A Low Power VLSI Architecture for Mesh-Based Video Motion Tracking

Wael Badawy, *Member, IEEE,* and Magdy A. Bayoumi, *Fellow, IEEE*

***Abstract*—This paper proposes a low-power very large-scale integration (VLSI) architecture for motion tracking. It uses a hierarchical adaptive structured mesh that generates a content-based video representation. The proposed mesh is a coarse-to-fine hierarchical two-dimensional mesh that is formed by recursive triangulation of the initial coarse mesh geometry. The structured mesh offers a significant reduction in the number of bits that describe the mesh topology. The motion of the mesh nodes represents the deformation of the video object. The architecture consists of a motion estimation and motion compensation units. The motion estimation architecture generates a progressive mesh code and the motion vectors of the mesh nodes. It reduces the power consumption, it uses a simpler approach for mesh construction, approximates the mesh nodes motion vector by using the three step search algorithm and it uses parallel motion estimation core to evaluate the mesh nodes motion vectors. Moreover, it maximizes the lifetime of the internal buffers. The motion compensation architecture uses a multiplication-free algorithm for affine transformation, which significantly reduces the complexity of the motion compensation architecture. Moreover, using pipelined affine units contributes to the power savings. The video motion compensation architecture processes a reference frame, mesh nodes and motion vectors to predict a video frame. It implements parallel threads in which each thread implements a pipelined chain of scalable affine units. This motion compensation algorithm allows the use of one simple warping unit to map a hierarchical structure. The affine unit warps the texture of a patch at any level of hierarchical mesh independently. The processor uses a memory serialization unit, which interfaces the memory to the parallel units. The architecture has been prototyped using top-down low-power design methodology. The performance analysis shows that this processor can be used in online object-based video applications such as in MPEG and VRML.**

***Index Terms*—Affine transformation, digital video processing, HDTV, low power architecture, mesh-based motion tracking, motion compensation, motion estimation, multimedia, VLSI.**

I. Introduction

MOTION estimation and compensation through a deformable mesh is one of the more advanced methods for video sequence representation. It outperforms the famous and widely used block matching algorithm (BMA) since warping is capable of producing a more accurate representation of the dynamics of a video sequence and thus is an attractive tool for many applications in digital video processing.

The advantage of the mesh-based method is the ability to generate continuously varying motion fields. This is especially important because natural video sequences generally give rise to smooth or slowly varying motion fields. In particular, camera zoom, movement of the objects toward or away from the camera and rotational motion of objects are much better described in a mesh-based model than in the traditional block-based motion and the overall motion field results in the smallest possible error metric [1].

The mesh-based transformation was used to compensate for the distortions introduced by the imaging system in the field of remote sensing [2]. Image warping is also used to generate special effects in computer graphics [3], [4]. Mesh-based systems for motion compensation and video compression are used in digital video processing [5]–[9]. A hierarchical version of the mesh motion is introduced [10]. A finite-element approach to the mesh warping and its application in video coding [11]–[14]. The mesh model is used for manipulation of synthetic objects as well as representations in joint natural-synthetic environments [15], [16]. Mesh-based methods have also been used with great success in processing medical image sequences [17]–[19].

The mesh-based motion model uses a mathematical transformation that maps one set of polygons in a frame to another set of polygons in the following frame. The pixels within the polygons are mapped from one frame to another through interpolation. Regular mesh and delaunay meshes are two main approaches to design a two-dimensional (2-D) mesh-based triangular mesh [20]. Using a regular mesh reduces the mesh code size since the mesh is known in both encoder and decoder but it lacks the efficient video representation since it does not capture the video dynamics. On the other hand, the delaunay triangular mesh suffers from the complexity of delaunay algorithm and the overhead of sending the location of the mesh nodes. A possible approach that reduces the mesh code size is to implement the delaunay algorithm in both the encoder and the decoder.

Hierarchical mesh representation provides rendering at various levels of detail (quality, scalability) and it allows progressive transmission of the mesh geometry and the motion vectors [21]. Using the delaunay mesh, the hierarchical representation can be achieved by ordering the mesh nodes according to their importance and then different disjoint sets are generated, where each set represents a resolution. The union of all sets will generate the fine resolution. In summary, the complexity of the 2-D mesh-based video representation is based on extracting the video dynamics, designing the inner nodes and coding the mesh nodes.

Manuscript received May 6, 2001; revised September 5, 2002. This work was supported in part by the U.S. Department of Energy (DoE) under Grant DE-FG02 97-ER 12220, Energy and Environmental Technology Applications Program (EETAP, 1997–2000), and in part by the State of Louisiana Board of Regents Support Fund (BoRSF, formerly LEQSF, 1996–1999) Program under Grant RD-B-13. This paper was recommended by Associate Editor T. Stouraitis.

W. Badawy is with the Department of Electrical and Computer Engineering, University of Calgary, Calgary AB T2N 1N4, Canada (e-mail: badawy@enel.ucalgary.ca).

M. A. Bayoumi is with the Center for Advanced Computer Studies, University of Louisiana, Lafayette, LA 70504 USA (e-mail: mab@cacs.louisiana.edu).

Digital Object Identifier 10.1109/TCSII.2002.805248

1057-7130/02$17.00 © 2002 IEEE

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 11:59:48 EDT from IEEE Xplore. Restrictions apply.

Performance and power consumption are the main objectives of the video coding for mobile applications. Special architectures are required for efficient hardware solutions delivering sufficient video processing performance and low power consumption [22], [23]. Surveying the literature, only a few very large-scale integration (VLSI) architectures for video motion tracking codecs that use 2-D mesh-based motion estimation and compensation exists.

