The CSI Journal on
Computer Science and Engineering
Vol. 9, No. 2 & 4 (b), 2011
Pages 22-33
Regular Paper

GPU Implementation of Edge Histogram Descriptor and Color Moments Fused Features for Efficient Image Retrieval

Alireza Ahmadi Mohammadabadi Abdolah Chalechale Hadis Heidari

Faculty of Engineering, Razi University of Kermanshah, Kermanshah, Iran

Abstract

In this paper, we implement the MPEG-7 Edge Histogram Descriptor and color moments fused features on the Graphics Processing Unit (GPU) to accelerate computations using CUDA. As the GPU can be used for data processing and deal with computationally enormous applications, in this study it is employed for efficient feature extraction phase of image retrieval. The Edge Histogram Descriptor describes the distribution of various types of edges with a histogram that can be a tool for image matching. These features are applied to search images from a database which are similar to a query image. We evaluated the accuracy of our method by the Precision-Recall graph. The average Precision and the average Recall of presented method are 71.53% and 55.00% respectively. GPU parallel computing led to a speedup of 15.12 over CPU sequential processing. Therefore, it demonstrates that parallel computing using a GPU can achieve good performance in image retrieval.

Keywords: Color Moments, Content Based Image Retrieval, Edge Histogram Descriptor, GPU, Parallel Computing.

1. Introduction

Digital images available on Internet have been increased recently and there has been a growing need for efficient information retrieval methods. So it is necessary to have parallel and efficient indexing techniques in image retrieval. Image retrieval as one of the interesting applications of image processing is an appropriate case to be implemented in parallel, because in this application, images are usually divided into several parts and each part is processed separately and similarly. The main idea of image retrieval is to analyze image information by the visual features like color, shape and texture. Content based image retrieval (CBIR) is regarded as one of the most effective ways of accessing visual data and also is a research area, which targets to develop tools for retrieval of visual information using its perceptual contents. The CBIR process consists of calculating a feature vector and storing in the feature database. The feature vector describes some image properties and CBIR system computes the feature vector for the query image. CBIR systems use low-level features like color, shape and texture to represent images and have found applications in different fields like scientific database management, geographical information systems, law enforcement and criminal investigations, communication systems and image search on the Internet [1]. Typical examples of the CBIR systems include PicSOM, Virage, Photobook, VisualSEEK, MARS, Web Seek, Cortina, Candid, Chatbot, FIDS, Blobworld, Netra and SIMPLIcity [2].

The main features used in CBIR are further discussed here. These include texture, color and shape. Texture is an important visual feature that refers to innate surface properties of an object [3]. In [4], texture was modeled by the fusion of marginal densities of subband image DCT coefficients. Usually texture features are derived from Gabor wavelets, discrete wavelet frame and the conventional discrete wavelet transform. Zhang et al. [5] extracted the rotation invariant texture feature extraction based on Gabor texture features by a circular shift of the feature elements so that all the images have the same dominant direction. Kokare et al. [6] applied dual tree rotated complex wavelet filter and dual tree complex wavelet transform for effective rotation

invariant texture feature extraction. Pun [7] proposed a texture based image retrieval system using polar transform followed by an adaptive row shift invariant wavelet packet transform. The image retrieval system introduced in Huang et al. [8] is based on the wavelet decomposition and gradient vector and uses a coarse feature descriptor and a fine feature descriptor with each image. Both descriptors are extracted from the wavelet coefficients of the original image.

Color is one of the most reliable visual features and is independent of image size. It is not sensitive to translation, rotation and scale changes. A color histogram is the most used method to extract color features that describes the global color distribution in an image but holds two shortcomings. They are unable to fully accommodate the spatial information and also not robust. Two dissimilar images with similar color distribution can produce similar histograms. Many researchers suggested the use of color correlogram for avoiding incompatibilities involving the spatial information.

Shape is an important feature for perceptual object recognition and classification of images and object shape features can also provide powerful information for image retrieval, because usually humans can recognize objects solely from their shapes. Shape representation methods include Fourier descriptors, polygonal approximation, invariant moments, B-splines, deformable templates, and curvature scale space [9]. Shape techniques can be classified into two categories: region based and boundary based [10]. In region based techniques, all the pixels within a shape are taken into account to obtain the shape representation and often use moment descriptors such as geometrical moments, Zernike moments, pseudo-Zernike moments, and Legendre moments to describe shapes [11]. Boundary based shape methods are more popular in the literature and tend to be more efficient for handling shapes that are described by their object contours. The most common boundary based shape descriptors are chain codes, Fourier descriptors, wavelet descriptors, and contour displacement [12].

The edges in images are an essential feature to represent their contents. Also, human eyes are sensitive to edge features for image perception [13]. There is a descriptor for edge distribution in the image which is called Edge Histogram Descriptor (EHD). The Edge Histogram Descriptor is one of the descriptors specified by MPEG-7 Visual Standard for measuring similarity in images [13]. An edge histogram in the image space represents the frequency and the directionality of the brightness changes in the image [13]. The Edge Histogram Descriptor can express only the local edge distribution in an image with a histogram. That is, since it is important to keep the size of the descriptor as compact as possible for efficient storage or transmission, the normative MPEG-7 edge histogram is designed to contain only 80 histogram bins describing the local edge distribution. These bins are the only standardized semantics for MPEG-7 EHD. However, it may be insufficient to only use the local histogram bins for representing global features of the edge distribution. In order to improve its expressiveness, Won et al. [13] proposed the use of global and semi-global edge histograms generated directly from the local histogram.

For extracting edge features, the different parts of the image must be processed separately, thereby consuming long time especially when processing high resolution images on large databases. So it is essential to investigate speed up techniques, satisfying time constraints for the various applications. Since EHD extraction is a data-parallel task described in following, it is expected to reduce execution time by using multi-core processors. To extract edge features for an image, the image is divided into 16 non-overlapping image blocks. Each image block is divided into several sub-images as well, such that each sub-image is processed similar to other sub-images. So in each image block, the type of edge is recognized independently. In fact, the same instruction is executed by multiple processors using different data streams (SIMD). Consequently these processes can be performed in parallel. To accelerate feature extraction, multi-core processors can be useful, because of multiple thread execution of algorithms.

It is important to mention that one of the simplest ways for extraction of the color features in images is to use color moments. In this study, to extract the color features, some color moments including mean, standard deviation, and skewness are exploited. To extract these features, the image is first divided into small non-overlapping image blocks and then each image block is processed concurrently in parallel, motivating utilization of parallel processers/GPU for getting a better performance.

