Telecommunication Systems 28:2, 185–210, 2005

Software-Based Video/Audio Processing for Cellular Phones

JIN-HWAN JEONG and CHUCK YOO {jhjeong;hxy}@os.korea.ac.kr
Korea University, Anam 5 Ga, SeongBuk-Gu, Seoul, Republic of Korea

Abstract. Nowadays, most cellular phones are used beyond voice communication. Although the processing power of cellular phones is sufficient for most data applications, it is difficult to play video and audio contents in software because of their computational complexity and lack of basic tools for multimedia processing, so software-based multimedia processing on cellular phones is a challenging issue. Several transcoding methods are introduced to address this issue, but they are mainly of the DCT-domain conversion. Hence, they are only applicable to high-end cellular phones. To develop a solution for low-end and mid-tier cellular phones, we begin this paper by analyzing the complexity of existing video standards to see if it is possible to play them on cellular phones by software. Next, various coding profiles as combinations of subalgorithms are studied, and we select a profile that adapts its complexity to the processing power of cellular phones. Also, an efficient dithering algorithm called out-of-order dithering is developed. We implement the profile with out-of-order dithering in an actual cellular phone software environment and present the performance results. The performance results show that software based video/audio processing is indeed possible on low-end cellular phones.

Keywords: video/audio processing, cellular phone, adaptation

1. Introduction

Most current and future cellular networks like GSM-GPRS, UMTS, CDMA-2000, or IMT-2000 are serving various packet-oriented transmission modes. In terms of the bandwidth, it is broad enough for cellular phones to be ready for new downloading services or streaming various multimedia formats such as H.263 [28], MPEG-x [4; Legall, 12; Talluri, 26], etc. Also, the third generation partnership project (3GPP) has selected several multimedia codecs for inclusion into its multimedia specifications. As a result, there is a great level of interest to use a wireless cellular phone for multimedia applications like MMS (Multimedia Messaging Service) [24], PSS (Packet-switched Streaming Service) and PCS (Packet-switched Conversational Service).

Although the wireless network speed is sufficient for multimedia data transmission, it is not simple to provide multimedia services because of two major constraints: (1) network constraint: packet losses and bit errors peculiar to wireless networks and (2) terminal constraint: insufficient hardware resources. The former is concerned with bit error detection, error collection, and error concealment as well as jitter delay management. As network issues are actively researched elsewhere in this issue, we will not discuss them anymore. The latter is concerned with minimizing terminal complexity that

is a key to success in wireless multimedia. This paper is focused on the algorithmic complexity of processing multimedia data and decoding capabilities of wireless terminals, especially cellular phones and whether they have enough power to process multimedia data without additional hardware supports.

Usually, as the processing pattern of multimedia applications is CPU-intensive or memory-intensive, efficient implementation has been an interesting issue for the last decade. As a result, many coding standards are well analyzed in terms of complexity and performance, and then various optimization techniques are introduced. However, most of these results are targeted and tested on general-purpose processors. On the contrary, these results ironically show that the current coding standards, especially video coding standards cannot be decodable on low-end cellular phones in software. The reason is not only because of its heavy computational requirements, but also because of a lack of the "idle" processing power ("idle" means the remaining CPU power excluding CPU time for tasks for phone management) of cellular phones. Some service providers transcode existing contents to alleviate computational requirements as well as to meet channel constraints or size constraints. However, most transcoding techniques [Assuncao and Ghanbari, 3; Dugad and Ahuja, 5; Han et al., 9; Merhav and Bhaskaran, 15; Morrison et al., 16; Sostawa et al., 25; Youn and Sun, 31; Zhu et al., 33] are focused on reducing bit-rate or spatial down-sampling rather than on lightening complexity. Therefore, another transcoding focusing on lightening computational complexity is needed.

The fundamental reason why common video standards cannot be decodable in software on cellular phones is that the goal of the video coding algorithms is to maximize the coding efficiency. This approach makes sense in the highly congested Internet where high performance PCs are connected, but under wireless network environments mainly composed of cellular phones, this approach overloads a low-powered processor in cellular phones. In principle, the more data is compressed, the more it requires computation for decoding at the same quality [Jeong and Yoo, 10]. Needless to say, the compression itself is an important factor because it reduces the size of data transmission and the space of contents; but the balance between computation and compression is indispensable in some cases. In a sense, this point coincides with the purpose of a transcoder that balances between two aspects at the wireless proxy server. Recent video standards are trying to cope with the balancing issue with profile levels, but it is not sufficient (we did some experiments, and will show the results later in this paper).

To address above problems, we introduce two adaptation mechanisms of multimedia coding algorithms to the capabilities of target cellular phones: processing adaptation and displaying adaptation. The former is about an adaptation of computational complexity to the processing power, and the second adaptation is about an efficient dithering algorithm that is indispensable with low resolution displayers.

This paper is organized as follows. In section 2, we mention backgrounds about cellular phone hardware configurations and MPEG-4 video standards with experiment results, related works, and explain the motivation of transcoding for cellular phones. Next, in section 3, we divide several video coding algorithms into multiple subalgorithms

to see how much decoding power is required for each subalgorithm. We roughly classify subalgorithms into four groups: transform coding, entropy coding, quantization coding, and prediction coding. In section 4, quantitative analyses of each subalgorithm in terms of two aspects (compression ratio & computation requirements) are discussed. Also, we will envision a light-weight codec (LC) as an example of video processing adaptation for low-end low-powered cellular phones, and an audio codec specialized in cellular phones, and explain what criteria were used to design our LC. Also, the displaying adaptation that uses an "out-of-order" dithering mechanism at the cost of graceful quality degradation for low resolution devices is explained. In section 5, some implementation issues related with cellular phones such as synchronization and watch dog are mentioned, and then we will show experiment results. Finally, we conclude this paper in section 6.

2. Backgrounds and related work

2.1. Hardware configuration of cellular phones

Before going through the correlation of two aspects – computation requirements and compression ratio, we describe three important resources of a cellular phone: processor, memory, and display device. While there are several other resources affecting multimedia processing, these are dominant factors.

First, we examine the core processor of the most widespread cellular phones. ARM7TDMI is one of the most representative core processors of cellular phones today. It is a 32-bit RISC processor that executes 32-bit instruction (ARM mode instruction) as well as 16-bit instruction (THUMB mode instruction) for code density. It is a high performance embedded processor, but, even in an idle state, it always runs a few tasks in the cellular phone system. The tasks are mainly for managing a cellular phone system itself such as searching for a base station, which consumes significant CPU time. Usually, these tasks occupy 70–80% of CPU time. In addition, after making a call, it burns CPU cycles to run specific protocol stacks. At last, CPU time left is merely 1-2MIPS (in the case of MSM-5000 [18]) for multimedia applications, so "lightness" is an indispensable property of multimedia applications. Table 1 shows the example specification of Qualcomm chipsets.

The ARM7TDMI [2] processor operates at 13.5 MHz clock speed (usually CDMA-2000 1x phone) in its idle state. It is quite obvious that its clock speed is too low to

Table 1
Widely used Qualcomm chipset.

Chipset	Processor	Freq.**	Cache
MSM-3100	ARM7TDMI	19/19 MHz	N/A
MSM-5000*	ARM7TDMI	13.5/27 MHz	N/A

* CDMA-2000 1x chipset.
** The first is for idle state, the second is for call state.

decode video standards such as MPEG-4 even 1 frame per second by software. Even though a faster processor that is powerful enough to decode video standards is available in the market, it is not easy to equip a cellular phone with a high speed processor because of the hardware cost and its power consumption that is a scarce resource in cellular phones. In other words, the development speed of the rechargeable battery technology is much slower than that of the processor technology so that cellular phones cannot be equipped with a higher speed processor easily.

Second, cellular phones are suffering from a memory shortage because of small RAM size and an absence of a memory management mechanism. After a cellular phone is powered-up, the free memory size for run-time execution is typically a few hundred kilobytes that are not sufficient for decoding. Moreover, the memory access speed is very slow. Furthermore, ARM7TDMI has no level 1 cache memory so that frequent memory reference operations may degrade the overall system performance. To make matters worse, frequent memory operations drain battery power quickly.

Lastly, display devices of cellular phones are characterized by low dot brightness, high dot reaction time and low resolution. Because of the battery power management, the brightness of cellular phone's LCD is dimmer than that of a normal LCD, so it is hard to notice the very slight color difference represented on a cellular phone's LCD. In addition, some STN LCD can display images up to 3 fr/s due to high dot reaction time, and some TFT-LCD can do up to 15 fr/s at most. In the case of color resolution, the color depth varies from 1 bit to 16 bits. If the LCD color depth is below 16 bits, the number of colors is not sufficient for natural representation, which requires special rendering mechanisms like dithering. Actually, the fact that cellular phones are equipped with a low resolution LCD has its advantages and disadvantages. The advantage is related with the fact that the precision of color representation on target display devices is low; hence, video pixel data does not need to be processed at full precision throughout. On the other hand, a low-resolution display below 16 bits color depth raises the dithering issue that is the main disadvantage. Dithering is very CPU-intensive because the input data of dithering algorithms is every pixel of an image. It is important to select a proper dithering algorithm and to merge it with the decoder and cellular phone system. Table 2 shows the specifications of various LCD types used for cellular phones on the market.

