Affine schemes in Mesh-Based Video Motion Compensation

Anant Utgikar1, Wael Badawy2, Guna Seetharaman1, Magdy Bayoumi1.

1 CACS, Univ. Louisiana at Lafayette, USA, 2 Univ. Calgary, CA.

ABSTRACT

We evaluate performances of different implementations of affine transform in motion compensation architecture based on a hierarchical adaptive structured mesh. The architecture predicts the next video frame using the reference frame, mesh code and mesh node motion vectors. It achieves significant reduction in describing the mesh topology by coding the splitting in recursive triangulation of the initial coarse geometry. It uses a memory serialization unit and one simple warping unit to map the hierarchical structure. The affine unit warps the texture of a patch at any level of hierarchical mesh independently. We compare Shift-After-Difference (SAD), LookUp Table (LUT) and naive implementations of the affine unit. Computing affine transform using multiplication-free SAD algorithm significantly reduces the complexity of the architecture. We establish from simulation results that our multiplication-free SAD affine computation requires far lesser power and area than other schemes. We discuss the limitations and advantages of either motion compensation schemes. Performance analysis shows that this scheme is suitable for video applications like MPEG and VRML.

I. INTRODUCTION

Recent advances in video signal processing [17] have played an important role in developing application areas like digital television, videophone, video-on-demand, video conferencing, and multimedia communication. All these applications involve processing or transmitting huge amount of information in a short period of time. In most cases, the available channel capacity is few orders of magnitude lower than the bandwidth needed by raw data. To circumvent this, several video compression techniques and hardware architectures have been developed. The computation intensive nature of these video applications and demand for real-time processing necessitate the use of an efficient VLSI implementation.

There exists inherent temporal redundancy in video sequences that many compression techniques exploit. To tap the correlation between consecutive video frames, Block Matching algorithms (BMAs), are widely deployed due to their regular processing scheme and simple control structures. Motion estimation and compensation through a deformable mesh outperforms BMAs since mesh warping is capable of capturing the dynamics of a video sequence more accurately. They generate continuously varying motion fields to represent more general types of motions, such as rotation, scaling, reflection, translation, and shear [1] and the overall motion field results in smaller error metric [2].

Warping has been used to generate special effects in computer graphics [3], [1] and to compensate for the distortions introduced by the imaging system in remote sensing [4]. The mesh model is also used for manipulation of synthetic objects as well as representations in joint natural-synthetic environments [5] and medical image sequences [6].

Mesh-based systems for motion compensation and video compression [7],[8] generally use regular or delaunay 2-D meshes [9]. A regular mesh has lower mesh code size since the mesh is known in both encoder and decoder but it lacks the efficient video representation, as it does not capture the video dynamics. The delaunay triangular mesh suffers from the complexity of delaunay algorithm and the overhead of sending the location of the mesh nodes.

Hierarchical mesh representation [10] provides rendering at various levels of detail and allows progressive transmission of the mesh geometry and the motion vectors [11]. The complexity of the 2D mesh-based video representation is thus based on extracting the video dynamics, generating the mesh and coding the nodes.

Video applications in mobile systems target high performance and low power consumption for video coding. High-speed data and computation-intensive processing of these applications cause much higher power dissipation than traditional applications. Hence, special architectures are required for efficient hardware solutions delivering sufficient video processing performance and low power consumption [12].

As operation speed increases, increased power dissipation reduces circuit reliability and hence the interest in low-power circuits.

Due to quadratic dependence, lowering supply voltage contributes a significant power saving though increasing the delay at the same time. The delay can be compensated for, at the system
level design by partitioning or scheduling. The power savings by reformulating the algorithms and mapping them to efficient low-power VLSI implementations are in the range of 70%–90% [12]. Approximations, parallel design and pipelining can compensate for the delay [13].

This paper compares different schemes to implement affine unit in motion compensation architecture for video motion tracking that uses a novel 2-D hierarchical mesh-based video model [11]. The motion compensation architecture uses parallel-pipelined threads, which implement scalable affine units. The architecture employs various techniques to reduce computations and power consumption and can be used as a building block for MPEG codec. Corresponding motion estimation architecture also generates a coarse-to-fine content-based mesh.