GPU has been developed as a supplementary arithmetic device of image processing [14] and can be used to accelerate a wide range of applications. GPUs are well suited to arithmetic-intensive computation workloads that are highly data-parallel [15] and many researchers have already applied GPUs to implement many algorithms in various areas such as image processing, computational geometry, and scientific computation, as well as computer graphics [16]. Not all algorithms can be effectively implemented on the GPU. The GPU can only be faster than the CPU on numerical problems that are inherently parallel. Our work exploits extensive usage of highly multithreaded architecture of GPU using Compute Unified Device Architecture (CUDA). The main contributions of this paper are given in the following:

- Both the Edge Histogram Descriptor and color moments are simultaneously used for more utilization of the features and compensation of their disadvantages.
- With respect to the data-parallel nature of the mentioned methods, efficient parallel algorithms are proposed and implement on GPU using CUDA platforms.

The rest of this paper is organized as follows. Section 2 presents an overview of the GPU architecture used in this study. Section 3 illustrates the details of our new approach for the CBIR. Section 4 reports experimental results to evaluate the robustness and retrieval effectiveness of the proposed scheme and finally, conclusion and future work are given in Section 5.

2. The GPU Architecture

By increasing market demands for real time and high-definition 3D graphics, improvements in GPU turned it into a highly parallel, multithreaded, many-core processor with tremendous computational power and very high memory bandwidth [17]. The potential of modern GPUs is a motivation to be applied by user for other data-parallel and intensive computations. General Purpose Graphics Processing Unit (GPGPU) [18] refers to the fact that the

GPU can be used as general purpose processor so that it is specialized for highly parallel and computationally intensive processes. To apply the parallel computing capability for solving many complex computational problems, NVIDIA Corporation introduced CUDA as a general purpose parallel computing architecture with a new parallel programming model and instruction set architecture. In fact, CUDA is an extended model of standard C language with specific extensions for parallel computing that allows the user to program own algorithms on the GPU easily. CUDA as a GPU programming model is supported by all the latest NVIDIA graphics hardware such as GeForce, Quadro-FX, Tesla, and ION.

Figure 1 illustrates the architecture of a GPU. A GPU has a series of Multiprocessors (MP), and each multiprocessor has a number of Scalar Processors (SP). The memory bank of a MP can be accessed by all its SPs and a large global memory is shared across all the MPs [19]. A typical CUDA program consists of a host code and a device code, where CPU and GPU are the host and the device, respectively. The host performs the nonparallel computations and passes data to the global memory in the GPU and launches a kernel and data-parallel portions of an application are implemented as kernels. The kernel executes the computations using parallel threads on the SPs. The threads are grouped into blocks and blocks are grouped further into grids. A thread block is a 3, 2 or 1-dimensional group of threads and also the grid consist of the blocks which are organized in 3, 2 or 1-dimensional manner. In fact the CUDA programming model organizes threads into a three-level hierarchy as shown in figure 2 Threads within a block can cooperate among themselves by sharing data through shared memory and synchronize their execution to coordinate memory accesses. Threads in different blocks cannot cooperate.

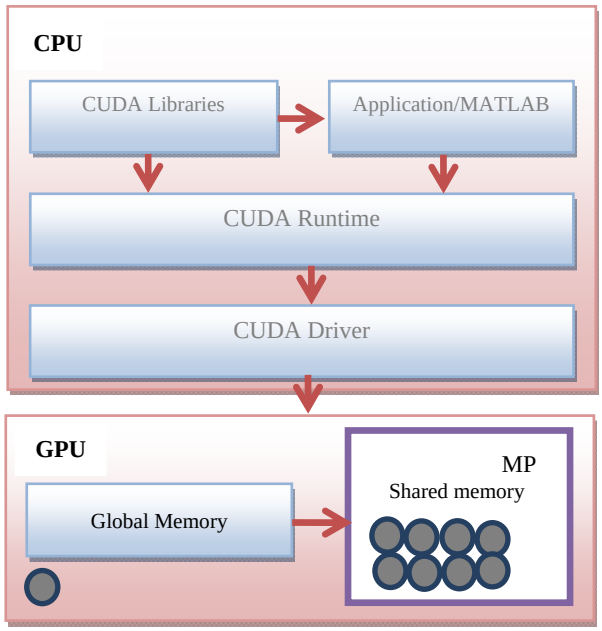


Figure 1. The NVIDIA GPU architecture

The number of threads per block is constrained by the limited memory resources of a processor core. The multiprocessor partitions its thread blocks into groups of 32 parallel threads called warps. When diverging a thread caused by data-dependent conditional branch, the warp including the thread executes each branch path serially. Consequently, the threads that are not on that path are disabled. However, after completing all paths, the threads converge back to the same execution path [17].

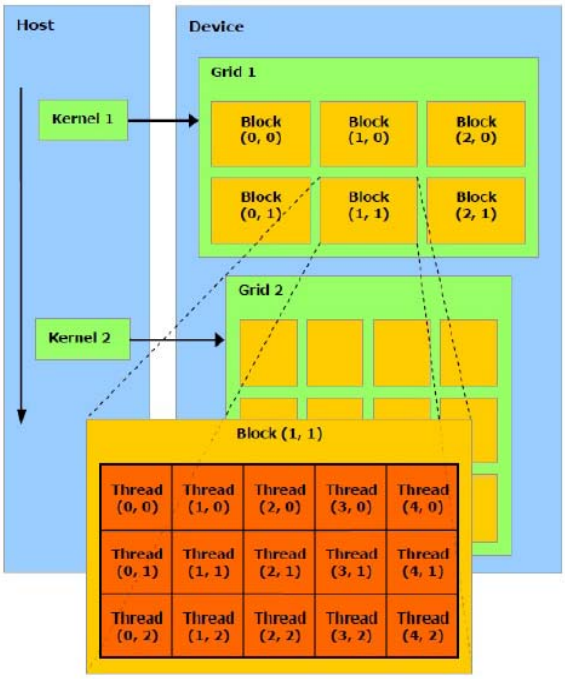


Figure 2. CUDA Software Architecture [20]

3. The Proposed Image Retrieval Procedure

In this section, we provide some information about the basic concepts of CBIR domain and proposed image retrieval procedure.

