Fast H.264 Picture in Picture (PIP) Transcoder with B-slices and Direct Mode Support

Yan Michalevsky [#], Tamar Shoham [*]

Electrical Engineering Department, Technion, Israel Institute of Technology
Technion Campus, Haifa 32000, Israel
[#] ymcrcat@gmail.com
[*] tshoham3@gmail.com

***Abstract*—H.264, the modern standard for video coding, has become increasingly popular in recent years, and offers solutions for many applications requiring video compression. Some of these applications require insertion of content into an already compressed video. This operation incurs high computational cost if a naive approach of full re-encoding is taken. Previous efforts in this field targeted the Baseline Profile of the H.264 standard. The Main Profile, that offers very efficient encoding modes using B-slices and Spatial Direct encoding, has not been addressed. We present *Guided Encoding* - a novel solution for efficient Picture-in-Picture embedding using the coding parameters of the original compressed bit stream. We also present an algorithm for handling Spatial Direct mode encoding to prevent propagation of errors caused by the embedding of new content into the original sequence. We reduce the computational time by an average factor of five in comparison to performing full re-encoding while preserving the video quality.**

I. Introduction

Video compression research gained momentum over the last decade, resulting in numerous practical applications. These applications are of high commercial value for various media industries, converged audio-video communication, mobile services and many more. H.264, also known as MPEG-4 Part 10 or AVC, is a joint standardization effort by ITU and the ISO MPEG for video compression. Having been designed for a wide range of scenarios and different applications it is of high complexity, and incurs high computational costs. Our aim is to support insertion of content, such as a static channel logo, an advertisement, sign language simultaneous translation, or any other video sequence inside the original video. The approach of simply decoding the video, insertion of the desired content into each picture and re-encoding the resulting sequence again will be referred to as the **naive approach**. It provides good video quality and reasonable bit-rate, but is very time-consuming.

Previous works on content insertion focus on different aspects of performance: bitrate, runtime and quality. An early article by Chang and Messerschmitt [1] discusses compositing of motion compensated video and proposes a method of video composition in the pixel domain. In [2] they offer a method of composition in the transform domain. An application to real MPEG-1 bitstreams is proposed in [3]. These algorithms require perfect alignment of the inserted content with the 8x8 block grid. In an article by Roma and Sousa [4], an algorithm for transform domain transcoding is proposed. It overcomes some of the limitations in previous works and aims to provide a better re-estimation of the motion vectors that results in a more compact residual. While working in the transform domain is beneficial for MPEG-2 and other classic coders, in the case of H.264 the transform calculation itself is negligible, and most of the coding complexity stems from choosing the optimal mode for encoding each macroblock. In [5], Li et. al. present a transcoding scheme, based on partial re-encoding (PRET) of macroblocks affected by content insertion, with optimization of motion vector re-estimation. In [6], they present PIP transcoding with slice groups and PIP transcoding using an auxiliary bitstream containing the content to be inserted into the background sequence.

We propose a method of ***Guided Encoding***. The basic underlying concept is reusing the encoding decisions from the original video for the areas of the pictures that are not affected by the inserted content. Previous works on content insertion for H.264 ([5], [6]) addressed only the Baseline Profile of the H.264 standard, and therefore only supported I and P slices. The standard De Facto for high-quality video in the entertainment industry as well as in other applications of H.264 is the Main Profile. One of its most important features are B slices, that significantly increase compression efficiency by offering Bi-directional prediction and Temporal or Spatial Direct encoding modes [7], [8]. The goal of this research was to apply *Guided Encoding* to additional modes, which are part of the Main Profile and were not covered by previous work. To evaluate the proposed algorithm we implemented modified versions of the reference implementations of an H.264 decoder and encoder that apply the concept of *Guided Encoding* to video streams that utilize Bi-directional Prediction, Weighted Prediction, Temporal Direct mode and Spatial Direct mode.

The structure of this paper is as follows: it starts with a short technical background referring to several concepts of the H.264 standard, which are required for understanding our transcoding method. In the Content Insertion section we explain our method of *Guided Encoding*, covering the methods of handling different encoding modes supported by the Main Profile. In section IV we present an evaluation of our method and compare it to the naive approach, referring to run-time, quality and bit-rate measurements. Finally, we summarize our work and present our conclusions.

