Fast Gabor Texture Feature Extraction with Separable Filters using GPU

Anonymous

Abstract

Gabor wavelet transform is one of the most effective texture feature extraction techniques and has resulted in many successful practical applications. Real-time applications, however, cannot enjoy this technique because of the high computational cost arising from the large number of small-sized convolutions which require over 10 minutes to process an image of 256×256 pixels on a dual core CPU. As the Gabor filtering computations are parallelizable, it is promising to accelerate the feature extraction process using GPU. Conventionally, this can be achieved simply by accelerating 2D convolution directly, or by expediting the CPU-efficient FFT-based 2D convolution. Indeed, the latter approach, when implemented with small-sized Gabor filters, cannot fully exploit the power of GPU due to the architecture of graphics hardware. This paper proposes a novel approach tailored for GPU in accelerating texture feature extraction algorithm with separable 1D Gabor filters used to approximate the non-separable Gabor filter kernels. Experimental results indicate significant improvement in timing performance with minimal error introduced. The algorithm is specifically designed and optimized for Compute Unified Device Architecture (CUDA) and achieves a speed of 16 fps. It is potentially applicable for real-time applications in areas such as motion tracking and medical image analysis.

1 Introduction

Texture feature extraction is the basis of many texture analysis techniques. Gabor feature extraction, as a multiresolution filtering technique, is one of the most important feature extraction techniques and has been proved to be advantageous for many applications. However, it is not suitable for real-time applications because of high computation complexity. Gabor filters are usually defined by a set of scales and orientations (e.g. 4 scales and 6 orientations). Convolution between each pixel of an image and the 2D Gabor kernels in different directions and orientations are required to extract the features. Here, the number of multiplication and addition operations required is proportional to the second order of the kernel dimension. When fast Fourier transform (FFT) is used to perform convolution, the complexity is $\log$ WH, where W and H are the width and height of the image. To tackle these problems, we propose to improve the computational speed by approximating the originally non-separable Gabor filter kernels to separable ones, by which the number of operations required in Gabor filtering becomes linearly proportional to the kernel dimension. Furthermore, the computations can then be implemented on GPU to enjoy the benefit of hardware acceleration.

In the proposed implementation, we attempt to exploit and maximize the parallel computational power of GPU. First, all pixel windows are processed for convolution with all Gabor kernels simultaneously, as they are all independent of each other. Second, the convolutions are optimized to employ coalesced access or shared memory in order to minimize memory access latency.

We have carried out a series of experiments to analyze the performance of the proposed approach on image segmentation in terms of speed and quality. The results show that our approach only introduces minor errors when compared

with the ground truth result, and their image segmentation performance is on par with each other. No significant difference can be found visually in the result, while the computational complexity is reduced and the speed is faster than other possible implementations on GPU. The acceleration provided by our method enables high frame rate of 16 fps which is useful for many real-time applications like motion tracking and medical image analysis.

2 Related Work

There are several existing methods to extract textural features, many of them are reported in the literature and widely used in texture analysis applications [1, 2, 3, 4, 5]. The extraction of such features usually account for the invariant characteristics of translation, rotation and scaling of textures. As a result, high dimensionality and high computational cost to obtain the texture feature are expected.

The Gabor wavelet transform is one of the robust methods and extensively being used in texture classification and discriminations [6, 7]. Apart from the theory that Gabor filters resemble responses of simple cells in visual cortex [8], many of the experiments, like the one carried by Ahmadian and Mostafa [9], proved that non-separable Gabor wavelet transform outperforms the classical dyadic wavelet transform for texture classifications. Nearly 7% better in classification rate is achieved, and an even higher rate of 13% for brightened textures.

There were several attempts to accelerate convolution with Gabor filters. Recursive filtering of Gabor kernel proposed by Young et. al. [10] can achieve a complexity of only O(N), however, it is obvious that recursive computation on GPU is not recommended. Geusebroek et al. [11] proposed a method to decompose an anisotropic Gaussian filter, which is a critical component in the Gabor wavelet formulation. In their method, a 2D filtering is performed first with a 1D Gaussian filter in x direction, followed by another 1D Gaussian filter along a line in particular orientation. The proposed method achieved good acceleration and approximation of arbitrary oriented Gaussian filter. However, it requires a special shear filtering instead of the standard horizontal and vertical filterings which is difficult to program and does not fit well on GPU. Similarly, the idea of separating Gabor filter kernels for performance boost is also suggested by Areekul et al [12], but the proposed method allowed only particular oriented Gabor filters and lacks of generality and flexibility to suit many applications.