Table 2
Cellular phone LCD specifications and cellular phone prices.

Model*	Color depth	RGB format	Price
SCH-X250	8 bits	3:3:2	180$
SCH-X290	12 bits	4:4:4	227$
SCH-X590	16 bits	5:5:6	300$
SCH-V300**	18 bits	6:6:6	590$

* Samsung Electronics Co. Ltd.
** Hardware VOD (MPEG-4) cellular phone.

Table 3
MPEG-4 decoding time.

Clip	Compression ratio	Dimension	Decoding time
51.m4v	51 : 1	176 X 144	4039 us/fr
49.m4v	49 : 1	240 X 96	3797 us/fr
38.m4v	38 : 1	240 X 112	4853 us/fr

Pentium III 750 MHz, 1 GB RAM.

2.2. Decoding performance of MPEG-4 simple profile

To quantify the video playback on a cellular phone equipped with the above hardware configuration, we ran an optimized MPEG-4 decoder on a PC platform and measured the performance. The test system consists of Pentium III at 750 MHz CPU and 1 Gbytes memory. Three test clips are used. As shown in table 3, the pure decoding time is about 4039 us/fr on average. Even if we suppose that the processing power of ARM7TDMI is equivalent to that of Pentium III (actually, Pentium III processor operates much more efficiently at the same clock speed), we can estimate that the pure decoding time on ARM7TDMI is 898[1] ms/fr. Taking display time into account, we roughly conclude that cellular phones can play an MPEG-4 clip at 1 fr/s by software at best.

2.3. Multimedia service system overview

To overcome the above performance issue, many cellular phone manufacturers have tried hardware solutions. To play video contents on a cellular phone, they attached a specific media coprocessor to a cellular phone. For example, SCH-V300 [21] is equipped with an MPEG-4 decoder chip for video playback. It shows 10–15 fr/s capability to decode a QCIF-sized MPEG-4 clip. The hardware approach increases the frame rate easily, but it has two major drawbacks.

The first is about flexibility. A media coprocessor usually handles a specific video format, which means that an MPEG-4 decoder chip can only decode MPEG-4 clips. The second is the cost. An additional media coprocessor drives the architecture of a cellular phone system complex and raises the overall hardware cost. Table 2 shows the comparison of a hardware VOD cellular phone (SCH-V300) and non-VOD cellular phones. Although SCH-V300 has an outstanding LCD, the price point is very high.

Figure 1 illustrates the overall architecture for a typical mobile multimedia service system. The contents are authored by contents providers, and they are also transcoded by the authoring server to meet the capabilities of cellular phones. Cellular phones are equipped with a WAP browser and a suitable multimedia player. With a phone, a subscriber connects to a server that contains clips. A clip is selected from the menu, and it is streamed or downloaded from the pumping server.

[1] 898 ms = 897,556 us = 4039 us* (750 MHz/13.5 (clock speed of ARM7TDMI in a idle state) MHz)* (100%/25% (available CPU time)).

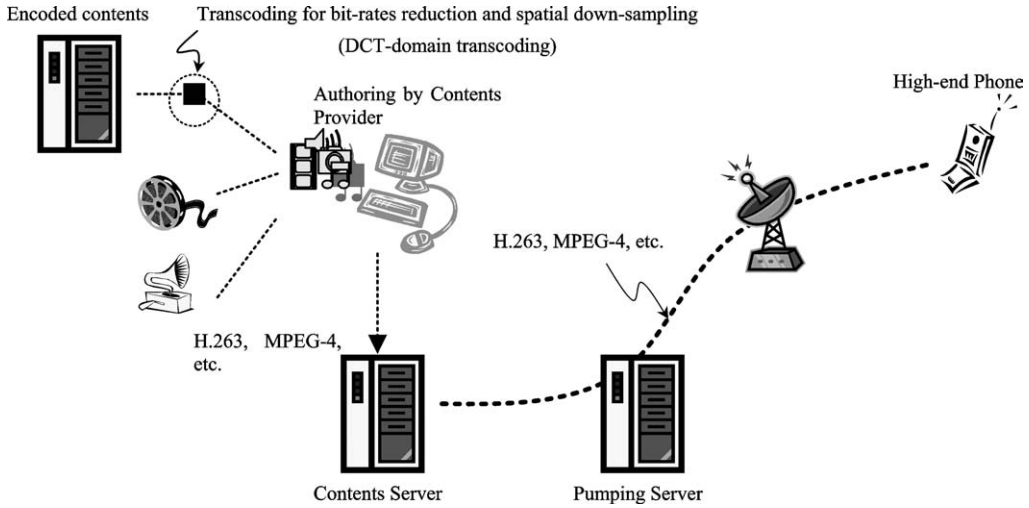


Figure 1. Overview of a multimedia service system for cellular phones.

As mentioned in the previous paragraph, high-end cellular phones with a media co-processor can download or stream standard video files such as H.263, MPEG-4, etc. (dotted line in figure 1). The transcoding in figure 1 is done in the authoring tool for bit-rate conversion [Assuncao and Ghanbari, 3; Han et al., 9; Morrison et al., 16; Sostawa et al., 25; Youn and Sun, 31], spatial down-sampling [Dugad and Ahuja, 5; Merhav and Bhaskaran, 15; Zhu et al., 33], and format conversion [Shanableh and Ghanbari, 22]. All of these are used for resizing the image to the cellular phone's LCD, for adjusting bit-rate to reduce stream size or frame rate to channel constraints, and for supporting various video formats. However, since these kinds of transcoding are still CPU-intensive, only high-end phones can decode the transcoded contents properly. We need a new transcoding method for low-end cellular phones without a media coprocessor, which is the goal of this paper.

3. Analysis of a video coding standard

The most widely used video coding standard is block-based hybrid video coding in which each video stream is divided into blocks of fixed size and each block is more or less processed independently, and is therefore named "block-based". The word "hybrid" means that each block is coded using a combination of motion compensated temporal prediction and transform coding. In this paper, we chose MPEG-4 as a representative block-based hybrid coding for the performance analysis because it is widely used.

We measure the decoding times of MPEG-4 in table 4 where chip names mean their respective compression ratio. VLD, IDCT, Pred, and Dequant mean variable length decoding, inverse DCT transform, syntax parsing and inter/intra frame prediction, and dequantization with scaling respectively. The intra and inter blocks are measured sep-

Table 4
Decoding time of subalgorithms.

Clip	Block	Count	VLD	IDCT	Pred	Dequant	Comp. ratio
51.m4v	Intra	4170	2.11	2.81	2.70	1.36	12.08
	Inter	134232	1.15	N/A	3.78	0.16	52.21
49.m4v	Intra	14538	1.84	2.44	2.76	1.19	21.23
	Inter	497382	1.20	N/A	4.03	0.16	49.82
38.m4v	Intra	15756	2.25	2.59	2.71	1.38	11.65
	Inter	623064	1.39	N/A	4.56	0.14	38.67

Test platform: Pentium III 750 MHz, 1 GB RAM, Unit: us/block.

Table 5
Decoding time of each function.

Rank	Function name	Decoding time	Description – subalgorithm
1	DCTInv_8 × 8_16s	13.43%	Inverse DCT (one block) – IDCT, Pred
2	MC_blk_XYHalf	11.13%	Motion compensation in half XY pixel mode – Pred
3	MC_blk_Int	8.52%	Motion compensation (interpolation) – Pred
4	VlcDecTCOEF	4.91%	Variable length decoding of coefficients – VLD
5	MC_blk_XHalf	3.60%	Motion compensation in half X pixel mode – Pred
⋮	⋮	⋮	The rest is committed

Test platform: Pentium III 750 MHz, 1 GB RAM.

arately. Also, we measure the CPU time at the level of functions and show in table 5 by the decoding time. A function of table 5 is an implementation of a subalgorithm of table 4 or an implementation of a part of a subalgorithm.

Table 4 shows the computationally intensive subalgorithms listed in order, IDCT, Pred, VLD, and Dequant. In the case of Pred, it means spatial prediction for intra block and temporal motion prediction for inter block. Hence, Pred requires several major functions (DCTInv_8 × 8_16s, MC_blk_XYHalf, MC_blk_Int, and so on). For this reason, we classify subalgorithms by functions. "Subalgorithm" of "Description – subalgorithm" column of table 5 means subalgorithms mentioned in table 4. Prediction coding algorithm (Pred) seems to be most intensive, but Pred is the sum of IDCT time and pixel compensation time. Actually, as shown in table 5, IDCT is the most CPU-intensive.

To analyze the video codec in terms of the compression ratio and computational complexity, we decompose various video standards into four subalgorithms (transform coding, prediction coding, entropy coding, and quantization coding), and summarize their computational requirements and contribution for compression ratio respectively. As a matter of convenience, we categorize the four subalgorithms into reference frame coding (transform coding, entropy coding, and quantization coding) and prediction frame coding (pixel compensation and error term coding).

