SEGMENT-BASED MOTION ESTIMATION USING A BLOCK-BASED ENGINE

Patrick Meuwissen[*,1], *Ramanathan Sethuraman*[1], *Fabian Ernst*[1], *Harm Peters*[1] *and Rafael Peset Llopis*[2]

[1]Philips Research Laboratories, Prof. Holstlaan 4 (WDC-31), 5656 AA Eindhoven, the Netherlands
[2]Philips Consumer Electronics, P.O. Box 80002 (SFJ-644), 5600 JB Eindhoven, the Netherlands
{patrick.meuwissen, ramanathan.sethuraman, fabian.ernst, harm.peters, rafael.peset.llopis}@philips.com
[*] phone: +31-40-2744523; fax: +31-40-2744639

ABSTRACT

Motion estimation is a key function in scan rate conversion, advanced picture quality improvement, 2D-to-3D content conversion, and many other video processing steps. For hardware efficiency reasons, most motion estimation implementations are block-based. As object boundaries commonly do not coincide with block boundaries, artifacts may be visible at object boundaries using the block-based approach. Motion estimation for irregular shapes, such as image segments, can accurately track motion boundaries, but a straightforward translation of block-based motion estimation algorithms to segment-based ones leads to inefficient hardware implementations. Therefore, this paper proposes a modified segment-based motion estimation algorithm utilizing the efficiency of block-based processing. We demonstrates an efficient very large instruction word (VLIW) application-specific instruction-set processor (ASIP) implementation of this algorithm.

1. INTRODUCTION

Motion estimation is a key technology underlying many video processing applications such as video coding [1,3], scan-rate upconversion [2], motion-compensated deinterlacing [4] and 2D-to-3D video conversion [5, 6]. Block-based motion estimation algorithms, although suffering from only block-accurate motion vectors, are popular due to ease of implementation (for achieving real-time performance) [3]. On the other hand, image segmentation is seen as a vital ingredient for content-based video processing, for instance in the domain of content-based retrieval (e.g. MPEG7, [1]), object tracking [7] and 2D-to-3D video conversion [6]. Current coding standards such as H.264 [8] use variable block sizes and shapes to, amongst others, reduce block-related artifacts; this can be considered as an intermediate between block-based approaches and full segment-based approaches.

Segment-based motion estimation (SBME) replaces the fixed blocks of traditional motion estimation algorithms with segments having arbitrary shapes and sizes. As motion vectors are now assigned to segments instead of blocks, it allows for motion discontinuities at their true, pixel-accurate, positions in the image (see Figure 2). Segment-based approaches (e.g. [9]) have shown the potential for highly accurate motions in a benchmark test [10]. Furthermore segments, as content-dependent entities, can be tracked over multiple frames of a video sequence. This functionality, that blocks can not provide, is useful for temporal filtering or other multi-frame processing.

However, from a hardware implementation point of view, SBME has a significant disadvantage: Since segments can be of arbitrary shape and size, a straightforward implementation of a SBME algorithm will either suffer from inefficient use of data memory bandwidth or suffer from irregular data addressing, which also results in poor bandwidth utilization of modern memories that are optimized for burst accesses (e.g. SDRAMs). Figure 1 illustrates this in detail. This disadvantage often precludes the use of SBME in real-time video applications. Here, efficient memory bandwidth utilization is a must because the size of frame buffers typically requires the use of off-chip memories, which have a limited data bandwidth compared to the processing capabilities (and thus the data bandwidth requirements) of logic chips. In this paper, we propose a SBME algorithm that applies block-based processing to calculate segment-based motion vectors. Thus, this modified SBME algorithm achieves an efficient use of data memory bandwidth, without sacrificing the regularity of segment data addresses. Further, the modified algorithm exhibits massive parallelism which also facilitates real-time implementations. The parallelism and the block-based memory addressing are exploited by our VLIW ASIP implementation, which has several Application Specific Units (ASUs) for accelerating inner kernels of the algorithm in a SIMD-style fashion and for buffering blocks of data that are used multiple times, thus reducing the bandwidth requirements of the data memory.

For SBME, we only require from the segmentation that no motion discontinuity occurs inside a segment [5, 6]. As the motion is determined on a segment basis, this is a chicken-and-egg problem. However, a color or luminance segmentation (see Figure 2) in general fulfills this requirement. A vast amount of algorithms exist for color segmentation (e.g. [11, 12]). In this paper, we focus on the SBME algorithm itself, assuming that an appropriate segmentation is provided by any (e.g. one of the above) segmentation algorithm.

