# H.264-BASED STREAM MORPHING WITH SCALABLE MOTION CODING

*James Macnicol, Michael Frater, John Arnold*

School of Information Technology & Electrical Engineering
University College, the University of New South Wales
Australian Defence Force Academy, Canberra

## ABSTRACT

Stream morphing is a technique that allows for two or more video bitstreams coded from the same original material with different quality settings to be simulcast in an efficient way. This paper describes our second generation stream morphing system that is based on the H.264 standard rather than MPEG-4 as used in our original implementation [2]. While the basic operation of stream morphing remains largely unchanged for H.264, a number of enhancements have been made. Firstly, enhancement layers are coded using the same low-complexity, adaptive arithmetic coder utilized by H.264 itself. Furthermore, a scalable motion model has been added which allows for the use of very low bit rates in the base layer compared to the previous approach that used the same motion vectors and macroblock partitioning modes in all layers.

## 1. INTRODUCTION

The recently ratified H.264 video coding standard is the latest in a line of hybrid motion-compensated prediction/block transform video codecs that have been internationally standardized. While the basic operation of the new codec is substantially similar to previous systems such as H.263, MPEG-2 and MPEG-4, it adds many new features, especially in terms of improved temporal prediction techniques [1]. Stream morphing is a technique for exploiting redundancy between sets of bitstreams generated from the same source material using encoders of this type. This has application both in scalable video compression and content distribution. In this paper we describe the design and performance of a stream morphing system based on the H.264 standard and outline the changes that were made from our first generation system that was built around MPEG-4 [2, 3].

This paper is organized as follows. Section 2 provides a brief overview of the stream morphing process as previously described in [2, 3] in the context of MPEG-4. Section 3 describes the differences between the stream morphing processes that we have used for MPEG-4 and H.264, most of which deal with differences in the original standards. Section 4 shows results from a 5 layer SNR scalable H.264 stream morphing system that we have constructed. Finally, Section 5 draw conclusions from the paper.

## 2. STREAM MORPHING OVERVIEW

Consider two bitstreams $A$ and $B$ coded from the same source material using one of the hybrid block transform/motion-compensated prediction codecs such as H.263/H.264/MPEG-4 etc. Intuitively, we would expect there to be significant redundancy between $A$ and $B$, even if they were coded at different quality levels: the motion fields encoded in the two bitstreams were derived from the same original source, areas of activity (i.e. where non-zero transform coefficients are found) will coincide, the spectral content of the two will be similar (so long as the framestores in the two systems are sufficiently well matched) and so on. Given a low-quality bitstream $A$ and some appropriate additional information $S$, we can generate a higher-quality bitstream $B$. The size of $S$ should be significantly smaller than $B$ if the redundancy between $A$ and $B$ is successfully exploited. Therefore, provision of both $A$ and $S$ is equivalent to the simulcast of $A$ and $B$, the amount of data involved has been reduced at the expense of some additional computation required to recover $B$. This process of transforming one bitstream into another is known as *stream morphing*; another name for the effect this produces is *virtual simulcast*. Stream morphing can be used as a scalable video coding system: $A$ is the base layer which should always be available, the stream $S$ is the enhancement layer which, if available, provides for a higher quality service since $B$ can then be recovered. Note that, unlike many scalable coding schemes, the decoder for stream morphing can be considered to be a pre-processing stage (where the bitstream with the highest available quality is recovered) followed by an unmodified single-layer decoder. In this way, single-layer decoder designs can be re-used to provide the bulk of the design of a scalable decoder based on the stream morphing concept. A second application for stream morphing is content delivery networks where multiple descriptions of the one video sequence coded at different bit rates is required (e.g. for onward transmission to multiple clients requiring different bit rates). In this case,

0-7803-8554-3/04/$20.00 

stream morphing allows for sets of bitstreams to be transmitted across the network in a more compact form rather than simply simulcasting them.

Figure 1 provides a pictorial representation of the stream morphing process. We can consider the two input bitstreams $A$ and $B$ to be a sequence of symbols that code primitive elements, e.g coded block pattern elements or location and value of non-zero transform coefficients. The *delta* bitstream $S$ has one symbol for each base layer symbol describing how the symbol has changed in going from $A$ to $B$ as well as defining any additional symbols that were not present in $A$. The number of symbols in $S$ is very large but their individual entropies are low since many symbols do not change between layers. For this reason $S$ must be generated using an arithmetic coder (or similar), variable length (Huffman) coding cannot be efficiently utilized for such bitstreams.

