Available online at www.sciencedirect.com

SCIENCE @ DIRECT®

J. Vis. Commun. Image R. 17 (2006) 564–572

JOURNAL OF VISUAL Communication & IMAGE Representation

www.elsevier.com/locate/jvci

Trading off quality and complexity for a HVQ-based video codec on portable devices

Marco Cagnazzo [a], Francesco Delfino [b], Luca Vollero [a,b,*], Andrea Zinicola [c]

[a] *Università degli Studi di Napoli "Federico II", Italy*
[b] *ITEM—Laboratorio Nazionale CINI*
[c] *LABCOM—Laboratorio Nazionale CNIT*

Received 15 December 2004; accepted 1 December 2005
Available online 26 January 2006

Abstract

Bandwidth and processing requirements of multimedia applications typically exceed capabilities of portable terminals with current technology. Applications should hence be able to accommodate their requirements to run on these devices. In this paper, we provide a performance characterization of a video codec based on techniques such as hierarchical vector quantization which trade off complexity and reproduction quality. Comparison with standard codecs shows a remarkable reduction of coding times, such that real-time coding/decoding of video becomes possible even on low-power devices. This complexity reduction is counterbalanced by reproduction quality impairment. Nevertheless, for application such as video-conference, subjective quality seems to be fairly acceptable. Our analysis also quantifies some limitations of low-power devices with current technology.
© 2005 Elsevier Inc. All rights reserved.

Keywords: Multimedia; Coding techniques; Palmtop devices; Performance evaluation

1. Introduction

The evolution of communication networks and the fast improvements of microprocessors performance have enabled a new set of multimedia applications. However, computer and network architectures are still strongly heterogeneous, above all when we consider recent trends toward mobile wireless systems. Thus, applications must be able to adapt to widely different conditions in terms of network bandwidth, visualization capabilities and, especially, available processing power. In particular, for low-power devices, the available computational power is still an issue, and advances in battery technology and low-power circuit design will not alone meet the demands of multimedia applications, where both network bandwidth and processing capabilities of clients may easily become a bottleneck. This paper focuses on experimental assessment of a recently proposed video coder which trades off complexity and reproduction quality, potentially enabling real-time

[*] Corresponding author. Fax: +39 081 768 4219.
E-mail addresses: vollero@ieee.org, vollero@unina.it (L. Vollero).

1047-3203/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jvcir.2005.12.001

processing of multimedia on low-power devices [1,2]. We provide a characterization of this codec and a comparison with more conventional client architectures and with standard coding schemes like H.263 and MPEG-4. We do not consider the last video coding standard H.264 because its improved performance comes at cost of a remarkable increase in complexity [3], making it unsuit to very low-complexity architectures. Our experiments show that the considered codec has good performance (in terms of encoding and decoding times) on low-power devices, but this is achieved by giving up some reproduction quality. However, for applications like video-conference, this degradation is limited, above all if we consider subjective quality instead of objective parameters. Our analysis also quantifies some limitations of low-power devices with current technology and gives hints on how applications can be ported on these devices with acceptable performances. The paper is organized as follows. In Section 2, we describe the low-cost coding strategy; in Section 3 we discuss the experimental setup of our tests and report our experience in porting the codec on a palmtop device. In Section 4, we present the experimental results. Finally, in Section 5 related work is briefly reviewed, and general conclusions are drawn.

2. Low-cost video coding techniques

The current video coding standards assure very good performances in a wide range of conditions, but have a remarkable encoding complexity that cannot be reduced too much without completely changing approach. In order to enable real-time video coding/decoding on low-power devices, we must resort to simpler compression algorithms than those used in current standards, even at the cost of increased rate or impaired quality. In this respect, a low-complexity video coding system has been proposed by Chaddha and Gupta [2]. From now on, we refer to it as the CG coder. Like all video coding algorithms, the CG coder achieves compression by exploiting the *temporal* redundancy (i.e., the resemblance among successive frames) and *spatial* redundancy (i.e., the similitude among different parts of a single frame). Usually, the most important contribution to video coding algorithms complexity lies in the temporal compression step, which involves heavy operations like motion estimation (ME) and motion compensation (MC). With MC-based compression, each frame was divided in blocks of pixel, usually called macroblocks (MB). To encode a MB of the current frame, it is compared to another MBs around its position in a reference frame, looking for the most similar one (this is the ME stage), which is then used as prediction of the current MB. Finally, the prediction error is encoded and sent.