The affine units can be used with any level of hierarchical mesh and they warp the mesh patches independently. The rest of the paper is organized as follows: Section II describes the 2-D mesh-based motion compensation architecture and affine computation, Section III presents a discussion of the schemes and Section IV, the simulation results and section V, conclusion.

II. MESH-BASED VIDEO MODEL AND MOTION COMPENSATION

A 2-D mesh is a planar graph that partitions an image into n-sided regular polygonal patches like triangles (n=3) or quadrangles (n=4). In mesh-based motion modeling, the polygonal patches in the current frame are deformations of polygonal patches in the reference frame due to the movement of the mesh nodes. The texture inside each patch in the reference frame is warped into the current frame using parametric mapping as a function of the node-point motion vectors. A video frame is segmented into independent video objects tracked individually and represented using a hierarchical mesh and texture [11]. A hierarchical mesh-based motion estimation and compensation technique [9] uses multiplication-free affine transformation.

Fine-to-coarse and coarse-to-fine are two approaches of hierarchical mesh representation. The former removes selected nodes from a full-resolution mesh and in the latter, the number of nodes and triangles increases gradually based on a specified threshold function. The hierarchical adaptive structured mesh [14] is a technique borrowed from 3-D modeling. It implements a coarse-to-fine mesh construction as shown in

---

**Figure 1** (a) Triangulation (b) Tree construction and (c) Bit coding

Initial coarse mesh has a fixed triangular mesh patch size at level 0. Each coarse patch is then processed to generate finer mesh granularity using a hierarchical adaptive structured approach to reduce the error among the motion vectors of the triangular patch [14]. All possible patches are processed until the max level of splitting is reached or the patch splitting condition is not satisfied.

Adding new mesh nodes is based on successive triangulation to minimize the mesh code that describes the mesh topology. Representing hierarchical mesh by progressive coding technique describes each mesh level, while the non-progressive one describes the mesh of each coarse patch.

---

**Figure 2 Hierarchical Adaptive Structured mesh construction from image.**

A. Mesh-Based Motion Estimation

The motion estimation algorithm processes current and previous frames to generate a 2-D hierarchical structured mesh from one frame and estimate motion vector for each mesh node. The motion vectors carry the transformation of the video object across the video sequence. They can be evaluated using the three-step search (TSS)
algorithm [15], which reduces the computational complexity while maintaining a performance comparable with the full-search schemes.

B. Mesh-Based Motion Compensation

The motion compensation algorithm processes the reference frame, mesh code and estimated motion vectors to predict the next frame. The predicted frames are synthesized by mapping the patches of the previous frame onto the corresponding patches of the current frame. Transmitted motion vectors compensate for the movement of the mesh nodes at the receiving end. Affine transformation maps the patch texture and has a small computational cost regardless of the shape of the patch [16]. Zoom of a mesh unit requires interpolation of values which being computationally expensive can be expedited using Lookup tables or approximations and is not considered here.

The motion compensation architecture as shown in Fig. 3, uses four scalable affine units, each having independent access to the frame buffer. The frame buffer is divided into four quadrants with an overlap region between them to accommodate the parallel operation. The degree of overlap is dependent on range of values motion vector can take, and dimension of the image since a bordering patch may warp (partially or completely) into neighbouring block. Fig.4 shows the implementation of the splitting logic.

![Figure 3 Motion Compensation Architecture](image-url)

(i) Scalable Affine Unit warps the triangular patch and maintains a queue to store the mesh at each level. The triangular patch is split into four patches and subsequently stored in the queue. If the patch needs to be triangulated, the patch will be forwarded to another affine unit.

(ii) Mesh Code Buffer stores the code of the adaptive structured mesh topology, which records the splitting operation. With each splitting, the adaptive technique generates four triangular patches, adds three nodes and estimates three motion vectors.

(iii) Memory Serialization Unit serializes memory-write requests issued by parallel affine units. The affine unit generates the memory address and data for the predicted frame. A control line serializes the parallel data to be transferred to the main memory.