3.1. *Reference frame coding*

Transform coding. The most widely used transform coding techniques are categorized into Fourier Transform (FT) and Wavelet Transform (WT) [Graps, 8]. In this paper, we investigate DCT (FT), SPIHT (WT), and HAAR (WT) algorithms for their computational requirements.

At first, the DCT algorithm [Arai et al., 1; Feig and Winograd, 6; Loeffler et al., 14] is the most popular transform coding that has been adopted by MPEG-x, H.26x, etc. Many algorithms have been proposed for the efficient calculation of the two-dimensional IDCT. Some algorithms transform 2D pixel values into 2D frequency coefficients directly, but some algorithms compute 1D IDCT by row and column in sequence for 2D IDCT. The notable algorithm is AAN [Arai et al., 1].

The AAN algorithm is a one-dimensional, pre-scaled DCT/IDCT algorithm and is very efficient to compute 2D IDCT. First, the eight input coefficients are prescaled by eight prescale values that require 8 multiplications. Then the algorithm is applied to the prescaled coefficients, which requires 5 multiplications and 29 additions for the transform. Although the 1D AAN algorithms is less efficient than other 1D algorithms, when applied to 2D 8×8 IDCT, this algorithm takes only 64 multiplications for the pre-scale, and 80 multiplications and 464 additions for the transform, so the overall cost for one 8×8 pixel block is 144 multiplications and 464 additions.

The HAAR transform is one of the simplest wavelet transforms. The HAAR transform is expressed recursively by averaging function and differencing functions, and it requires only addition (and shifting for efficiency). Therefore, the HAAR transform algorithm takes only about 244 additions for one 8×8 pixel block.

Said and Pearlman [20] have introduced a variant of coding of wavelet coefficients by successive approximation, that even without arithmetic coding outperforms EZW by Shapiro [23]. They call it Set Partitioning In Hierarchical Trees (SPIHT). The crucial parts of their coding process are the way the subsets of the wavelet coefficients are partitioned and the significant information is conveyed. This technique is so fast in execution that it is used in real-time playback software on the legacy system. Unlike the above two algorithms (DCT and HAAR), this algorithm performs comparison and assignment operations instead of arithmetic operations. Therefore, it is difficult to compare the SPIHT algorithm with two algorithms directly, but some experiments show that its execution time is about 3 times longer than that of the HAAR transform. Table 6 shows the summary of performance in terms of computation cost for an 8×8 pixel block.

Table 6
Comparisons of three transform coding algorithms.

Algorithm	Main operation	Execution time
DCT(AAN)	80 MUL. and 464 ADD.	$10T$
HAAR	224 ADD.	T
SPIHT	Comparisons and assignments	$3T$

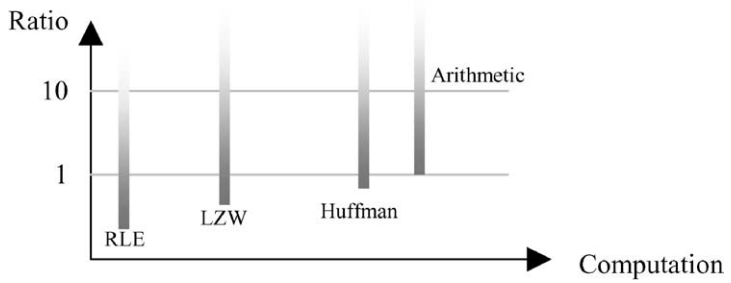


Figure 2. Entropy coding: Compression ratio & Computation.

Entropy coding. The entropy coding is the basis of a compression algorithm usually used in a stand-alone compression application [Lelewer and Hirschberg, 13]. The most famous algorithms in this category are RLE (Run Length Encoding), LZW (Lempel–Ziv–Welch), Huffman coding, and arithmetic coding. The key concept of RLE and LZW is to eliminate the duplication of the input string. On the other hand, Huffman coding and arithmetic coding depend on the occurrence probability of input string. RLE is such a simple and intuitive method, but its performance is the worst. In the worst case, it produces data 2 times its original size. In contrast, the performance of arithmetic coding is remarkable while it requires frequent bit operations and table look up.

Figure 2 displays the correlation of the compression ratio and computation amount for each algorithm. The X axis of the figure 2 is the amount of computation; the Y axis is the compression ratio for each entropy coding. RLE is a very simple encoding algorithm with a 1–2 : 1 compression ratio on average. On the other hand, the Huffman or arithmetic coding result in a higher compression ratio, but they need more CPU cycles and IO operations.

To achieve a high compression ratio, some applications adopt arithmetic coding or a mix of several entropy codings. For instance, MPEG-x uses both run-length coding and Huffman coding in turn.

Quantization coding. A quantizer maps a signal with a range of values X to a quantized signal with a reduced range of values Y. It should be possible to represent the quantized signal with fewer bits than the original since the range of possible values is smaller [Richardson, 19]. Quantization coding methods are usually categorized into scalar quantization and vector quantization. A scalar quantizer maps one sample of the input signal to one quantized output value and a vector quantizer maps a group of input samples (vector) to a group of quantized values (codeword). Scalar quantization is done by rounding a quotient divided by a quantization step; hence, it is simple but lossy. Vector quantization depends on the design of the codebook and efficient searching to find an optimal vector. Obviously, vector quantization is much more CPU-intensive than scalar quantization.

3.2. Prediction frame coding

Prediction coding. Motion compensated prediction assumes that the current picture can be locally modeled as a translation of the pictures of some previous time. In most standards, each picture is divided into blocks of $2^n \times 2^n$ pixels, called a *macroblock*. Each macroblock is predicted from the previous (called P frame) or future frame (called B frame), by estimating the amount of motion in the macroblock during the frame time interval.

Unlike transform coding or entropy coding, the performance of prediction coding is difficult to quantify for comparison. For example, backward prediction coding and bidirectional prediction coding are different prediction codings, but the latter includes the former. In other words, the computation requirements and compression ratio increase together by moving the degree of prediction coding from the simple backward prediction coding (one frame as a predictor without compensation) to the complicate bidirectional prediction coding (multiple frames as predictors with compensation).

Motion compensation is effective to produce very low bit streams at the cost of moderate CPU time and memory space. However, the main operations of motion compensation are memory operations such as block read/write, block copy, and block interpolation. These operations possess a system bus. It means that the bandwidth of the system bus and the memory access speed are critical for motion compensation performance as well as CPU power. Motion compensation is also involved in battery consumption in portable devices. Memory operation consumes more battery power than arithmetic operation, so frequent memory operations drain the battery power quickly [Joo et al., 11].

4. Video/audio codec for low-end cellular phones

In this section, we envision a light-weight video codec (LC) and an audio codec. It can be used as a transcoder for standard video/audio formats at the server and as a decoder for low-end cellular phones. Also, we introduce an efficient dithering algorithm for low-resolution display devices of cellular phones. We first explain how an LC can be designed, and then we explain our new dithering algorithm.

4.1. Design of profile X

Reference frame coding. The reference frame coding is the basic coding of motion pictures, but it consumes significant CPU cycles. Therefore, the key point of an LC, especially, as a decoder for low-powered cellular phones, is that it should consume as little as possible CPU time. Undoubtedly, the compression ratio is still an important factor.

Subalgorithms in the previous section are transform coding, quantization coding, and entropy coding. Figure 3 shows the possible combinations of the subalgorithms for reference frame coding. We added a subsampling group to reference frame coding combinations because subsampling is very common and it is treated as a starting point of coding. We will explain prediction frame coding later.

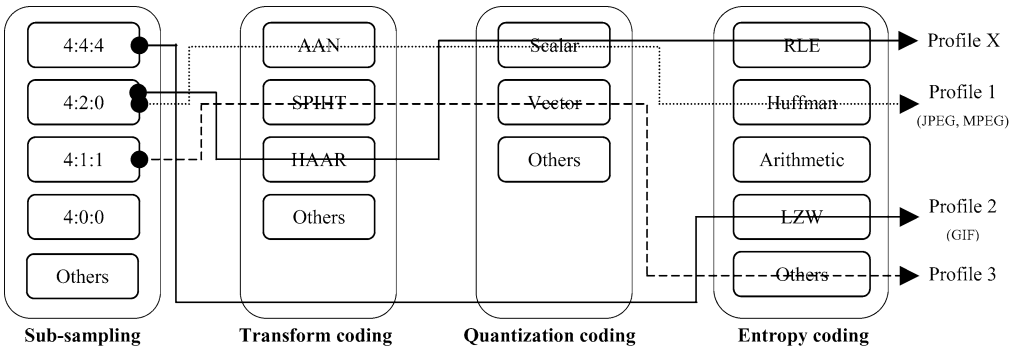


Figure 3. Various compression algorithms.