This paper presents a mesh-based motion tracking architecture that uses a novel 2-D hierarchical mesh-based video model. It reduces the computational operations and employs techniques to reduce the power consumption as well. The new video model and its architecture are suitable for mobile video applications and the architecture can be used as a building block for MPEG codec. The proposed motion tracking architecture is divided into motion estimation and motion compensation architectures. The motion estimation architecture generates a coarse-to-fine content-based mesh. It optimizes the overhead of sending the mesh's geometry by coding the splitting locations instead of the mesh nodes' locations. The coarse level is split into independent triangular patches that can be processed in parallel and each patch is hierarchically triangulated using a structured technique, which simplifies the processing unit. It uses the three steps motion estimation algorithm [24], [25] to simplify the motion estimation architecture. The motion compensation architecture uses parallel-pipelined threads, which implement scalable affine units. 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 low power design methodologies, Section III describes the proposed video model, Section IV describes the 2-D mesh-based motion estimation architecture, Section V describes the 2-D mesh-based motion compensation architecture, Section VI presents the performance of the proposed video representation and the motion tracking architecture, Section VII presents the prototyping methodology and finally Section VIII concludes this paper.

II. Low-Power Design Methodologies

With the advances of the mobile communication and personal communication services (PCSs), there is a trend to run video applications on mobile systems. High speed data and computation-intensive processing will lead to much higher power consumption than traditional portable applications. Due to the limited power-supply capability of current battery technology, current research efforts face two conflicting design challenges. One is to explore high-performance design and implementation techniques that can meet the stringent speed constraints for real-time systems. The second is to consider low-power design approaches to prolong the operating time of portable devices.

Power consumption has been introduced as a major design parameter due to the fast growing technology in mobile communications, which require portable power supplies. There are three main sources of power dissipation in a complementary–metal–oxide semiconductor (CMOS) circuit: 1) *switching power*, which is consumed by the circuit capacitance during transistor switching; 2) *short-circuit power*, which is due to current flowing from power supply to ground during transistor switching; and 3) *static power*, which is due to leakage and static currents. The switching and short circuit power are known as dynamic power, which constitutes about 99% of the total power dissipation in CMOS VLSI circuits. Dynamic power P_{dynamic} is given by the following equation:

$$P_{\text{dynamic}} = f_{\text{clk}} \cdot V_{\text{dd}} \cdot ((C_i \cdot V_{\text{swing}\,i} \cdot p_i) + I_{\text{sc}} \cdot p_i)$$

where f_{clk} is the system clock frequency, V_{dd} is the power supply voltage, C_i is the capacitance at node I, $V_{\text{swing}\,i}$ is the voltage swing at node i, p_i is the switching probability at node i, and I_{sc} is the short circuit current at node i.

As system speed increases, the increase in power consumption becomes a major problem. The increase in the power dissipation will generate extra heat and reduce the reliability of a circuit. The development of techniques allowing low-power circuits is crucial to the practicality of future products. Due to its quadratic dependence, lowering V_{dd} contributes a significant power saving. However, the delay in processing time will increase as V_{dd} decreases. To meet the throughput requirements, the delay must be compensated for at the system and algorithmic/architecture design level. Simply, a new version of the design that allows more latency is required.

Low-power system-level approaches involve two main techniques. One is partitioning the system into modules and shutting down modules that are not working such as multichip modules (MCMs) [39]. The other approach is to use a monitor scheduler to distribute the workload over several functional units. The scheduler can be a hardware or software monitor.

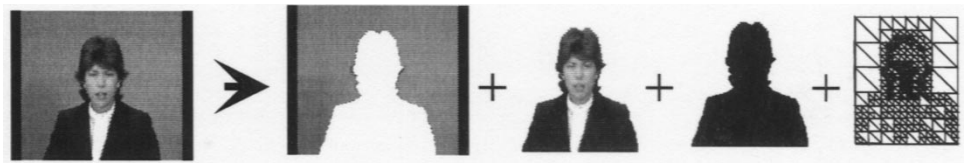
Besides the system-level low-power approaches, the algorithmic/architectural approach is the most promising one [40]. The algorithmic/architectural low-power design can be achieved by reformulating the algorithms and mapping them to efficient low-power VLSI architectures to compensate for the speed penalty caused by low supply voltage. The power savings of the algorithmic/architectural level are in the range of 70%–90% [23]. The current goal is to reduce the total power dissipation of the electronic systems to two orders of magnitude less than what it would have been with the conventional technology [41]. Approximations, parallel and pipelined design approaches are used to compensate for the delay penalty [40], [42].

Low-power and high-performance architectures can be achieved using approximation approaches which trade high performance with the cost of the circuits, pipelining which trades the overhead of extra buffers with latency and parallel architectures which trade the area with latency.

In the approximation approaches, the constraints are relaxed in order to reduce the cost and yield acceptable performance. Approximation approaches generally lead to much simplified architectures, which then add to the savings in total implementation cost. For example, three-step motion estimation [30] is usually employed to reduce the computational complexity while maintaining a performance comparable with the full-search motion estimation schemes [43].

Pipelining is a commonly used technique to achieve high-speed data processing. The main idea behind pipelining

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 11:59:48 EDT from IEEE Xplore. Restrictions apply.



(a) (b) (c) (d) (e)

Fig. 1. Video frame representation. (a) Original frame. (b) Frame background. (c) Object texture. (d) Binary layer. (e) Hierarchical mesh layer.

a design is shortening the delay through the critical path by inserting latches between consecutive pipeline stages. As a result, the speed of the system is faster than that of the original system at the penalty of increasing the latency by the number of pipelining stages. On the other hand, the pipelining can be used to compensate for the delay incurred in the low-power design when the supply voltage drops significantly. Some derivations of the pipelining techniques are asynchronous circuit pipelining [44], two-level (word/bit-level) pipelining [45] and bit-level pipelining approaches [46].