C. Affine Transformation

It scales distances between points uniformly. Mathematically, [1]:

\[
\begin{align*}
    f(x,y) &= a_1 x + a_2 y + a_3 \\
    g(x,y) &= a_4 x + a_5 y + a_6 \\
    f(x+1,y) &= a_1 (x+1) + a_2 y + a_3 = f(x,y) + a_1 \\
    g(x+1,y) &= a_4 (x +1) + a_5 y + a_6 = g(x,y) + a_4 \\
    f(x,y+1) &= a_1 x + a_2 (y+1) + a_3 = f(x,y) + a_2 \\
    g(x,y+1) &= a_4 x + a_5 (y +1) + a_6 = g(x,y) + a_5
\end{align*}
\]

where \( f(x,y) \) and \( g(x,y) \) are the new coordinates of the pixel initially at \( (x,y) \) and \( a_n, (n=1-6) \) are the affine parameters. The affine parameters can be calculated from \( p_i \) and \( q_i \), the motion vectors of node \( (x_i,y_i) \), as:

\[
a_1 = \frac{(f(x_2,y_1)-f(x_2,y_3)) y_3 + (f(x_3,y_1)-f(x_1,y_1)) y_2, (f(x_2,y_2)-f(x_2,y_3)) y_1}{(x_1-x_2) y_3 + (x_2-x_3) y_2 + x_2 - x_3) y_1}
\]

where \( f(x,y) = p_i + x_i \) and \( g(x,y) = q_i + y_i \).

Scan-line algorithm [9] simplifies the calculation of the affine transformation. Instead of using four multiplication operations and four additions, the algorithm uses only two additions per pixel and uses only four affine parameters. It must still evaluate these parameters for each mesh unit from motion vectors \( (p_i,q_i) \) and locations \( (x_i,y_i) \) of the node.

The algorithm is based on the fact that the pixels are integer values and the mapping can be evaluated using the mapped values of a neighboring pixel as shown above. Multiplication-Free Algorithm uses a finite space
approach and processes the mesh patches independently. The affine parameters can be evaluated at the three nodes of the triangular patch from following simplified equations, for triangular topology \( d-e-f \) with sides along the coordinate axes as:

\[
\begin{align*}
    a_1 &= 1 + \frac{p_f - p_e}{c} \\
    a_2 &= \frac{p_d - p_e}{c} \\
    a_4 &= \frac{q_f - q_e}{c} \\
    a_5 &= 1 + \frac{q_d - q_e}{c}
\end{align*}
\]

**SAD Scheme:** If \( c \) is power of 2, the division can be replaced by a bit shift operation. Using finite space analysis, each triangular patch is translated to the origin \((0, 0)\) and the edges aligned with the coordinate axes. Thus, there are two pairs of nodes, which differ in exactly one coordinate. The affine parameters can now be evaluated with simple expressions given above and the texture is warped. The patch is then retranslated to its correct position. The assumption of the size to be power of 2 will relax the patch translation to be only address mask.

**LUT Scheme:** We can store the affine parameters in a LookUp Table. The six motion vectors, 2 for each of the 3 mesh nodes are used to index into the memory. Note that the mapping from motion vectors to affine parameters is not one-to-one but many-to-one. This is expected because the degree of freedom has been reduced from 6 to 4 and affine parameters depend only on the difference of motion vectors. The size of the table builds up as an exponential function of both, precision of affine parameters or motion vectors and the search window of motion vectors. This requires tremendous amount of storage and the trade offs between performance and cost come into picture.

### III. DISCUSSION

The 2D mesh is generated by successive triangulation to minimize the mesh code that describes the mesh topology as shown in Fig.1. If each split produces \( N \) new nodes (3 for tri-split, 4 for quad-split) then each split would require \( \log_2 N \) extra bits for representation. In the simplest case let us assume that over a wide variety of video sequences[20] the object and the background appear with equal probability and object exhibits motion that needs to be captured. Thus, we assume that though actual splits may happen anywhere in the tree structure, statistically all depths of splits are identical. Since each split produces \( N \) new nodes whose representation by coding needs \( \log_2 N \) more bits than its parents, the number of bits increases by \( O(N \log_2 N) \) with each split. For a mesh with \( k = N^r \) splits, coding the splitting of mesh nodes, requires only \( kN+1 \) bits whereas coding the mesh nodes requires \( O(N^{r+1} \log_2 N) \) bits. Exact improvement would depend on statistical distribution and will vary in each case.