The CG coder exploits temporal redundancy by means of conditional replenishment (CR). When CR is employed, ME and MC are not performed. Current MB is just compared to the MB in the same position in the reference frame, and it is transmitted only if it is not predictable from the reference MB, i.e., if their distance with respect to a suitable metric, as the mean square error (MSE), is greater than a given threshold. Note that this MSE-based CR test accounts for much of the encoder complexity. Non-predictable MBs are compressed by means of hierarchical vector quantization (HVQ) [2] which has a negligible complexity, much lower than transform coding (employed in standard algorithms), because it only requires a sequence of table look-ups to encode a block. For example, an 8-pixels block X is encoded by a three stage scheme as in Fig. 1A. Each couple of pixel is vector quantized by looking up in first level table, in which the best representing codeword for every possibly input couple has been stored. A couple of first level codewords (representing a 2×2 pixels block) is then sent as input of the second stage (which uses different tables), and a further stage is required for 8-pixels blocks. In order to compute the tables, we just need three codebooks (one for each stage)

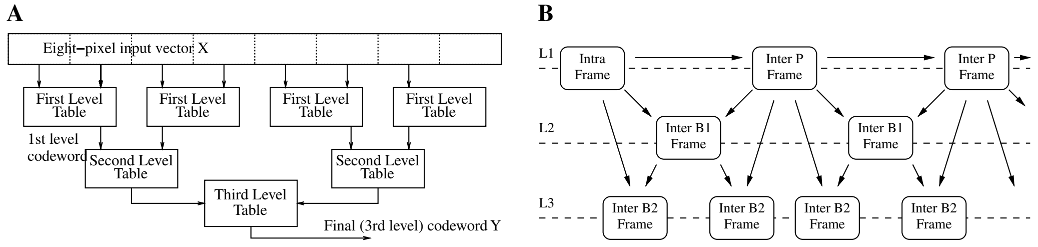

Fig. 1. (A) A 3-stage HVQ encoder. (B) Layer structure for the low resolution stream.

which, however, can be designed at will. The original version of the CG codec employs a tree-structured codebook to supply quality scalability. The improved version of the codec [1] uses ordered codebooks, needed to reduce the encoder complexity, as we show further on. The CG algorithm provides a good degree of *scalability*, as well. An encoded stream is said to be scalable when it is composed by embedded subsets, which are efficient compressed representations of original data, but at different resolutions (both spatially or temporally) or qualities. Spatial scalability is ensured by pyramidal coding on two layers: a low resolution one, and another containing the fine details, called enhanced layer. Temporal scalability is obtained by structuring the sequence of frames in multiple temporal layers. In Fig. 1B we show temporal layers structure for a group of pictures (GOP) in the low resolution stream. The complementary high resolution stream has the same three-level structure. The arrows represent CR dependencies: thanks to this scheme, frames on layer L_n only need frames on layer L_m (with $m \leqslant n$) to be encoded or decoded. Note that, in each GOP, there is one "intra" frame which does not depend on other frames for encoding and decoding. Other frames are called "inter" frames. The CG coder exhibits worse RD performance with respect to more sophisticated techniques, but also a much lower complexity which, together with its good scalability, makes it a good candidate for applications where computing power is severely limited.

In this paper, we use the improved version of the CG codec proposed in [1], where the CG codec was analyzed in detail to reduce its remaining complexity. It was found that residual bottlenecks lie in the CR, which accounts for about 50% of the overall CPU time, and in the anti-aliasing filtering and interpolation required by pyramidal coding, accounting for 40% of the CPU time. Experimental results proved that the new codec is about twice as fast as the CG codec, and also provides a slight improvement in reproduction quality. The main modifications introduced in [1] to the CG codec are the following.