Parallel data processing decomposes the desired function into independent and parallel small tasks. These tasks are executed concurrently and individual results are combined to obtain the desired computation result. It is a "divide-and-conquer" strategy in solving problems. The fast Fourier transform (FFT) [47] is a famous example where the algorithm first decomposes an N-point FFT into two N/2-point FFTs and expands the computation recursively to form a parallel computing network.

III. Proposed Mesh-Based Video Model

The dynamics of a video sequence can be modeled as block or 2-D mesh. An advantage of the 2-D mesh representation over the block-based model is its ability to represent more general types of motions, such as rotation, scaling, reflection, translation, and shear [4], [27].

The video object tracking model tracks the changes of the video frames by tracking independent video objects. The video frame is segmented into independent video objects, with each object represented using a hierarchical mesh and texture [37]. The motion vectors of the mesh nodes carry the transformation of the video object across the video sequence. The deformation of a triangular patch represents the motion of the image points on a common rigid plane. The predicted images are synthesized by mapping the patches of the previous frame into the corresponding patches of the current frame. This mapping is easily realized since the transformation between two triangles is described by a 2-D affine transform.

A hierarchical mesh-based motion estimation and compensation technique is proposed in [21]. It uses a hierarchical mesh representation with multiplication-free affine transformation. The basic idea is to divide the image frame [Fig. 1(a)] into a background [Fig. 1(b)], a video object texture [Fig. 1(c)], a binary layer [Fig. 1(d)], and a hierarchical triangular patch [Fig. 1(e)] and then estimate the motion vectors of the nodes of the triangular patches.

The motion estimation algorithm processes two frames and generates a 2-D mesh and the motion vector for each mesh node. The motion compensation algorithm processes a reference frame, a mesh and motion vectors to predict a frame. The predicted frames are synthesized by mapping the patches of the previous frame onto the corresponding patches of the current frame. The image construction uses an affine transformation to map the patch texture. It has been proved that the affine transformation has a small computational cost regardless the shape of the patch [27].

The proposed mesh-based video representation uses a hierarchical triangular mesh to model the dynamics of the video sequence. The motion vectors of the mesh nodes carry the deformation across a video sequence. The predicted images are synthesized by mapping the patches of the previous frame into the corresponding patches of the current frame. This mapping is easily realized since the transformation between two triangles is described by a 2-D affine transformation [15], [26]–[33].

The hierarchical adaptive structured mesh [20] is a technique borrowed from the 3-D modeling for mesh construction. It uses a coarse-to-fine mesh. The number of mesh nodes and triangular patches increases gradually according to a threshold function. Adding new mesh nodes is based on successive triangulation to minimize the mesh code that describes the mesh topology.

A coarse-to-fine mesh (Fig. 2) is generated as fixed triangular mesh at level 0, then each patch is split according to a threshold function and the mesh nodes increase gradually. Recording the splitting operation generates the mesh topology. Note that each time three nodes are added, three motion vectors are estimated at most. The successive triangulation is shown in Fig. 3. In this approach, an adaptive technique generates four triangular patches. Fig. 3(c) shows the coding for the triangular patch, where '1' means that the mesh is split into three triangles and '0' means that no further splitting.

The generated hierarchical mesh topology can be represented using progressive or nonprogressive code. The progressive coding technique describes each mesh level, while the nonprogressive one describes the mesh of each coarse patch. Fig. 4 shows two different mesh code representations. Fig. 4(a) shows the progressive code, where each level is a refinement of the previous mesh level. Fig. 4(b) shows the mesh code where each patch in the level 0 is coded sequentially.

A. Mesh Nodes Motion Estimation

The motion of the mesh nodes describes the deformation of the mesh topology. It represents the dynamics of the video across a sequence. To estimate a mesh node motion vector, the node is surrounded by a block. The three steps motion estimation algorithm [30] is used to estimate the motion vector of the mesh nodes (Fig. 5). The three-step motion estimation [30] reduces the computational complexity while maintaining the performance comparable with the full-search motion estimation schemes.

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 11:59:48 EDT from IEEE Xplore. Restrictions apply.

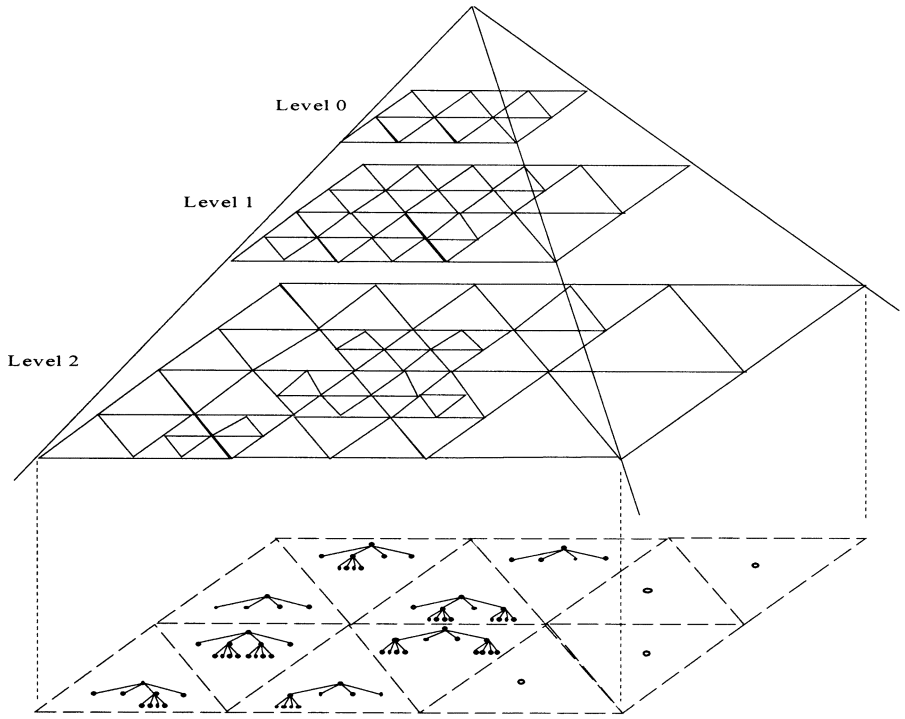


Fig. 2. 2-D mesh construction.

