A New Framework for Automatic Extraction of Key Frames Using DC Image Activity

Kang-Wook Kim
Mobile Communication Division of Samsung Electronics Co., Ltd
129, Samsung-ro Yeongtong-gu, Suwon-si, Gyeonggi-do 443-742, Korea
[e-mail: ekans999@gmail.com]

Received September 3, 2014; revised October 26, 2014; accepted October 29, 2014; published December 31, 2014

Abstract

The effective extraction of key frames from a video stream is an essential task for summarizing and representing the content of a video. Accordingly, this paper proposes a new and fast method for extracting key frames from a compressed video. In the proposed approach, after the entire video sequence has been segmented into elementary content units, called shots, key frame extraction is performed by first assigning the number of key frames to each shot, and then distributing the key frames over the shot using a probabilistic approach to locate the optimal position of the key frames. Moreover, we implement our proposed framework in Android to confirm the validity, availability and usefulness. The main advantage of the proposed method is that no time-consuming computations are needed for distributing the key frames within the shots and the procedure for key frame extraction is completely automatic. Furthermore, the set of key frames is independent of any subjective thresholds or manually set parameters.

Keywords: Key Frame, Extraction, Compressed Video, Android

http://dx.doi.org/10.3837/tiis.2014.12.017

1. Introduction

Many services, such as VOD (video on demand) and pay television, are provided in digital form to consumers and a rapidly increasing number of interactive multimedia documents, including text, audio, and video, are now available. Consequently, it is widely recognized that there is a need for intelligent management and search methods particularly for visual information in multimedia documents and digital videos. Efficient access to video data located in a distributed database is a very difficult task, mainly due to the large bandwidth requirements imposed by the large amount of video information. Traditionally, video is represented by numerous consecutive frames, each of which corresponds to a constant time interval. However, such a representation is not adequate for new emerging multimedia applications, such as content-based indexing, retrieval, and browsing. Furthermore, tools and algorithms for the effective organization and management of video archives are still limited [1]. There is also an essential need to automatically extract key information from images and videos for the purpose of indexing, fast and easy retrieval, and scene analysis. In order to allow the user to efficiently browse, select, and retrieve a desired video part without having to deal directly with GBytes of compressed data, several activities have to be carried out in preparation for such a user interaction. For videos, a common first step is to segment the videos into temporal "shots," each representing an event or continuous sequence of actions. A shot is what is captured by the camera between a record and a stop operation. Further scene analysis and interpretation can then be performed on such shots. Segmented video sequences can also be used for browsing, in which only one or a few representative frames, i.e., key frames of each shot are displayed [2]-[5]. The main goal of the above procedures is to provide the user a compact and easily understandable overview of the complete stored video information. Most existing approaches to key frame extraction [6]-[8], based on measuring the differences between the last selected frame and the remaining frames and extracting a subsequent key frame if the measured difference exceeds the given threshold, are typically sequential processes leading to unpredictable results. Particularly, the final number of key frames for entire sequence can’t be estimated and a large number of key frames or too few key frames can be allocated. This makes it difficult to predict the capacity needed to store extracted key frames. Moreover, the dependency on subjective and usually data dependent thresholds, limits its applicability in fully automated systems and leads to bad results.

This paper proposes a new and fast method for extracting key frames from a compressed video. The proposed algorithm can operate directly on various MPEG compressed videos. After the entire video sequence is segmented into elementary content units, called shots, key frame extraction is performed by first assigning the number of key frames to each shot and then distributing the key frames using a probabilistic approach to locate the optimal position of the key frames. The main advantage of the proposed method is that no time-exhaustive computations are needed for distributing the key frames over the shot, plus the procedure of key frame extraction is fully automatic. In addition, the set of key frames is independent of any subjective thresholds or manually given parameters.

Section 2 briefly reviews several previous approach and their drawbacks. In section 3, we explain the concept of our proposed framework for key frame extraction step by step along with simple test results. Experimental results on various video sequences are presented in section 4, demonstrating the performance and validity of the proposed method. Section 5 gives details of software implementation process of the key frame extraction application in Android,

demonstrating the class view and UI structure of the proposed method. Lastly, section 6 gives some final conclusions.

2. Related Work

The use of key frames to represent the content of a video has been previously discussed in many research papers as an efficient way of preserving the temporal information of sequence based on a small amount of data. The underlying assumption is that if these frames are extracted using an appropriate video sampling method, the visual content of each segment of the sequence can be easily understood by looking at given samples. Accordingly, such compact video representation can be suitable for the purpose of video browsing. Furthermore, for query processes involving the search for video parts containing specific objects, the concept of key frames can also be useful.

There have been many previous reports on extracting key frames from an entire video sequence. One simple method for selecting key frames is to take the first frame of each shot. Whereas a more reliable content representation requires a non-uniform sampling of the video shot. In [5], Pentland et al. found that the frames at the beginning and the end of a shot, in the middle of no-motion segments, or in the middle of segments where the camera is tracking a foreground object, are good key frames to represent the content of a shot. Some other approaches [3],[6],[7], based on measuring the differences between the last selected frame and the remaining frames and extracting a subsequent key frame if the measured difference exceeds the given threshold, are typically sequential processes leading to unpredictable results. In particular, since the final number of key frames for an entire sequence can not be estimated, either too large a number of key frames or too few key frames can be allocated, which is ineffective for indexing and browsing. This also makes it difficult to predict the capacity needed to store the extracted key frames in spite of reducing the already obtained key frames. It is also hard, especially in [3] and [6] to relate any parameter value using a threshold setting to the key frame collection resulting from such a setting. Moreover, the dependency on subjective and usually data dependent thresholds, limits the applicability to fully automated systems and produces bad results. A mathematical optimization-based approach to key frame extraction is presented in [8], where certain measures defined in terms of color features are used. However, the drawback of this scheme is that no key frames are allocated to a shot with a short duration.