Stream morphing as described can be easily applied to systems with three or more layers. In such cases one delta bitstream is generated "between" each pair of input streams. For scalable systems with many layers it is also possible to recover the original single-layer stream in one of the intermediate layers and to then form another scalable system using the remaining higher enhancement layers unchanged. This is similar to transcaling [4] but with the advantage that the new base layer service is much easier to generate (does not require transcoding).

## 3. CHANGES FOR H.264

The bulk of the stream morphing process as sketched in Section 2 and detailed in [2, 3] is retained for H.264. Listed below are the major changes that were made, either to deal with issues specific to the newer standard or to improve upon design decisions made for the original MPEG-4 system:

### 3.1. Scalable Motion Coding

MPEG-4 supports only a relatively small number of partitioning modes for inter coded macroblocks with either one or four motion vectors per macroblock in P frames and one or two vectors in B frames. H.264 has a much richer motion model with up to a maximum of 32 motion vectors per macroblock, two (forwards and backwards) for each 4x4 block. In the standard (single-layer) case, as the video quality is increased so does the use of smaller partitions as finer details are resolved and small-scale motions can be identified. For our MPEG-4 stream morphing system the same partitioning scheme was always used for coincident macroblocks in all layers. This resulted in a small increase in typical base layer bit rates due to the use of 8x8 partitioning in P frames that is useful in the top-most layer but not the base layer. This did not, however, affect the overall scalability of the system. For highly scalable H.264, however, the use of the top layer partitioning in the base layer causes a very large increase in base layer bit rate as shall be demonstrated in the experimental results to be shown in Section 4.

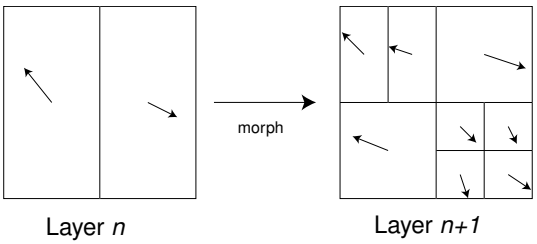


**Fig. 2**. Scalable motion coding

A solution to this problem is to allow the partitioning mode to change between layers so that large partitions using few motion vectors are used in lower layers with those partitions being sub-divided as the video quality increases. Figure 2 shows an example of this process where one layer uses the 8x16 partitioning and the higher layer uses a more complex set of partitions. The delta bitstream syntax for H.264 has additional symbols for coding both the change in macroblock partitioning mode between layers and the coding of the additional motion vectors (which are predicted either from the layer below or using the standard differential prediction in the same layer, whichever is more efficient). Changing a motion vector between layers alters the spectral content of the difference signal in the upper layer and so morphing of quantized texture is not effective. For this reason macroblock mode changes are only allowed where there are no non-zero quantized coefficients in the lower layer.

### 3.2. Intra Coefficient Coding

Intra coefficient prediction in MPEG-4 is performed in the DCT domain and as such it is easy to recover the full value of each coefficient (to within the error introduced by quantization). In the SNR scalable case we know that the quantization bins in all layers must overlap since the original DCT coefficient before quantization must be the same in all layers. Our original MPEG-4 stream morphing implementation used the quantization bin overlap in lower layers to assist with the coding of coefficients in higher layers [2].

H.264, however, predicts the pixel domain texture of intra coded macroblocks in the spatial domain, prior to the transform and quantization being applied. It is therefore not possible to compare the quantization bins in each layer without fully decoding the surrounding parts of the frame. Although this is possible, this adds significantly to decoder complexity and is not in the spirit of the stream morphing concept which seeks to do all processing in the transform domain. For this reason, non-zero intra coefficients for H.264 are treated in the same way as those from inter