II. Background

We assume basic knowledge of the H.264 standard by the reader. Hence we provide only the background necessary to understand the innovations in our encoding method and refer to [9] wherever we make use of otherwise unexplained concepts.

A. Slice Types

In our work we used pictures that contain a single slice, but this is not an inherent limitation of the algorithm. Intra (I) slices contain only I macroblocks, where each macroblock is predicted from previously coded data in the same picture. This picture type is supported by all profiles of H.264. Predicted slices (P) are Inter slices that contain I and P macroblocks. Each P macroblock in a P slice is predicted from a single reference picture. This slice type is supported by all profiles. Bi-directionally predicted slices (B) are generalized Inter slices. This type of slice is the focus of the current work. They are supported by the Main Profile. Each B macroblock in a B-slice may be predicted from List 0 and/or List 1 reference picture. B slices can contain P and I macroblocks as well.

B. Bi-directional Prediction

Bi-directional prediction is enabled in B-slices and utilizes two lists of reference pictures, List 0 and List 1 ([9]). The difference between the two lists is in the method of ordering past and future reference pictures. List 0 enables efficient access to past reference pictures while List 1 enables efficient access to future reference pictures. In Bi-directional prediction the samples are the weighted average of the samples from the reference pictures specified by the two lists, and the motion estimation is based on the two sets of provided or deduced motion vectors.

C. Weighted Prediction

In weighted prediction mode each block is predicted as a weighted sum of two reference blocks. The weighting factor, *alpha* is either explicitly provided in the bitstream or deduced according to the temporal distances of the reference pictures.

D. Direct Encoding

Direct encoding assumes a strong correlation between temporally or spatially adjacent macroblocks. When direct coding mode is used for coding a macro-block no motion vector is transmitted. List 0 and List 1 vectors are calculated based on previously coded vectors. Direct mode could be utilized within P (skip mode) or B slices. Two types of correlation are exploited - temporal and spatial correlation ([9], [8]). Direct prediction is available for 16×16 and 8×8 partitions. It is not available for 16×8 or 8×16 partitions.

E. Calculation of Motion Vectors

During decoding, Motion Vectors (MVs) are calculated as a sum of two parts: a PMV (Predictive Motion Vector) [10], that is calculated by the decoder according to prediction parameters, and a DMV (Differential Motion Vector) that is transmitted as part of the encoded bitstream. In Direct encoding the DMV is not present as no data is transmitted for the macroblock in this encoding mode. As we shall see later, this has implications on our method of handling Spatial Direct macroblocks.

III. Content Insertion

In H.264, transform domain manipulations could save only a small portion of the encoding time and therefore the content insertion is done in the Pixel Domain. Obviously it requires decoding of the H.264 bitstream as a first step, but decoding time is negligible compared to the encoding time, even when optimizations such as Fast Mode Decision or our *Guided Encoding* are applied. Therefore we concentrate on optimizing the re-encoding process. The output picture is constructed by a combination of the input picture and the inserted content, or *"logo"*. The logo may be opaque or transparent to a degree defined by the factor γ (Eq. 1), a real value in the range [0, 1], where 1 indicates an opaque logo and 0 indicates complete transparency. γ is specified as a parameter to our encoder. Fig. 1 illustrates content insertion with $\gamma = 0.5$.

$$out[m,n] = \begin{cases} \gamma \cdot logo[m,n] + (1-\gamma) \cdot in[m,n] & \text{m,n} \in \text{logo support} \\ in[m,n] & \text{m,n} \notin \text{logo support} \end{cases} \tag{1}$$

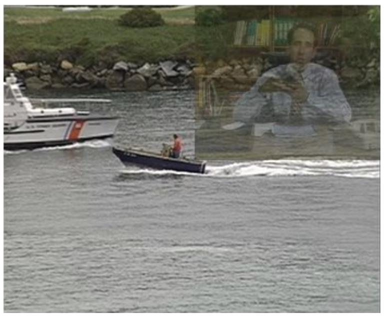

Fig. 1. Example of content insertion with $\gamma = 0.5$

Our method of performing efficient encoding of video along with content insertion is based on reusing the information that exists in the original encoded video. An encoded H.264 stream contains many useful parameters which are the results of the decision-making process performed while encoding the original video. To extract this metadata we first run a decoder that stores slice-level information and macroblock-level information. To insert content into the video sequence we operate on the decoded YUV sequence. We run an encoder that takes the decoded video, the YUV sequence of the content to be inserted and the metadata extracted by the decoder and produces an encoded H.264 stream, combining the new content with the original. Our decoder and encoder are based on the H.264 reference implementation JVT-JM ver. 11.0 [11]. Fig. 2 illustrates our content insertion process.