3. Proposed Key Frame Extraction Algorithm

We propose a new three-step key frame extraction method for efficient video content representation. A block diagram of the proposed architecture is illustrated in **Fig. 1**, and consists of three modules: video segmentation, key frame allocation, and key frame distribution. These three modules are described in the current section. First the video sequence is segmented into distinct video shots, then a mathematical analysis of the video information flow is applied to the frames of each shot. Such an approach provides a more meaningful description of the video content, therefore, the key frame extraction can be implemented more efficiently.

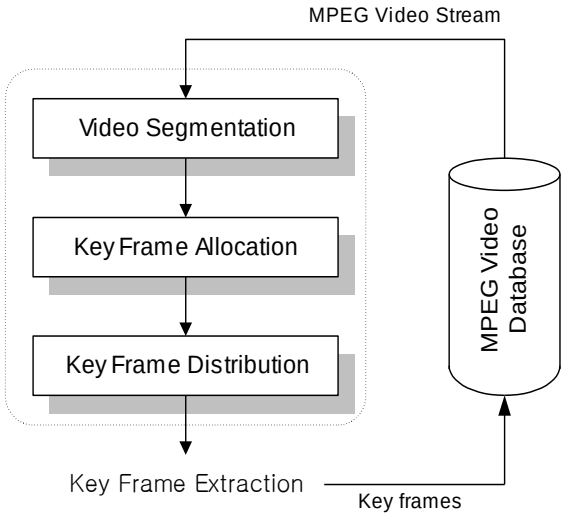


Fig. 1. Block diagram of proposed architecture

3.1 Video Segmentation Using DC Image

DC images are spatially reduced versions of original images. Such spatially reduced images, once extracted, can also be used for other applications beyond scene change detection, for example, the efficient comparison of video shots, automatic generation of compact documents, and nonlinear video browsing applications. This section briefly outlines how a DC image and DC sequence can be efficiently extracted from a compressed video, and illustrates why they are useful for fast and efficient video segmentation operations. A video stream conforming to the MPEG standard is generally composed of I, P, and B type frames. A DC image is obtained from block-wise averages of an 8×8 block. For the I type frame of an MPEG coded video, each pixel in the DC image corresponds to a scaled version of the DC coefficient of each DCT (discrete cosine transform) block. Each DC image is thus reduced 64 times compared to the original image. The challenge is to also extract DC images from P and B type frames, which are coded using motion compensation to exploit the temporal redundancy of video. A generic situation is shown in **Fig. 2**. Here, P_{ref} is the current block of interest, $P_0,\ldots,P_3$ are the four original neighboring blocks from which P_{ref} is derived and the motion vector is $(\Delta x,\Delta y)$.

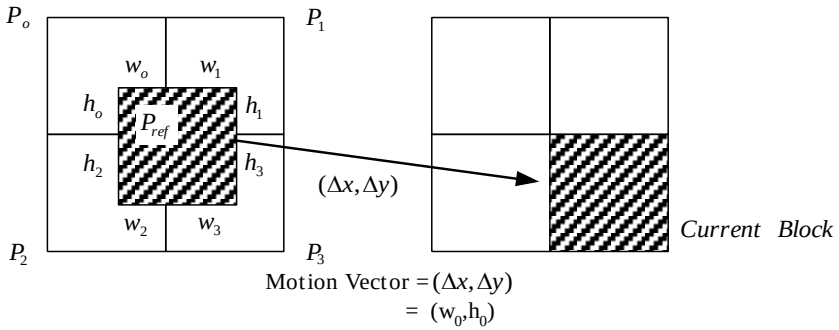


Fig. 2. Reference block (P_{ref}), motion vectors, and original blocks

The shaded regions in $P_0,\ldots,P_3$ are moved by $(\Delta x,\Delta y)$. We are thus interested in deriving the DC coefficients of P_{ref}. By denoting the 2D DCT of an 8×8 block P as $DCT(P)$, the linearity of DCT operations also means that the DC coefficient of $DCT(P_{ref})$ can be expressed as:

$$(DCT(P_{ref}))_{00} = \sum_{i=0}^{3}(\sum_{m=0}^{7}\sum_{l=0}^{7} w_{ml}^{i}(DCT(P_i))_{ml}) \tag{1}$$

, for some weighting coefficients w_{ml}^{i}. The weight w_{00}^{i} is the ratio of overlaps of block P_{ref} with block P_i, i.e., $w_{00}^{i} = h_i w_i / 64$. An approximation, called the first-order approximation, approximates $(DCT(P_{ref}))_{00}$ by

$$DC(P_{ref})^1 = \sum_{i=0}^{3}\frac{h_i w_i}{64} DC(P_i) \tag{2}$$

This approximation, when applied to B and P type frames, yields good results in practice. Such approximation only requires motion vector information and DC values in the reference frames. Several algorithms [9]-[11] to extract DC images from an MPEG compressed video using DCT DC coefficients in an I type frame and motion compensated DCT DC coefficients in P or B type frames have already been proposed. **Fig. 3** illustrates an original image of size 352×240 and its DC image of size 44×30.

Fig. 3. Full image at 352×240 and DC image at 44×30

It has been demonstrated that even at such a low resolution, global image features useful for specific classes for content-based operations with MPEG compressed video streams are well preserved. After extracting the DC image from an MPEG compressed video, the next step is to detect the cuts, i.e., shot boundaries to segment the video into individual shots. To minimize the influence of non-relevant temporal variations, global frame visual features such as color and intensity histograms need to be used to detect a shot boundary. The proposed approach adapts the method proposed in [9] and defines an *activity function* $AF(k)$ for describing the relevant difference between frames k and $k-1$ as:

$$AF(k) = \sum_i \sum_j \left| I_{DC}^{k}(i, j) - I_{DC}^{k-1}(i, j) \right| \tag{3}$$

, where k is the frame index, and $I_{DC}^{k}(i, j)$ means the pixel value at position (i, j) in the DC image. $AF(k)$ measures the relative changes between two consecutive frames, thereby indicating the magnitude of any changes. An $AF(k)$ curve is used to detect the cuts, as illustrated in [9]. The method of [9] uses a sliding window to examine a few successive frame differences. Here, a scene change from frame $k-1$ to k is declared if

1) $AF(k)$ is the maximum within a sliding window of size $2W$, and
2) $AF(k)$ is n times the second largest maximum in the sliding window.