A number of researches exploiting GPU for image processing purposes emerged early this decade. But until recently, Wang and Shi [13] tried to accelerate Gabor filtering with GPU. Their method decomposes the 2D filtering into three 1D filterings in order to reduce computation complexity. However, the performance is not very satisfactory, and it is not very clear if the method is advantageous for texture feature extraction which involves computing simultaneously many small sized pixel windows. Inspired by these previous works, we proposed a novel approach turning Gabor filters into separable ones which can then be implemented more easily into current GPU architecture for accelerated texture feature extraction.

The idea to decompose non-separable filters into separable ones was first proposed by Treitel and Shanks [14]. They provided a general framework and idea of approximating filtering with finite number of one-dimensional recursive filterings and minimal error. Later, Kubota [15] presented similar but improved method especially suitable for orientational filters. Our algorithm is also closely related to these previous work, while, we focus on the the application to a specific filter, i.e. the Gabor filter. Moreover, the massive computation of Gabor filtering on GPU for the purpose of per-pixel computation of texture feature has not been studied previously in the literature.

3 Method

The Gabor function $g(x, y)$ is written as [7]

$$g(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} exp\left[-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_y^2}\right)+2\pi jWx\right]. \quad (1)$$

The Gabor function form a complete but nonorthogonal basis set. Gabor wavelets are a class of self-similar functions derived form the Gabor function. Dilation and rotation of the

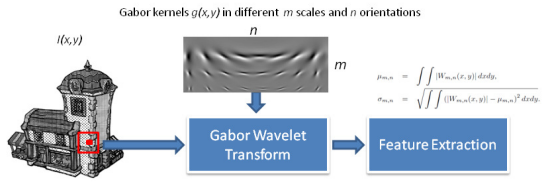

Figure 1: Typical workflow of Gabor wavelet feature extraction.

mother function $g(x, y)$ generate the following functions:

$$g_{m,n}(x,y) = a^{-m} g(x', y'), \quad a > 1,$$
$$x' = a^{-m}(x cos\theta + y sin\theta),$$
$$y' = a^{-m}(-x sin\theta + y cos\theta),$$

where m and n are integers, $\theta = n\pi/K$, K is the total number of different orientations and a^{-m} is a scale factor which ensures the energy is independent of m. To reduce the redundancy of Gabor wavelets caused by the nonorthogonal nature of the Gabor function, σ_x and σ_y are assigned appropriately as discussed in [7].

For an image $I(x, y)$, its Gabor wavelet transform is then defined to be

$$W_{m,n}(x,y) = \int I(x_1, y_1)\overline{g_{m,n}}(x - x_1, y - y_1)dx_1 dy_1, \quad (2)$$

where $\bar{\cdot}$ is the complex conjugate operation. As local texture regions are assumed to be spatially homogeneous, so the mean $\mu_{m,n}$ and the standard deviation $\sigma_{m,n}$ of the transform coefficients' magnitudes are used as the Gabor wavelet features. For a 6 scales and 4 orientations texture feature, they are defined as $[\mu_{m,n}, \sigma_{m,n}]_{m=1,\cdots,6, n=1,\cdots,4}$ in a per-pixel manner:

$$\mu_{m,n} = \int\int |W_{m,n}(x,y)|\, dxdy,$$
$$\sigma_{m,n} = \sqrt{\int\int (|W_{m,n}(x,y)| - \mu_{m,n})^2\, dxdy}. \quad (3)$$

Throughout the paper, S and R are used to represent respectively the total number of scales and orientations. Figure 1 illustrated the major procedures involved in Gabor wavelet-based texture feature extraction. From the above formulations, the most computation intensive part occurs at equation 2, that is the convolutions between the image and several Gabor filter kernels of different orientations and scales. When a huge number of images or image windows is involved, the computational cost is high and usually a bottleneck.