3.1. The Basic Concepts of CBIR Systems

An effective retrieval system is needed to retrieve the images by the visual features like color, shape and texture. CBIR is performed in two steps include indexing and searching. In the indexing step, features of the images are extracted and stored in feature vectors in the feature database. In the searching step, user query image feature vector is constructed and compared with all feature vectors in the database for similarity to retrieve the most similar images to the query image from the database [21]. In fact, the relevance comparison is performed by using some distance measurement technique, and the minimum or permissible distances are the metrics for the matched or similar images. figure 3 shows the architecture of CBIR systems.

3.2. CBIR System Using the Fusion of Edge Histogram Descriptor and Color Moments

In this section, Edge Histogram Descriptor and color moments, as two important features used for image retrieval are investigated. The procedure for generating local, global and semi-global edge histograms proposed by Won et al. [13] is also explained in details.

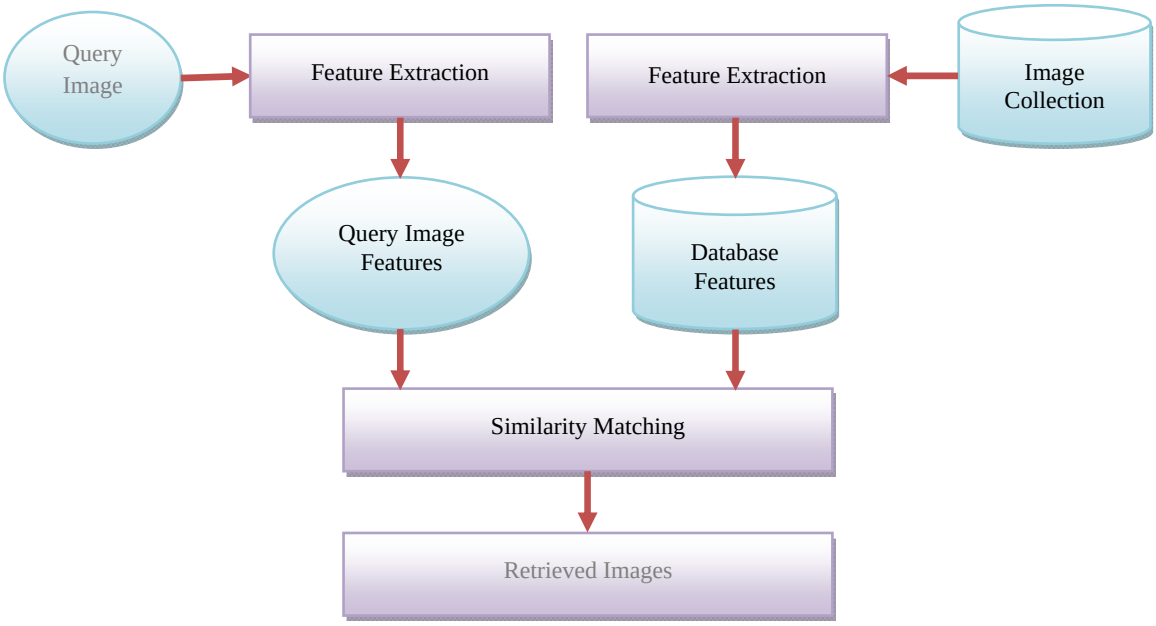


Figure 3. Architecture of CBIR systems

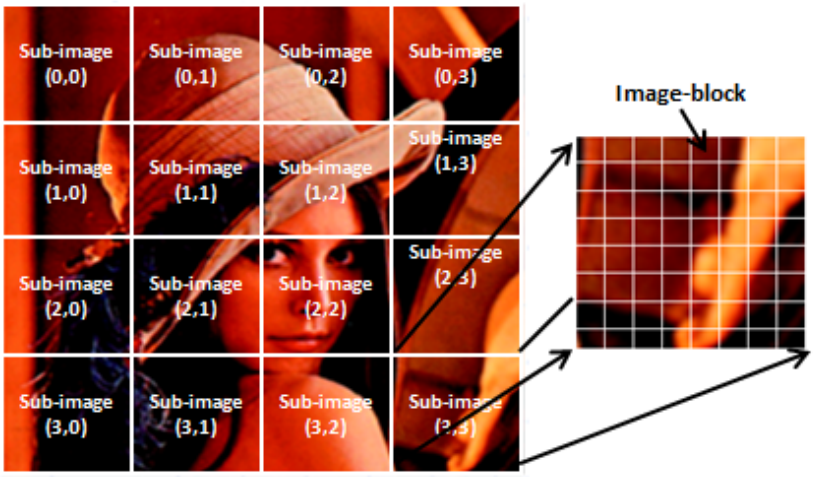


Figure 4. Definition of sub-image and image block

(a) vertical edge (b) horizontal edge (c) 45 degree edge (d) 135 degree edge (e) non-directional edge

Figure 5. Five types of edges [13]

3.2.1. Feature Extraction Using Edge Histogram Descriptors

A given image is first divided into 16 sub-images (4×4), and local edge histograms are computed for each sub-image. To compute the edge histogram, each of the 16 sub-images is further subdivided into image-blocks as depicted in figure 4. To describe the sub-image, a histogram of edge distribution for each sub-image is created. As shown in figure 5, edges in the sub-images are classified into 5 types: vertical, horizontal, 45-degree diagonal, 135-degree diagonal and non-directional edges. Each local histogram includes 5 bins. Each bin corresponds to one of 5 edge types. Each histogram bin value should be normalized. For this purpose, the number of edges for each bin is divided by the total number of image-blocks in the sub-image. Each image-block is classified into one of the 5 types of edge blocks or a non-edge block.

For normalizing process, the amount of each histogram bin is divided into the total number of image blocks. To extract directional edge features, defining small square image-blocks in each sub-image is performed. The image space is divided into non-overlapping square image-blocks and then the edge information from them is extracted. Applying digital filters in the spatial domain is one of the most useful methods for extracting edge features in the image-block [13]. In the first step, the image-block is divided into four sub-blocks as depicted in figure 6. By assigning labels for four sub-blocks from 0 to 3, the average gray levels for four sub-blocks at (i,j)th image-block as $a_0(i,j)$, $a_1(i,j)$, $a_2(i,j)$, and $a_3(i,j)$, respectively is presented.