The first aim was to reduce CR complexity. This is achieved by performing this step *after* the spatial compression and not before it (as usually happens). Of course this is possible only if, in the spatial compressed domain, it is still feasible to compute a similarity measure among MBs. We observe that, after HVQ, a MB is made up of the addresses of the codewords with which it has been encoded. We can tell how much two MBs are similar just by looking at their vector quantization addresses, only if "ordered codebooks" are used. In an ordered codebook, if two addresses A' and A'' are close, then the squared error $d(Y', Y'')$ between the corresponding codewords is small, while if A' and A'' are far apart, quite likely $d(Y', Y'')$ will be large. Therefore, the CR can be simplified by having it check codeword addresses produced by the HVQ encoder, rather than the MB pixels, obtaining the new test: $\sum_{k=1}^{N}|A(k) - A_r(k)| > T$, where N is the number of codewords in a MB, $A(k)$ is the address of the kth codeword, and $A_r(k)$ is the address of the corresponding codeword in the reference MB. The threshold T trades off reproduction quality and compression ratio. For $T = 0$, only *unchanged* MBs are not sent, achieving the best quality but also the higher bit-rates, while with a high threshold, only heavily changed MBs are sent, resulting in a little bit-rate and a worse quality. The actual threshold can be decided by the encoder, to adapt to network bandwidth. Anyway, in our tests the choice $T = 5N$ often achieved a good trade-off between quality and coding rate. Remark that this choice means that replaced codeword addresses differ at most of 5 in the average. In conclusion, the new CR test (first proposed in [1]) remarkably reduces the CR complexity, since it only requires a few integer sums for each MB, with respect to the heavy pixel-by-pixel computations needed by ordinary CR. We also note that the off-line design of an ordered codebook is not trivial, but it has long been investigated [4], and some efficient solutions exist in literature [5].

The other major source of complexity in the CG codec is the filtering needed by spatial scalability. This problem is solved by generating a hierarchy of tables that accept the pixels in the filter support as input, and provide in output an approximated filtered value. For small filters, (4 taps in each direction) the approximation is extremely good, and the processing very fast.

3. Experimental setup and methodology

We used in our experiments an HP iPaq 3850 as low-power device. It is a palmtop that meets minimum requirements for hosting non trivial multimedia applications. To characterize codec performances on the iPaq platform, we carried out a preliminary evaluation of its capabilities with respect to a Pentium-based workstation (WS). The characteristics of both platforms, including the Operating System and the C compiler used, are

Table 1
Platforms characteristics

Platform	iPaq 3850	Workstation
Processor Type	Intel StrongARM 1110, 206 MHz	Intel Pentium III, 600 MHz
Memory	64 MB	384 MB
Cache L1/Cache L2	24KB (I-16, D-8)/not present	32 KB(I-16, D-16)/512 KB
OS	Linux2.4.18	Linux2.4.7-10
C compiler	gcc 2.95.2	gcc 2.96

reported in Table 1. This experimental setup can reasonably be assumed as representative of realistic scenarios in the mid-term period, as both CPU performances and complexity of multimedia application are expected to increase.

We ran two benchmarks to assess differences in memory and CPU performance. The first one consists in copying memory blocks of different sizes, providing information about all levels of memory hierarchy. The second benchmark consists of four nested loops, in which we transfer data among structures used in the CG codec. This benchmark has been designed so that all memory operations are performed in the L1 cache, to roughly evaluate the CPU speed and the effects of compiler optimizations. Results are reported in Table 2 (first three lines). For CPU performance, we reported the time needed to perform the test, normalized to the time of the best configuration (all optimizations enabled on the WS platform). In the third column of the table it is reported the iPaq slowdown factor with respect to the WS. From these tests, we conclude that, if a relevant fraction of memory accesses are out of cache, iPaq is at least one order of magnitude slower than the WS. This difference can be even higher for in-cache computations. Moreover, we note that only -O1 compiler optimizations seem to have effect on the iPaq platform. We argue that the main reason for this is that the slow memory system makes performance almost independent of further code optimizations, providing that the memory access pattern is unchanged (as in our benchmark). To assess the effectiveness of the CG approach, we decided to compare our implementation with some standard algorithms. We ported on iPaq two open source codecs: the FFMPEG suite, including an H.263[1] codec, and XviD,[2] an MPEG-4 compliant codec The most recent standard H.264/AVC was also considered, as it provides huge gains in compression efficiency (up to 50%) compared to previous standards [3]. However, the decoder complexity is about four times that of MPEG-2 or H.263, and two times that of MPEG-4 [3], and the encoder complexity has a similar increase (even though it depends on the implementation). Since we are considering low-complexity codecs, we did not consider the H.264 encoder in the following.