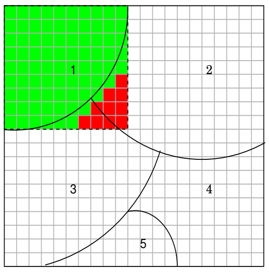

Figure 1: Illustration of the inefficiency of SBME algorithm: If all blocks within the bounding box are fetched for each segment (green and red blocks for segment 1), this results in simple segment data addresses for fetching the required blocks (and pixels) of the segment; however, fetching of non-segment blocks (red blocks for segment 1) results in inefficient use of the data memory bandwidth. Alternatively, if only those blocks that are part of a segment are fetched (green blocks for segment 1) irregularity of the segment data addresses leads to computational overhead as well as inefficient use of data memory bandwidth.

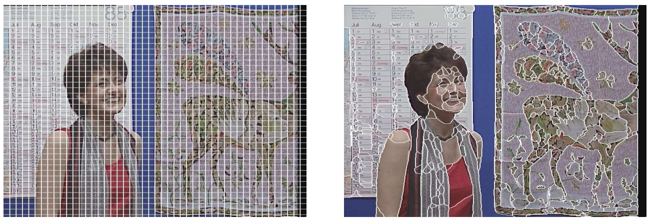

Figure 2: Left: Block-grid overlay of a frame in Renata video sequence. Right: Segmentation of the same frame. If segments instead of blocks are used for motion estimation, motion boundaries can be obtained with pixel accuracy.

The remainder of the paper is organized as follows. In Section 2 a straightforward translation from a block-based motion estimation algorithm to a segment-based motion estimation algorithm is presented. Section 3 discusses modifications which enable efficient hardware implementation. Section 4 shows an implementation of the modified SBME algorithm which can also perform traditional block-based motion estimation. We conclude in Section 5.

2. SEGMENT-BASED MOTION ESTIMATION

Motion estimation algorithms find for a given domain D such as a block its best matching position in another (previous or next) frame by minimizing a so-called *match penalty* (MP):

$$\mathbf{m}_{\text{opt}} = \text{argmin}_{\mathbf{m}\in\mathrm{M}} MP(\mathbf{m}) = \sum_{\mathbf{p}\in D} w(\mathbf{p})|I_1(\mathbf{p}+\mathbf{m}) - I_0(\mathbf{p})|, \quad (1)$$

where $\mathbf{m}$ is a motion vector (MV), M the set of MVs which are evaluated, $\mathbf{p}$ denotes pixels, w is a weight for each pixel, I_0 is the current frame and I_1 is the frame to be matched to. Different motion estimation algorithms differ in the choices of M (e.g., a full search algorithm tests all MVs in a certain range), the weights w (e.g., skipping every second pixel for efficiency reasons) and the minimization strategy.

A very successful motion estimator is the 3D recursive search (3DRS) block-based motion estimation algorithm [2, 13], which is fast, accurate, proven in hardware and currently "best in class" for applications where the true motion field is required. Its main distinguishing feature lies in the choice for M. The blocks are processed in scan-line order, and for each block under consideration M is very limited: Two MVs from blocks in the neighbourhood which already have been processed for the current frame (for spatial smoothness), one MV of the previous frame from a block in the neighbourhood which has not been processed yet (for temporal smoothness), $\mathbf{m} = \mathbf{0}$ (to handle the 'no-motion' case efficiently) and two MVs which are equal to one of the previously mentioned MVs with a small random perturbation (for handling motion changes).

A straightforward SBME algorithm [6] directly translates this philosophy to segments, where the domain D now consists of all pixels in a segment instead of all pixels in a block, and the set of candidates M consists of MVs of neighbouring segments. However, it suffers from the inefficiencies discussed in the Introduction and shown in Figure 1. Further, while in the 3DRS algorithm each block is visited only once, for accuracy reasons it may be beneficial to do multiple iterations over all segments. The total computational effort for SBME is proportional to the number of iterations times the number of candidate vectors per segment.