The extracted coding domain metadata includes different slice-level and macroblock-level encoding parameters. Slice-level parameters include: slice type (I, P or B), quantization parameter and weighted prediction type. For B-slices,

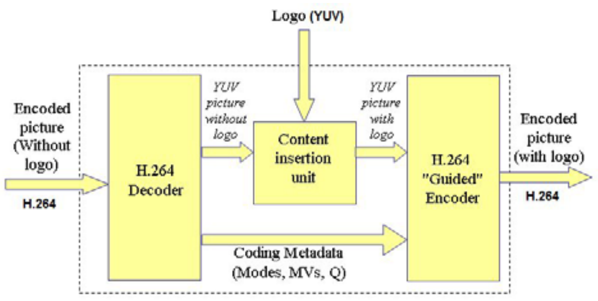

Fig. 2. Content Insertion Scheme

macroblock-level parameters include: quantization parameter, macroblock type (partitioning and prediction mode), encoding mode for each block in case of partitioning, forward and backward reference list (values of List 0 and List 1), List 0 and List 1 motion vector values, forward and backward alpha parameter and prediction direction (forward, backward or bi-directional). In case of P-slices only the relevant subset of these parameters is extracted. For Intra slices (I) block types and prediction modes are extracted per macroblock.

A. Macroblock Classification

Macroblocks are classified into *affected*, *indirectly affected* and *unaffected*. A macroblock is defined as *affected* when at least one of its pixels is in the new content region. A macroblock is *indirectly affected* when the pixels it uses for prediction belong to an *affected* macroblock. Otherwise, it is marked as *unaffected*. An Intra macroblock is *indirectly affected* if the macroblock above or to the left of it is *affected*. An Inter macroblock is *indirectly affected* when its motion vector points to pixels that belong to *affected* macroblocks. This is illustrated in Fig. 3.

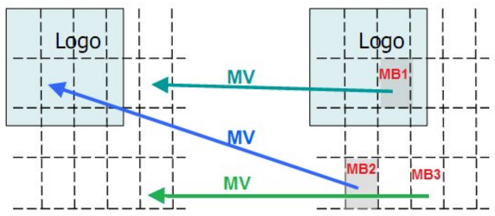

Fig. 3. Macroblock classification. MB1 is directly affected by new content, MB2 is indirectly affected because its motion vectors points to new content and MB3 is unaffected.

Unaffected macroblocks are encoded according to the parameters saved in the metadata extracted by our decoder. Encoding parameters for *affected macroblocks* in each picture have to be fully recomputed since the old parameters are no longer relevant. We also have to recalculate encoding parameters for *indirectly affected macroblocks* since they use affected macroblocks as their prediction references, and the reference information is no longer relevant. In the case of B-slices the determination of *indirectly affected* macroblocks becomes quite complicated since macroblock pixel values depend on backward as well as forward prediction. In the case of Spatial Direct encoding mode the classification algorithm was further extended to determine *indirectly affected* macroblocks.

B. Bi-directional Prediction Support

To support B-slices we need to handle both List 0 and List 1. List 1 reference picture identifiers were included in the metadata exported by the decoder. The Prediction Direction parameter is also extracted for each sub-partition of a macroblock. During the guided encoding process we consider the Prediction Direction value for each block and operate on List 0 and/or List 1 accordingly, i.e. we take into account only reference pictures from the relevant lists.

C. Weighted Prediction

Support for Weighted Prediction requires extraction of the forward and backward alpha parameters from the original stream for every sub-partition of each macroblock, and applying these parameters during the encoding of *unaffected* macroblocks. Also, on the picture level, one needs to extract the flag that determines whether Weighted Prediction is to be applied for this picture, and a flag that determines the mode to be used, i.e. implicit or explicit weighted prediction. We set these flags for each picture and the forward and backward alpha factors for each macroblock to reuse weighted prediction in the re-encoded video.