All measurements reported in the following refer only to coding/decoding operations, excluding frame acquisition and visualization. The CG and H.263 codecs results have been obtained by code instrumentation, using a high resolution timer. Measurements concerning the XviD codec were directly provided by its open source implementation. We performed all measurements on two video sequences (referred to as *video 1* and *video 2*): the first one is a typical video-conference video, with a large fixed background and small movements of objects. The second sequence is a typical trailer with scenes characterized by sudden changes and fast moving objects.

As far as porting and optimization are concerned, the codec has been ported on the iPaq platform from an original WS-oriented implementation. This required minor modifications to the code and we run a first set of tests using this version of the codec. In the attempt to further improve performance, we also manually introduced into the code two kinds of simple optimizations. First, individual bytes copies from one data structure to another were reorganized with block transfers through calls to `memcpy`. Second, we manually performed the complete unrolling of loops cycling a small number of times (typically 3 or 4). Tables 3A and B report frame elaboration times on WS and iPaq, with and without setting compiler optimizations. Data refer to base stream P-frames decoding for the first video but they are representative also of the second test. In all cases, compiler optimizations reduce decoding times, meaning that they improve memory accesses locality. Manual

[1] [Online]. Available: `http://ffmpeg.sourceforge.net`.
[2] [Online]. Available: `http://www.xvid.org`.

Table 2
Performance comparison between platforms for different compiler optimization flags

Parameter	iPaq	WS	Slowdown
Memory BW [MB/s]	30.5	250	8.2
L2 cache BW [MB/s]	—	1080	—
L1 cache BW [MB/s]	37.5	1400	37.3
CPU (no opt.) time ratio	36.6	9.1	4.0
CPU (-O1) time ratio	28.8	4.4	6.5
CPU (all opt.) time ratio	28.8	1.0	28.8

Table 3
Single frame decoding times and speedup factor

Platform	Original (ms)	Manually optim. (ms)	Speedup
(A) w/o compiler optimizations			
WS	3.8	3.2	1.18
iPaq	21.7	14.5	1.50
(B) With compiler optimizations			
WS	1.8	1.7	1.06
iPaq	9.6	9.1	1.05

optimizations improve performance as well, but their effect is limited when compiler optimizations are enabled. In our experiments we used the manually optimized version of the codec with all compiler optimizations enabled.

4. Performance analysis

In this section we compare CG codec to H.263 and XviD on the iPaq platform. To make the comparison fair, we set the coding parameters of both codecs so as to generate a video with the same frame size, the same compression ratio, and the same GOP structure of the one generated by the CG codec. Since neither the H.263 standard nor the current implementation of XviD do support B-frames, we do not use them in the CG coder neither, i.e., we used only the first level of temporal hierarchy, see Fig. 2B. Test configuration parameters are the following: the base stream has a resolution of 176×144 pixels and is encoded at 80 Kb/s for a compression ratio of 16.2%. The enhanced stream has 352×288 pixels, and is encoded at 300 Kb/s with a compression ratio of 15.4%. For both the base and enhanced layers, the frame rate is 6.25 frame/s and there are 24 P-frames per each I frame.

