scale by a factor of five. The global memory is used. We group 16 local regions as a block and assign threads for all the regions. Multiresolutional orientation maps are used for the dominant orientations, in the same manner as the histogram computation for the descriptors.

Orientation map: Multiresolutional orientation maps with the eight quantized directions shown in Fig.2 are calculated from the scale space pyramid obtained by the gradient filter. The orientation maps are generated in the global memory. The EC is the same as that for the **Feature point**. As shown in Appendix A, a weighed scheme is used to reduce the boundary effect. The scales of Gaussian filters applied to the multiresolutional orientation maps are 1.2 times larger than those used in the **Gaussian filter**. The scales correspond to the sizes of the cells in local regions, as explained in Appendix B. The memory usage and the EC are the same as those for the **Gaussian filter**.

Descriptor: We calculate the descriptors in local regions with the sizes obtained by multiplying their characteristic scales by a factor of 20. The global memory is used. The EC is the same as that for the **Dominant orientation**. Local regions are divided into 4×4 cells as shown in Fig.4. Using convolved orientation maps enables us to calculate the histograms of gradient orientations from the centers of the cells. When orientation maps are not generated, all the pixels in the cells are used to calculate the histograms. We shift the histograms based on the dominant orientations of local regions for rotation invariance, as explained in Appendix C. The descriptors are obtained by normalizing vectors that consist of the histograms of the cells for illumination invariance.

In the next section, we present the measurements of the computational times for the implementations with and without orientation maps on the GPU and the CPU.

6. Experimental Results

We used an Intel Quadcore Xeon (3.16GHz/12MBL2) CPU and a NVIDIA GeForce GTX 280 GPU for the experiments. The local invariant features were calculated on the Xeon and GeForce, and another GPU, an NVIDIA Quadro FX4600, was used solely for the purpose of display. The GPU for computing has 240 cores, and its compute capability is 1.3[19]. A single core was used in computing on the CPU. Single precision floating-point (float) arithmetic was used in all the implementations.

We compared the computational times for feature extraction for the "Tour de France" image shown in Fig.1[1] and the first image in the set called "Graffiti" obtained from the feature detector evaluation sequences[7]. The resolution of the Tour de France image was 720×480. The resolution of the Graffiti image was reduced to permit the measurement of computational time for the QVGA size. The numbers of octaves was five for the Tour de France image and four for the Graffiti image. As the number of scales in an octave was three, we generated 120 and 96 multiresolutional orientation maps, respectively, with the eight quantized gradient orientations.

Table 1 lists the computational times. In this table, "CPU–C" and "GPU–C" are the implementations without using orientation maps on the CPU and the GPU, respectively, whereas "CPU" and "GPU" stand for the corresponding implementations using orientation maps. By comparing the computational times for "Dominant orientation", "Orientation map" and "Descriptor" among the implementations, we can see that both the introduction of orientation maps and the use of the GPU are very effective in terms of fast feature extraction. Although the effectiveness depends on the resolution of the image and the contents of the scene, the method is more than 30 times as fast as the conventional method on the CPU.

The sizes of local regions were determined by the factor for the characteristic scales. The factor was 20 in the experiments shown in Tab.1. We can change the area of the overlaps among local regions by altering this factor. This means that the computational dependency among local regions varies with the factor. The use of a smaller factor reduces the number of memory access conflicts in the overlaps, which increases the efficiency of the parallel computing. The top illustration in Fig.7 shows the changes in the ratio of the computational times of the CPU and the GPU as a function of the factor for the Tour de France image. The ratio of the computational times reflects the efficiency of parallel computing because the computational cost is the same for both the CPU and the GPU. When no orientation maps were used, the efficiency of parallel computing fell as the factor was increased. On the other hand, the efficiency was obviously maintained when orientation maps were used. One reason for this improvement is the reduction in the number of memory access conflicts in the overlaps, and another one is the fast implementation of the Gaussian filters for the multiresolutional orientation maps. The bottom illustration in Fig.7 shows the advantage of the parallel implementation of the Gaussian filters with the shared memory in comparison with the CPU-based implementation. Because the areas of the cells increase as the factor is increased, we need Gaussian filters with larger scales, which yield a large number of convolution operations. However, no degradation of the efficiency of the parallel computation was observed as a result of data reuse in the shared memory. Similar results were obtained for the Graffiti image.

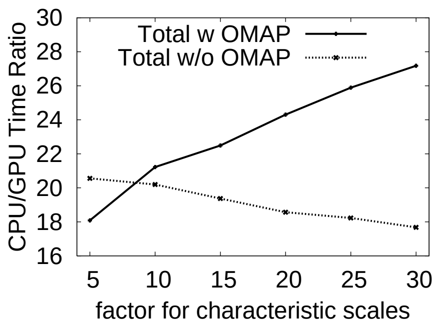
In summary, we have shown that the use of multiresolutional orientation maps for calculating the descriptors has two advantages in GPU computing: an improvement in the efficiency of parallel computing as a result of a reduction in the number of memory access conflicts in the overlaps among local regions, and the utilization of the fast implementation of Gaussian filters using the shared memory for large numbers of convolution operations for orientation maps.