A combination of subsampling, transform coding, quantization coding, and entropy coding leads to a profile, which has different characteristics in terms of computation requirements and compression ratio. For example, profile 1 (JPEG [Wallace, 29] or MPEG-x) consists of $4:2:0$ subsampling, AAN transform, scalar quantization, and Huffman coding, so it focuses on maximizing compression ratio; profile 2 (GIF) focuses on minimizing computation time. From figure 3, there can be $5 \cdot 4 \cdot 3 \cdot 5$ profile combinations, and the computational requirements and compression ratio are summarized as follows:

- *Profile 1: Heavy computation, high compression.*
- *Profile 2: Light computation, low compression.*
- *Profile 3: Medium computation, medium compression.*
- ...
- *Profile X: Light computation, medium compression.*

Profile 1 represents most of the current video standards and has a high compression ratio, so it can be applicable to high-end cellular phones. On the other hand, profile 2 represents simple image formats such as GIF that are inadequate for video playback due to a low compression ratio. A question is how to build a profile (let us call it profile X) from those combinations for an LC. The following explains how each subalgorithm is chosen. The suitability of the chosen profile for an LC is verified in section 5.

As mentioned in the previous sections 2 and 3, most cellular phones cannot run a software-based MPEG-4 decoder. Considering table 5, it is clear that the main overhead comes from IDCT. 80 multiplications and 464 additions generate approximately one and half thousand cycles on ARM processor, which is equivalent to about 0.4[2] ms per block. Hence, IDCT is too heavy for cellular phone processors. On the contrary, HAAR is famous for light computation [Mulcahy, 17; Wen et al., 30]. Table 6 (Computation requirements of transform coding) shows that HAAR transform coding takes $1/10$

[2] 0.4 ms = 0.1 ms (the elapsed time for one and a half thousand cycles on the 13.5 MHz ARM7TDMI)* (100%/25% (available CPU time)).

times of the IDCT computation time, and therefore it is a good candidate of profile X. As transform coding is the most critical factor of all subalgorithms for the decoding performance, the "lightness" is very crucial. Especially, HAAR is a process of averaging and differencing which is also a kind of very light processing in the viewpoint of processor logic, so it has been widely used as a software real-time codec for off-the-shelf processors since the mid 1990s. Detailed performance effects of transform functions in actual video decoding will be shown in section 5.

Next, the profile X codec adopts a scalar quantization coding as a quantization coding. Since profile X uses a block-based coding and a transform coding algorithm, scalar quantization needs to boost the entropy coding performance by controlling the step size. In addition, scalar quantization coding has a merit in implementation and computational performance as compared with vector quantization coding because it can be implemented by table look-up. Its compression ratio depends on a quantization table as well, which is a useful tool for managing bit-rates.

In the case of entropy coding, the computational pattern as well as algorithmic complexity is important. Although Huffman decoding can be done by table look-up, it is possible for a processor to be overburdened due to numerous bit operations. As shown in table 1, ARM7TDMI has no CPU cache. Therefore, looking up a codeword table with an input symbol generates many memory references, which degrades the system performance and drains the battery quickly. For this reason, we adopt a modified RLE algorithm as entropy coding of profile X. As it is well known, RLE has a shortcoming regarding compression ratio; especially in the case that the run length of its elements is frequently 1.

To overcome this drawback, we added a normalization step into profile X. The normalization step is inserted after the quantization coding phase and before the entropy phase to boost the RLE coding power. Coefficients of AC terms computed by transform coding are normalized in the manner that the MSB (Most Significant Bit) of AC coefficients can be used as a tag bit of succession. That is, if the MSB is set, the coded value is a pair of run length and AC coefficient. Otherwise, it is just an AC coefficient. In the normalization step, coefficients can be distorted. However, as we normalize AC coefficients that are odd numbers and are greater than 32 while it is hard to find an AC coefficient that is greater than 32 actually, quality degradation by normalization is less than 1–2 dB. Though there is a normalized AC coefficient, quality degradation is negligible to the human eye, because an image is displayed on the cellular phone's LCD. Under the circumstance that the color depth is less than or equal to 16 bits and the dot brightness of a cellular phone's LCD is low, it is hard to notice a very slight color difference.

Summarily, profile X that consists of $4:2:0$ subsampling, HAAR transform, scalar quantization, and modified RLE seems to be a good candidate of the reference frame coding for low-end cellular phones because its combination consists of subalgorithms of the low computation. We will verify the suitability of the implementation of profile X in actual low-end cellular phones.

Prediction frame coding. Now, we specify profile X for the prediction frame coding. As pointed out in the previous section, the performance of prediction frame coding is related with the memory system (e.g., memory bus bandwidth, memory access speed) as well as the processing power, which means memory-intensive. In the case of low-end cellular phones, a memory bus line is 8 bits and the speed of memory access is relatively slow (in real cellular phones, the memory fetch speed is much slower than the CPU ALU instruction speed). Therefore, a sophisticated prediction coding algorithm takes longer time to execute.

Furthermore, in a real situation, since it is impossible to play video clips at 24 fr/s or higher on a cellular phone due to lack of CPU power, a time interval of interframe is relatively large. For this reason, the compression contributed by prediction frame coding is not significant. That is to say, an elaborate prediction algorithm fails to achieve very low bit rates because many blocks are "mismatch". Even though a block is "match", it is possible to produce a significant size of error term data (predicted spatial data) for compensation that makes the compression ratio low and causes excessive memory operations on the decoding side. In this case, two heavy subalgorithms, inverse transform for error term data and pixel compensation with error term data, are required (table 5). Under the cellular phone environment, it is possible for a sophisticated prediction algorithm to overburden the processor while resulting in low compression.

Specifically, the effectiveness of most subalgorithms of prediction frame coding is related with block "match" performance. Let the cost of calculating the transform coding for the I-block be $TCost(x)$, the cost for the transform decoding for error term be $TECost(x)$, the cost for the motion vector decoding be $MCost(x)$, and the cost for the pixel compensation be $ICost(x)$. Similarly, let the effect on compression ratio for the transform coding for the I-block be $TEffect(x)$, the effect for the transform coding for error term be $TEEffect(x)$, the effect for the motion vector decoding be $MEffect(x)$, where the probability of a block "match" is $p(x)$. The decoding cost for one block is expressed as $DCost(x)$[3] $= TCost(x)\cdot(1-p(x))+(TECost(x)+ICost(x)+MCost(x))\cdot p(x)$, and the effect is expressed as $DEffect(x)$[4] $= TEffect(x)\cdot(1-p(x))+(TEEffect(x)+MEffect(x))\times p(x)$. If $p(x)$ is close to 1 and the size of the error term is small, $TECost(x)$ and $ICost(x)$ ($TECost(x)$ and $ICost(x)$ are 0 for perfect "match") are small and $TEEffect(x)$ is big; otherwise, $TECost(x)$ and $ICost(x)$ are big and $TEEffect(x)$ is small. In the cellular phone system, $TECost(x)$ and $ICost(x)$ are very high because of relatively small $p(x)$ and slow memory access speed. Actually, through the observation of some tests, we found that when the frame rate is below 15, $p(x)$ is so small that the prediction frame coding with the error term coding degrades the overall system performance and drains battery power very quickly. Therefore, the profile X video codec adopts the simple backward prediction coding per block with vector coding and without error term coding, that is, $TECost(x)$ and $ICost(x)$ are 0 by sacrificing $TEEffect(x)$.

[3] $TCost(x) > TECost(x) > ICost(x) > MCost(x)$ in common cases, and $TCost(x)$ and $MCost(x)$ are independent on $p(x)$ and the size of error term.

[4] $TEffect(x) < TEEfect(x) < MEffect(x)$ in common cases.

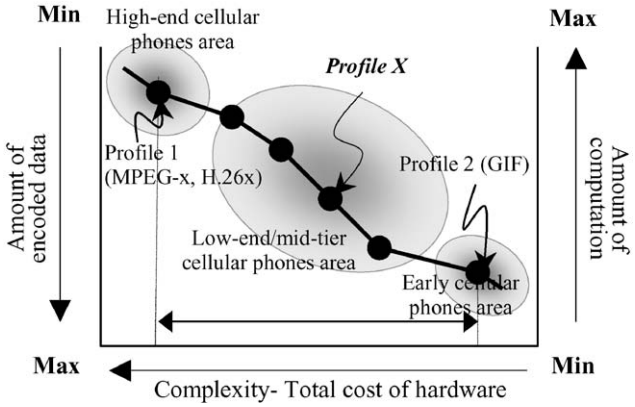


Figure 4. Spectrum of correlation between computation and compression.

Profile X. Figure 4 illustrates the spectrum of correlation between the computation requirements and the compression ratio, and shows the position where profile X is located. As well known, most of video standards such as MPEG-4 are positioned in the left-top corner targeting for high performance cellular phones with a media coprocessor. On the contrary, GIF that is of still image format is used for early cellular phones that are too weak to process moving pictures. In summary, profile X, one of several combinations of subcoding algorithms, adapts the compression ratio to the processing power for mid-tier/low-end cellular phones.