W is set to be smaller than the minimum duration between two scene changes. For example, setting $W = 15$ for a 15 frames/s video means that there cannot be two scene changes within one second. It has been found that values of n ranging from 2.0-3.0 produce the best result. This method also reduces false detections in the case of significant object or camera motions. **Fig. 4** illustrates the plot of $AF(k)$ versus k for a 1000-frame clip from a SBS (Seoul Broadcasting System) TV sports news program. From the $AF(k)$ curve, the video sequence was determined to consist of 7 shots.

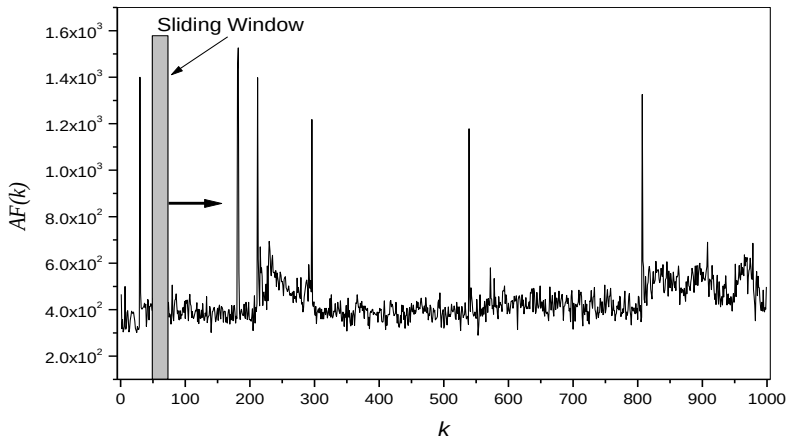


Fig. 4. Plot of $AF(k)$ versus k for 1000 frames

If the entire video sequence is segmented into shots by the above mentioned method, the next step is that we should properly assign the number of key frames to each shot and then distribute the key frames over the shot. In the following sections, we will refer to these procedures.

3.2 Key Frame Allocation to Shots

To represent video shots, it is important to properly decide on the number of key frames (or representative frames) and then select these key frames from each shot. Generally, this is not an easy or automatic task because the decision is subjective to each person. Selecting one key

frame for each shot is presented in [12]. However, a single key frame is very often unable to provide sufficient information about the video content of a given shot, especially for shots with a long duration. Moreover, important shots of small duration may have no key frames, while shots of longer duration may be represented by multiple frames with a similar content.

We propose a simple intuitively appealing algorithm for allocating the number of key frames for each shot. This algorithm may not be optimal, but it allocates key frames to shots incrementally, one key frame at a time, in a way that yields a good assignment. The basic idea is that in each of a total of K_T key frames, one key frame is allocated where it will do the most good at this point. Let $M_i(K_i)$, called the content function, denote the content of the i th shot for the key frame allocation of K_i key frames. The content function of each shot is defined by

$$M_i(K_i) = g_i CAF_i(L) 2^{-2(K_i-1)} \tag{4}$$

, where $CAF_i(n)$ is the accumulated value of $AF(k)$ from the beginning up to the final summation position n and $g_i = g$ is a constant independent of i for simplicity. $CAF_i(n)$ can be calculated as follows:

$$CAF_i(n) = \sum_{k=1}^{n} AF(k) \tag{5}$$

, where i, k are the shot and frame index, respectively. If the summation of Eq. (3) stretches through the entire frame within a shot, $CAF_i(L)$ can be easily calculated. In $CAF_i(L)$ of Eq. (4), L is the number of frames in the shot.

Let $K_i(m)$ denote the total number of key frames allocated to the i th shot after iteration m, i.e., after m key frames have been allocated to the shots. Now the request $Q_i(m)$ associated with the i th shot after the m th iteration of the allocation algorithm can be defined according to:

$$Q_i(m) = M_i(K_i(m)) \tag{6}$$

That is, the request $Q_i(m)$ after the m th key frame has been assigned is simply the content of the i th shot as regards its current key frames. The proposed algorithm assigns K_i key frames to shot i as below.

Step 0. Initialize the key frame allocation to one, so that $K_i(0) = 1$ for each i th shot and $m = 0$. Set $Q_i(0) = M_i(K_i(0))$ as the initial values of request.
(The reason for $K_i(0) = 1$ is that at least one key frame must be allocated to each shot.)

Step 1. Find the shot index j with the maximum request.

Step 2. Set $K_j(m+1) = K_j(m) + 1$, and set $K_i(m+1) = K_i(m)$ for each $i \neq j$, then set $Q_i(m+1) = M_i(K_i(m+1))$.

Step 3. If $m < K_T - T - 1$, increment m by 1 and go to step 1. Otherwise stop.

T is the number of shots in the entire sequence. This algorithm carries out a very simple and

intuitive idea. That is, simply give away key frames to the most needy shot, one key frame at a time until you run out of key frames. The degree of neediness of each shot is measured based on the content it will yield if it were to operate with its current key frame assignment.

Table 1. Results of key frame allocation ($K_T = 10, g = 1, T = 7$)

Shot index *i*	**1**	**2**	**3**	**4**	**5**	**6**	**7**
K_i	1	1	1	1	2	2	2
L	50	171	41	93	276	252	209
$M_i(K_i(0))$	10314	62290	11404	42502	93864	124628	98970

By spreading the given maximal number of key frames K_T along the entire video sequence, each shot of the sequence gets assigned a fraction of the given K_T key frames according to its share of the content relative to the total content of the sequence. **Table 1** illustrates the result of key frame allocation for the $AF(k)$ curve in **Fig. 4**.

3.3 Key Frame Distribution over a Shot

Here, $l_u (u = 1, \cdots, K_i)$ are the temporal locations of the key frames, while n_{u-1} and n_u are the breakpoints between the shot segments represented by key frame l_u. Notice that n_0 and n_{K_i} are the known temporal beginning and end points of the i th shot. The basic idea can be seen in **Fig. 5** with K_i assigned key frame.