The efficiency of SAD scheme for affine transformation dictates that mesh be generated in such a way as to ensure \( c \), the dimension of a patch in the mesh, is a power of 2. This can be achieved by simply restructuring the initial conditions since, a general sequence of frames may have any dimensions. Each quad-split during mesh generation automatically ensures that each of the resultant polygons have dimensions exactly half of that of the parent. For an image of arbitrary dimension, rounding up to smallest power of 2 larger than dimension, would introduce variable overhead. However, the Lookup Table scheme is valid for any arbitrary value of \( c \) since it does not require simplification in computation of affine coefficients. In fact, this scheme may be generalized further for any alignment and location of patch, thereby amplifying the size of the table. Either scheme is valid for any mesh code as input.

**Figure 4 Mesh Splitting logic**

The motion compensation must use the same algorithm for mesh generation as the corresponding motion estimation to recognize correspondence between nodes and motion vectors, unless mesh code is input. It is then not necessary to transmit the mesh code since the
receiver can generate the same mesh from its previous frame. The choice of threshold function for generating the mesh decides the accuracy of tracking object motion. A function that depends on some property of mesh nodes alone, like difference in motion vector of vertices, might ignore significant intricate details by not decomposing a patch. This is a limitation of mesh construction due to choice of threshold function, and not of motion compensation. Either of the affine schemes in motion compensation discussed earlier are transparent to this and behave identically. Statistical peak signal to noise ratio (PSNR) determines visual quality and is:

$$PSNR = 10 \times \log \left( \frac{255^2}{\sum_{m=1}^{M} \sum_{n=1}^{N} (c_{m,n} - \bar{c}_{m,n})^2} \right)$$

where $c_{m,n}$, $c_{m,n}$ are pixels of the original and the reconstructed frame. PSNR for a sequence is a property of the algorithm and would not vary for different implementation schemes discussed above. PSNR for one, two and four-level (Q-1,2,4) hierarchical mesh with 32 pixel resolution is plotted in Fig. 5.

IV. RESULTS

Both implementations schemes reconstruct frame from input data identically. Simulation figures for 5x5 inputs processed in one step have been tabulated. We compare the two schemes for estimated Dynamic Power, Area, Number of Cells, and Number of nets as obtained from Synopsys simulation at a global voltage of 5V, in Table 1. It should be noted that the naïve computation scheme involves six multiplications and a division operation, which being extremely demanding on all critical resources, is not comparable and hence not included here. As can be seen from the table, SAD computation of affine unit in the architecture obtains significant savings over the LUT scheme in all parameters considered. The percentage savings in each parameter are included in the table (in braces). Also, increment in motion vector size builds up a cumulative effect, running the required memory size and hence the simulation time exponentially. We also present schematic generated for the two schemes by Synopsys Design Analyzer Tool for identical number of ports.

![Figure 6 Schematic for SAD scheme](image)

![Figure 7 Schematic for LUT scheme](image)

<table>
<thead>
<tr>
<th>Measure Scheme</th>
<th>SAD (% Savings)</th>
<th>LUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area</td>
<td>1564</td>
<td>1683</td>
</tr>
<tr>
<td>Dynamic Power</td>
<td>19.85 mW</td>
<td>21.92 mW</td>
</tr>
<tr>
<td>Cells</td>
<td>742</td>
<td>862</td>
</tr>
<tr>
<td>Nets</td>
<td>865</td>
<td>977</td>
</tr>
</tbody>
</table>

Table I. SAD and LUT scheme comparison.

V. CONCLUSION

This paper compares affine design schemes for motion compensation architecture based on hierarchical mesh. Adaptive structured mesh captures the contents of the video sequence with significant reduction in the mesh topology coding by coding the splitting. The motion compensation architecture offers low power performance by decomposing mesh for parallel processing and by simplifying the processing unit. Multiplication-free affine transformation significantly reduces the complexity and pipelining the affine unit improves efficiency at the cost of storage, delay. We establish from simulation results that SAD scheme offers about 7-15% savings in various measures, over the
LUT scheme. The performance results show that SAD motion compensation can be used for online applications and lower power consumption makes it suited for mobile applications.

Acknowledgement: Authors would like to thank GSO (ULL), Frank Ducrest (CS, ULL) and Xilinx® for financial and infrastructure support. HariH OM shri kR^iShNAarpaNamastu [21].

VI. REFERENCES