4.2. Dithering

A dithering algorithm [Ulichney, 27] is usually applied when the color depth of the target display device is below 12 bits. Most cellular phones today have a low resolution display whose color depth varies from 4-level gray to 16 bits color. It is obvious that low-end LCD (below 12 bits) devices require dithering, but, the 16 bits LCD on a cellular phone also needs dithering because of mobile LCD peculiarity. Therefore, dithering is indispensable. Figure 5 illustrates the common rendering process. Since dithering is required for the human eye (it may be a kind of noise in the viewpoint of a computer), it is applied at the last phase of rendering, that is, just before displaying the image on LCD. Most dithering algorithms are categorized into two groups. One group is type A that dithers a pixel with its own value and its coordinates (some algorithms use the average value of an image). Type A algorithms are simple and easy to use, but these produce moderate image quality. ORDERED dithering is a representative algorithm of type A. On the other hand, type B is known as diffusing algorithm. It means that the target pixel value is diffused into its neighboring pixels and dithered. Compared with type A, type B requires more complex arithmetic computation including floating point arithmetic and memory references to diffuse the pixel value. However, it produces graceful images with a small number of colors. Figure 5 also illustrates the difference of two dithering groups.

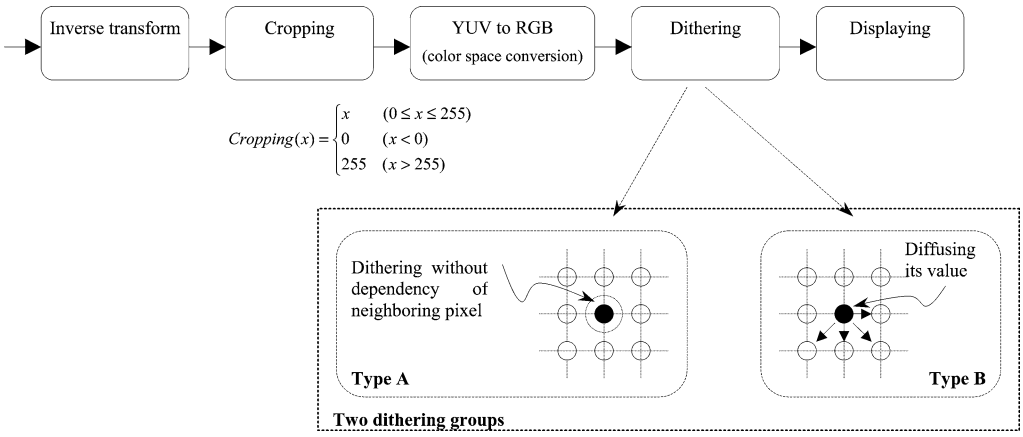
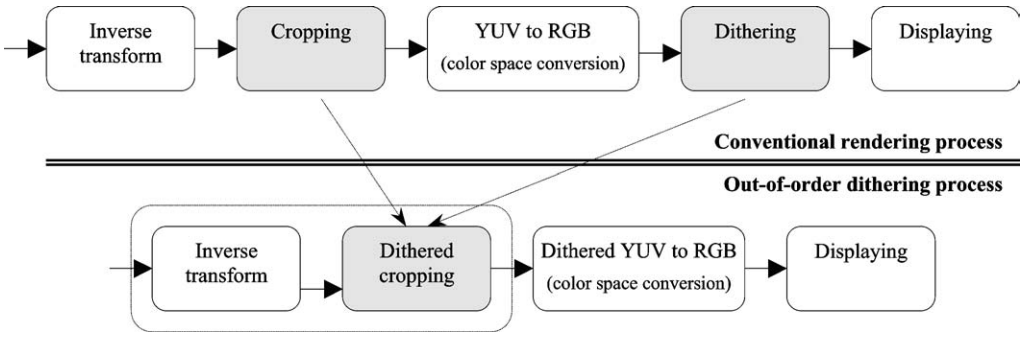


Figure 5. Common rendering process and two dithering groups.

Because dithering algorithms are applied for all pixels of each frame, any dithering algorithm overburdens a mobile processor. Through experiments, we found that the frame rate with ORDERED dithering is half of that without dithering. To address this performance issue, we characterized each phase in figure 5 with the following observations:

1. *Pixel value (YUV) is calculated at the inverse transform phase.*
2. *Pixel coordinates are known at the inverse transform phase.*
3. *Cropping is done after inverse transform phase for clamping YUV value.*
4. *Cropping can be done by comparison and assignment or by a table look-up operation.*
5. *Type A dithering algorithms require pixel value and its coordinates to dither.*
6. *Type A dithering can be done by comparison and assignment or by a table look-up operation.*

In the conventional rendering, since the color space conversion phase is located between the cropping phase and the dithering phase, a cropping table and a dithering table cannot be merged. Hence, two kinds of table look-up operations are required. If we relocate the dithering phase between the cropping phase and the color space conversion phase, then we can merge the cropping table and the dithering table (we call it out-of-order dithering). In other words, the original decoding phases are expressed as a display pixel $P_{x,y} = Dither(RGB(Cropping(T^{-1}(Q^{-1}(C_n)))))$, where RGB is a function converting YUV to *RGB*, T is a transform coding, Q is a quantization function, and C_n is coded bits. Out-of-order dithering is to reorder the rendering functions like this: $P'_{x,y} = RGB(Dither(Cropping(T^{-1}(Q^{-1}(C_n)))))$. Figure 6 illustrates the comparison of the original rendering process and the out-of-order dithering process.



$$Cropping(x) = \begin{cases} x & (0 \leqslant x \leqslant 255), \\ 0 & (x < 0), \\ 255 & (x > 255), \end{cases} \qquad Dithered\_cropping(x) = \begin{cases} Dither(x) & (0 \leqslant x \leqslant 255), \\ Dither(0) & (x < 0), \\ Dither(255) & (x > 255). \end{cases}$$

Figure 6. Out-of-order dithering.

Out-of-order dithering has good and bad points. A good point is that the dithering function, $Dither(x)$ and the cropping function, $Cropping(x)$ are combined into one table, $Dither\text{-}Cropping(x)$. By doing so, the dithering computation time can be almost absorbed into one table look-up operation time; hence, the frame rate with dithering is almost equal to that without dithering. Also, the data size of dithering is usually 12 bits or smaller, so color space conversion, $RGB(x)$ can be done by a table look-up operation, which is a further computational merit. In a 4069-color LCD, the above 3 tables can be merged into one table of reasonable size. In addition, brightness adjustment or gamma correction can be done by a color table, so additional computation is not required for color adjustments.

However, a bad point is that the final pixel values of out-of-order dithering, $P'_{x,y}$ is not equal to those of the traditional dithering, $P_{x,y}$. The difference between $P'_{x,y}$ and $P_{x,y}$ is due to the color space conversion (floating point rounding error) and *YUV* dithering (in color space conversion, dithering Y affects R, G, and B at the same time). We measured that the difference varies from -20 to 20. In the worst case (the difference is -20 or 20), the quality degradation can be noticeable on 24 bits, but it is not distinguishable to the human eye on low resolution LCD because of following reasons. Suppose that an image is rasterized on a 12 bits LCD, 8 bits dithered pixel component value is shifted right by 4 bits, so its difference gets smaller. In addition, the reconstructed R, G, and B values from the dithered Y, U, and V can be calculated in advance and stored in a color table in order to reduce the difference. Therefore, the image quality of out-of-order dithering can be much the same with the quality of the original dithering on low resolution LCD.

Figure 7 is an illustration of reconstructed images, and shows the quality of out-of-order dithering originated from the ORDERED dithering. The left images are of 24 bits, and the right images are of out-of-order dithered 12 bits. In the "Description" column of figure 7, "Color count" means the number of colors to present images for each. For example, "2412, 59" of the first row in figure 7 is the number of colors for a 24 bits

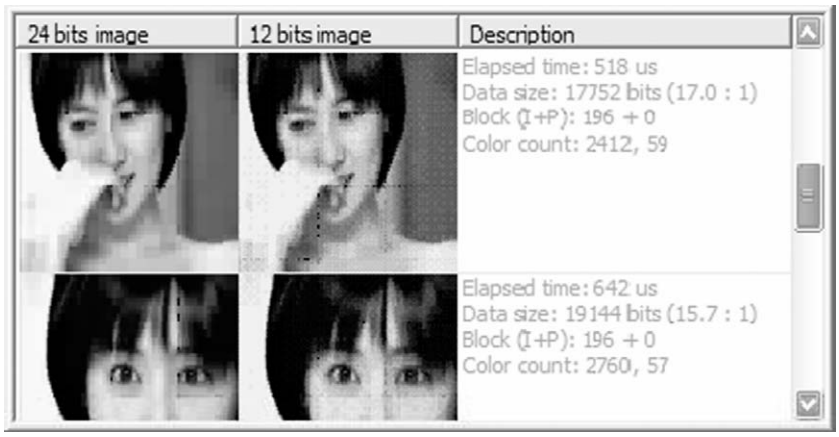


Figure 7. Out-of-order dithering images.

image and for a 12 bits dithered image, respectively. As shown in figure 7, there is no noticeable artifact on out-of-order dithered images.