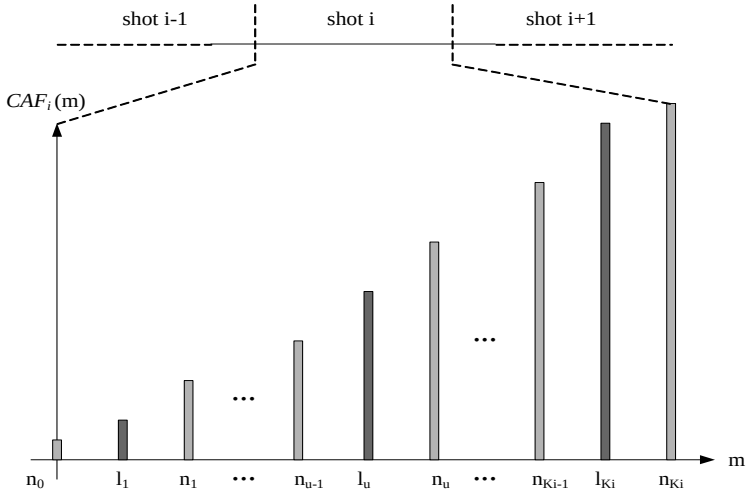


Fig. 5. Key frame distribution within i th shot using assigned K_i key frames

To find the positionb of $l_u (u = 1, \cdots, K_i)$, we propose a fast and effective which uses a probabilistic approach to locate the optimal position of the key frames. First, the normalized $CAF_i(m)$ $(= NCAF_i(m))$ is calculated for the i th shot, which is assumed to be composed of $n_{K_i} - n_0 + 1$ frames between frame n_0 and n_{K_i}. $NCAF_i(m)$ is computed as follows:

$$NCAF_i(m) = Int\left[n_0 + (n_{K_i} - n_0) \cdot \frac{CAF_i(m) - CAF_i(n_0)}{CAF_i(n_{K_i}) - CAF_i(n_0)} + 0.5 \right], m = n_0, \cdots, n_{K_i} \quad (7)$$

, where $Int[x]$ represents the integer part of x. Using Eq. (5), the discrete $CAF_i(m)$ values that are not interpolated are normalized into integer values lying between the interval $[n_0, n_{K_i}]$. Next, the histogram $H(m)$ of $NCAF_i(m)$ is calculated, then the pmf (probability mass function) $P(m)$ and cdf (cumulative density function) $F(m)$ can be obtained from $H(m)$ using the following relations:

$$P(m) = \frac{H(m)}{n_{K_i} - n_0 + 1},$$

$$F(m) = \sum_{\alpha=n_0}^{m} P(\alpha), \; m = n_0, \cdots, n_{K_i} \quad (8)$$

The pmf $P(m)$ is referred to as the probability of change in the shot content. Consequently, only $H(m)$, $P(m)$, and $F(m)$ need to be calculated before distributing the key frames. The remaining key frame distribution procedure is performed by first computing the value q_u such that $F(q_u) = u / K_i$ then finding $n_u = x$ such that $NCAF(x) = q_u$ for $u = 1, \cdots, K_i$. From the above computed n_u, the key frame positions can be easily decided sequentially as follows:

$$l_u = \frac{n_u + n_{u-1}}{2}, \; u = 1, \cdots, K_i \quad (9)$$

,where n_0 and n_{K_i} are the known temporal beginning and end points of the i th shot.

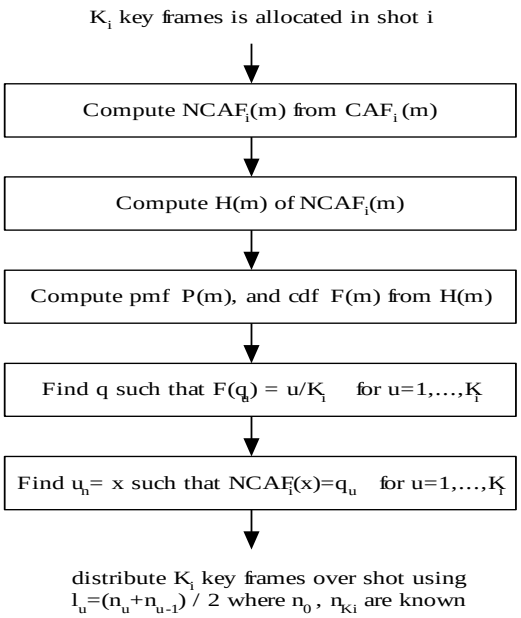


Fig. 6. Flow chart of proposed key frame distribution algorithm

This procedure of distributing K_i key frames over the i th shot is very simple and fast. In addition, the proposed method does not require any recursive computations and is performed sequentially. It is intended that the given key frames are distributed over the shot according to the probability of a change in the shot content. **Fig. 6** illustrates a summary of the steps involved in the proposed algorithm.