D. Temporal Direct Mode Handling

In Temporal Direct mode, predicted macroblocks are less susceptible to changes in reference picture content compared to Spatial Direct and therefore Temporal-Direct encoding mode is handled similarly to other Inter modes. Encoding mode, reference pictures and motion vector parameters are enforced for *unaffected* macroblocks.

E. Spatial Direct Mode Handling

Due to the nature of Direct macroblocks any change in a reference macroblock, which may be spatially quite far from the current Direct macroblock, affects its parameters. As mentioned, our encoder extracts the values of the Motion Vectors (MV) and stores them as part of our metadata. In the case of Direct macroblocks no data is written regarding the motion vectors and there is no residual (DMV) that could fix a PMV (Predicted Motion Vector) to match the value of the MV we seek. The value is entirely dependent upon calculations performed within the decoder. Therefore, if the data changes in the area of a Direct macroblock, the predicted pixels may be incorrect. This is due to the inconsistency between the original data and the new data. We need to recalculate the motion vectors to be consistent with the new content of the picture. We reuse the original JVT ([11]) algorithm that calculates motion vectors for Direct macroblocks, and avoid overriding those values with the values that appear in the encoding metadata and are no longer relevant. During transcoding we have to either recalculate the motion vectors or choose a different encoding mode for that macroblock. We first examined an approach where macroblocks encoded in a Spatial

Direct mode, that were found to be *unaffected* by the logo insertion, were set to Bi-predictive 16x16 mode, overriding the original mode. Then List 0 and List 1 reference indices are set according to the metadata extracted in the decoding stage, and motion vector predictor references are set using the original function of the encoder that finds suitable reference pictures. We abandoned this approach since changing mode from Spatial Direct to 16x16 Bi-predictive increases the bit rate significantly, because this type of encoding requires more information to be written compared with direct encoding. As a result, special handling of Spatial Direct mode is done in two stages of our algorithm - in the stage of classification of macroblocks into *affected* and *unaffected* (described in detail bellow), and in the stage of choosing encoding parameters for an *unaffected* macroblock. In the first stage we check whether the Direct macroblock should be re-encoded from scratch, or whether we can mark it as *unaffected* and set its parameters according to the previously extracted metadata. If a Spatial Direct macroblock is eventually marked as *unaffected* we maintain its mode in the output bitstream.

***Affected/Unaffected Macroblocks Determination**:* In classification of macroblocks as *affected* or *unaffected*, Spatial Direct macroblocks are treated as a special case. We find the co-located blocks and check whether one or more of them are *affected*. To determine whether a macroblock should be marked as *affected* we receive information about the co-located macroblocks, similarly to the method used in the Motion Vector prediction algorithm that looks at co-located blocks A, B, C and D to determine the motion vector for macroblock E [8]. The neighbor blocks are illustrated in Fig. 4.

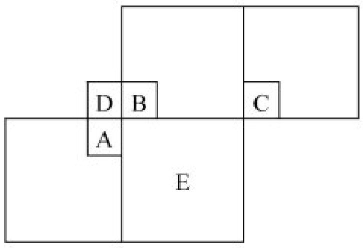

Fig. 4. Neighbors used for Direct Macroblocks classification

If one of those neighbors is marked as *affected* we mark our macroblock as *affected* as well. To find out whether one of the neighbor blocks is affected we store an *"affected macroblocks map"* that is reset before beginning the encoding of each slice. It is a 2-dimensional array with an element for each macroblock. The classification of each macroblock (as affected/unaffected/indirectly affected) is stored in this map. The function *get_affected(block)* looks up a value in this matrix and return the block *Affected* status. If the macroblock is marked as *affected* it is re-encoded, which prevents the effect of the inserted content on surrounding macroblocks. This is necessary since the new content creates inconsistencies between the original references and motion vectors and the visual data they refer to. Fully re-encoding the Direct macroblocks that have affected neighbors prevents this inconsistency. The classification algorithm for Spatial Direct macroblocks is described in Algorithm 1.