Key design choices for the SBME algorithm are: Which neighbours should provide candidate vectors, and how many neighbours should be used? Note that, unlike the block-based case, there is no a-priori knowledge of the number of neighbouring segments and their geometry. To obtain the most relevant neighbours - the ones most likely to provide a suitable MV - the neighbours are sorted first based on color difference to the current segment, as similarly colored segments are more likely to belong to the same object and hence have a similar motion. The dashed lines in Figure 3 show the typical convergence behaviour as a function of the number of candidates for different video sequences; it implies that selecting maximal 5 neighbours is sufficient. Statistics on many sequences show that this corresponds to the average number of neighbours per segment, using our segmentation method [6]. An impression of the sequences and their segmentation is shown in Figure 4.

The SBME algorithm then becomes:

- **Step 1** ***(choice of candidate MVs)*****:** select candidate MVs for each segment of the current frame;
- **Step 2** ***(motion estimation kernel)*****:** evaluate MP (see equation (1)) for each segment and for each candidate MV of a segment;
- **Step 3** ***(convergence check)*****:** perform global convergence check; go to Step 1 if convergence fails; otherwise, provide the best matching MV per segment. Convergence has been reached when the average match penalty for all segments does not improve anymore. In practice the number of iterations should be bounded to a maximum in order to guarantee real-time performance.

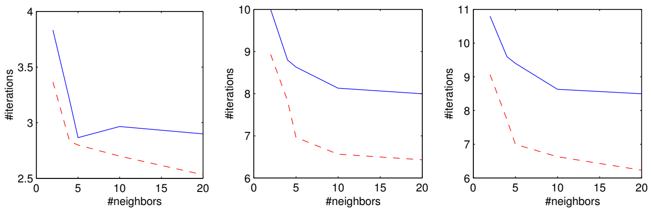

Figure 3: Required average number of iterations until convergence for 30 frames of Renata, Obst and Flikken sequences (left to right) as function of used number of neighbours for the parallel SBME algorithm (solid line) compared to the sequential one (dashed line). Due to the simple constant translational motion in the Renata sequence, convergence is fast.

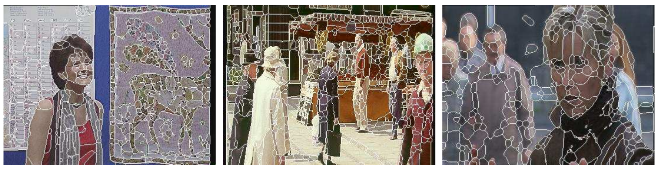
Figure 4: Example frames of the sequences and their segmentation which are referred to in this paper (Renata, Obst and Flikken).

In Step 1, the candidate MVs are: The current segment MV, the current segment MV with a random update; and 5 neighbouring segment MVs with 50% of these vectors having random updates. In the first iteration, also $\mathbf{m} = 0$ is tried.

3. MODIFICATIONS FOR IMPLEMENTATION EFFICIENCY

This section discusses two modifications which, together, allow for an efficient hardware implementation: Segment-based parallellization and block-based computations. They allow to address the inefficient use of data memory bandwidth explained in the Introduction.

Parallelization. One of the key features of the SBME algorithm is that, as in the original 3DRS algorithm, the chosen best candidate MV for the segment under consideration is a candidate MV for some neighbouring segments in the same iteration. This key feature enables the convergence of this algorithm in relatively few iterations (typically 4–6 iterations; see Figure 3). However, it also makes the algorithm inherently sequential. Often, parallel implementations are a must to meet real-time requirements in video applications. For the block-based 3DRS this is not a major issue, especially not in a streaming environment, but due to the multiple iterations for SBME and the irregular shape of the segments this issue needs to be addressed here.

To enable parallelization, we do not take the MVs of neighbouring segments from the *current* iteration as candidates, but from the *previous* iteration. Since this removes the intra-iteration dependency between segments of a frame, parallel evaluation of the match penalties for all segments is possible. Figure 3 shows the convergence behaviour for the parallel approach compared to the sequential approach discussed previously. The number of *iterations* until convergence increases by 1 or 2, but the number of *candidate MVs* per segment need not be increased. Because the modified algorithm is easily parallelizable (up to all segments at once), a significant gain of the overall computation time can be achieved. For example, the use of 4 processing elements already leads to a gain of roughly a factor of 3 compared to the original SBME algorithm (the other factor of 1.25 being lost in the additional required iterations).