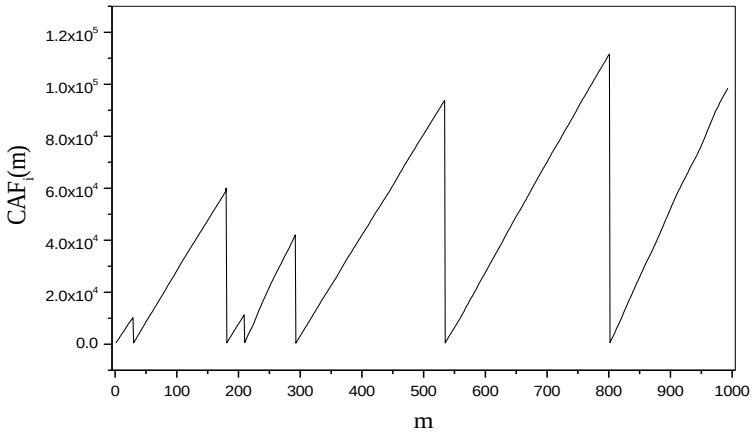


Fig. 7. Plot of $CAF_i(m)$ versus m for test sequence

Fig. 7 illustrates the plot of $CAF_i(m)$ versus m for the same sequence as used in **Fig. 4**. This plot shows the level of content variation for each shot. Therefore, based on the slope and relative magnitude, the importance of each shot can be estimated. The steeper parts correspond to more substantial changes, whereas and flatter parts indicate a more stationary shot variation. The proposed algorithm is then used to locate the key frames. The result of the key frame distribution when using the proposed algorithm is shown in **Table 2.** The key frames are arranged in a temporal order and extracted in a content-based manner, instead of just simple sub-sampling. For a shot with little or no variation, one key frame (e.g. the first frame) is sufficient. Yet for a long shot or shot with a lot of variations, multiple key frames are chosen. **Table 2** shows the effective condensing of 1000 frames of a TV sports news video clip into 10 key frames. For shots 1-4, only one key frame represents the content of each shot, however, for shots 5-7, two key frames are selected.

Table 2. Results of key frame distribution for 7 shots

Shot index *i*	**1**	**2**	**3**	**4**	**5**	**6**	**7**
K_i	1	1	1	1	2	2	2
$l_u(u=1,\ldots,K_i)$	12	126	183	212	327 557	666 789	815 951

4. Experimental Results

The proposed key frame extraction method was validated by experiment using several long video sequences, as listed in **Table 3**. The test data were digitized at a 704× 576(4CIF) spatial

resolution from consumer-grade video recordings of TV broadcasts and then compressed in MPEG-4 format at 30 frame/s. The sequences were also available as DC sequences, obtained from MPEG streams with (slightly modified) frame sizes of 88×72. We used HEVC contents as well, which have a resolution of 1280×720 compressed at 30 frame/s as MP@L9.3.

Table 3. Video sequences used in experiments

Video sequences	No. of frames	Bit rate	min:sec
TV news ("news.mp4")	10,121	1.300 Mbps	6:33
music video ("music.mp4")	12,541	1.394 Mbps	7:58
sports ("sport.mp4")	20,026	1.300 Mbps	11:17
Animation ("happyfeet.mp4")	27,630	1,054 Kbps	15:22
Documentary ("shark.mp4)	36,560	1,354 Kbps	20:22

The reduced DC sequences were first extracted using the algorithm described in section 3.1. Next, the shot boundaries were detected using the method from section 3.1. Generally, recall and precision are used as performance criteria for shot boundary detection methods [12]. The results of the video segmentation are presented in **Table 4**.

Table 4. Results of video segmentation

Video sequences	No. of frames	No. of shots T	Recall / Precision	$K_T (= T \times 1.5)$
TV news	10,121	60	0.9153 / 0.8443	90
Music video	12,541	68	0.8992 / 0.8367	102
Sports	20,026	77	0.9029 / 0.8662	116
Animation	27,630	89	0.9123 / 0.8235	134
Documentary	36,560	102	0.9234 / 0.8974	153

Key frame extraction was then performed using the individual shots obtained after video segmentation. In the experiments, the only parameter set was the maximal number of key frames. **Table 4** depicts the key frame extraction results obtained for the test sequences. The maximal number of key frames K_T was set at 1.5 times the number of shots T for each sequence. However, K_T can be adjusted by the user according to a pictorial summary and storage capacity. Unlike scene change detection, it is hard to define an objective performance analysis method for the assessment of a key frame extraction algorithm. To objectively assess the performance of our proposed method and to compare it with existing method, we define a criterion function P_{S_i} as Eq. (10) in our own way and refer to it as key frame dissimilarity.

$$P_{S_i} = \frac{1}{K_i - 1} \sum_{j=1}^{K_i - 1} \left\{ \sum_{m=1}^{M} \sum_{n=1}^{N} \left| f_{key}^{l_{j+1}}(m,n) - f_{key}^{l_j}(m,n) \right| \right\} \tag{10}$$

where S_i means the i th shot and $f_{key}^{l_j}(k,m)$ is the pixel value at position (m,n) of $M \times N$ key frame at temporal location l_j. If $K_i = 1$ in Eq. (10), we take P_{S_i} as 0. Then, we define $P_T = \sum_i P_{S_i}$ as the overall performance measure of key frame extraction method for entire sequence. The smaller value P_T has, the more similar selected key frames are. Performance test has been performed in the following way. The proposed method is compared with Bede Liu's method, which is typical scheme for extracting key frame. The reason why we do not compare our algorithm with other conventional methods such as motion analysis based or shot boundary based methods is that they are inefficient for video indexing and not easily applicable to compressed domain scheme due to many computations and unstableness. P_T is a better indicator of performance as similarity or dissimilarity between selected key frames. This means that if the selected key frames are not similar, these frames are a good representative set of frames to represent a video shot. The comparison of performance is performed in terms of overall dissimilarity according to percentage of selected frames. The results are shown in **Table 5**.

Table 5. Comparison of performance

% selected frames	Proposed method		Bede Liu's method	
	Time	***Dissimilarity,*** P_T	***Time***	***Dissimilarity*** , P_T
1 % (100 K-frames)	2.7 s	846.78	9.1 s	689.91
2 % (200 K-frames)	5.8 s	1368.15	21.1 s	1213.25
3 % (300 K-frames)	8.1 s	1287.38	31.3 s	1053.96
4 % (400 K-frames)	12.0 s	1112.03	44.3 s	921.12
5 % (500 K-frames)	13.1 s	1253.69	55.3 s	870.14

Fig. 8 shows the plot of P_{S_i} vs. shot index i according to K_T. As expected, proposed scheme has better performance than Bede Liu's method in all cases. When 5% of sequence is selected as key frames, the proposed method shows 1253.69 of P_T, while the Bede Liu's method shows 870.14 of P_T. It is shown that the key frames selected by proposed method are more dissimilar on another than those chosen by Bede Liu's method. It is observed in **Fig. 8** that for some shots, for example, shots 10-15, the dissimilarity in these shots has relatively higher value than in other shot. Key frame dissimilarity P_{S_i} can be computed by Eq. (10) when two more key frames exist. Because our method selects only one key frame for shots 10-15, we make zero of the resulting dissimilarity for these shot. Although Bede Liu's method seems to have high dissimilarity value for specific shot, the average dissimilarity value of proposed method is higher than that of Bede Liu's method as shown in **Table 5**.