Also, Won et al. [13] represented the filter coefficients for vertical, horizontal, 45-degree diagonal, 135-degree diagonal, and non-directional edges as $f_v(k)$, $f_h(k)$, $f_{d\text{-}45}(k)$, $f_{d\text{-}135}(k)$, and $f_{nd}(k)$, respectively, where k=0,…,3 represents the location of the sub-blocks. Now, the respective edge magnitudes $m_v(i,j)$, $m_h(i,j)$, $m_{d\text{-}45}(i,j)$, $m_{d\text{-}135}(i,j)$, and $m_{nd}(i,j)$ for the (i,j)th image-block can be obtained as follows:

$$m_v(i,j) = \left| \sum_{k=0}^{3} a_k(i,j) \times f_v(k) \right| \quad (1)$$

$$m_h(i,j) = \left| \sum_{k=0}^{3} a_k(i,j) \times f_h(k) \right| \quad (2)$$

$$m_{d-45}(i,j) = \left| \sum_{k=0}^{3} a_k(i,j) \times f_{d-45}(k) \right| \quad (3)$$

$$m_{d-135}(i,j) = \left| \sum_{k=0}^{3} a_k(i,j) \times f_{d-135}(k) \right| \quad (4)$$

$$m_{nd}(i,j) = \left| \sum_{k=0}^{3} a_k(i,j) \times f_{nd}(k) \right| \quad (5)$$

If the maximum value among five edge strengths obtained from (1) to (5) is greater than a threshold T_{edge} (T_{edge} = 11), then the image-block is considered to have the corresponding edge in it [13], otherwise, the image-block contains no edge. In MPEG-7 XM Document [22], a set of filter coefficients, depicted in figure 7, is recommended. To achieve a high retrieval performance, the local histogram alone is not enough. Beside the local histogram, Won et al. [13] generated the global edge histogram and some semi-global edge histograms. The global edge histogram represents the edge distribution for the whole image space. Since there are five edge types, the global edge histogram also has five bins. For the semi-global edge histograms, Won et al. clustered some sub-images as shown in figure 8.

There are 13 different clusters and for each cluster they generated edge distributions for five different edge types. Consequently, there are total 80 bins(local) + 5 bins(global) + 65 bins (13×5, semi-global) = 150 bins. Note that the bin values for all global and semi-global histograms can be obtained directly from the local histogram. For the similarity matching, Won et al. [13] calculated the distance D(A,B) of two image histograms A and B using (6).

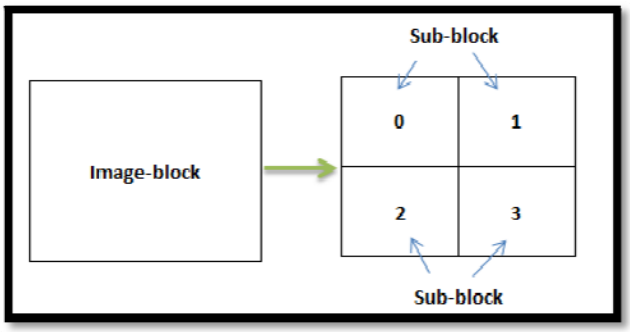


Figure 6. Sub-blocks and their labeling

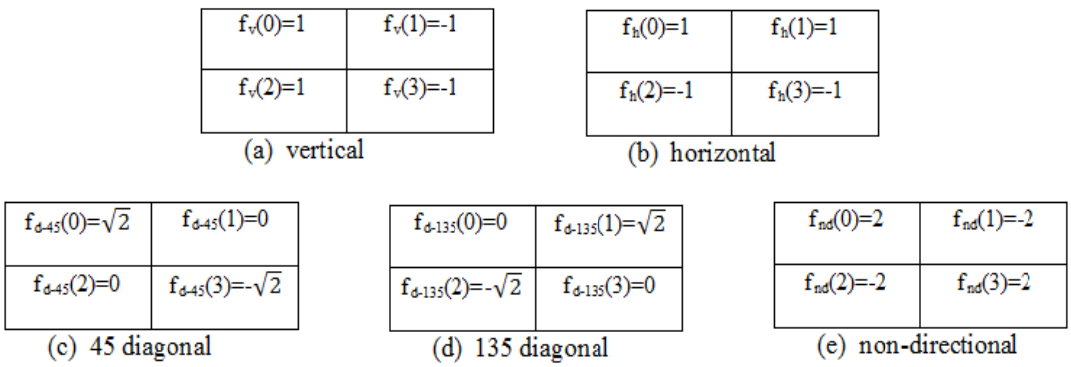


Figure 7. Filter coefficients for edge detection

$$D(A,B) = \sum_{i=0}^{79} \left| Local\_A[i] - Local\_B[i] \right| + 5 \times \sum_{i=0}^{4} \left| Global\_A[i] - Global\_B[i] \right| + \sum_{i=0}^{64} \left| S\_Global\_A[i] - S\_Global\_B[i] \right| \quad (6)$$

Where Local_A[i] represents the reconstructed value of Bin-Count[i] of image A. Similarly, Local_B[i] is the reconstructed value of Bin-Count[i] of image B. Global_A[] and Global_B[] represent the normalized bin values for the global edge histograms of image A and image B, respectively. Similarly, Semi_Global_A[] and Semi_Global _B[] represent the normalized histogram bin values for the semi-global edge histograms of image A and B, respectively. Since the number of bins of the global histogram is relatively smaller than that of local and semi-global histograms [13], a weighting factor of 5 is applied in (6).

3.2.2. Feature Extraction Using Color Moments

There are three important methods for image retrieval: extracting color, shape and texture futures. Utilization of color moments is one of much useful techniques in color based image retrieval. In this paper, we used color moments including mean, standard deviation, and skewness to extract color features. The sub-image is defined by dividing the image space into 4×4 non overlapping blocks. Thus, the image partition always yields small equal-sized sub-images regardless of the size of the original image. To characterize the sub-image, we then generate color moments for each sub-image.

The first-order (mean), the second (standard deviation), and the third-order (skewness) color moments have been proved to be efficient and effective in representing color distributions of images. If the value of the i-th color channel at the j-th image pixel is P_{ij}, then the color moments are as follows.

Moment 1: Mean

$$\mu_i = \frac{1}{N} \sum_{j=1}^{N} P_{ij} \quad (7)$$

Moment 2: Standard deviation

$$\sigma_i = \sqrt{\frac{\sum_{j=1}^{N} (P_{ij} - \mu_i)^2}{N}} \quad (8)$$

Moment 3: Skewness

$$S_i = \sqrt[3]{\frac{\sum_{j=1}^{N} (P_{ij} - \mu_i)^3}{N}} \quad (9)$$

For color image, color moments are very compact representation features compared to other color features. When a query image is submitted for image retrieval, its color feature is extracted and matching operation is performed between query image features and the image features stored in database, then the closest results to the query image are retrieved from the database.