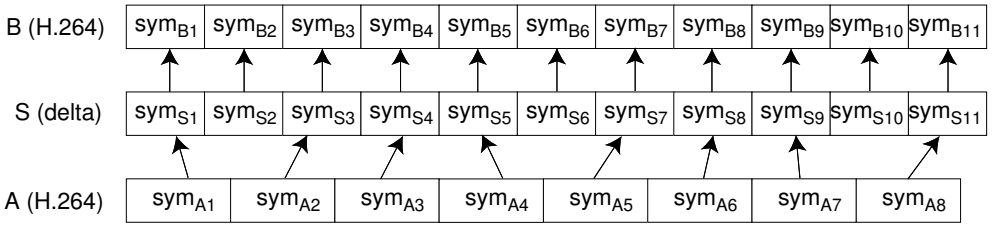


**Fig. 1**. Stream morphing process

macroblocks. We do not believe this has a detrimental impact on performance, indeed the improved H.264 intra prediction scheme [1] reduces the energy of the signal to be coded/morphed.

### 3.3. Arithmetic Coding

Our MPEG-4 stream morphing system used the fixed (i.e. non-adaptive) range coder for coding of the delta bitstreams. The statistical properties of the symbols being coded in the delta bitstreams are highly dependent on the quality difference between the two bitstreams concerned; if they are very close together then almost all symbols in the original bitstreams are identical. Conversely, if the quality difference is large then many changes are required. To use a non-adaptive arithmetic coder in this scenario required the use of multiple probability models that were selected using a layer spacing heuristic. This is an ad-hoc solution which adds complexity not only in computing the layer spacing metric but also in storage required for the additional probability models. The range coder itself is also computationally complex, requiring integer multiplication and in some cases division. Both of these issues have been corrected for H.264 by using the low-complexity (multiplication- and division-free) adaptive arithmetic coder defined in the H.264 standard.

### 3.4. Implementation Re-Use

Our MPEG-4 stream morphing implementation was written from scratch without using any code from an existing MPEG-4 system. One goal for H.264 was to re-use as much code as possible from the reference implementation [5]. Generating a delta bitstream requires first reading all the information (motion vectors, locations and values of non-zero quantized transform coefficients etc.) from the input bitstreams, followed by the morphing process itself which codes the differences between the bitstreams. The bitstream reading code was created by modifying the reference decoder and removing all (unnecessary) parts that operate on pixel domain information. For recovering bitstreams the base layer needs to be read in the same way, the morphing information is then applied and the resulting bitstream(s) written out. This writing process can likewise be performed by the reference encoder which has had its pixel domain operations removed and which receives the contents of each macroblock directly from the output of the inverse morphing process. Alternatively, rather than write out the bitstreams it can be decoded immediately, in which case the pixel domain operations that were removed from the reference decoder to create the bitstream reader can be used. As noted above, reading and writing of the delta bitstreams uses the H.264 arithmetic coder so this code can also be re-used.

### 3.5. Other Features

In addition to SNR scalability, temporal scalability has also been implemented along the lines of the MPEG-4 technique using B frames as described in [6]. Temporally-coincident frames are coded in the same way as for SNR scalability, additional B frames are then provided and passed through the delta bitstream in their original form.

Since each layer in a stream morphing system corresponds to a single-layer bitstream, H.264's notion of SP/SI switching frames can be used to move between layers in a scalable system without having to wait for the next I frame. Bitstream switching in standard H.264 is equivalent to layer switching in stream morphing.

## 4. RESULTS & DISCUSSION

Figures 3 and 4 show the performance of 5 layer SNR scalable systems with the low-motion "Mother & Daughter" and high-motion "Foreman" sequences respectively. In both cases results with and without scalable motion coding are shown. It can be seen that the use of the scalable motion coding cuts the bit rate in the base layer by around 50%. This comes at the cost of increased overhead associated with updating the motion model in one or more enhancement layers. The non-scalable results use slightly fewer bits when all layers are available since it is clearly more efficient to code the motion information once rather than progressively. For these systems stream morphing has achieve the same effect as simulcast with a bit rate saving of around 30%.