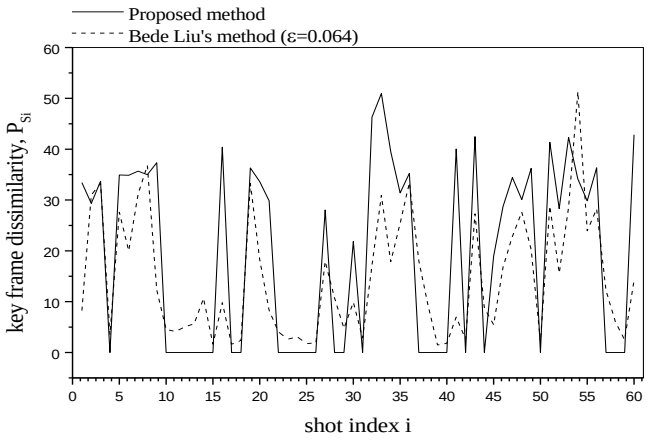


Fig. 8. Comparison of dissimilarity P_{S_i} between key frames for K_T (5%)

Shown in **Fig. 9** are extracted key frames by using proposed and Bede Liu's method. As shown in **Fig. 9**, proposed method is superior to Bede Liu's method in respect to dissimilarity measure and a subjective point of view for shot 9 for 5% selected frames.

Frame 3269 Frame 3367 Frame 3465

(a)

Frame 3222 Frame 3297 Frame 3410

(b)

Fig. 9. Extracted key frames for shot 9 (a) Proposed method ($P_{S_9} = 68.35$) (b) Bede Liu's method ($P_{S_9} = 60.21$)

5. Design and Implementation of Key Frame Extraction

In this section, we describe the architectural design of the proposed key frame extraction method and its implementation on the Android system using the framework described above. In Android, the functions packaged in the form of library (DLL, SO) or executable file, such as Assembly, C and C++ can be called on Java layer through JNI (Java Native Interface). JNI comes from the following reasons: First, the application has to use the system-related

functions, while Java does not support or is hard to implement. Second, there are many useful libraries written in other languages. Java programs can reuse them. Third, for higher performance issues, the developer has to use assembly or C/C++ code to implement some specific program modules [13][14]. For these requirements, Android platform supports the JNI method. In this paper, we use JNI because of the second reason, for reusing already implemented C codes for key frame extraction. JNI layer is exchanging the key frame data between Application UI and MPEG decoder library. Meanwhile, it provides the interface for controlling the DC images decoding.

Normally, native C code executes faster than Java code [15]. In view of the efficiency requirements of key frame extraction application, and the characteristics of Android hierarchy, the DC image extraction engine is located between Linux kernel layer and applications layer and realized by C/C++ programming language. In the DC image extraction engine, the function of Linux kernel and libraries are called to decode DC image from video stream and calculate the differences between DC images. Functions in Android application layer call the service provided by DC image extraction engine using JNI interface. The architecture of key frame extraction is designed into four layers as shown in **Fig. 10**. In Android, applications are developed with Java programming language based on Android SDK, but key frame extraction engine is based on C programming language. In this paper, we develop dynamic linked library based C programming language (.so) by JNI, and then pack the ".so" file and the Java application as a ".apk" file by Android NDK. The advantage of this approach is we can upgrade and reuse each layer because only changing the common library allows us to develop new applications. Combining hierarchical and modular design, the key frame extraction engine consists of mainly four layers, including the user interface, scene change detection, key frame allocation, and key frame distribution. This type of design approach can simplify video information processing. It is useful to develop and maintain video processing application using the key frame extraction engine. It is also easy to add a new functionality to our proposed design scheme. There is a mapping table between native functions and Android Java functions, which is registered to Dalvik VM.

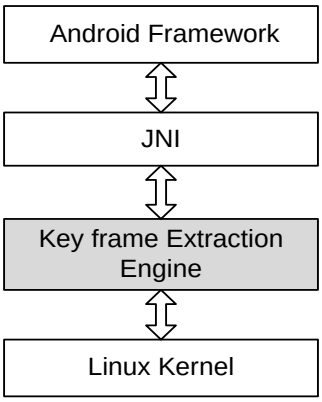


Fig. 10. Structure of Android key frame extraction application

The user interface layer is the interface of key frame extraction engine, and it is a JNI interface package of key frame extraction engine. Java applications can call the corresponding key frame extraction engine functions through JNI interface. The layer controls the flow of command from one layer to another. Related APIs implement the control of key frame number, widow size setting for scene change detection, and view options for display on screen. The

main flow of the functional call is shown in **Fig. 11**. Every step's function is as follows:

- *mm_decode_DC( )* : decode DC image from compressed video
- *mm_video_seg_from_DC( )* : segment video into shots using extracted DC images
- *mm_kframe_num_alloc( )* : allocate the number of key frames to a shot
- *mm_kframe_distributor( )* : distribute key frames over a shot

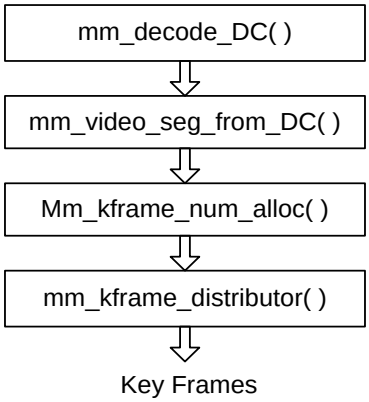


Fig. 11. The flow chart of key frame extraction function

We need to use a fragment structure for application UI which is provided by Android 4.2 APIs as shown in **Fig. 12** because we can easily compose different functionality for each layout in activity. An activity in Android OS represents a single screen with a user interface. In a multiple activities application, generally, an activity is defined as the "main" activity, which is presented to the user when user first executes the application program. A main activity of key frame list view is an object of activity type and it provides interface to users and communicates with the common library. View pager contains content providers provided by system to get key frame information from the common library. These components need to cooperate with each other in order to extract and show key frames on Android platform.