3.3. Parallel Implementation Using CUDA

As the image is two dimensional, execution of the kernel creates a two dimensional grid to efficiently map threads on the image pixels. Each thread has its own id, thus making it possible to assign a specific cell to each thread. In our problem, there are 16 blocks in the grid and also 64 threads in each block. The image contains 16 sub-images and the total number of image-blocks within the sub-image is set to 64. In order to get this aim, grid dimensions can be set to 4×4 and block dimensions to 8×8. So for visual feature extraction, each sub-image is mapped to one block and each image-block is processed by one thread.

Therefore edge type recognition and color feature extraction is performed in all image-blocks in parallel by CUDA threads. In fact, the kernel is responsible for edge detection and color feature extraction in the image-block. The image is an input arguments of the kernel and a data structure of features is the output. All the threads run the same kernel instructions but with different data. Thread position and image-block which should be processed by the thread can be determined by the blockIdx and threadIdx as:

```
data_per_thread_x = image_height /
  (gridDim.x*blockDim.x);

data_per_thread_y = image_width /
  (gridDim.y*blockDim.y);

row_index = (blockIdx.x*blockDim.x +
        threadIdx.x)*data_per_thread_x;

column_index = (blockIdx.y*blockDim.y +
        threadIdx.y)*data_per_thread_y;
```

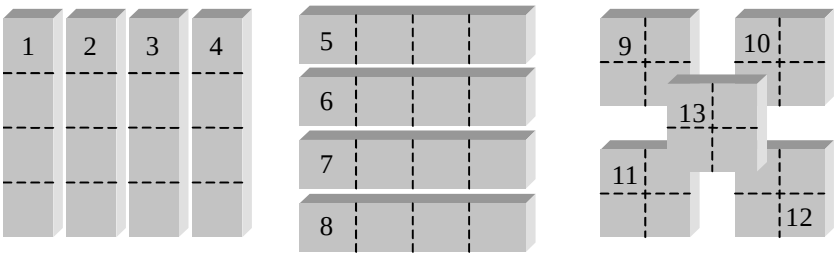


Figure 8. Cluster of sub-images for semi-global histograms [13]

Where image_height and image_width are the horizontal and vertical image resolutions, and row_index, column_index are two indexes to the image matrix that specify the beginning of the image-block. Also, the grid dimensions are given by variables gridDim.x, gridDim.y and the block dimensions are given by variables blockDim.x, blockDim.y. The threads of each block compute the number of edge types and color features in a sub-image. When a thread specifies an edge, or computes color features, its relevant elements in the data structure of features is updated. Therfore, there are 16×64 parallel threads to process 16×64 independent parts of the image. The threads are well organized, so that not only there is an efficient mapping between the thread blocks and the image parts, but also the number of threads in each block (64) is multiple of number of threads of warp (32). It means processing adjacent image parts by the same core. The following pseudo code describes the proposed strategy.

1- The data structure of features is composed of two parts. One part is a two-dimensional array for edge features and another is a vector to store 9 color features including Mean, Standard deviation and Skewness, for the three components (red, green and blue) in each sub-image. The two-dimensional array has 80 columns (corresponding to 80 local edges) and 64 rows for 64 threads in each block. Elements of the data structure of features are initially set to zero.

2- The Grid dimensions and block dimensions are determined, so that there are 16×64 parallel threads to process 16×64 independent parts of the image.

3- Query image is transmitted to the device memory and the kernel is called.

3- Each thread processes its corresponding image-block and then updates its relevant elements in the data structure of features. In order to extract color and edge features, the image-block is processed as described in sub-sections 3.1 and 3.2.

4- The data structure of features is transmitted to the host memory.

5- Local-query vector containing number of local edges is created by the two-dimensional array. For this, the elements of each column of the two-dimensional array which are 0 or 1 (depending on the recognition of its corresponding thread) are summed up and resultants are stored in Local-query vector, containing 80 elements.

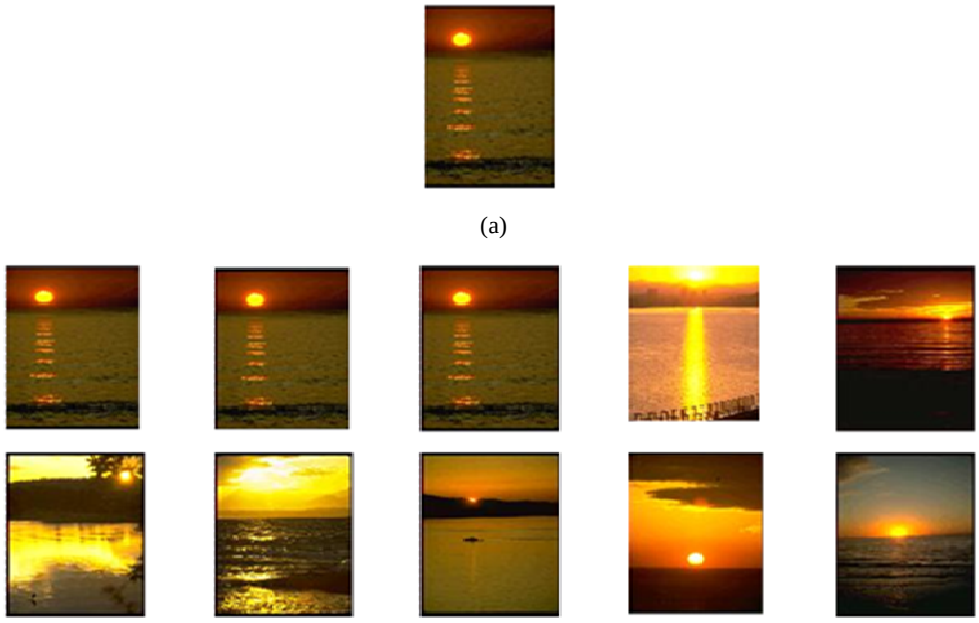
6- The Global-query vector and Semi-global vector are created by means of Local-query vector as described in sub-section 3.2.