Efficient use of data memory bandwidth and regular addressing. As mentioned before, this is a main requirement for efficient hardware implementation. Note that in each iteration, for all segments and all their candidate MVs the match penalty (1) needs to be evaluated. As there are no intra-frame dependencies in the choice of the MVs for a segment, the order in which the absolute difference of pixel pairs is computed is irrelevant. This allows the use of block-based kernels for SBME as follows: We evaluate all pixels of each block of a frame and evaluate all blocks successively. This approach is typical to conventional block-based motion estimation implementations. However, in this case a single block B may overlay multiple segments $S_1, S_2, \ldots, S_n$. Thus, for pixels in B belonging to S_1, different candidate MVs may have to be evaluated than for pixels belonging to S_2. This is, however, straightforwardly possible by evaluating expression (1) n times per block with every time a different set of candidate MVs, and setting weight $w(\mathbf{p}) = 0$ the i'th time for all pixels not belonging to segment S_i. Again, since processing each block in an iteration is independent of the results of other blocks, massively parallel implementations can be realized. The results of the MP evaluation for all sub-segments (the part of a segment in a single block) are accumulated for each candidate MV and for each segment of the frame, as segments typically overlay multiple blocks. With this approach, all pixels belonging to a block are effectively used; the price for this is an accumulation buffer for the MP for all candidate vectors of all segments.

Since processing resources are typically restricted, we need to make sure that there is a reasonable upper bound to n, the number of times a block is processed (e.g., 4 times). As the upper bound of n for a block can be quite high (worst case: number of pixels in the block), we have to do a pre-processing step:

- **Step 0 *(segmentation refinement)*:** lay a block (say, 16-by-16 pixels) grid on the segmented current video frame (see Figure 1); ensure that each block contains contributions from not more than four segments (note that this depends on the chosen block-size and segmentation method).

In practice, we select for each block B those four segments which have the largest sub-segments within B, and set $w(\mathbf{p}) = 0$ for the pixels of all other segments in B. While this may seem a significant modification, analyzing many segmented frames revealed however that in our implementation, few blocks in a frame have contributions from more than four segments; on average 0.2-0.3% of pixels are excluded.

The above steps of the modified SBME algorithm ensure the following:

- all pixels of a block accessed from the data memory are used by the motion estimation process and hence this scheme ensures efficient use of data memory bandwidth;
- accesses to the data memory are block oriented, again ensuring efficient usage of the bandwidth of this (typically off-chip) memory.

The penalty of more iterations to convergence for the modified SBME algorithm is far out-weighed by the benefits of the modifications. In the following section we present a processor implementation of the modified SBME algorithm.

4. IMPLEMENTATION

The VLIW ASIP. Figure 5 depicts an application specific instruction-set processor (ASIP) based on a very large instruction word (VLIW) processor architecture template. This processor implements the modified SBME algorithm. Apart from several general purpose functional units like an arithmetic-logic-unit (ALU), a multiplier (MULT) and an address-computation-unit (ACU), this ASIP also contains a number of so-called Application Specific Units (ASUs), tailored for accelerating the inner kernels of video signal processing algorithms in general, and SBME in particular. For SBME, we use a processing block size of 16×16 pixels, which we refer to as macro-blocks (MB) to avoid confusion with the 8×8 blocks stored in memory. However, other block sizes are possible as well. The granularity for synchronisation between the system controller and the ASIP coprocessor has been chosen as one line of MBs, which is a good trade-off between memory sizes and synchronization overhead.

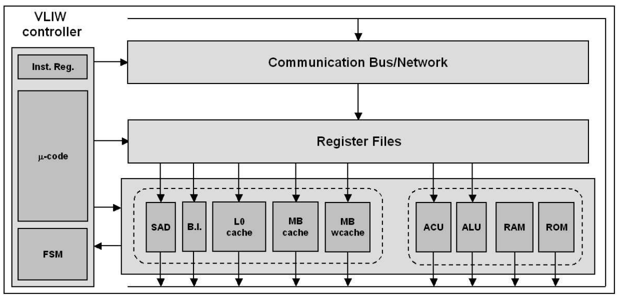

Figure 5: An application specific instruction-set processor for SBME. See main text for details and abbreviations.

We discuss the implementation and the ASUs based on the following pseudo code of the modified SBME algorithm:

```
For each block of block line:
  Fill macroblock cache
  Fill MB wcache
  Fill L0 cache with complete search-area
  For all sub-segments:
    For all vectors:
      Read candidate vector
      Initialize Application Specific Units
      Read first block line from L0 cache
      Supply first block line to BI
      For 16 block lines:
        Read block line from macroblock cache  : R
        Read weights from MB wcache            : W
        Read block line from L0 cache
        Supply block line from L0 cache to BI
        Read interpolated line from BI         : P
        Supply results R, W, P to SAD unit
      Read block-SAD result from SAD unit
      Store block-SAD result and accumulated weight
```

For clarity and compactness of exposure, bilinear interpolation (BI) is not discussed in this paper.