Algorithm 1 Affected Macroblocks Classification for Direct Macroblocks

```
if (MB is Direct) and (direct mode type is Spatial) then
  get co-located blocks (A, B, C)
  if block C is not available then
    use block D instead
  end if
  for each block do
    if block is Affected then
      current macroblock ← Affected
    end if
  end for
end if
```

Notice that this approach causes the *affected* area to grow, propagating from the "Logo area" to its surroundings through neighboring macroblocks. Tests showed that in many slices about 70% of the direct macroblocks are re-encoded. Although 20% of them were in the "Logo area" so they would have been marked as *affected* anyway, there are still 50% of them, which increases the encoding time of B-slices. To improve run-time we wish to minimize the amount of Direct macroblocks being marked as *affected*. We introduce a hierarchical status scheme. In this scheme macroblocks could be marked *indirectly affected* with several levels of indirection. Each time a Direct macroblock is marked as *affected* due to an *affected* neighbor its *indirection level* is decreased and when it reaches zero it becomes *unaffected*, and therefore won't be re-encoded. As part of the attempt to minimize the *affected* area by adding *indirect affection levels*, the *Affected* status which was a boolean flag, is changed to an enumeration of several levels of affection. The modified algorithm is described in Algorithm 2. A possible example for assigning different indirection levels to macroblocks is presented in Fig. 5.

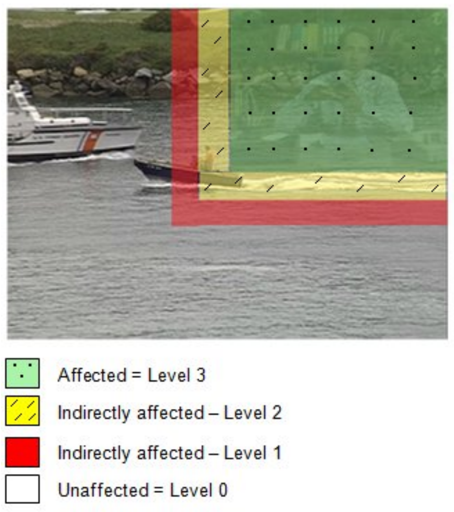

Fig. 5. Indirection Level in Affected Macroblocks Classification

Algorithm 2 Affected Macroblocks Classification with Indirection Levels

```
if (MB is Direct) and (direct mode type is Spatial) then
    get co-located blocks (A, B, C)
    if block C is not available then
        use block D instead
    end if
    for each block do
        block.affected ← get_affected(block)
        affected ← MAX(affected, block.affected)
        if affected > 0 then
            affected ← (affected - 1)
        end if
    end for
end if
```

IV. Performance

A. Run-time

To evaluate the proposed algorithm performance we compared the time required to insert new content to an encoded video sequence using our encoder and the naive approach. The tests were performed with CIF and SIF sequences and varying values were used for the transparency factor. The frame rate specified to the encoder was 30 FPS. For the results presented here, the inserted sequence was ”news” (resized to 140x90) with transparency factor $\gamma = 0.7$. The run-time testing was performed on video sequences of various lengths, and we noticed that the sequence length does not have any significant effect on run-time measurements. We therefore proceeded with measurements on short video sequences of 45 pictures. The original H.264 bitstreams were encoded so that the picture types used in encoding were ordered as follows: I-B-B-P-B-B-P... . We worked with bitrate around 500 KB/sec, varying between different video sequences. Only the first picture was coded as Intra. Adding more Intra slices would only bias the run-time results in our favor because Intra slices are very fast to encode, hence we avoided it. The logo size used in the tests was approximately 15% the size of the picture. We tested the encoder on a Linux machine, with no other tasks executed at the same time. Thus, encoding the same video several times provided consistent result. Several representative run-time measurements, performed using the naive approach and using our method are presented in Table I. The comparison between the two methods is presented graphically in Fig. 6.

We can see significant improvement in run-time for all tested modes. For certain scenarios we observed a run-time improvement of close to 90%. Although the results may vary subject to specific clip characteristics, the average improvement is significant for all encoding modes. The average time-reduction is 77% for Bi-directional predictive coding without Direct mode, 86% Temporal Direct and 75% for Spatial Direct.

B. Video Quality

To ensure that we preserve video quality we performed PSNR measurements for the different encoding modes we supported. The evaluation setup is illustrated in Fig. 7. We decoded the H.264 output bitstream of the *Guided Encoding* chain (1) and the output of the naive transcoding chain, getting two YUV sequences. We compared the quality of both outputs by calculating the PSNR between the areas that do not contain the logo in both videos. We excluded the logo area from the comparison since it is fully re-encoded in both setups and does not contribute to our comparison. We are interested in the quality of encoding in the *unaffected* area where *Guided Encoding* comes into play. The calculated PSNR for all tested encoding modes was above 45.5 dB, indicating that *Guided Encoding* does not cause a degradation in video quality.