7- Now there are two feature vectors, one edge features (local, global and semi global) and another color features. These two vectors are used to determine most similar images based on the lowest difference of the elements of the vectors. This difference of two images A and B is calculated as:

$$\begin{aligned} D(A,B) = & \sum_{i=0}^{79} \left| Local\_A[i] - Local\_B[i] \right| + \\ & 5 \times \sum_{i=0}^{4} \left| Global\_A[i] - Global\_B[i] \right| \\ & + \sum_{i=0}^{64} \left| S\_Global\_A[i] - S\_Global\_B[i] \right| \\ & + \sum_{i=0}^{47} | \mu\_A[i] - \mu\_B[i] | + \\ & \sum_{i=0}^{47} | \sigma\_A[i] - \sigma\_B[i] | + \sum_{i=0}^{47} | S\_A[i] - S\_B[i] | \end{aligned} \quad (10)$$

Where, color features are applied by means of the three last summations. For simultaneous retrieval of different images, the query images are integrated in one array and then are transmitted to device memory

Figure 9. Samples of MPEG-7 image database



(a)

(b)

Figure 10. Content based image retrieval results (a) input image for retrieval (b) using the proposed method

N×16 blocks and *N*×16×64 parallel threads are considered to extract edge features. The number of images which are simultaneously transmitted to the GPU depends on the memory capacity.

It is important to note that there are factors limiting the speed up such as:

- The image should be copied to GPU global memory before execution of the kernel and Bin-Count array should be copied to CPU's memory after execution of the kernel. The data communication overhead has a significant impact on the speed up.
- Logical operations may force the threads within a warp to be serialized, reducing the speed up. However, the studied application does not require to any further optimization process due to its data-independent and parallel nature.

4. Experimental Results

Recall and Precision are two common set-based measures used to evaluate the performance of an information retrieval system. The Recall is a measure of the ability of a system to present all relevant items and is defined as the ratio of the number of relevant items retrieved and the number of relevant items in class. The Precision is a measure of the ability to present only relevant items and is defined as the ratio between the number of relevant items retrieved and the total number of items retrieved [24].

Precision and Recall is defined as:

$$\mathrm{Re}call = \frac{\text{Number of relevant images retrieved}}{\text{Total number of relevant images}} \tag{11}$$

$$\mathrm{Pr}ecision = \frac{\text{Number of relevant images retrieved}}{\text{Total number of images retrieved}} \tag{12}$$

The average Precision for the images that belongs to the qth category (A_q) has been computed by:

$$Average\_\mathrm{Pr}ecision = \sum_{k \in A_q} \frac{p(i_k)}{|A_q|} \tag{13}$$

The database contains 230 images organized in 23 equivalence classes that samples of MPEG-7 image database are shown in figure 9. Serial implementation of the image retrieval technique is done in C language using a PC with Intel Pentium 2.5 GHz and 4 GB RAM. The GPU calculation speed is 15.12 times as fast as the CPU calculation speed, when running on a GPU named NVIDIA GeForce GT610M. The GeForce GT610M is an entry-level card with 48 CUDA cores and 900 MHz core clock speed. figure 10 shows the results generated from our proposed system that show the efficiency of our proposed approach and have an average retrieval time as 0.26 seconds. These results show that the performance of the proposed method is better than the other methods.

The average Precision and the average Recall of our proposed method are 71.53% and 55% respectively. One graph that describes the performance of the system is the Precision-Recall graph. It provides a meaningful result when the database is known and has been used be some earlier systems. To evaluate our proposed system, we use the Precision-Recall graph, that Precision can be plotted against Recall after each retrieved item. Precision and Recall are widely used in combination to characterize retrieved performance, usually giving rise to the well-known

hyperbolic graphs from high Precision, low Recall towards low Precision, high Recall values.

We select all images from each class in the database to use them as queries to calculate the Precision and Recall. For each image, the Precision of the retrieval result is obtained by increasing the number of retrieved images. figure 11 (a) to (c) show the Precision-Recall graph for when the color feature, the edge histogram descriptor, and the fusion of color moments and edge histogram descriptor respectively extracted from images. In figure 11 (d), the average Precision and Recall curves of the proposed algorithm for different number of retrieved images is plotted. It can be seen from figure that the proposed algorithm achieved the best retrieval accuracy. In addition, for the presented method, the maximum average Precision of 100% at Recall value is 10%, and the Precision value decreases to 44.4367% at 100% of Recall. To evaluate our proposed system, we use each image in our database to be a query image and submit it to the system. We calculate the Precisions for each query in all classes. Then we take the average of all calculated Precisions as shown in table 1. Recall, Precision, Average Precision (AP) and Average Recall (AR) for the proposed method are in table 1.

Content based retrieval method manages to find images similar to query image in database but with a drawback of a lot of time consumption. It is evident that the serial implementation on CPU does not need steps 2, 3, 5 and 6 as described in sub-section 3.3. We profiled the CPU implementation and found that from 78.4% to 88.3% of the execution time is spent on step 4 which is parallelized by the CUDA implementation. The ratio depends on the image size as shown in table 2.

With larger image sizes, the portion is raised, since the number of pixels which must be processed is increased. On the other hand, steps 1, 7 and 8, consume approximately a constant time (due to the equality of feature vectors for different image sizes). Consequently, as shown in table 3, the achieved speedup is increased with higher resolution images. table 3 also shows the execution times for varying sizes, using 2000 iterations.