In the following subsections, we will introduce various approaches of accelerating the large number of Gabor filtering, discussing the pros and cons, as well as introducing the implementation details on GPU using CUDA.

3.1 Gabor filtering with 2D convolution

The most straightforward method of performing the Gabor filtering is by direct 2D spatial convolution. A direct convolution in spatial domain with an M×M filter kernel requires a M^2 multiplications and additions. Spatial convolution can be easily parallelizable on GPU as each kernel value multiplies independently. We, therefore, implemented a CUDA version of Gabor filtering for experimental comparisons.

Suppose that the width and height of an image I are W an H respectively. We will run W blocks of CUDA kernels, and within each block, H threads are launched to compute one row of the image independently. For efficiency, our implementation puts the image into shared memory, while the filter kernels are left in global memory.

To compute per-pixel texture feature, we will compute all the features in parallel. Instead of running H blocks, we run R×2×NUM by S×H blocks of image window centered at every pixel. Here, NUM is the total number of pixels in the whole image.

The timing statistics of Gabor filtering using 2D spatial convolution are reported in Section 4, the average run-time on GPU needs about 6.2s per frame for an image size of 256×256, far from real-time requirements.

3.2 Gabor filtering with FFT on CUDA

Convolution in frequency domain is another standard approach for computing Gabor filtering. FFT is usually preferred to achieve higher efficiency. Thus, the convolution can be done by

first applying FFT to both the input data and the convolution kernels, point-wise multiplying between them, and finally performing inverse FFT to converts the result back into the spatial domain.

Many experiments have shown that FFT based convolution is more efficient than 2D direct spatial convolution implementation on CPU when large kernel is used, as the complexity of per-pixel multiplications and additions involved is $\log$ WH. However, in our case, only small or mediate sized windows are involved, and the largest window is 128×128 pixels. This problem also occurs in the corresponding GPU version. For example, the standard CUFFT algorithm that parallelizes FFT in CUDA is most efficient for the cases where both the kernel and the window size are large [16].

Apart from small sized windows, parallelizing FFT on GPU also faces the problem that each Gabor filter scale and orientation as well as each individual pixel window can only be processed sequentially, and thus cannot fully capitalize the parallel processing power of GPU.

The timing statistics of Gabor convolutions in frequency domain are reported in Section 4, the average run-time on GPU is about 87.1s per frame for an image size of 256×256, even worse than the 2D convolution implementation.

3.3 Our approach

3.3.1 Gabor filtering using separated 1D convolution

Another possible approach for 2D filtering is to approximate the computation with two separable 1D convolutions, in which only requires 2M multiplications and additions. This is significantly smaller than the M^2 operations in direct 2D convolution, and is comparable to $\log$ WH depending on the filter and window sizes. However, this only applies to separable filters. Unfortunately, some Gabor kernels are non-separable. Separation of these filter kernels can only approximate the effect of the original filter, and inevitably introduces errors in the convolution results. In the following of this section, we will introduce a method to best separate the filter and minimize the convolution errors. We will discuss the details and advantages of using 1D filters in the GPU implementation.

The best way to separate a wavelet kernel in the least square error sense can be accomplished by singular value decomposition (SVD) of a wavelet kernel g:

$$U \Lambda V^T = svd(g). \tag{4}$$

Here Λ is a diagonal matrix with all the eigenvalues. The decomposition of g forms a set of r and c (i.e. the eigenvectors) which are ordered based on the significance. As a result, the best separation takes the first eigenvectors from the matrix U and V as follows :

$$\begin{aligned} r_1 &= \Lambda(1,1)^{\frac{1}{2}} \times u_1^T \\ c_1 &= \Lambda(1,1)^{\frac{1}{2}} \times v_1^T \end{aligned} \tag{5}$$

where $\Lambda(1,1)$ is the first element in matrix Λ, u_1 and v_1 are the first (left most) column vector in matrix U and V respectively. In general, separable kernels sets r_z and c_z can be formed using the remaining eigenvectors and eigenvalues as

$$\begin{aligned} r_z &= \Lambda(z,z)^{\frac{1}{2}} \times u_z^T \\ c_z &= \Lambda(z,z)^{\frac{1}{2}} \times v_z^T \end{aligned} \tag{6}$$

where z ranges from 1 to the kernel size of wavelet g. The introduction of the remaining components in the SVD result improves the accuracy of the convolution results.