We first report coding and decoding times of the three codecs for the test sequences. Figs. 2A–D report mean coding and decoding times and their standard deviation (in milliseconds) of the three codecs for the low resolution stream, in the cases of video 1 and video 2, respectively. These results highlight the symmetry and the low complexity of the CG coder: coding and decoding times are very close each other, while standard codec have much slower encoding times: for video 1 the CG codec is about 3 times faster than the H.263 codec and almost 5 times faster than the MPEG-4 codec in the coding phase. The speedups increase to 4 and 6.5, respectively, for video 2, which requires a more expensive motion compensation. As to the decoding phase, differences are much less remarkable and, moreover, times are all compatible with real-time operation. In particular, for video 1 H.263 is slightly faster than CG and XviD, while for video 2, CG has slightly higher decoding times than both H.263 and XviD, with higher variability. Nevertheless, even in this case decoding times widely remain within acceptable limits.

Results concerning the enhanced stream are shown in Figs. 2E–H. In this case, the CG codec is 2–4 times faster than H.263 and XviD in the coding phase and slightly slower in the decoding phase. Table 4 reports codecs performances on the WS, to evaluate the differences in codecs behavior on the two platforms. We note that in this case the codecs have closer performances, with the H.263 codec proving to be the fastest and the XViD the slowest. We conclude that CG scheme is well adapted to