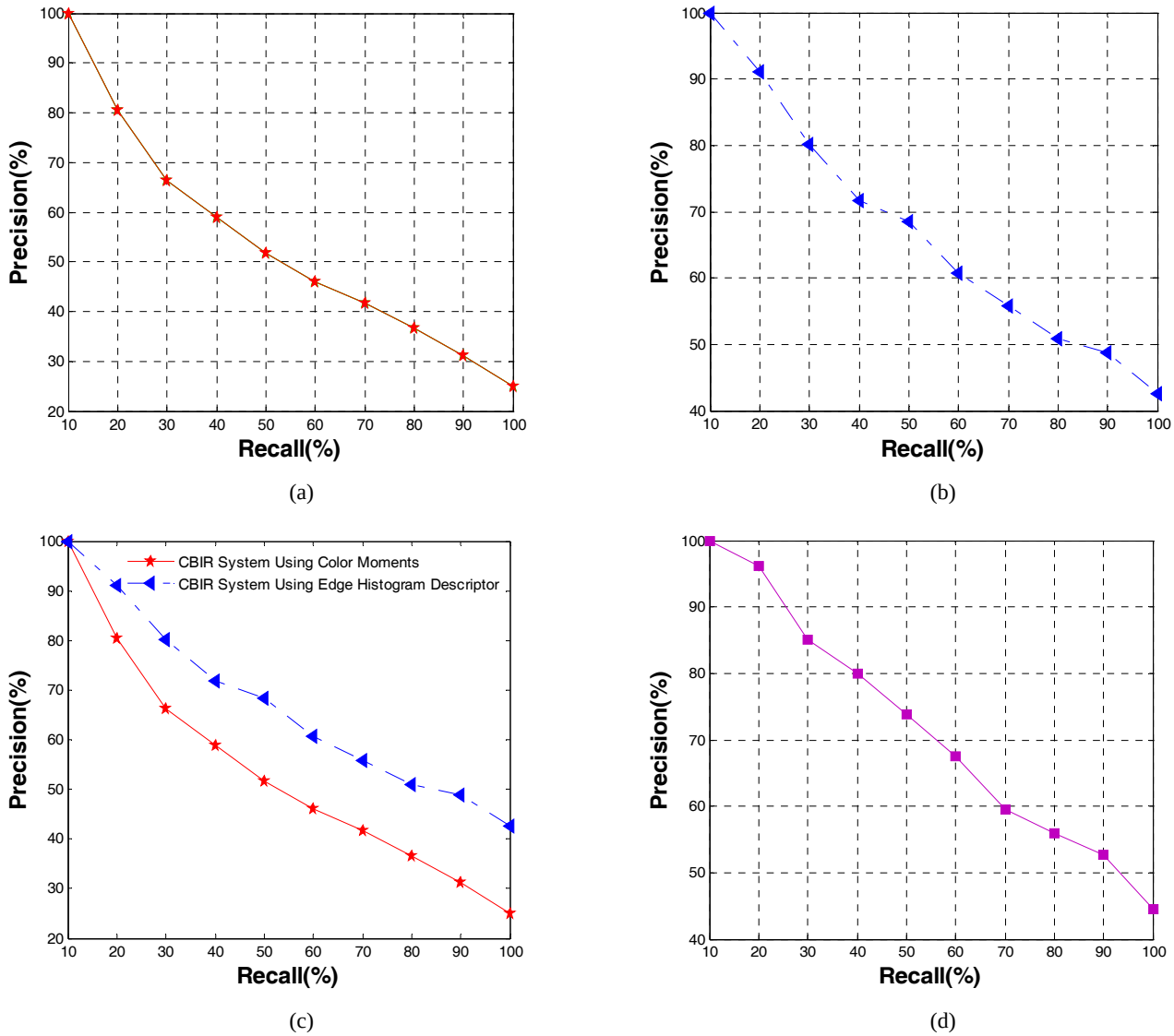


Figure 11. The average precision/recall chart for (a) color based image retrieval system (b) edge histogram descriptor based image retrieval system (c) CBIR system using the fusion of color moments and edge histogram descriptor (d) the proposed system

Table 1. Precision and recall of the proposed method

Recall (%)	Precision (%) for the proposed method
10	100
20	96.123
30	85.1354
40	79.987
50	73.7654
60	67.5546
70	59.567
80	55.9953
90	52.7683
100	44.4367
AR = 55%	**AP = 71.53%**

Table 2. Profiling CPU code for the portion with parallelization potential

Resolution (a×a)	The Time Spent for Step 4 Per Total Execution Time (%)
300	78.4
400	81.7
500	85.1
600	88.3

It is important to mention that there are some overheads that reduce desired performance. These are: (1) organization of the data structure of features and creation of Local-query vector; (2) transferring data between host and device. However, there are some reasons for getting reasonable speedup. These are: (1) a significant portion of the algorithm is parallelized by the CUDA implementation; (2) the studied application has a data-independent nature and we have used an intensive data-parallel algorithm. Here, an appropriate organization of threads has been established and different portions of the image are efficiently mapped on the CUDA threads. As a significant result, an average speed up of 15.12 was achieved by our CUDA implementation.

Table 3. Execution times for CPU and GPU implementations

Resolution (a×a)	CPU Runtime [s]	GPU Runtime [s]	Speed up	Speed up Average
300	1.97	0.19	10.37	15.12
400	3.73	0.24	15.54	
500	4.78	0.29	16.48	
600	5.61	0.31	18.1	

The average retrieval time on GPU and CPU is shown in Figure 12. As we can see, the simple parallel implementation obtains almost 15 times speedup, illustrating difference between CPU and GPU performance in inherent data-parallel applications.

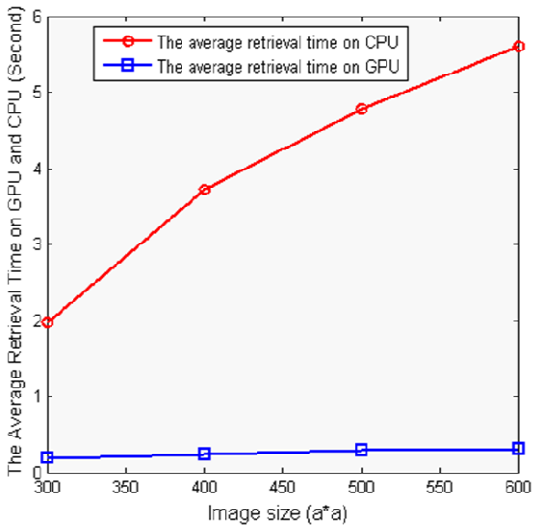


Figure 12. The average retrieval time on GPU and CPU for varying the problem sizes for 2000 images

5. Conclusion and Future Work

In this paper, parallel implementation was performed for Edge Histogram Descriptor and color moments fused features using CUDA programming model to run on GPU. This descriptor characterizes the distribution of various types of edges with a histogram that can be a tool for image matching. Experimental results showed that the use of GPU, reduced the retrieval time by a factor of 15.12. Our method was evaluated using Precision, Recall, and average Precision measures. The average Precision and the average Recall of proposed method are 71.53% and 55.00% respectively.

The main purpose of this research is parallel implementation of EHD extraction and color moments fused features on the GPU (GeForce GT610M), as compared to serial implementation on the CPU. Since our case study is an intensive data-parallel application, it is well-suitable to implement on many-core architecture like the GPU. However, CPU code can be implemented using Open-MP to run on a multi-core CPU in parallel, achieving higher performance. This can be considered as a new research direction. Furthermore, the work can be extended for translation, rotation, and scale invariance properties, so that the retrieval efficiency can be more increased.

References

[1] S. Youssef, "ICTEDCT-CBIR: Integrating Curvelet Transform with Enhanced Dominant Colors Extraction and Texture Analysis for Efficient Content-based Image Retrieval," *Journal of Computers and Electrical Engineering*, vol. 38, no. 3, pp. 1358-1376, 2012.

[2] J. Yue, Z. Li, L. Liu, and Z. Fu, "Content-based Image Retrieval Using Color and Texture Fused Features," *Journal of Mathematical and Computer Modeling*, vol. 54, no. 4, pp. 1121-1127, 2011.