4.3. Audio codec

Before designing the audio codec, we surveyed audio codecs suitable for cellular phones. Some focus on audio quality and others are superior in compression. Although audio codecs generally consume less processor cycles for decoding than video codecs, they still require moderate processor time. All cellular phones have a hardware audio codec (vocoder) such as EVRC (Enhanced Variable Rate Codec) for a CDMA phone. Because an audio codec was implemented as a hardware chip, it does not require additional core processor power. Moreover, audio codecs for a cellular phone has strength in compression and bit error. For these reasons, we decided to use a hardware vocoder as our audio codec.

In cellular phone environment, the vocoder generates an interrupt periodically (50 ms in case of the CDMA phones) to sound the voice. When an interrupt is generated, the audio interrupt handler is called. At this time, the interrupt handler checks the header of the encoded voice data and pushes to the vocoder's buffer queue, and then the vocoder sounds the voice. Therefore, to decode and play audio data, a player just tosses the encoded audio data to the interrupt handler. That is to say, we use a new interrupt handler. A new interrupt handler reads the encoded data, manipulates the header, and calls the original interrupt handler. Figure 8 illustrates an audio decoding process.

This mechanism makes the player architecture simple, and it is easy to synchronize with video data in a non-preemptive system without time drifting. Synchronization will be explained in the following section. GSM phones also support similar audio mechanisms.

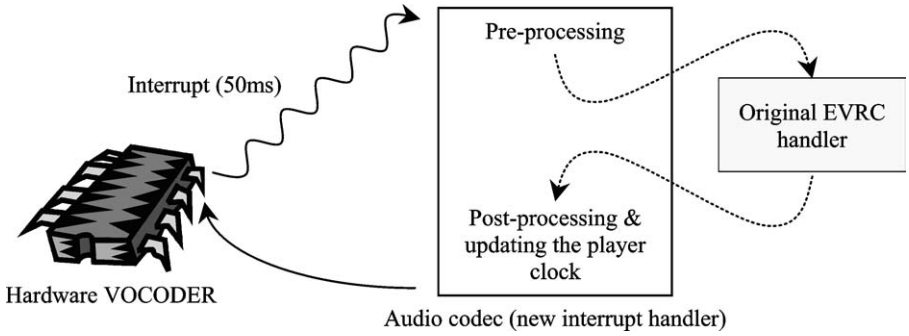


Figure 8. Audio codec processing.

5. Implementation and experiments

This section explains how profile X and the out-of-order dithering are implemented in an actual cellular phone software environment. Our implementation is done on a Samsung CDMA phone, and the performance is measured with our implementation.

5.1. Phone software environments

Operating systems for a cellular phone are embedded, non-preemptive, and single threaded, and they adopt a priority-based task scheduling policy. For simplicity and software lightness, they do not have sophisticated subsystems such as a virtual memory system, and use very simplified subsystems. REX that operates on CDMA phones by Qualcomm is an example of OS for cellular phone.

Because of the non-preemptive and single-threaded nature, OS for cellular phone can not take a processor resource by force from a task that is currently running, so the task occupying a processor should yield to a next task within a given time quantum voluntarily; otherwise, the WATCHDOG exception that guards and resets the system triggers when one task holds a processor beyond the allowed time quantum. This is because a cellular phone always needs to execute some tasks periodically in order to communicate with a base station.

As the main job of the OS for cellular phones is interrupt handling, there needs a special task called the UI task for managing the phone system and its states. The UI task runs regularly and checks the system states, for example, which key is pressed, which task should be launched, etc. That is, the UI task acts as a system shell of a regular OS, and it does specific processing that the state says. When an event happens, the event handler simply change the state, at first, by setting or clearing the UI task's flags, and then the UI task starts an appropriate task by interpreting the state when it is scheduled. Therefore, most events are processed asynchronously, and the UI task can access most phone system resources for task management. The UI task has most system libraries such as file system, network system, LCD module, audio module, etc. Figure 9 illustrates the overall phone software architecture.

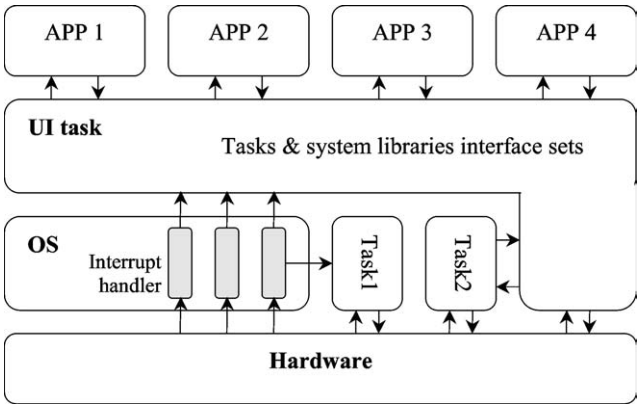


Figure 9. Cellular phone software architecture.

$$\sum_{k=i}^{n} DT(PKT_k) + \mathrm{Max}DT < WT,$$

- DT: Decoding time for a packet,
- PKT_k: kth packet,
- MaxDT: Predefined maximum decoding time for a packet,
- WT: WATCHDOG time.

Figure 10. Decoding time for a given time quantum.

5.2. *Synchronization*

Due to the absence of a multi-threading mechanism in the phone OS, the synchronization between the video and audio is tricky. Our solution in the implementation for synchronization is to allow partial decoding of a frame. When the player starts decoding, the video decoder decodes a group of small sized-encoded data (called packet – PKT) for a given time quantum (WT) and yields the processor. If the video decoder tries to decode a whole frame, it is possible that the WATCHDOG exception occurs. To avoid the WATCHDOG exception, our player keeps track of the decoding time and yields the processor within a time quantum. Figure 10 is a formula to avoid the WATCHDOG exception.

The meaning of figure 10 is as follows: The video decoder decodes packets from the ith packet to the nth packet. If the video decoder cannot finish decoding a whole frame within a time quantum or the sum of decoding time from the ith packet to the $(n + 1)$th packet is greater than a time quantum (actually, as the video decoder does not decode the $(n + 1)$th packet yet, its decoding time is replaced with MaxDT), then the video decoder stops decoding and yields the processor. At the next turn, the video decoder resumes decoding the $(n+1)$th packet. After decoding, the video decoder inserts a decoded frame into a frame list.

The OS for a cellular phone does not offer a fine-grained timer, so if the player clock used for synchronization is updated by the system timer, the player clock is not reliable. In order to update the player clock accurately, we use the audio interrupt that occurs at a fixed time interval as mentioned in section 4.3. When an audio interrupt is

```
Video decoder {    // Timer Handler
  if (Read_Clock_AND_Test) {
    Display_Frame;
    Yield_Processor;
  }
  Time_Quantum = MaxDT;
  while (Time_Quantum < WatchDog) {
    Decode_One_Packet;
    if (Build_Frame) {
      Insert_Frame;
      Yield_Processor;
    }
    Update_Timer_Quantum;
  }
}
```

```
Audio decoder {    // EVRC Interrupt handler
  Update_Clock(TIME_INTERVAL);
  Read_Data;
  Attach_EVRC_Header;
  Call_Original_EVRC_Handler
}
```

Figure 11. Pseudo code for audio and video decoders.

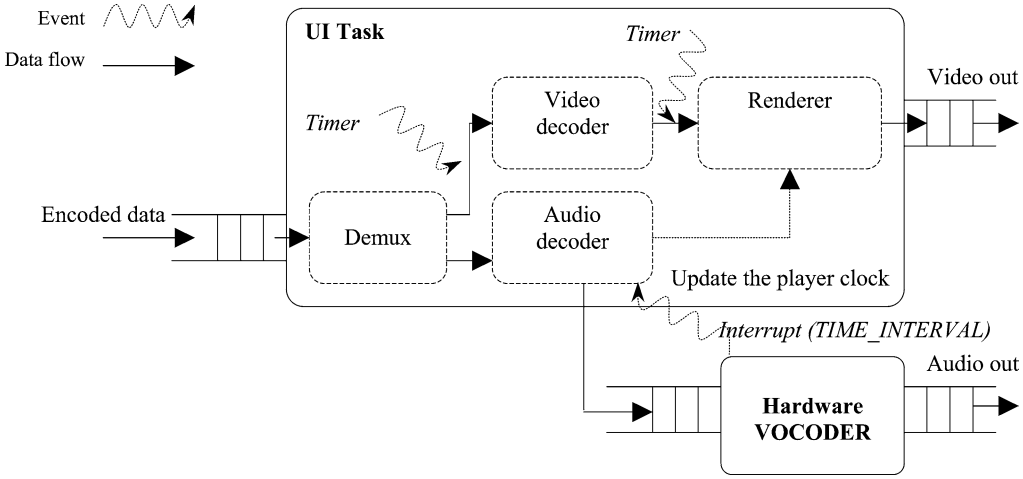


Figure 12. Overall architecture of profile X player.

triggered, the audio interrupt handler updates the player clock (add the time interval). Figure 11 illustrates the pseudo code of the decoding and interrupt handler.