The outer loop of the pseudo-code iterates over all the macroblocks in a line. Within this loop body, all three caches (L0 cache, MB cache and MB wcache) are filled from the off-chip memory. All inner loops re-use the same cached data. The loop over all vectors (maximum 8 in our implementation) is similar to block-based algorithms. We now discuss the specific ASUs:

Macroblock cache (MB cache): This cache stores the pixel data of the macro-block of 16 × 16 pixels from the *current* frame (I_0).

Macroblock weight cache (MB wcache): This block is very similar to the MB cache. It stores sub-segment mask data and pixel weight data for the current MB. The sub-segment mask data consists of 2 bits per pixel, indicating to which of the up to 4 sub-segments within the MB that pixel belongs. During the loop which iterates over the (up to) 4 sub-segments within a macro-block, the current sub-segment ID (*cssid*) is fed to the MB wcache to generate the binary mask w_{mask} required by the SAD unit. The pixel weight data $w(\mathbf{p})$ is currently not used, but can be handled transparently through the MB wcache as well.

Level 0 cache (L0 cache): This cache stores the entire search area required by the algorithm. The size of the search area depends on the maximum allowed length of the MVs. Its purpose is to allow a line, consisting of 16 pixels, at an arbitrary position within the search area, to be retrieved very efficiently without the need to access the data memory. The current L0 cache size is limited to 48×32 pixels.

Sum of Absolute Differences (SAD): This ASU calculates the sum of absolute differences[1] of up to 16 pixels in parallel. The corresponding pixels are provided by the MB cache and the L0 cache, respectively. What separates our SAD unit from designs for block-based algorithms, is that it has a third input which is supplied by the MB wcache. It is used to specify which pixels are part of the current

[1]We use the term SAD to denote any sum of absolute differences of pixel values. The term MP is reserved for the SAD of a complete segment.

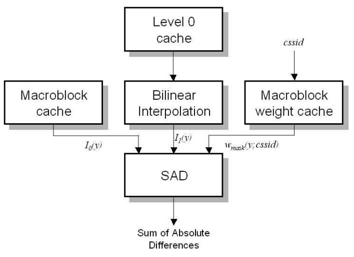

Figure 6: Flow chart of the pixel processing performed by the ASUs. Bilinear interpolation is not discussed in this paper and can be considered a 'no-operation' here.

sub-segment (indicated by *cssid*), by means of a binary mask w_{mask}. By sequentially processing consecutive lines, the SAD for an entire macro-block is calculated:

$$SAD(cssid) = \sum_{y=0}^{15} \sum_{x=0}^{15} |I_0(x,y) - I_1(x,y)| \cdot w_{mask}(x,y;cssid)$$

Figure 6 depicts the flow of data between the various ASUs. All ASUs operate with a throughput of 16 pixels per cycle, but because of pipelining (both inside and between ASUs) the loop takes slightly more than 16 cycles to execute. Note that when bilinear interpolation (BI) is added, it will process the output of the L0 cache before it is sent to the SAD unit. This increases the latency by two cycles and requires an increase of the line size in the L0 cache to 17 pixels. Once BI is included, MVs can be calculated with quarter-pixel accuracy instead of the current pixel accuracy.

After the SAD has been calculated in the inner loop, it is written into buffers to be sent back to the system controller. Note that selection of the best candidate cannot be performed by the ASIP itself, because the best candidate for a sub-segment is not necessarily the best candidate for the entire segment. The SADs of all sub-segments first have to be integrated by the system controller, to get the MPs for the entire segment. These MPs are then used to select the best candidate for the entire segment.

Results. The SBME ASIP was designed using the A|RT toolset, now marketed as OptimoDE by ARM Ltd. [14]. Table 1 summarizes the synthesis results of the design. For this simulation, a CIF input sequence was used. The maximum number of iterations per frame was set to 12. Caching was applied to exclude candidate MVs that had already been tried in previous iterations. This reduced the amount of calculations required by approximately 75%.