[3] X. Wang, Y. Yu, and H. Yang, "An Effective Image Retrieval Scheme Using Color, Texture and Shape Features," *Proc. IEEE Intl Conf. Computers Standard & Interfaces*, pp. 59-68, 2011.

[4] C. Theoharatos, V. Pothos, N. Laskaris, and G. Economou, "Multivariate Image Similarity in the Compressed Domain Using Statistical Graph Matching," *Proc. IEEE Intl Conf. Pattern Recognition*, pp. 1892-1904, 2006.

[5] D. Zhang, A. Wong, M. Indrawan, and G. Lu, "Content-based Image Retrieval Using Gabor Texture Features," *IEEE Trans. Patt. Anal. and Mach. Intell.*, vol. 12, no. 10, pp. 1-4, 2000.

[6] M. Kokare, P. Biswas, and B. Chatterji, "Texture Image Retrieval Using New Rotated Complex Wavelet Filter," *IEEE Trans. on Systems*, vol. 10, no. 2, pp. 1168-1178, 2005.

[7] C. Pun, "Rotation-invarient Texture Feature for Image Retrieval," *Proc. IEEE Intl Conf. Computer Vision and Image Understanding*, pp. 24-43, 2003.

[8] P. Huang, and S. Dai, "Image Retrieval by Texture Similarity," *Proc. IEEE Intl Conf. Pattern Recognition*, pp. 665-679, 2003.

[9] J. Vogel, and B. Schiele, "Performance Evaluation Optimization for Content-based Image Retrieval," *Proc. IEEE Intl Conf. Pattern Recognition*, pp. 897-909, 2006.

[10] D. Tralic, J. Bozek, and S. Grgic, "Shape Analysis and Classification of Masses in Mamographic Images Using Neural Networks," *Proc. IEEE Intl Conf. Signal and Image Processing*, pp. 1-5, 2011.

[11] W. Kejia, Z. Honggang, C. Lunshao, and H. Ying, "A Comparative Study of Moment-based Shape Descriptor for Product Image Retrieval," *Proc. IEEE Intl Conf. Pattern Recognition*, pp. 355-359, 2011.

[12] T. Adamek, and E. Connor, "A Multiscale Representation Method for Nonrigid Shapes with a Single Closed Contour," *IEEE Trans. Patt. Anal. and Mach. Intell.*, vol. 12, no. 10, pp. 742-753, 2004.

[13] C. Won, D. Park, and S. Park, "Efficient Use of MPEG-7 Edge Histogram Descriptor," *Proc. IEEE Intl Conf. Pattern Recognition*, pp. 23-30, 2002.

[14] Y. Komura, and Y. Okabe, "GPU-based Single-cluster Algorithm for the Simulation of the Ising Model," *Proc. IEEE Intl Conf. Computational Physics*, pp. 1209-1215, 2012.

[15] J. Stone, D. Hardy, I. Ufimtsev, and K. Schulten, "GPU-accelerated Molecular Modeling Coming of Age," *Proc. IEEE Intl Conf. Molecular Graphics and Modelling*, pp. 116-125, 2010.

[16] F. Yi, I. Moon, J. Lee, and B. Javidi, "Fast 3D Computational Integral Imaging Using Graphics Processing Units," *Proc. IEEE Intl Conf. Pattern Recognition*, pp. 714-722, 2012.

[17] "NVIDIA Corporation, NVIDIA CUDA C Programming Guide 4.1," May 2011.

[18] "General Purpose GPU Programming (GPGPU)," http://www.gpgpu.org, June 2010.

[19] P. Sattigeri, J. Thiagarajan, K. Ramamurthy, and A. Spanias, "Implementation of a Fast Image Coding and Retrieval System Using a GPU," *Proc. IEEE Intl Conf. Emerging Signal Processing Applications*, pp. 5-8, 2012.

[20] N. Sawant, and D. Kulkarni, "Performance Evaluation of Feature Extraction Algorithm on GPGPU," *Proc. IEEE Intl Conf. Pattern Recognition Communication Systems and Network Technologies*, pp. 536-540, 2011.

[21] F. Malik, and B. Baharudin, "Analysis of Distance Metrics in Content-based Image Retrieval Using Statistical Quantized Histogram Texture Features in the DCT Domain," *Proc. IEEE Intl Conf. Computer and Information Science*, pp. 1-12, 2012.

[22] M. Swain, and D. Ballard, "Color Indexing," *Proc. IEEE Intl Conf. Computer Vision*, pp. 11-32, 1991.

[23] D. Donno, A. Esposito, L. Tarricone, and L. Catarinucci, "Introduction to GPU Computing and CUDA Programming: A Case Study on FDTD," *IEEE Tran. Antennas and Propagation*, vol. 5, no. 1, pp. 116-122, 2010.

[24] C. Ruberto, and L. Cinque, "Decomposition of Two-Dimensional Shapes for Efficient Retrieval," *Proc. IEEE Intl Conf. Image and Vision Computing*, pp. 1097-1107, 2009.

Alireza Ahmadi Mohammadabadi received the B.S. degree in Computer Engineering from Azad University of Kashan, Iran in 2011. He obtained his M.Sc. degree in Computer Engineering from Razi University, Kermanshah, Iran. His research interests are in the fields of parallel architectures, evolutionary algorithms, image and multimedia signal processing.
E-mail: alireza.ahmadi.computer@gmail.com

Abdolah Chalechale Born in Kermanshah, Iran, received his B.S. and M.Sc. degrees in Electrical Engineering (Hardware) and Computer Engineering (Software) from Sharif University of Technology, Tehran, Iran. He received his Ph.D. degree from Wollongong University, NSW, Australia in 2005 and currently is with Razi University, Kermanshah,

Iran. His research interests include image processing, machine vision and human-machine interactions.
E-mail: chalechale@razi.ac.ir

Hadis Heidari received the B.S. and M.Sc. degrees in Computer Engineering from the University of Razi, Kermanshah, Iran, in 2011 and 2013, respectively, where she is currently pursuing the Ph.D. degree in Computer Architecture from the same University. She was first rank of students in Computer Engineering in the Razi University. Her interests include operating systems, computer networks, reconfigurable computing, image and multimedia signal processing, with particular attention to computer vision applications for content based image retrieval and image processing.
E-mail: hadis.68.heidari@gmail.com

Paper Handling Data:

Submitted: 03.10.2013
Received in revised form: 03.02.2014
Accepted: 06.04.2014
Corresponding author: Dr. Abdolah Chalechale,
Faculty of Engineering, Razi University of Kermanshah, Kermanshah, Iran.