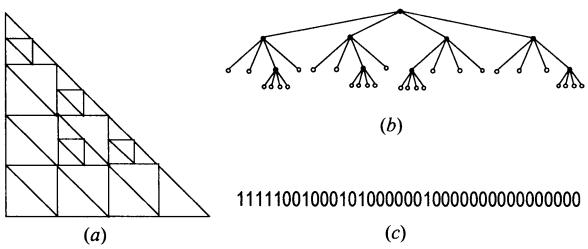


Fig. 3. Triangular patch coding. (a) Patch triangulation. (b) Tree representation. (c) Bit code representation.

level 0 = 11110011110 0

level 1 = 00000100010100000100101101000100 1

level 2 = 00000000000000000000000000000000000000

(a)

1 0000 1010000 101010 0000000

10000 0 0 100 100000 1011000000000

110000000 1100100000000 0 0

(b)

Fig. 4. (a) Progressive and (b) nonprogressive mesh code.

B. Mesh-Based Motion Estimation Algorithm

The mesh-based motion estimation algorithm generates a 2-D hierarchical structured mesh and a motion vector for each mesh node. The proposed algorithm processes an initial coarse mesh with a fixed patch size. The motion vector of each mesh node is estimated. The motion vector of each mesh node is evaluated using the three-step search (TSS) algorithm. Then each coarse patch is processed to generate finer mesh granularity using a hierarchical adaptive structure approach. The hierarchical structure mesh is constructed to reduce the error among the motion vectors of the triangular patch [20]. For each new node, the motion vector is estimated. The algorithm will process all possible patches until the max level of splitting is reached or the patch splitting condition is not satisfied. The 2-D mesh-based video motion estimation algorithm is listed in Fig. 6.



Fig. 5. Mesh nodes motion estimation.

C. Mesh-Based Motion Compensation Algorithm

A 2-D mesh is a planar graph that partitions a 2-D image region into polygonal patches consisting of triangles or quadrangles. The vertices of the polygonal patches are referred to as node points. Mesh-based motion modeling differs from block-based motion modeling in that the polygonal patches in the current frame are deformed by the movement of the node points into polygonal patches in the reference frame. 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.

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 11:59:48 EDT from IEEE Xplore. Restrictions apply.

For each patch, pi is represented as triangle abc,
1: Given the maximum number of levels l_{max}.
2: $l_{current} = 0$
3: Compute the motion vectors v_a, v_b, v_c.
4: If the motion vector is not sent
  5: Send X and Y component
  6: Mark node for send
7: If ($l_{current} < l_{max}$)
  8: If ($|v_a\text{-}v_b| > th$ OR $|v_a\text{-}v_c| > th$ OR $|v_b\text{-}v_c| > th$)
    9: Locate new nodes d, e, f
    10: Generate v_d, v_e, v_f
    11: For each node a, b, c
    12: If the motion vector is not sent
      13: Send X and Y component
      14: Mark node for send
    15: Output '1'
    16: Increment ($l_{current}$)
  17: Else output '0'
18: Repeat steps 2-17 until all patches are visited

Fig. 6. Proposed 2-D mesh-based motion estimation.

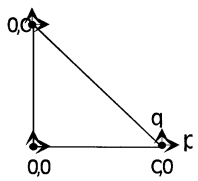


Fig. 7. Multiplication-free algorithm triangular topology.

D. Affine Transformation

The affine transformation is widely used in texture mapping and it defined as follows [4]:

$$\mathbf{f}(\mathbf{x},\mathbf{y}) = \boldsymbol{a}_{i1}\mathbf{x} + \boldsymbol{a}_{i2}\mathbf{y} + \boldsymbol{a}_{i3}$$
$$\mathbf{g}(\mathbf{x},\mathbf{y}) = \boldsymbol{a}_{i4}\mathbf{x} + \boldsymbol{a}_{i5}\mathbf{y} + \boldsymbol{a}_{i6}$$

where $f(x,y)$ and $g(x,y)$ are the new pixels location, (x,y) is the pixel location, and a_{i1}–a_{i6} are the affine parameters. The affine transformation can be computed once the parameters can be evaluated.