7. Conclusions

In this paper, we have considered GPU computing with multiresolutional orientation maps for calculating the local descriptors consisting of the histograms of gradient orientations. Aggregating gradient magnitudes for quantized gradient orientations through the use of Gaussian filters, we can reduce memory access for the overlaps among local regions, thereby increasing the efficiency of parallel computing by the GPU. The fast implementation for a large number of convolution operations that is required for aggregation can

Table 1. Computational times for extracting local invariant features. The units are microseconds. We measured the mean time for 100 trials, because a non-realtime OS (Linux) was used. The total times include the computational times for processes other than the tasks that were examined, e.g., memory allocation.

Image	Tour de France				Graffiti			
Image size	720×480				320×240			
Task/Implement	CPU–C	CPU	GPU–C	GPU	CPU–C	CPU	GPU–C	GPU
Image transfer	N/A	N/A	2.563	2.597	N/A	N/A	0.600	0.600
Y component	3.922	3.747	0.108	0.107	0.934	0.938	0.062	0.061
Down sampling	0.854	0.866	0.140	0.140	0.207	0.165	0.083	0.082
Gaussian filter	263.740	277.149	8.766	8.716	56.435	58.623	4.854	4.830
ALoG-CD filter	308.480	309.437	13.639	13.639	63.758	63.805	3.812	3.817
Gradient filter	219.952	220.617	4.394	4.405	45.636	45.759	1.358	1.360
Feature point	158.199	160.817	7.538	7.546	34.718	35.501	1.957	1.948
Edge sampling	11.179	12.486	7.312	7.335	3.541	3.885	1.833	1.836
Dominant orientation	196.260	92.575	5.977	3.548	119.797	77.340	3.145	1.392
Orientation map	N/A	1509.556	N/A	45.076	N/A	308.565	N/A	24.079
Descriptor	2091.589	51.782	122.483	10.772	954.991	21.989	76.171	5.119
Total	3255.619	2640.544	176.641	107.536	1280.372	616.980	96.381	47.619
#feature	5731				2706			



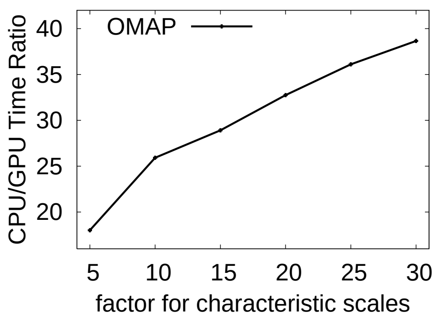


Figure 7. The changes in the ratio of the computational times of the CPU and the GPU as a function of the factor for the characteristic scales. This result was obtained from the Tour de France image. "OMAP" stands for orientation maps. Top: the ratio of the total times. Bottom: the ratio of the times for calculating orientation maps.

be realized by reuse of data by means of the shared memory. The usefulness of GPU computing in extracting local invariant features was confirmed by the experimental results. We believe that the considerations on GPU computing presented here will facilitate the development of local descriptors that can be efficiently calculated in parallel, because the strategy of applying the processes required to compose descriptors for all the pixels in a scale space pyramid can be applied to other information for descriptors such as colors, and to other method for arranging the cells in a scale space pyramid.

References

[1] An image of the Tour de France. The image was obtained from the broadcast of J SPORTS.

[2] S. Ando. Consistent gradient operators. *IEEE Trans. PAMI*, 22(3):252–265, 2000.

[3] N. Cornelis and L. V. Gool. Fast scale invariant feature detection and matching on programmable graphics hardware. In *Proc. Workshop on Computer Vision on GPU's (in conjunction with CVPR08)*, 2008.

[4] C. Csurka, C. R. Dance, L. Fan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. In *Proc. Workshop on Statistical Learning in Computer Vision*, pages 1–22, 2004.

[5] CUDA Zone. http://www.nvidia.com/object/cuda_home_new.html.

[6] N. Dalal and B. Triggs. Histograms of orientated gradients for human detection. In *Proc. Int. Conf. Comp. Vis. Patt. Recog.*, volume 1, pages 886–893, 2005.

[7] Feature detector evaluation sequences. http://lear.inrialpes.fr/people/mikolajczyk/Database/.

[8] L. Fei-Fei and P. Perona. A bayesian hierarchical model for learning natural scene categories. In *Proc. Int. Conf. Comp. Vis. Patt. Recog.*, volume 2, pages 524–531, 2005.

[9] S. Heymann, K. Müller, A. Smolic, B. Fröhlich, and T. Wiegand. SIFT implementation and optimization for general-purpose GPU. In *Proc. Int. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, pages 317–322, 2007.

[10] C. H. Lampert, M. B. Blaschko, and T. Hofmann. Efficient subwindow search: A branch and bound framework for object localization. *IEEE Trans. PAMI*, 31(12):2129–2142, 2009.

[11] T. Lindeberg. Feature detection with automatic scale selection. *Int. J. Comp. Vis.*, 30(2):79–116, 1998.

[12] D. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comp. Vis.*, 60(2):91–110, 2004.