TABLE I
RUN-TIME COMPARISON

Video Sequence	Mode	Transcoding Time (seconds)		Speed-up factor
		Naive	Guided	
mobile	No Direct	44	9	4.89
	Temporal Direct	70	9.75	7.18
	Spatial Direct	70	16	4.38
flower garden	No Direct	54	10	5.4
	Temporal Direct	81.6	10.24	7.97
	Spatial Direct	83	20	4.15
Paris	No Direct	54	16	3.38
	Temporal Direct	77	9.8	7.85
	Spatial Direct	83	22	3.77

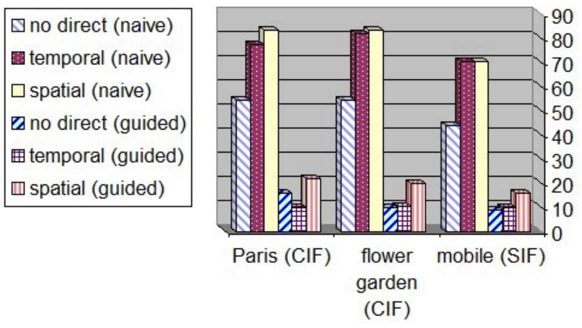

Fig. 6. Run-time comparison (time is in seconds)

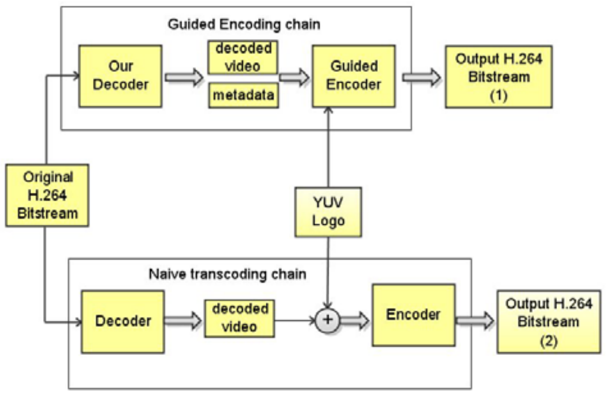

Fig. 7. Performance evaluation setup

C. *Bit-rate*

While having a significant improvement in encoding run-time, in most cases we experience an increase in bit-rate. This is due to the sub-optimality of the encoding decisions used in *Guided Encoding*, which do not consider the new inserted content. We measured the increase in bit-rate in comparison to the naive solution using multiple video clips with different characteristics. For videos with P-slices only, the increase in bit-rate was 2.5% in the worst case. In case of Bi-directional predictive encoding without Direct mode, an average increase of 30% in bit-rate was measured. Approximately the same increase was measured for Temporal Direct. For some ”busy” clips having a lot of motion (Stefan) the increase in bit-rate reached 40%. For Spatial Direct mode a much lower average increase of about 15% was measured. This result is expected since a relatively big portion of the macroblocks is marked as *affected* and enforced to be re-encoded using their original, Direct, mode. This results in relatively low bit-rate compared to other modes. Part of the increase in bit-rate might be explained by our method of testing. The naive approach bit-rate testing involves decoding the output bitstream of our decoder (which contains the logo in it) and re-encoding it with the original JVT-JM encoder. This method involves multiple applications of the de-blocking filter within the decoder and the encoder that possibly smooth the pictures and decrease the amount of information, which results in lower bit-rate. The smoothing may well cause some macroblocks to be encoded using Direct mode, which was not the case in the original video. Our encoder operates on a video that is closer to the original, having gone through less encoding and decoding stages. This of course does not explain all of the bit-rate increase, and most of it is caused, as mentioned, by encoding using parameters deduced for the original video. We should be aware that the output bit-rate is actually not controllable since the information in the inserted content may well be much denser than in the original video. To have a predictable and controlled bit-rate, rate control functionality has to be incorporated.