The affine transformation suffers from the cost of evaluating the affine parameters and the multiplication operations. To calculate the affine parameters, six equations are needed. We can solve the equations at the triangle vertices using the X and Y components of the motion vector. A scan-line algorithm [27] is proposed to simplify the calculation of the affine transformation. Instead of using four multiplication operations and four additions for calculating the (X,Y) pair in the two equations above, the algorithm uses only two additions per pixel to generate the (X,Y) pair. The algorithm is based on the fact that the pixels are integer values and the mapping can be evaluated using the values of neighboring pixel as shown in the following equation:

$$\mathbf{f}(\mathbf{x}+\mathbf{1},\mathbf{y}) = \boldsymbol{a}_{i1}(\mathbf{x}+\mathbf{1}) + \boldsymbol{a}_{i2}\mathbf{y} + \boldsymbol{a}_{i3} = \mathbf{f}(\mathbf{x},\mathbf{y}) + \boldsymbol{a}_{i1}$$
$$\mathbf{f}(\mathbf{x},\mathbf{y}+\mathbf{1}) = \boldsymbol{a}_{i1}\mathbf{x} + \boldsymbol{a}_{i2}(\mathbf{y}+\mathbf{1}) + \boldsymbol{a}_{i3} = \mathbf{f}(\mathbf{x},\mathbf{y}) + \boldsymbol{a}_{i2}$$
$$\mathbf{g}(\mathbf{x}+\mathbf{1},\mathbf{y}) = \boldsymbol{a}_{i4}(\mathbf{x}+\mathbf{1}) + \boldsymbol{a}_{i5}\mathbf{y} + \boldsymbol{a}_{i6} = \mathbf{g}(\mathbf{x},\mathbf{y}) + \boldsymbol{a}_{i4}$$
$$\mathbf{g}(\mathbf{x},\mathbf{y}+\mathbf{1}) = \boldsymbol{a}_{i4}\mathbf{x} + \boldsymbol{a}_{i5}(\mathbf{y}+\mathbf{1}) + \boldsymbol{a}_{i6} = \mathbf{g}(\mathbf{x},\mathbf{y}) + \boldsymbol{a}_{i5}.$$

Moreover, the scanline algorithm uses only four affine parameters but it suffers from the cost of evaluating these parameters. The affine parameters can be evaluated at the three nodes of the triangular patch as shown in the equations at the bottom of the page, Where (p_i, q_I) are the motion vectors and (x_i, y_i) are the location of the node i.

The scan-line algorithm reduces the computational cost of calculating the pixel value and reduces the number of parameters to four, but it suffers from the computation cost for evaluating the affine parameters. In the following we will present a totally multiplication free algorithm that can be used to design the video object compensation algorithm with only addition operations.

E. New Multiplication-Free Algorithm

The new multiplication-free algorithm uses a finite space approach and processes the mesh patches independently. Each patch will be mapped using affine algorithm. To illustrate the novel multiplication-free algorithm for affine transformation, consider the triangular topology shown in Fig. 7. The four affine parameters that are required by the scan-line algorithm will be evaluated as follows:

$$a_{i1} = -\frac{(p_i - p_{i+i})}{C}$$
$$a_{i2} = \frac{(p_{i-1} - p_i)}{C}$$
$$a_{i4} = -\frac{(q_i - q_{i+i})}{C}$$
$$a_{i5} = \frac{(q_{i-1} - q_i)}{C}.$$

If $C = 2^n$, the division can be replaced by a shift operation.

The multiplication-free algorithm is shown in Fig. 8. Using the finite space analysis, each triangular patch is translated to

$$a_{i1} = \frac{(p_i - p_{i+1}) \times y_{i-1} + (p_{i+l} - p_{i-l}) \times y_i + (p_{i-1} - p_i) \times y_{i+1}}{(x_i - x_{i+l}) \times y_{i-1} + (x_{i+1} - x_{i-1}) \times y_i + (x_{i+1} - x_i) \times y_{i+1}}$$
$$a_{i2} = \frac{(p_i - p_{i+1}) \times x_{i-1} + (p_{i+l} - p_{i-l}) \times x_i + (p_{i-1} - p_i) \times x_{i+1}}{(x_i - x_{i+l}) \times y_{i-1} + (x_{i+1} - x_{i-1}) \times y_i + (x_{i+1} - x_i) \times y_{i+1}}$$
$$a_{i4} = \frac{(q_i - q_{i+1}) \times y_{i-1} + (q_{i+l} - q_{i-l}) \times y_i + (q_{i-1} - q_i) \times y_{i+1}}{(x_i - x_{i+l}) \times y_{i-1} + (x_{i+1} - x_{i-1}) \times y_i + (x_{i+1} - x_i) \times y_{i+1}}$$
$$a_{i5} = \frac{(q_i - q_{i+1}) \times x_{i-1} + (q_{i+l} - q_{i-l}) \times x_i + (q_{i-1} - q_i) \times x_{i+1}}{(x_i - x_{i+l}) \times y_{i-1} + (x_{i+1} - x_{i-1}) \times y_i + (x_{i+1} - x_i) \times y_{i+1}}.$$

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 11:59:48 EDT from IEEE Xplore. Restrictions apply.

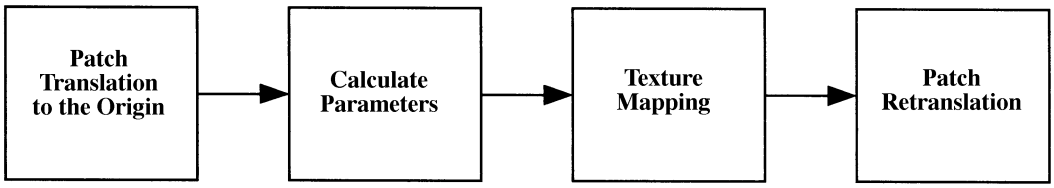


Fig. 8. Multiplication-free algorithm for general mesh.

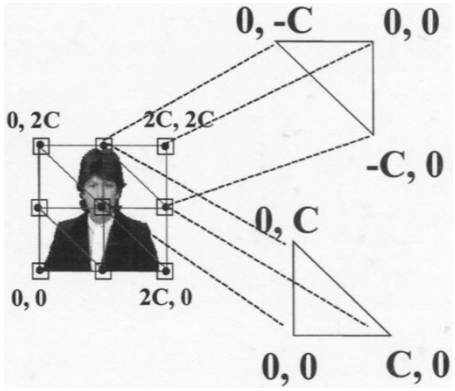


Fig. 9. Mesh patches are mapped to the origin.

the origin (0, 0). The parameters are evaluated and the texture is warped. Then the patch is retranslated to its correct position. Notice that the assumption of the size to be 2^n will relax the patch translation to be only address mask. Fig. 9 shows the mapping of the mesh patches to the origin.