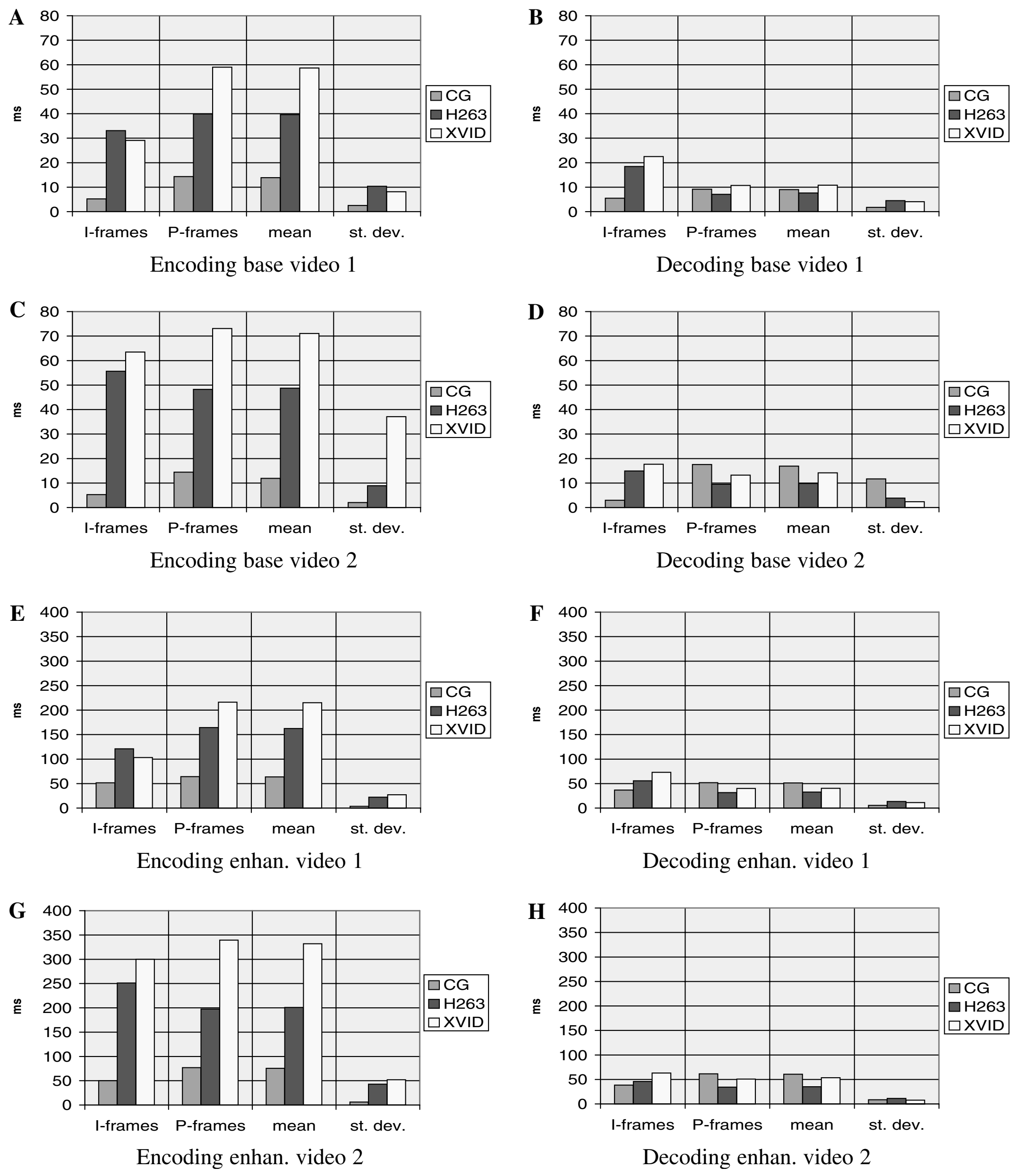

Fig. 2. Coding and decoding times on the iPaq platform.

Table 4
Mean coding time [ms] for P-frames in the high resolution stream on the WS platform

Sequence	CG	H.263	XviD
Video 1	12.0	10.6	17.9
Video 2	17.4	11.6	25.9

low-power platforms, while it loses its advantages on workstations. This is explained by the fact that the H.263 and XViD implementations for WS benefit from assembler optimizations (as the use of MMX and explicit cache allocation instructions) which are not available for the palmtop. On the contrary, the

Table 5
Mean coding and decoding times (milliseconds) of B-frames for the CG codec

	Coding	Decoding
(A) Base		
Video 1	11.20	7.72
Video 2	11.22	7.73
(B) Enhanced		
Video 1	58.93	51.97
Video 2	56.20	47.74

Fig. 3. Screen shots at low (QCIF) resolution from video 1 (1st row) and video 2 (2nd row).

CG coder implementations (neither for WS nor for palmtop) has not explicit assembler optimizations, which could further improve its performances.

All these data confirm that the CG coding scheme achieves reduced coding times with respect to standard codecs. From a quantitative point of view the speedup achieved on a low-power platform in the coding phase is between 2 and 6, depending on input sequences and on coding parameters. Decoding times are close to those that standard codecs can achieve. These considerations hold as well in the case B-frames are considered. Coding and decoding times of B-frames on the iPaq platform for the CG codec are reported in Table 5. Recalling that the number of B-frames in a GOP is three times the number of P-frames, it seems that when all the temporal layers are considered, the CG codec has even better mean performance than those presented so far. However, as other codecs do not support B-frames, it would not be possible a fair comparison, so we do not further discuss this matter.

Mean coding and decoding times per frame of CG on the iPaq are compatible with real-time applications (at 25 frames/s), if only low resolution stream is used. On such a device, H.263 or MPEG-4 streams with same frame size and compression ratio can be decoded in real-time (e.g., video streaming), but cannot be encoded (e.g., video conference). High resolution streams cannot be encoded in real-time on a portable device when all temporal layers are used (i.e., a frame rate of 25 frame/s is required). However, the CG codec scalability allows discarding the last temporal layer and operating in real-time and full resolution at 12.5 frames/s. If such a frame rate is acceptable, both video streaming and video conference applications can run on a low-power device. Moreover, in the video streaming case, CG intrinsic scalability would not require multiple copies of the same video at the source site, while this would be mandatory for the other codecs, for which video streaming applications could use alternative formats only if the video has been coded off-line at the proper rate.

For a complete characterization of the codec we must consider reproduction quality as well. Quality measures of compressed multimedia data are divided into two categories: *subjective* measures (related to the decoded data quality which can be perceived by human observers) and *objective* measures (based on mathematical properties of data). The most common objective quality measures are based on mean square error (MSE) between original and decoded data, such as the Peak Signal-to-Noise Ratio. For video and images, it is defined as: $\text{PSNR} = 10 \cdot \log_{10}(255^2/\text{MSE})$. The PSNR has the advantage to be easily computable, and to be related to an Euclidean metric in the signal space, but it does not perfectly account for perceived quality. Objective quality measures single out that low complexity algorithms suffer from a relevant PSNR decrease, up to 10 dB, depending on the input sequence. However, it should be considered that subjective quality does not suffer from such a large penalization. This is shown in Fig. 3, where we report some screen-shots from the two video sequences. For the first sequence, characterized by limited and smooth motion, we see that quality is acceptable and consistent during reproduction. In the second video sequence there is some sudden movement, which is not easy to encode for our scheme as it does not perform motion compensation. This is quite evident in the second decoded frame, which is amid a camera panning. Here, reproduction quality is not acceptable. Afterward (third frame), the scene stabilizes, allowing again a sufficient quality. To compare the subjective quality of reconstructed video sequences, we performed several trials with observer judgements. We employed the usual five-levels scale for visual impairments [6], from 5 (imperceptible impairment) to 1 (very annoying impairment). We obtain a quality level of 2.8 for the CG codec, 3.0 for H.263 and 3.1 for XviD. These results confirm that the CG coder has an acceptable subjective quality with respect to H.263 and MPEG-4.