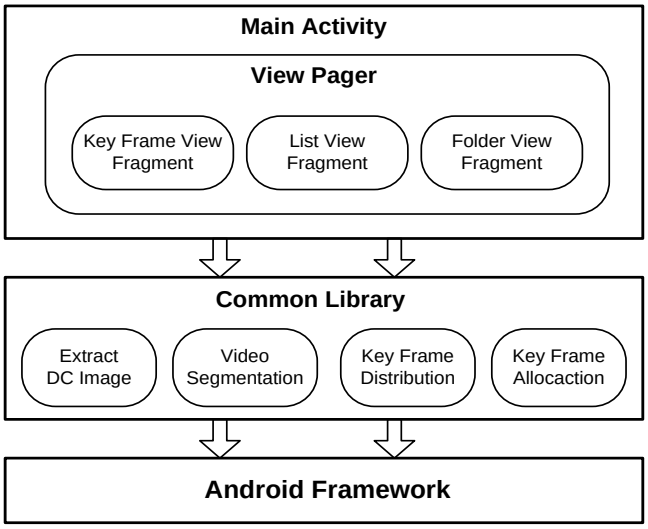


Fig. 12. Fragment structure for key frame extraction application UI

To extract key frames, the video common library should collect key frames then display them after decoding DC image, scene change detection, key frame allocation and distribution. According to the four steps, this paper designs the key frame extraction application based on this hierarchy. In the Java layer of key frame extraction application, the relationship of function classes and context view structure is shown in **Fig. 13**.

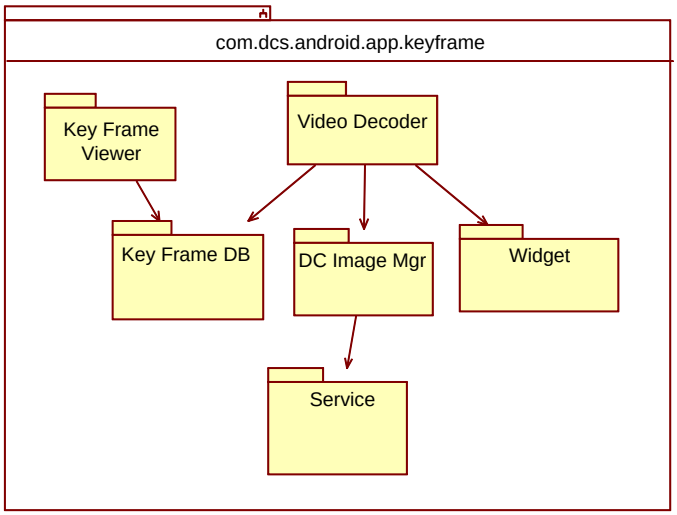


Fig. 13. Relationship of the classes and context view structure

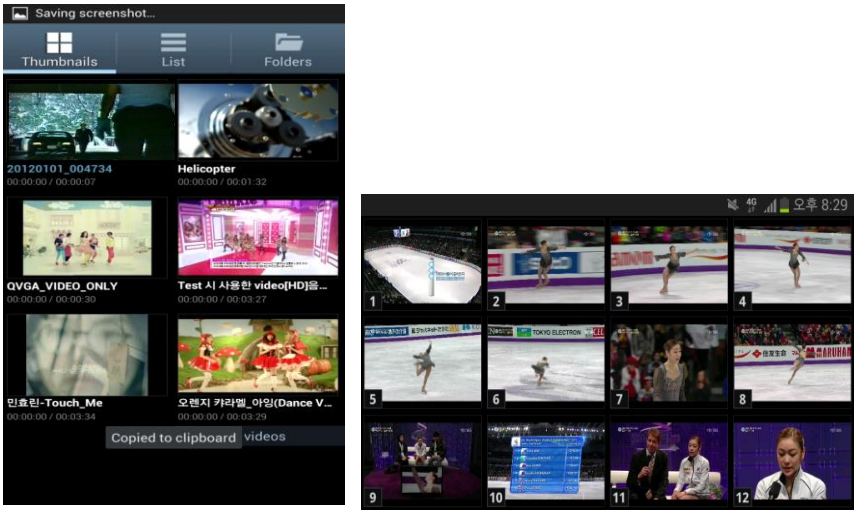
Since there are various formats of video files, our system need to parse most of universal video formats. The formats we support are as list in Table 6. So, we modify the original media player in Android to support various kinds of video files listed in table 1 and then use Android API to run media scanner service, which reads metadata from the file and adds the file to the media content provider.

Table 6. Supported video file formats

Supported video files			
video/mp4	**video/3gp**	**video/3gpp**	**video/3gpp2**
video/x-ms-asf	**video/x-ms-wmv**	**video/x-ms-wma**	**video/divx**
video/avi	**video/flv**	**video/mkv**	

After we implemented the application, compatibility testing was conducted on the application to evaluate the application's compatibility with the contents, device environment and Android OS version. Therefore, we tested various kinds of video sources, which have different video formats, audio formats and resolution. There are hundreds of devices with Android system. It is not easy to test application compatibility for all of the devices. So we choose several phones of major Android mobile phone manufacturers such as Samsung, HTC, and Google, which have different systems and hardware. We verified our application for various video formats and resolution to see if there are performance or compatibility issues using several Android phones. Test results showed that our proposed application is fully compatible for android 3.0, android 4.0 and android 4.3 and later version.