IV. 2-D Mesh-Based Motion Estimation Architecture

The proposed motion estimation architecture is based on the presented 2-D mesh-based motion estimation algorithm. It processes two video frames and generates a 2-D hierarchical mesh and motion vectors of the mesh nodes. The block diagram for motion estimation architecture is shown in Fig. 10. It consists of two memory blocks, three motion estimation units, a queue, a splitting unit and a register array. The architecture processes the initial mesh nodes in a sequence that maximizes the data reusability and minimizes the internal storage [21].

To estimate the motion vector of each mesh node, the node is surrounded by a block of pixels. Then it uses the TSS [25] and the mean absolute difference (MAD) [22] to estimate the motion vector of each node. The splitting unit adds new mesh nodes according to a threshold function. The new nodes represent the finer mesh granularity. The queue unit stores the motion vectors, this information is needed to perform the recursive triangulation operations. The register array stores the motion vector and nodes location. The control unit synchronizes the operation between different blocks.

In the presented prototype, each triangular patch in level j can be split into four triangles, which requires the addition of three nodes. As in Fig. 2, the mesh, at level 0, consists of fixed triangular patches. At level 1, new nodes are added depending on the threshold function. Note that each time three nodes are added, three motion vectors are estimated at most.

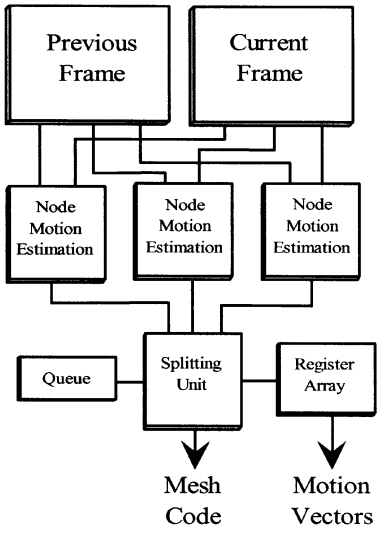


Fig. 10. Motion estimation architecture.

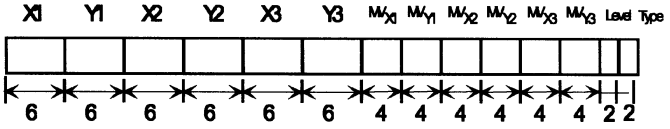


Fig. 11. Queue word entry.

A. Memory

The memory unit consists of two blocks. The first memory block contains the reference frame and the second one contains the current frame. The size of each block will be the size of the video frame. In this prototype, the size of the QCIF format is considered and the size of each block is 176×144 byte.

B. Queue

The queue unit stores the node locations; this information is needed to compute the recursive triangulation operations. The queue is used to generate the progressive coding. A stack with the same configuration can be used to generate the nonprogressive coding described in Fig. 4.

The queue word has nine entries, each one represents a mesh node (Fig. 11). Each queue entry consists of: 1) 36 b represent three mesh nodes; 2) 24 b represent three motion vectors; 3) 2 b are used to represent the level of the triangle; and 4) 2 b are used to define the patch type T. The total size of each queue entry in our case is 64 b and the queue length for a mesh with $N \times M$ nodes and up to L levels is $((N-1)^*(M-1)+3^*L)^*R^*2$, where R is the number of bits for each entry. Note that four bits

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 11:59:48 EDT from IEEE Xplore. Restrictions apply.

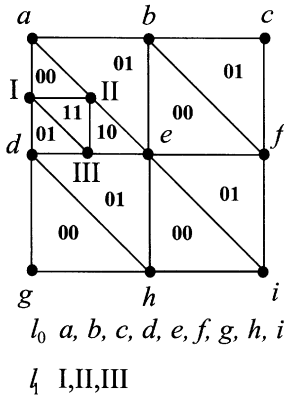


Fig. 12. Different patch type.

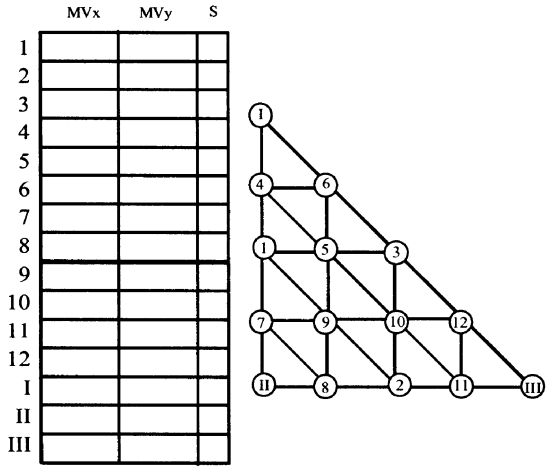


Fig. 13. Structure of the register array (3 levels triangulation).

are used for each motion component since the search window size is [−7, 7] for each node. The motion values are normalized [0, 14] and the value 15 is used as initial value to indicate that the motion is not estimated yet. The patch type is as shown in Fig. 12, the patch type is used for coding purposes.

C. Register Array

The register array unit stores the motion vectors and the locations of the mesh nodes. Fig. 13 shows the structure of the register array for three levels of triangulation. The register array stores the motion vectors of the mesh nodes, in addition to a status bit to indicate the validation of the current value. Notice that the nodes I, II, and III are coarse level nodes, while nodes 1–12 are the full splitting mesh nodes.