While these results demonstrate the feasibility of H.264-based stream morphing, it is important to note that there is significant scope for optimizing the single-layer bitstreams

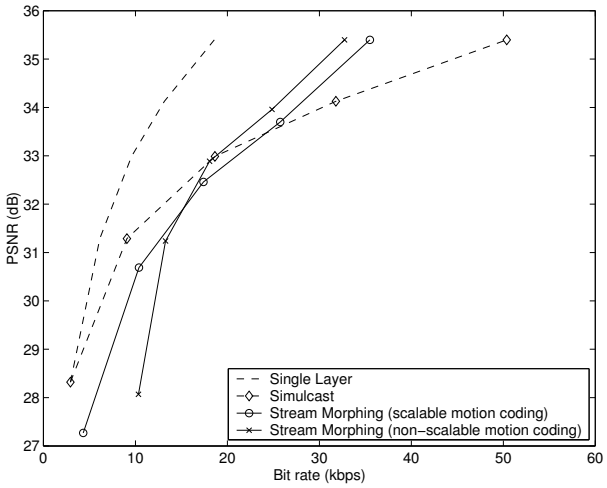


**Fig. 3**. 5 layer SNR scalability results for "Mother & Daughter" sequence (QCIF, 10fps)

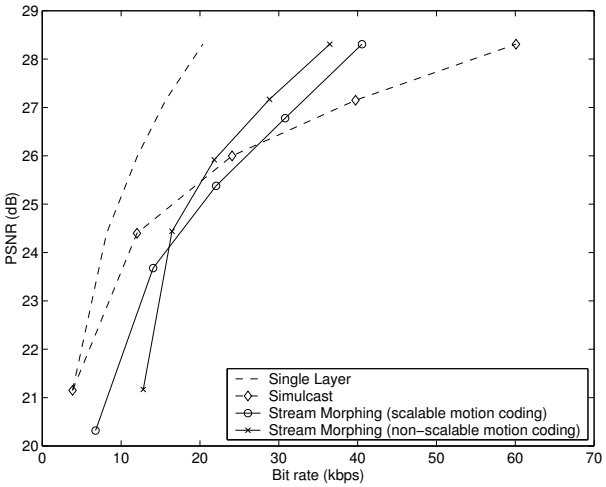


**Fig. 4**. 5 layer SNR scalability results for "Foreman" sequence (QCIF, 10fps)

that are input to the stream morphing process. We expect that such optimizations (which are the subject of current research) will provide large efficiency gains. At present the single-layer bitstreams input into the morphing process are generated using the standard H.264 reference encoder which has been modified only to match partitioning modes and motion vectors as detailed in Section 3.1. For the scalable case, the content in the previous layer has a large impact on the "cost" of coding entities in the current layer. For example, there is almost zero cost in coding a non-zero coefficient if there is a coincident non-zero coefficient in the previous layer. This is in contrast to the single layer case where non-zero coefficients are scanned and their values coded, operations that require on average many bits per coefficient. An optimized encoder would consider the effect of coding decisions across all layers, and potentially multiple frames. We believe this may also make spatial scalable stream morphing practical; experiments thus far have shown that framestore mismatch between layers destroys the correspondence between the motion-compensated prediction signals that makes stream morphing possible. By considering all layers over many frames it should be possible to control this mismatch to produce better results.

## 5. CONCLUSION

This paper has demonstrated that the stream morphing concept can be successfully applied to H.264. A number of enhancements have been made from our original MPEG-4 implementation, most notably the use of scalable motion coding. This has allowed for the use of much lower base layer bit rates at the expense of a small increase in bit rate when many layers are present. A significant saving has been shown when simulcasting a five layer system and this has been done without any encoder optimization which we believe will provide even better performance in the future.

## 6. REFERENCES

[1] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, July 2003.

[2] J. Macnicol, M. Frater, and J Arnold, "Scalable Video Coding By Stream Morphing," in *IEEE International Conference on Image Processing (ICIP)*, Rochester, NY, USA, September 2002, vol. 3, pp. 733–736.

[3] J. Macnicol, J. Arnold, and M. Frater, "Scalable Video Coding By Stream Morphing," *IEEE Transactions on Circuits and Systems for Video Technology*, 2004, (to appear).

[4] H. Radha, "TranScaling: A Video Coding and Multicasting Framework for Wireless IP Multimedia Services," in *Proc. ACM SIGMOBILE Workshop on Wireless Mobile Multimedia*, July 2001, pp. 13–23.

[5] K. Sühring, "H.264/AVC Software Coordination," http://bs.hhi.de/˜suehring/tml/.

[6] J. Macnicol, M. Frater, and J Arnold, "Stream Morphing Approaches to Temporal Scalable Video Coding," in *IEEE International Conference on Image Processing (ICIP)*, Barcelona, Spain, September 2003, vol. 3, pp. 637–640.