The most significant separable kernel set r_1 and c_1 can then be applied one after another by two 1D convolutions with the input image $I(x,y)$ to obtain the Gabor wavelet transform $W'(x,y)$,

$$W'(x,y) = I(x,y) * r_1 * c_1^T. \tag{7}$$

where $*$ is convolution operator. Notice that the computed $W'(x,y)$ is only an approximation of the $W(x,y)$.

3.3.2 CUDA implementation

Apart from the computational efficiency of 1D convolution, a significant advantage of this approach is that it makes possible the best utilization of the shared memory, which has low access

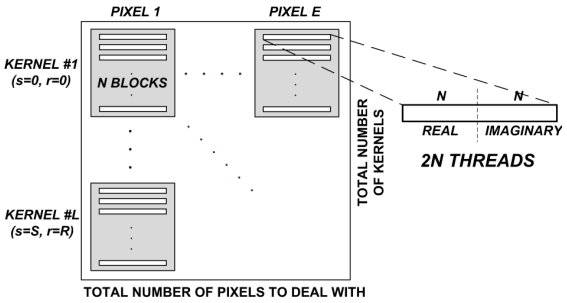

Figure 2: Memory access approach for 1D convolution on CUDA.

latency, in the computation. Our CUDA implementation of the 1D convolution approach is designed to avoid too many data being transferred to the global memory.

In our implementation, the image is first read into global memory as a whole in row-wise manner, as illustrated in Figure 2. N is the pixel window size, E is the total number of pixels to deal with, and L is the total number of kernels whose scale s is the maximum scale number S and orientation r the maximum orientation number R. Then, to deal with a single pixel location, N blocks are assigned and each of them has 2N threads. This is because each block will deal with one row in the pixel window, both the real part and the imaginary part, and so it is 2N in total. As a result, each block computes elements in the same row; each thread can deal with pixels in the same row simultaneously. This greatly and efficiently exploits the parallelization capability in GPU.

Here, column convolution is first performed. Each CUDA kernel is then working independently to convolve a single pixel with the kernel c, as shown in the pseudo code in Algorithm 1. After that, the row convolution is performed in a similar manner. Note that column-wise computation is done between image data and column kernel c in global memory, as shown in Figure 3. Then, each block reads all columns needed from global memory into shared memory for the following row-wise computation. The kernel r is also loaded into shared memory, before the row convolution takes place.

Our algorithm features simultaneous reading of a large amount of data from GPU while the operations in each thread involve light weight computation. This fully utilizes the GPU bandwidth and avoids potential bottleneck for the whole process. Moreover, in column convolution, the memory access pattern in threads is designed to be consecutive, so as to take advantage of coalesced access which minimizes the latency.

Algorithm 1 Pseudo code performing the column convolution in CUDA.

a is the index range, determined by threadID, it takes values between [0, N]
input from global memory to share memory: corresponding column vector
read in the index range a
for k=N-a to N, i=0 to a **do**
  $sum+ = filter[k] \times img[i]$
end for
store sum in share memory

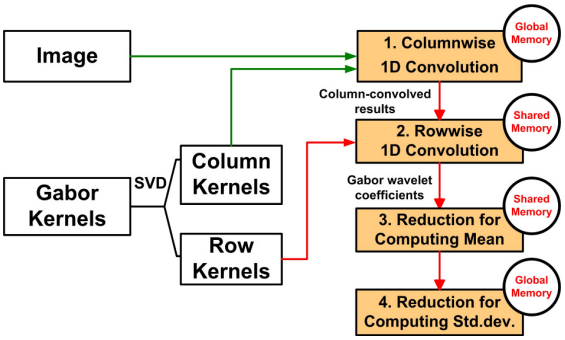

Figure 3: CUDA implementation of 1D convolution approach.