D. Control Unit

The control unit synchronizes the operation between different blocks. The control unit is designed as finite state machine and it is summarized in Fig. 14. The mesh-based motion estimation architecture uses 10 states. To describe the operation of the motion estimation architecture, assume a square mesh, as shown in Fig. 12 and the mesh, at level 0, has eight triangular patches.

In State 0, previous and current frames are loaded. In State 1, the queue is initialized. The mesh nodes, at level 0, are added in

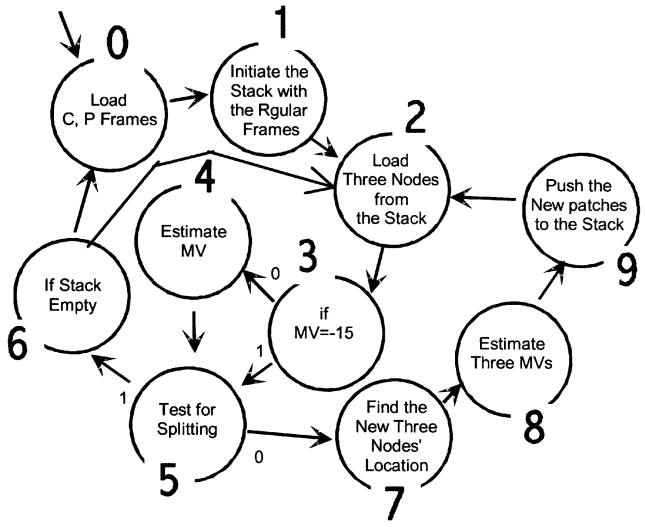


Fig. 14. Mesh coding and motion estimation FSM.

the sequence that is shown in Fig. 40. Each patch has one queue entry and the fields of the queue entry is set to its proper values as shown in Fig. 40.

E. Splitting Unit

The splitting unit generates the mesh nodes according to a threshold function. In this prototype, the threshold function is considered as the difference between the motion vectors. Fig. 16 shows the gate level implementation for the splitting nodes. In this implementation, 4 b motion vector representation is considered since the maximum threshold is of value four.

F. Mesh Node Motion Estimation Unit

The node motion estimation unit estimates the motion vector for each node in the mesh topology. It implements a block matching motion estimation architecture. It surrounds each node with a block of pels, then it uses the TSS and the MAD [22] to estimate the motion vector of each node. The three-step motion estimation algorithm optimizes the architecture for low power.

The node motion estimation unit is a real time three-step block matching motion estimation architecture. The proposed architecture is based on the real time three-step motion estimation architecture. It uses real-time three steps search algorithm, in which the search can be interrupted any time and the architecture reports the current motion estimation as a motion vector. The architecture benefits from the additive property of the three steps search algorithm. The performance analysis scrutinizes further the advantage of implementing of the MAD as block matching criterion. The performance analysis shows that the architecture requires 41 kHz for one block three step motion estimation and 17.6 kHz for only one step. The motion estimation core supports an extremely high bandwidth requirement with the minimum input–output (I/O) pins and hardware resources and it has the following specifications:

a) 16×16 pels block size;
b) window size -8 to $+7$ pels;
c) MAD is used as a block matching criteria;

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 11:59:48 EDT from IEEE Xplore. Restrictions apply.

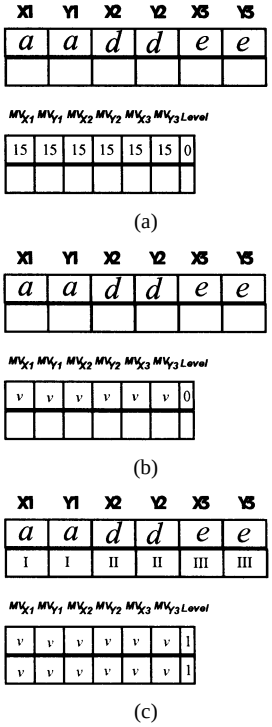


Fig. 15. Register array: (a) after loading the first patch, (b) after estimating the motion vectors, (c) after splitting occurs.

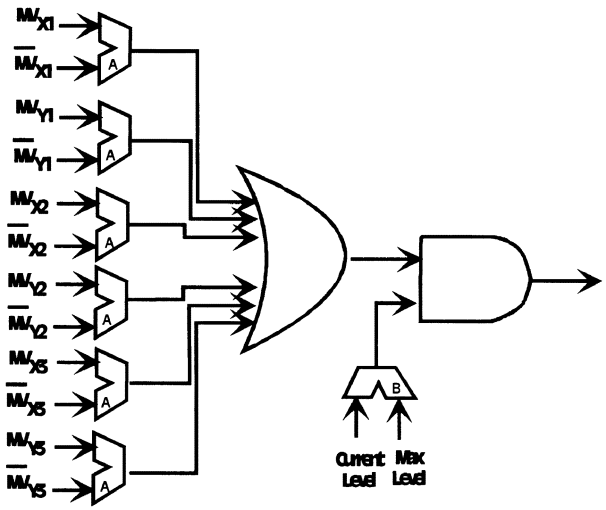


Fig. 16. Mesh splitting logic.

d) real-time search algorithm is used in the sense that the search can be interrupted at any step and the most recent motion vector will be reported.

Fig. 17 shows the block diagram for the proposed architecture. It consists of processor array unit and motion vector generation unit. The processor array is the computational core for the architecture. It calculates nine MAD values using the reference block and a search area. Then it sends them to the MVG, which compares the MADs and decide the motion vector with the minimum MAD. The control unit generates the memory address to supply the motion estimation unit with the exact addresses for the pels value in the current and previous frame.