IC technology:	0.18μm
Conditions:	25°C, 1.8V
Area:	2.48 mm^2
Clock frequency:	100 MHz
Power:	70.5 mW
Execution time:	55 ms/frame

Table 1: Synthesis and netlist-level power simulation results.

From these results we can conclude that the current design is capable of handling CIF video sequences at approximately 18 frames per second. With a number of additional optimizations we expect this can be improved to about 60 frames per second.

5. CONCLUSIONS

Motion estimation is a key function in many video applications. Unlike the block-based motion-estimation algorithm where the motion vectors are block-accurate, segment-based motion estimation (SBME) provides higher accuracy (especially at object/occlusion boundaries), but the irregular shapes and sizes of segments are disadvantageous for a hardware implementation.

In this paper, we have proposed a modified SBME algorithm that results in an efficient use of data memory bandwidth and exhibits massive parallelism which is essential for real-time implementations. Furthermore, we presented an implementation of the modified SBME algorithm, which can also perform traditional block-based motion estimation. This implementation is based on a VLIW ASIP, that contains several Application Specific Units to accelerate inner kernels of the modified SBME algorithm, and reduce the data bandwidth requirements of the (typically off-chip) data memory used for frame buffers.

This indicates that with slight modifications, an irregular and seemingly hardware-unfriendly algorithm such as SBME can still be implemented efficiently, by making use of a block-based engine. As such, the hardware implementability does not have to be a show-stopper for the introduction of advanced algorithms, such as advanced image enhancement and 2D-to-3D conversion algorithms, to mainstream video processing.

Acknowledgments

We would like to acknowledge the valuable contributions of our colleagues Aleksandar Beric, Carlos Alba Pinto, Jef van Meerbergen, Srinivasan Balakrishnan and Ghiath Alkadi.

REFERENCES

[1] (2005) MPEG home page. Moving Picture Experts Group. [Online]. Available: http://www.chiariglione.org/mpeg/

[2] G. de Haan and P. Biezen, "Sub-pixel motion estimation with 3D recursive search block matching," *Signal Processing: Image Communication*, vol. 6, pp. 229–239, 1994.

[3] A. van der Werf, R. Kleihorst, E. Waterlander, M. Verstraelen, T. Friedrich, F. Bruls, and R. Takken, "I.McIC: A single-chip MPEG-2 video encoder for storage," *IEEE Solid-State Circuits*, vol. 32, no. 11, pp. 1817–1823, Nov. 1997.

[4] C. Ciuhu and G. de Haan, "A 2-dimensional generalised sampling theory and application to de-interlacing" In *Proc. VCIP 2004*, San Jose, 2004, pp. 700–711.

[5] H. Tao, H. Sawhney, and R. Kumar, "A global matching framework for stereo computation," in *Proc. IEEE ICCV*, vol. 1, Vancouver, Canada, July 2001, pp. 532–539.

[6] F. Ernst, P. Wilinski, and K. van Overveld, "Dense structure-from-motion: an approach based on segment matching," in *Proc. ECCV*, vol. II. Copenhagen, Denmark: Springer, May 2002, pp. 217–231.

[7] C. Erdem, A. Tekalp, and B. Sankur, "Video object tracking with feedback of performance measures," *IEEE CSVT*, vol. 13, pp. 310–324, 2003.

[8] T. Wiegand, G.J. Sullivan, G. Bjontegaard and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE CSVT*, vol. 13, pp. 560–576, 2003.

[9] L. Hong and G. Chen. "Segment-based stereo matching using graph cuts". *Proc. IEEE CVPR*, Washington DC, 2004.

[10] (2005) Stereo Benchmark Page [Online]. Available: http://www.middlebury.edu/stereo

[11] L. Vincent and P. Soille, "Watersheds in digital spaces: An efficient algorithm based on immersion simulations," *IEEE PAMI*, vol. 13, pp. 583–598, 1991.

[12] L. Salgado, N. Garcia, J. Menendez, and E. Rendon, "Efficient image segmentation for region-based motion estimation and compensation," *IEEE CSVT*, vol. 10, pp. 1029–1038, 2000.

[13] A. Beric, G. de Haan, R. Sethuraman, and J. van Meerbergen, "Towards a high-quality low-power single-chip picture-rate up-conversion module," *Proc. ICIP*, Barcelona, 2003.

[14] (2005) OptimoDE tool set. ARM Ltd. [Online]. Available: http://www.arm.com