For synchronization, the video decoder decodes packets and checks the present time stamp of decoded frames with the player clock for displaying periodically, and the audio decoder manipulates the audio data as well as updating the player clock. By doing so, video and audio is well synchronized without time drifting.

We implement profile X player as a library and attach it to the UI task. Figure 12 shows the overall player architecture. The UI task runs periodically by a timer. When the UI task runs, the video decoder decodes packets that are read from the Demux, and the Renderer (actually, the video decoder) checks the present time stamp of the topmost

image from a decoded image list with the player clock, and then displays the image. In the case of audio, upon each interrupt, OS calls the audio interrupt handler (audio decoder) that is a function of UI task, and the audio decoder passes the audio packets to vocoder with updating the player clock.

5.3. Profile X performance

The performance of profile X is compared with the H.263 baseline codec and the MPEG-4 simple profile codec. The H.263 decoder is based on the reference code and runs on a PC, while the MPEG-4 decoder is highly optimized and runs on an ARM processor. The H.263 decoder is compared with the profile X decoder on a PC system, and the MPEG-4 decoder is compared with the profile X decoder on an ARM system. The image quality is measured along with computation requirements and compression ratio. It is not easy to measure the CPU performance of a real cellular phone because of the absence of reliable performance counter tools, so we use the ARM profiler to collect meaningful performance data that are the number of instructions and the number of cycles for a section of codes or the whole program. With these performance counters, we can estimate the actual performance from the CPU clock speed (e.g., 20 MHz ARM9 CPU executes 20,000 cycles/ms). Most core processors of embedded devices including cellular phones are ARM CPU or its variants. Therefore, the number of instructions and cycles (CPU instruction, not high level language instruction) is meaningful to other thin devices.

We carry 4 experiments. Three experiments are about the video decoders and the last one is about the rendering method. The size of test clips is QCIF, and the quality comparisons are done in 24 bits color in case of video decoder experiments. Table 7 shows 4 experiment parameters.

Table 8 and figure 13 show the results of experiment 1. Table 8 shows the "lightness" of profile X clearly while the PSNR of "ma.lc" is inferior to that of "ma.263" by 4 dB with a similar compression ratio. profile X decodes the same bit rate clip at the cost of 1/414 (115 us/47724 us) times while sacrificing 4 dB PSNR. The performance difference between the two decoders mainly comes from inverse transform (DCT vs.

Table 7
Experiment parameters.

No.	Targets	Clip name	Compression ratio (PSNR)	Test platform
Experiment 1	H.263	ma.263	139 : 1 (variable)	P-3 750 MHz
	Profile X	ma.lc	141 : 1 (variable)	
Experiment 2	MPEG-4	ma.mp4	1130 : 1 (35 dB)	ARM processor
	Profile X	ma.lc	141 : 1 (35 dB)	
Experiment 3	Profile X	ma.lc	141 : 1 (35 dB)	ARM processor
	Profile X	cf.lc	20 : 1 (35 dB)	
Experiment 4	Normal rendering	N/A	N/A	ARM processor
	Out-of-order	N/A	N/A	

Table 8
Performance result of "Miss America".

Codec	*CPU (us/fr)	PSNR (dB)	Bits/fr	Ratio
Profile X	115	32.74	4304	141 : 1
H.263	**47724	36.57	4369	139 : 1

* Pentium III 750 MHz, 512 MB RAM.
** It depends on implementation.

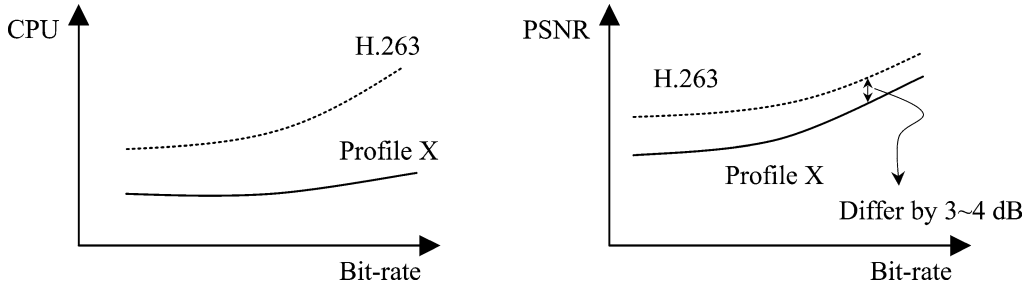


Figure 13. Comparison of CPU usage and PSNR between profile X (ma.lc) and H.263 (ma.263).

HAAR), entropy decoding (Huffman vs. modified RLE), and error term decoding. Especially, the inverse transform function makes a major difference. The IDCT function requires significant CPU time among several decoding subfunctions, while the inverse HAAR function is not heavy compared with other profile X subfunctions. Besides, the IDCT function is called frequently for I typed block and error term decoding. In the case of quality, 4 dB of PSNR is a big number, but if reconstructed images are displayed on a low resolution device like a cellular phone LCD, the quality degradation is blurred, and the quality difference is quite small to the human eye.

In figure 13, the performance gap is getting larger (profile X is better) as the bit rate increases. In the case of H.263, if the bit rate increases, the number of I typed blocks is bigger (many IDCT) and the length of run is shorter (many table look-up for Huffman decoding) due to fine quantization step sizes. Figure 13 also shows that the gap of image quality is getting smaller as the bit rate is increasing.

Figure 14 illustrates the results of MPEG-4 and profile X performance on an ARM processor, experiment 2. In this experiment, we qualify the adaptation of the computation at the same image quality. As shown in figure 14, "ma.mp4" requires about 354,590 instructions (587,140 cycles) per frame for decoding, but "ma.lc" requires about 7,800 instructions (13,600 cycles) per frame. Though the tested MPEG-4 decoder is highly optimized, it consumes the CPU time approximately 45 times more than profile X, which means that the MPEG-4 decoder is difficult to run on cellular phones but that the profile X is well adapted for cellular phones. In other words, profile X greatly reduces the computational requirements by adapting the compression ratio gracefully. In the case of the MPEG-4, the pure decoding frame rate for CDMA-2000 cellular phones (13.5 MHz

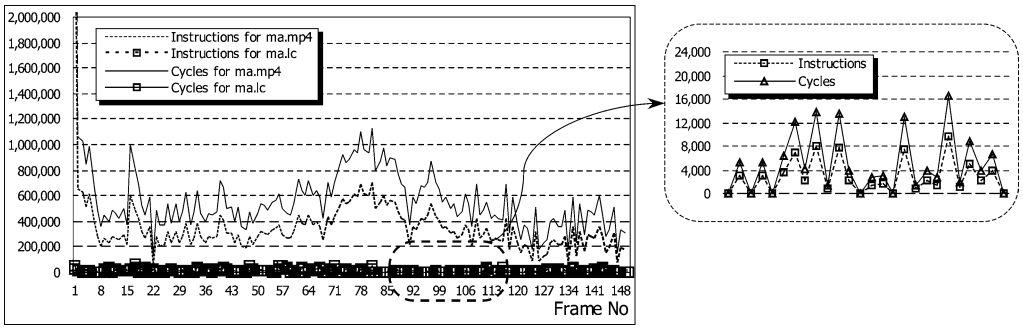


Figure 14. Performance comparison: profile X (ma.lc) and MPEG-4 (ma.mp4).

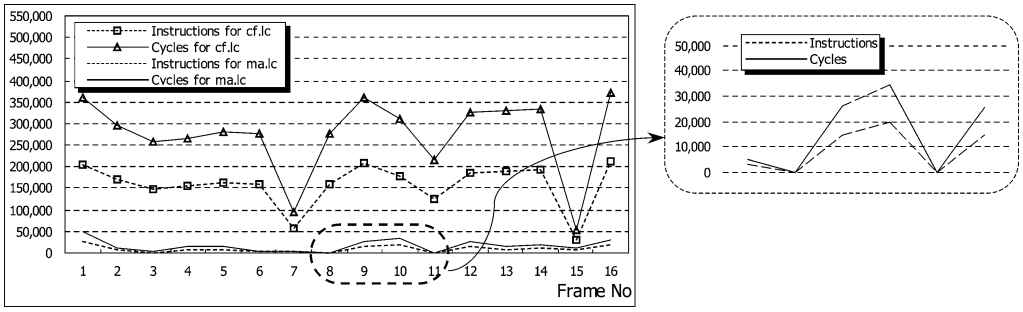


Figure 15. Performance comparison: “ma.lc” and “cf.lc”.

ARM7TDMI) is at most 5.7[5] fr/s, and the frame rate for playback is much lower due to the rendering computation time.

Figure 15 illustrates the number of thumb mode instructions and cycles per frame, experiment 3. As the compression ratios of two clips imply, “cf.lc” has more reference blocks than “ma.lc” because “cf.lc” has less predicted frames. As a result, “ma.lc” requires about 7,800 instructions (13,600 cycles) per frame, while “cf.lc” requires about 159,000 instructions (276,000 cycles) per frame. Like other codecs, profile X also shows that the higher the clip is compressed, the smaller the number of instructions is needed at the same quality. The main reasons are (1) the energy compaction property of transform coding, and (2) the effects of prediction frame coding.