After the column and row convolutions with all the filters, the coefficients are ready for computing the texture features. In equation 3, the features are extracted from the mean and standard deviation of each filtered results using different Gabor kernels. As the computation of mean and standard deviation is a parallel reduction procedure in CUDA, the well-known methods suggested by Harris [17] is used to achieve fast computation. As a result, two more CUDA kernels are introduced for this purpose.

4 Results and Discussion

We carried out a number of experiments to evaluate the timing performance and feature extraction quality of our proposed method. All the experiments are carried out on a PC equipped

	Convolution approach		
	2D	FFT	1D
C	2537 s	857 s	412 s
CUDA	6.2 s	87.1 s	0.032 s

Table 1: Timing performance of different convolution approaches. 256×256 image and 32×32 window size are used.

with Intel dual core CPU 2.66Ghz, 3GB RAM, and a nVidia GeForce 9800GTX graphics card with 600MHz graphics clock, 1500MHz processor clock and 57.6GB/Sec bandwidth.

4.1 Timing statistics

First, we compare the three approaches mentioned in Section 3 (namely, 2D-conv, FFT, and 1D-conv). Both C and CUDA versions are tested and the corresponding running times on CPU and GPU are recorded. Table 1 shows the results using a 256×256 image and 32×32 window size. Significant speed improvement is found in Gabor wavelet based texture extraction using GPU when compared with that using CPU. CPU version takes more than 10 minutes to process, while the GPU implement ion requires only 0.062s, approximately 16fps.

The effects of image size on the processing time are also investigated. A window size of 64×64 pixels is used in this test. As we seen from Table 2, computational times for image size between 128×128 and 512×512 pixels are all around 0.05 s; the increase with image size is minor. However, a larger image of 768×728 pixels in size hits the limit of hardware, as evident from the drastic increase in computational time (10 times longer). This is probably due to the memory bandwidth limit.

The effects of different window sizes (equal to the kernel size) on the timing performance of our method are also analyzed. Table 3 shows the test results using a image of 256×256 pixels. The CPU implementation shows an exponential increase in time with the window size. This is anticipated because the computational complexity is proportional to the number of elements processed in single-window convolution and the vector dimension in 1D convolution. The vector dimension is related to the number of elements involved in single element computation. However, such increase is not observed in the GPU implementation. As seen in Table 3, window size has a negligible effect on the GPU results when compared to that running on CPU.

4.2 Feature quality

Finally, the error of the 1D separable kernel implementation on CUDA is studied. The results are also compared with that of 2D convolution (ground truth). Table 4 shows the maximum and mean errors of the Gabor wavelet features when different kernel sizes are used. The error is measured by averaging (by length of the vector) the Euclidean distance between the ground truth and our Gabor wavelet feature vectors. Since elements of a Gabor wavelet feature can take arbitrary large value, the maximum value among the feature vectors is also provided in the table for reference. The maximum error is between 0.67 and 1.14, which is quite satisfactory comparing to the maximum values that are as large as 105 or above.

We provide segmentation results of various gray-scale images using our accelerated method for visual evaluation in Figure 4. Most of the testing images and textures are selected from the Brodatz texture database [18] and the Berkeley Segmentation Dataset [19]. For the sake of comparing the segmentation ability of texture features computed using our method and the original FFT-based method, we obtain the color-coded segments by the K-mean clustering algorithm. A 48 dimensional texture features together with the 2 dimensional pixel location form the attributes for clustering. The textured mosaic image shown in Figure 4(a) is used in our first experiment. The corresponding segmentation results using our proposed method and the original method by FFT are shown in the middle and on the right respectively. The latter serves as the ground truth for comparisons.

We can observe from the results that, with the same segmentation method, nearly the same results are obtained. This shows that the proposed approach does not affect the segmentation quality of the computed features. Figures 4 (b) to (f) show a few more sets of segmentation results demonstrating that the same conclusion also applies to natural images or even textured cartoons

	Image size			
	128×128	256×256	512×512	768×728
C	328 s	1321 s	5371 s	12634 s
CUDA	0.042 s s	0.047 s	0.062 s	0.67 s

Table 2: Timing performance of 1D convolution approach using 64×64 Gabor kernel for images with different size.