5. Related work and conclusions

Low complexity algorithm for multimedia processing is an active research field, but only a few experimental work is reported in the literature for what concerns low-power devices. In [7], Johanson analyzes the performance of a wavelet based codec. His work indicates that different implementations of performance-critical code are necessary to use it on different platforms. In our work, we use such a simple algorithm that we do not need to perform this tuning. In this way, we also obtain a source-level portable software. In [8], Sheikh et al. focus on a standard H.263 encoder and optimize its code for an embedded Digital Signal Processor. They demonstrate that access to external memory is a bottleneck for video systems with large memory requirements and they suggest to use implementation-dependent designs. Our work confirms that memory can be a bottleneck also on low-power devices, but our results show that the CG approach can lead to acceptable performance with minimal optimization efforts. More in general, processing requirements of expensive multimedia applications (e.g., real-time video processing) cannot be supported by devices like palmtops and smart-phones. Our work shows that the CG codec can perform video coding at low cost thanks to its peculiar encoding algorithm. This makes it well suited for time-constrained multimedia applications. We also show that state-of-the-art codecs for both streaming (MPEG-4) and video conference (H.263) cannot perform real-time video coding on low-power devices, although they lead to acceptable decoding performance. Another interesting aspect of CG codec is its high scalability, which allows to change reproduction quality on-the-source (server overloading control), on-the-middle (network congestion control) and on-the-device (computational overloading control) without computational overhead. The counterpart of these good features is a lower PSNR, which, nevertheless, does not necessarily correspond to unacceptable visual quality. On the contrary, for application like video conference, final video quality seems to be fairly acceptable to the user.

Finally, authors would like to acknowledge the support to this work provided by the *Centro Regionale di Competenze ICT* (Benevento, Italy) and by the *Ministero dell'Istruzione, Università e Ricerca* in the frameworks of the FIRB project "Middleware for advanced services over large-scale, wired-wireless distributed systems (WEB-MINDS)," and of the project "PETIT-OSA."

References

[1] M. Cagnazzo, G. Poggi, L. Verdoliva, Low-complexity scalable video coding through table lookup VQ and predictive index coding, in: Proceedings of IDMS-PROMS, Coimbra (Portugal), 2002, pp. 166–175.

[2] N. Chaddha, A. Gupta, A framework for live multicast of video streams over the internet, in: Proceedings of International Conference on Image Processing, 1996, pp. 1–4.

[3] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, T. Wed, Video coding with H.264/AVC: tools, performance, and complexity, IEEE Circuits Syst. Mag. 1 (2004) 7–28.

[4] G. Poggi, Address-predictive vector quantization of images by topology-preserving codebook ordering, Eur. Trans. Telecomm. Related Tech. 4 (1993) 423–434.
[5] T. Kohonen, Self-Organization and Associative Memory, Springer-Verlag, New York, 1988.
[6] Recommendation ITUR BT.500, Methodology for the subjective assessment of the quality of television pictures.
[7] M. Johanson, Implementation issues for scalable multimedia communication systems, Framkom tech. Report <http://w2.alkit.se/~mathias/>.
[8] H. Sheikh, S. Banerjee, B. Evans, A. Bovik, Optimization of a baseline H.263 video encoder on the TMS320C6x, in: Proceedings Texas Instr. DSP Educator's Conf., 2000.