[13] X. Ma and W. E. Grimson. Edge-based rich representation for vehicle classification. In *Proc. Int. Conf. Comp. Vis.*, volume 2, pages 1185–1192, 2005.

[14] K. Mikolajczyk, B. Leibe, and B. Schiele. Multiple object class detection with a generative model. In *Proc. Int. Conf. Comp. Vis. Patt. Recog.*, volume 1, pages 26–36, 2006.

[15] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Trans. PAMI*, 27(10):1615–1630, 2005.

[16] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. V. Gool. A comparison of affine region detectors. *Int. J. Comp. Vis.*, 65(1/2):43–72, 2005.

[17] K. Mikolajczyk, A. Zisserman, and C. Schmid. Shape recognition with edge-based features. In *Proc. Britich Machine Vis. Conf.*, volume 2, pages 779–788, 2003.

[18] E. Nowak, F. Jurie, and B. Triggs. Sampling strategies for bag-of-features image classification. In *Proc. European Conf. Comp. Vis.*, pages 490–503, 2006.

[19] *NVIDIA CUDA Programming Guide, Version 2.0, Sec.5.1.2*, 2008.

[20] V. A. Prisacariu and I. Reid. fastHOG – a real-time GPU implementation of HOG –. Technical Report 2310/09, Department of Engineering Science, Oxford University, 2009.

[21] C. Schmid and R. Mohr. Local greyvalue invariants for image retrieval. *IEEE Trans. PAMI*, 19(5):530–535, 1997.

[22] SiftGPU. http://www.cs.unc.edu/˜ccwu/siftgpu/.

[23] T. B. Terriberry, L. M. French, and J. Helmsen. GPU accelerating speeded-up robust features. In *Proc. Int. Symp. on 3D Data Processing, Visualization and Transmission (3DPVT)*, pages 355–362, 2008.

[24] E. Tola, V. Lepetit, and P. Fua. A fast local descriptor for dense matching. In *Proc. Int. Conf. Comp. Vis. Patt. Recog.*, 2008.

[25] M. Trajkovic and M. Hedley. Fast corner detection. *Image and Vision Computing*, 16:75–87, 1998.

A. Generating Orientation Maps

The left-hand illustration in Fig.8 shows voting from a pixel with a gradient orientation θ and a gradient magnitude g to orientation maps. The gradient orientation is assigned to a bin with a median τ and width d. Basically, the gradient magnitude g is voted to the bin. However, as a result of quantization of a gradient orientation, the method is not robust to deformation and changes in illumination, because if the distance between θ and one of the boundaries of the bin is small, variability in a scene causes an abrupt change in the bin assignment of θ. This *boundary effect* badly affects the invariant properties of local features based on a histogram. To avoid abrupt changes in the histogram of gradient orientations, a weighted voting scheme is used. We vote g to the bin with the median τ by multiplying it by the following weight:

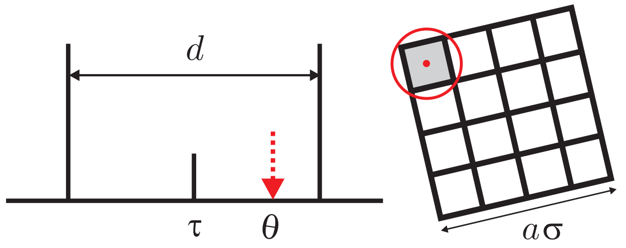


Figure 8. Left: Weighting for quantized gradient orientations to generate orientation maps. Right: Determining the scales of the Gaussian filters for orientation maps.

$$w_1 = 1 - |\theta - \tau| / d \,. \quad (1)$$

The quantity $(1 - w_1)\, g$ is voted to the left or right bin based on the sign of $(\theta - \tau)$. We generate orientation maps by applying the voting scheme to all the pixels in the scale space pyramid of edges.

B. Determining the Scales of Gaussian Filters for Orientation Maps

Here we describe a method for determining the scales of Gaussian filters for orientation maps. Although the 4×4 cells in Fig.4 are used, the following method can be adapted to other arrangements of cells.

The scale of an orientation map is denoted by σ. The factor a is multiplied by σ to compute the size of a local region. Because the local region is divided into 4×4 cells, the width of a cell is $a\sigma/4$. The scale of a Gaussian filter σ_o is determined by setting the value of the independent variable of the Gaussian function at the position of a circumscribed circle of a cell as shown in Fig.4. The value should be chosen carefully, because the scale σ_o directly governs aggregation for the gradient magnitudes in a cell. We use the value $3\sigma_o$, and thus the diameter of the circumscribed circle is $6\sigma_o$. Using the width of a cell and the diameter, we have the following scale:

$$\sigma_o = \sqrt{2} a\sigma / 24 \,. \quad (2)$$

In measuring the computational times shown in Tab.1, a factor of $a = 20$ was used, which leads to the scales of the Gaussian filters for the multiresolutional orientation maps as $\sigma_o \approx 1.2\sigma$.

C. Shifting Histograms

The histograms of cells are shifted for rotation invariance based on the dominant orientation of the local region. The weighted voting scheme explained in Appendix A is exploited using the weights determined from the ratio of the overlaps between the shifted histogram and the bins.