	Window size		
	16×16	32×32	64×64
C	112 s	412 s	1322 s
CUDA	0.029 s	0.032 s	0.047 s

Table 3: Timing performance of 1D convolution approach using windows with different size.

illustrations. We should focus on the ability of grouping texturally similar regions in all these images, for example the dot patterns in (b), the tree branches in (d), the rocks in (e). These results are difficult to be achieved without the involvement of texture features in segmentation.

5 Conclusion

In the paper, we propose to accelerate Gabor texture feature extraction using separable filters and to develop an effective approach to implement the computations on GPU. The experiments on image segmentation give promising results. The approximation only introduces minor errors which are not noticeable visually while the computational speed is significantly increased, attaining near real-time performance. Research will be carried out to optimize the algorithms to further capitalize and utilize the parallel processing power of GPU and the memory bandwidth. In the meantime, experiments will be conducted to use the Gabor texture feature extraction technique in motion tracking so as to demonstrate its feasibility for real-time applications.

Kernel size	Max Error	Mean Error	Max Value
16×16	1.14	0.86	105.22
32×32	1.89	1.49	116.61
64×64	0.67	0.55	121.79

Table 4: Error of Gabor texture feature at different kernel sizes.

6 Acknowledgements

The work is supported in part by some grants (to be advised, anonymized for blinded review).

References

[1] L Van Gool, P Dewaele, and A Oosterlinck. Texture analysis anno 1983. *Computer Vision, Graphics, and Image Processing*, 29:336–357, 1985.

[2] Todd R. Reed and J. M. Hans du Buf. A review of recent texture segmentation and feature extraction techniques. *CVGIP: Image Understanding*, 57(3):359–372, 1993.

[3] M. Tuceryan and A. K. Jain. Texture analysis, handbook of pattern recognition and vision. pages 235–276.

[4] Robert M. Haralick, K. Shanmugam, and Its'Hak Dinstein. Textural features for image classification. *IEEE Transactions on Systems, Man and Cybernetics*, 3(6):610–621, Nov. 1973.

[5] R. Haralick. Statistical and structural approaches to textures. In *Proceedings of the IEEE*, volume 67, pages 786–804. IEEE, 1979.

[6] M R Turner. Texture discrimination by gabor functions. *Biological Cybernetics*, 55(2-3):71–82, 1986.

[7] B S Manjunath and W Y Ma. Texture features for browsing and retrieval of image

data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):837–842, 1996.

[8] J. Daugman. Two-dimensional spectral analysis of cortical receptive field profiles. *Vision Research*, 20(10):847–856, 1980.

[9] A. Ahmadian and A. Mostafa. In *Proceedings of the 25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 2003.*, volume 1, pages 930–933, New York, NY, USA, Sept 2003.

[10] I.T. Young, L.J. van Vliet, and M. van Ginkel. Recursive gabor filtering. *IEEE Transactions on Signal Processing*, 50(11):2798 – 2805, November 2002.

[11] J.-M. Geusebroek, A.W.M. Smeulders, and J. van de Weijer. Fast anisotropic gauss filtering. *Image Processing, IEEE Transactions on*, 12(8):938 – 943, aug. 2003.

[12] Vutipong Areekul, Ukrit Watchareeruetai, and Sawasd Tantaratana. Fast separable gabor filter for fingerprint enhancement. In *Biometric Authentication*, volume 3072 of *Lecture Notes in Computer Science*, pages 1–42. Springer Berlin / Heidelberg, 2004.

[13] XinXin Wang and B.E. Shi. Gpu implemention of fast gabor filters. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 373 –376, 2010.

[14] S. Treitel and J.L. Shanks. The design of multistage separable planar filters. *Geoscience Electronics, IEEE Transactions on*, 9(1):10–27, 1971.

[15] T. Kubota. *Orientational Filters For Real-Time Computer Vision Problems*. PhD thesis, Georgia Institute of Technology, 1995.

[16] Ondrej Fialka and Martin Cadik. Fft and convolution performance in image filtering on gpu. In *Proceedings of the conference on Information Visualization 2006*, pages 609–614. IEEE Computer Society Press, 2006.

[17] Mark Harris. Optimizing parallel reduction in cuda. *NVIDIA Developer Technology*, 2008.

[18] P. Brodatz. *Textures: A Photographic Album for Artists and Designers*. Dover, New York, 1996.