V. Conclusion

We developed the concept of *Guided Encoding* and demonstrated that it provides superior performance to the naive approach. We further showed that it is applicable to Main profile features, starting with simple ones, like Weighted Prediction and ending with more complex features, like Direct Mode. Furthermore, this idea seems to be applicable to other profiles as well and possibly to other codecs. By using the *Guided Encoding* method we were able to significantly decrease transcoding time. Our transcoder is on average 5 times faster (subject to specific video characteristics) than the naive transcoder. The most significant improvement was achieved for Temporal Direct encoding mode - average run-time reduction was 86%. For Bi-directional prediction without Direct macroblocks an average improvement of 77% was achieved and 75% for Spatial Direct mode. We succeeded in showing solutions that speed up encoding without loss in quality, and in that sense met an important initial requirement for the research. We also suggested a sub-optimal solution in terms of quality for Spatial Direct encoding, leaving the run-time against video quality dilemma to the user. Obtaining much better run-time we bear the penalty of an increase in bit-rate, which is quite negligible for P-slices and more significant for B-slices. As we explained, to obtain a stable output bit-rate, rate control must be applied in any case.

Acknowledgment

We wish to thank Prof. David Malah, head of the Signal and Image Processing Lab (SIPL) at the Technion for reviewing the work and providing valuable comments, Yair Moshe for sharing his knowledge in H.264 and helping the research, SIPL staff Ziva Avni and Avi Rosen for providing us with a friendly environment that was so helpful to our work and Asaf Cidon for reviewing and providing comments. We wish to acknowledge the NEGEV consortium (http://www.negev-initiative.org) for support and sponsorship of the research. We would also like to express our gratitude to Assaf Tzabari, Itai Shpak and Naama Hait who had conducted research about content insertion into MPEG-2 encoded video, Yuval Bymel and Dan Vardi for their initial research of the methods to perform content insertion in the context of H.264 Baseline Profile, Natan Goldfarb and Ori Rottenstreich whose study of the Main Profile and the JVT-JM had put us in a good starting point for our work. Finally we wish to thank Nimrod Peleg for his constant support through all the stages of our work.

References

[1] S.-F. Chang and D. G. Messerschmitt, “Compositing motion-compensated video within the network,” *MULTIMEDIA ’92. 4th IEEE ComSoc Conference on Multimedia Communications*, pp. 40–56, Apr. 1992.

[2] S.-F. Chang and D. G. Messerschmitt , “Manipulation and compositing of MC-DCT compressed video,” *IEEE Journal on Selected Areas in Communications*, vol. 13, pp. 1–11, Jan. 1995.

[3] Y. Noguchi, D. G. Messerschmitt, and S.-F. Chang, “MPEG video compositing in the compressed domain,” *ISCAS ’96., ’Connecting the World’., 1996 IEEE International Symposium on Circuits and Systems*, vol. 2, pp. 596–599, May 1996.

[4] N. Roma and L. Sousa, “Fully compressed-domain transcoder for PIP/PAP video composition,” *PCS2007*, Nov. 2007.

[5] C.-H. Li, H. Lin, C.-N. Wang, and T. Chiang, “A fast H.264-based Picture-in-Picture (PIP) transcoder,” *IEEE International Conference on Multimedia and Expo (ICME’04)*, vol. 3, pp. 1691–1694, 2004.

[6] C.-H. Li, C.-N. Wang, and T. Chiang, “A low complexity picture-in-picture transcoder for video-on-demand,” *International Conference on Wireless Networks, Communication and Mobile Computing*, vol. 2, Jun. 2005.

[7] ISO/IEC 14496-10 and ITU-T Rec. H.264, *Advanced Video Coding*, 2003.

[8] A. M. Tourapis, F. Wu, and S. Li, “Direct mode coding for bipredictive slices in the h.264 standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, pp. 119–126, Jan. 2005.

[9] I. E. Richardson, *H.264 and MPEG-4 Video Compression*. John Wiley and Sons Ltd, 2003.

[10] J. Yang, K. Won, and B. Jeon, “Motion vector coding with selection of an optimal predictive motion vector,” *Optical Engineering Letters*, vol. 48, Jan. 2009.

[11] A. M. Tourapis, A. Leontaris, K. Suhring, and G. Sullivan, *H.264/14496-10 ACV Reference Software Manual*, ISO/IEC MPEG & ITU-T VCEG, July 2009.