To see the performance enhancement of out-of-order dithering, we measure the numbers of THUMB mode instructions and cycles for processing one pixel. These two rendering methods yield 12 bits-dithered pixels. In the case of the conventional method, the color space conversion function is a multiplierless version [Gordon et al., 7], and the dithering algorithm is the ORDERED dithering implemented by table look-up operations. As shown in table 9, the conventional rendering method requires significant amount of computation[6] in spite of highly efficient implementation. However, the out-

[5] $5.7 = (13{,}500{,}000/587{,}140) \cdot 0.25$ (available CPU power).

[6] 49 instructions/pixel = 827,904 $(49 \cdot 176 \cdot 144 \cdot 2/3)$ instructions/frame (4 : 2 : 0 subsampling: 2/3).

Table 9
Costs of two rendering methods for processing one pixel.

Rendering method	Instructions	Cycles
Conventional rendering	49	86
Out-of-order dithered rendering	14	37

Unit: THUMB mode instructions and cycles.

Figure 16. Profile X player (12 bits LCD) for SCH-X430.

of-order dithering method requires only 14 instructions that are about 29% of the conventional way. In case of the quality, a metric such as the PSNR for quality comparison is not accurate because of dithering. The quality of dithered images is determined by human-visual-system as well as by complicate metrics [Yu and Parker, 32]. For example, suppose a dithering matrix (A) and another matrix (B) for ORDERED dithering, and B is rotated some degrees from A. After being dithered by two matrixes, the two images are different in terms of a certain metric, but both are well dithered and come to the same thing to the human eye. Therefore, determining the quality improvement of the two rendering methods by a numerical metric comparison is not accurate. Instead, subjective tests are preferable. Actual images are shown in figures 7 and 16.

Finally, considering the number of instructions for decoding (including rendering) and the available CPU power mentioned in section 2, we can estimate the performance of profile X for a real cellular phone and calculate the frame rate approximately. Actually, we implemented profile X with EVRC codec on a CDMA-2000 cellular phone – SCH-X430. The Qualcomm chipset of SCH-X430 is MSM-5000 whose core processor is ARM7TDMI operating at 13.5 MHz clock speed in the idle state. We saw that a "normal" ("normal" means that it can do a call service) SCH-X430 plays a SQCIF-sized clip (about $20:1$ compression ratio – "cf.lc") at 6–8 fr/s with 12 bits color dithering. Figure 16 is snapshots of our player on SCH-X430.

6. Conclusion

It is generally believed that video processing on cellular phones is only possible with high-powered smart phones. It is because existing video coding standards emphasize the compression ratio and quality so that huge computation is required. Therefore, a low-powered processor cannot decode them by software. This paper attempts to show that software-based video processing is indeed possible even with low-powered cellular phones. It is done by designing a Light-weight Codec (LC) and an efficient rendering method. Specifically, an LC is designed by choosing a profile that balances computational requirements and compression ratio. The new rendering method is called out-of order dithering, and it alleviates significantly the conventional computation overhead for displaying. The LC and out-of-order dithering are implemented in an actual cellular phone environment, and their performance is measured. We measured that LC with out-of-order dithering needs about 20,000 instructions to play a $140:1$ compressed clip (SQCIF-sized "Miss America" clip). The experiment results imply that our approach can be applied to ubiquitous mobile devices to come in the future that are equipped with low-powered processors.

References

[1] Y. Arai, T. Agui and M. Nakajima, A fast DCT-SQ scheme for images, Transactions of IEICE 71 (1988) 1095–1097.
[2] ARM, `http://www.arm.com`.
[3] P.A.A. Assuncao and M. Ghanbari, Transcoding of MPEG-2 video in the frequency domain, in: *Proc. of IEEE Internat. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, Vol. 4 (April 1997) pp. 2633–2636.
[4] Coding of audio-visual objects, part 2: visual, ISO/IEC, 14496-2 (December 1999).
[5] R. Dugad and N. Ahuja, A fast scheme for downsampling and upsampling in the DCT domain, in: *Proc. of IEEE Internat. Conf. on Image Processing (ICIP)*, Vol. 2 (October 1999) pp. 909–913.
[6] E. Feig and S. Winograd, Fast algorithms for the discrete cosine transform, IEEE Transactions on Signal Processing 40 (1992) 2174–2193.
[7] B. Gordon, N. Ghaddha and T.H. Meng, Low-power multiplierless YUV to RGB converter bases on human vision perception, invited chapter, in: *Low Power/Low Voltage Integrated Circuits and Systems*, ed. E. Sanchez-Sinencio (IEEE Press, New York, 1994) pp. 408–417.
[8] A. Graps, An introduction to wavelets, IEEE Computational Science and Engineering 2(2) (1995) 50–61.
[9] R. Han, P. Bhagwat, R. LaMaire, T. Mummert, V. Perret and J. Rubas, Dynamic adaptation in an image transcoding proxy for mobile Web browsing, IEEE Personal Communications Magazine (1998) 8–17.
[10] J.-H. Jeong and C. Yoo, A server-centric streaming model, in: *Proc. of NOSSDAV 2000* (June 2000) pp. 25–34.
[11] Y. Joo, Y. Choi, H. Shim, H.G. Lee, K. Kim and N. Chang, Energy exploration and reduction of SDRAM memory systems, in: *Proc. of ACM Conf. on Design Automation*, USA (2002) pp. 892–897.
[12] D. Legall, MPEG – A video compression standard for multimedia applications, Communications of the ACM 34(4) (1991) 46–48.
[13] D.A. Lelewer and D.S. Hirschberg, *Data Compression, ACM Computing* (Springer, Heidelberg/New York, 1989).

[14] C. Loeffler, A. Ligtenberg and C.S. Moschytz, Practical fast 1D DCT algotithm with eleven multiplications, in: *Proc. of ICASSP* (1989) pp. 988–991.
[15] N. Merhav and V. Bhaskaran, Fast algorithms for DCT domain image down sampling and for inverse motion compensation, IEEE Transactions on Circuits and Systems Video Technology 7(3) (1997) 468–476.
[16] D.G. Morrison, M.E. Nilson and M. Ghabari, Reduction of the bit-rate of compressed video while in its coded form, in: *Proc. of Internat. Packet Video Workshop*, Portland, OR (September 1994) pp. D17.1–17.4.
[17] C. Mulcahy, Image compression using the Haar wavelet transform, Spelman Science and Mathematics Journal 1(1) (1997) 22–31.
[18] Qualcomm Corp., `http://www.qualcomm.com`.
[19] I.E.G. Richardson, *H.264 and MPEG-4 Video Compression* (Wiley, New York, 2003).
[20] A. Said and W.A. Pearlman, A new fast and efficient image codec based on set partitioning in hierarchical trees, IEEE Transactions on Circuits and Systems for Video Technology 6 (June 1996) 243–250.
[21] Samsung Electronics Co., `http://www.sec.co.kr`.
[22] T. Shanableh and M. Ghanbari, Heterogeneous video transcoding to lower spatio-temporal resolutions and different encoding formats, IEEE Transactions on Multimedia 2(2) (2000) 101–110.
[23] J.M. Shapiro, Embedded image coding using zero trees of wavelet coefficients, IEEE Transactions on Signal Processing 41(12) (1993) 3445–3462.
[24] SK Telecom, `http://www.sktelecom.com`.
[25] B. Sostawa, T. Dannemann and J. Speidel, DSP-based transcoding of digital video signals with MPEG-2 format, IEEE Transactions on Consumer Electronics 46(2) (2000) 358–362.
[26] R. Talluri, Error-resilient video coding in the ISO MPEG-4 standard, IEEE Communication Magazine 36 (June 1998) 112–119.
[27] R. Ulichney, A review of halftoning techniques, Proceedings of SPIE 3963 (2000) 378–391.
[28] Video coding for low bit rate communication, ITU-T Recommendation H.263, version 1 (November 1995), version 2 (January 1998), version 3 (November 2000).
[29] G.K. Wallace, The JPEG still picture compression standard, IEEE Transactions on Computer Electronics 38(1) (1991) 18–34.
[30] J. Wen, P. Meshkat and J. Villasenor, Structured trees for lossless coding of quantized wavelet coefficients, in: *Proc. of Asilomar Conf. on Signals, Systems, and Computers* (November 1996) pp. 3–6.
[31] J. Youn and M. Sun, Motion vector refinement for high performance transcoding, IEEE Transactions on Multimedia 1(1) (1999) 30–40.
[32] Q. Yu and K.J. Parker, Quality issues in blue noise halftoning, Proceedings of SPIE 3300 (1998) 376–385.
[33] W. Zhu, K. Yang and M. Beacken, CIF-to-QCIF video bitstream down-conversion in the DCT domain, Bell Labs Technical Journal 3(3) (1998) 21–29.