[19] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001.

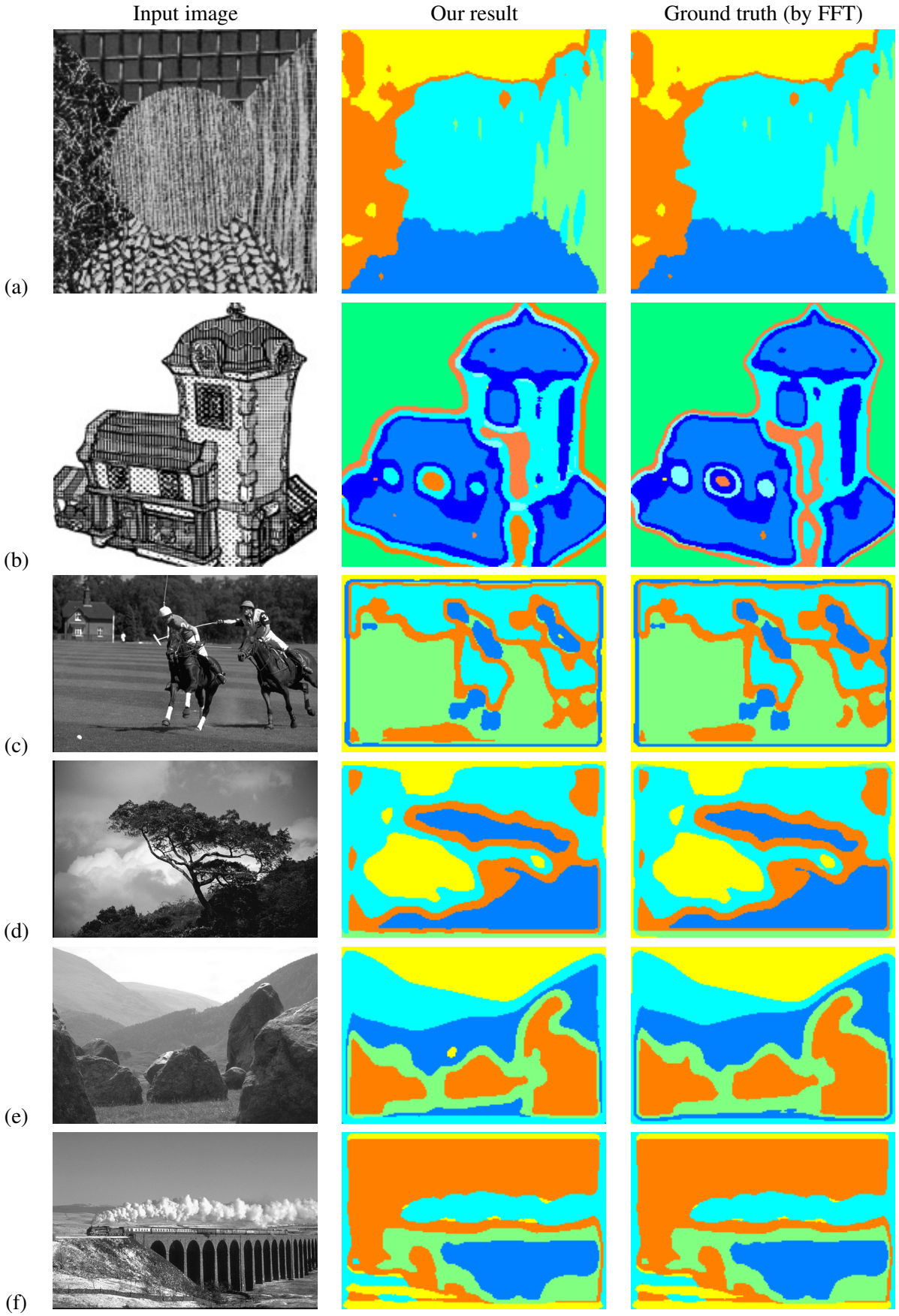

Figure 4: Segmentation results of various images containing rich textures. (a) and (b) are with size 200×200, while (c) to (f) are with size of 481×321. All the segmentations are obtained using K-mean clustering algorithm with K=5.