We installed the application on each of these phones. User interface is shown in **Fig. 14**. It shows a screen shot of the user interface of the application and its menu tabs. It has a simple and clear user interface with three tap buttons in the view layout. When we touch the key frame, video is played from the position of corresponding key frame. Due to the use of the standard Android development kits, the application can be easily built on all of these mobile phones without modifications to the engine source code. So far, section 5 has illustrated all the design, implementation, and testing. The main advantage of our proposed algorithm is that we can support various video file formats and time-exhaustive computations are not needed in distributing the key frames over the shot. In a smartphone environment, the speed performance of key frame extraction is an indication of the feasibility of the application. Application test results on target devices confirm the validity, availability and usefulness of the proposed method.



(a) (b)

Fig. 14. Screen shot of key frame display on Android phone (a) Portrait mode (b) Landscape mode

6. Conclusion

This paper proposes a new key frame extraction method for content-based video indexing and retrieval. The proposed method consists of three steps: video segmentation, key frame allocation, key frame distribution. The main advantage of the proposed method is that no time-exhaustive computations are needed for distributing the key frames over the shot, plus the procedure of key frame extraction is fully automatic. In addition, the set of key frames is independent of any subjective thresholds or manually given parameters. The proposed algorithm can operate directly on a wide range of video file formats. We implement our proposed framework on Android smartphone using JNI interface to confirm the feasibility and availability. Experimental results confirmed the validity and usefulness of the proposed method. Furthermore, the proposed key frame extraction framework can provide a sufficient platform for many multimedia applications, including the efficient management of large video database, access to video archives, and the automatic creation of video clip previews.

References

[1] Zeeshan Rasheed and Mubarak Shah, "Detection and representation of scenes in videos," *IEEE Trans. on Multimedia*, vol. 7, no. 6, pp. 1097-1105, December, 2005. Article (CrossRef Link)

[2] Lijie Liu, "Combined key-frame extraction and object-based video segmentation," *IEEE Trans. on Circuit and Systems for Video Technology*, vol. 15, no. 7, July, 2005. Article (CrossRef Link)

[3] Jian-quan Ouyang, "Interactive key frame selection model," *Journal of Visual Commun. and Image Representation*, vol. 17, Issue 6, pp. 1145-1163, December, 2006. Article (CrossRef Link)

[4] Lang Congyan, "Automatic key-frames extraction to represent a video," in *Proc. of IEEE ICSP'04*, vol. 1, pp. 741-744, December, 2004. Article (CrossRef Link)

[5] Guozhu Liu and Junming Zhao, "Key frame extraction from MPEG video stream," in *Proc. of IEEE ISIP*, pp. 423-427, 2010. Article (CrossRef Link)

[6] Kin-Wai Sze, "A new key frame representation for video segment retrieval," *IEEE Trans. on Circuit and Systems for Video Technology*, vol. 15, Issue 9, pp. 1148-1155, 2005. Article (CrossRef Link)

[7] Sang-Hyun Kim and Rae-Hong Park, "A novel approach to video sequence matching using color and edge features with the modified hausdorff distance," in *Proc. of ISCAS*, pp. II-57-II-60, 2004. Article (CrossRef Link)

[8] Janko Calic and Ebroul Izquierdo, "Efficient key-frame extraction and video analysis," in *Proc. of ITCC*, pp. 28-33, 2002. Article (CrossRef Link)

[9] B. L. Yeo and Bede Liu, "Rapid scene analysis on compressed video," *IEEE Trans. on Circuit and Systems for Video Technology*, vol. 5, no. 6, pp. 533-544, December, 1995. Article (CrossRef Link)

[10] Fangxia Shi and Xiaojun Guo, "Keyframe extraction based on K-means results to adjacent DC images similarity," in *Proc. of ICSPS*, pp. V1-611-V1-613, 2010. Article (CrossRef Link)

[11] B. L. Yeo and Bede. Liu, "Fast extraction of spatially reduced image sequence from MPEG-2 compressed Video," *IEEE Trans. on Circuit and Systems for Video Technology*, vol. 9, no. 7, pp. 1100-1114 1999. Article (CrossRef Link)

[12] Y. Nakajima, K. Ujihara, and A. Yoneyama, "Universal scene change detection on MPEG-coded data domain," in *Proc. of IS&T/SPIE Visual Commun. and Image Processing*, vol. 3024, pp. 992-1003, February, 1997. Article (CrossRef Link)

[13] Cheng-Min Lin, Jyh-Horng Lin, Chyi-Ren Dow, Chang-Ming Wen, "Benchmark Dalvik and Native Code for Android System," in *Proc. of the 2nd Int. Conf. on Innovations in Bio-inspired Computing and Applications*, pp. 320-323, December, 2011. Article (CrossRef Link)

[14] Damianos Gavalas and Daphne Economou, "Development platforms for mobile applications: status and trends," *IEEE Software*, vol. 28, no. 1, pp. 77-86, Feburary, 2011. Article (CrossRef Link)

[15] SJ Cho, KJ Kim, EH Hwang, SH Yoon and JW Jeon, "Benchmarking Java application using JNI and native C application on Android," in *Proc. of ICC*, pp. 284-288, 2012. Article (CrossRef Link)

[16] K.W. Kim, J.S. Lee, and S.G. Kwon, "Key frame assignment for compressed video based on DC image activity," *Journal of Korea Multimedia Society*, vol. 14, no. 9, pp. 1109-1116, September, 2011. Article (CrossRef Link)

[17] Chinh T. Dang and H. Radha, "Heterogeneity image patch index and its application to consumer video summarization," *IEEE Trans. on Image Processing*, vol. 23, no. 6, pp. 2704-2718, 2014. Article (CrossRef Link)

[18] Haojin Yang and Christoph Meinel, "Content based lecture video retrieval using speech and video text information," *IEEE Trans. on Learning Technologies*, vol. 7, no. 2, pp. 142-154, 2014. Article (CrossRef Link)

Kang-Wook Kim received the B.S., M.S., and Ph. D. degrees in Electronics Engineering from Kyungpook National University, Korea in 1996, 1998 and 2002 respectively. He is currently a principal engineer in R&D Group, Mobile Communication Division, Samsung Electronics Co., Ltd. His research interests include visual communication, mobile embeded system, and Android application.