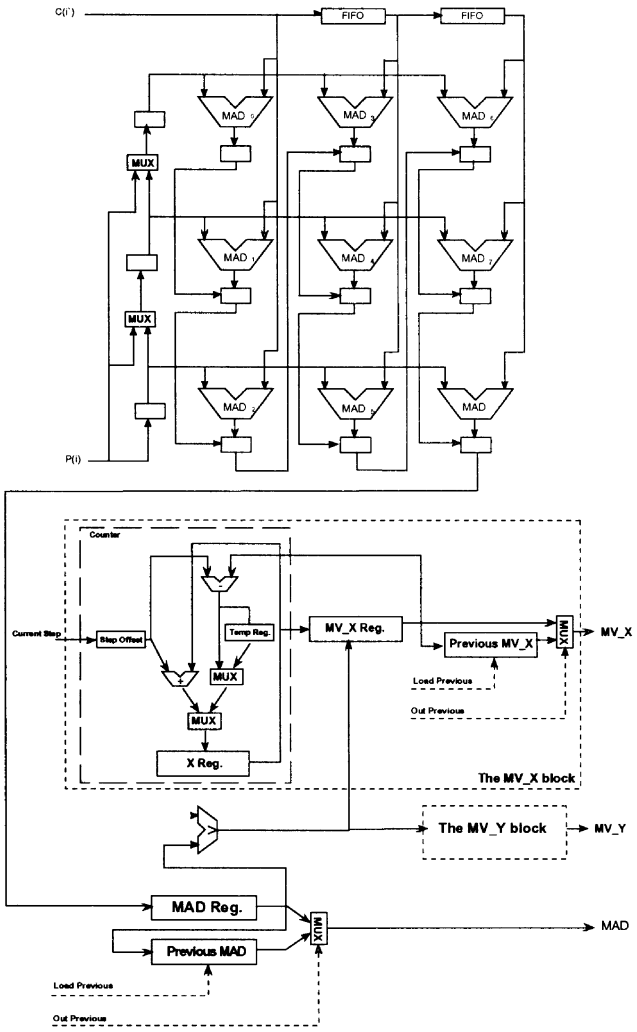


Fig. 17. Block diagram for the TSS motion estimation unit.

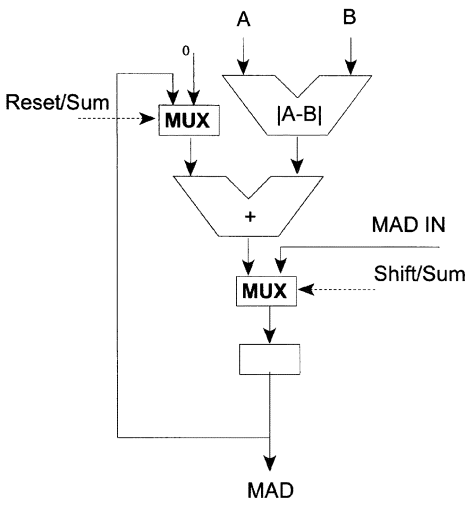


Fig. 18. Processor element structure (MAD architecture).

The processing array consists of nine MAD units, two first-in first-out (FIFO), and three registers. Each MAD unit processes a pel at a time. Each unit generates a MAD by using the block and its counter part in the previous frame. The $C(i)$ and $P(i)$ buses

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 11:59:48 EDT from IEEE Xplore. Restrictions apply.



Fig. 19. Frame allocation for Lena (512×512).

are used to deliver the 2-pels data for the MAD unit in each column calculate three MAD in the same column simultaneously. The two FIFOs are used to adjust the timing gaps needed for three columns MAD to achieve the maximum pipeline granularity.

The MAD processing element calculates the sum of the MAD between two blocks in two different frames. Fig. 18 shows the structure of one of the processor elements (PE). The PE processes a pel from each block at a time. It includes a mechanism to shift the MAD coming from a neighbor element, from right to left. All the motion vectors are moved to the MVG, which generates the motion vector by comparing the incoming MADs.

The motion vector generator collects the motion vectors from nine MAD units in each step. Fig. 18 shows only the details of the motion vector generator in the X direction and the Y direction is similar. The value that is stored in X register, at any instance, may be considered as a final or an intermediate motion vector. In another words, if the system runs out of resources, it will consider the current motion vector as final value, otherwise it will continue the three steps.

V. 2-D Mesh-Based Motion Compensation Architecture

The motion compensation architecture is shown in Fig. 20. It uses four scalable affine units, each having independent access to the frame buffer. The size of the frame to map an $N \times M$ image will be $(N + 14) \times (M + 14)$ since the motion vector value is limited by $[-7, 7]$. In this prototype, the frame buffer is divided into four quadrants as shown in Fig. 19. The buffers are not disjoined; there is an overlap region between buffers to accommodate the parallel operation. The mesh coding buffer unit loads the hierarchical mesh. The memory serialization is used to synchronize that data writing operation to the current frame. The control unit synchronizes the parallel affine unit operations. The operation of the control unit is summarized in Fig. 21.

A. Scalable Affine Unit

The scalable affine unit warps the triangular patch. It is scalable in the sense that it uses more cascaded units with the increase of the mesh hierarchical levels. The block diagram for the affine unit is shown in Fig. 22. The unit consists of a queue to store the mesh at level i. 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. The structure of the warping unit is shown in Fig. 23 and the steps of evaluating the constant are shown in Fig. 24.

B. Mesh Code Buffer

The mesh code buffer stores the code of the adaptive structured mesh. The mesh initially is a regular triangular mesh. Each triangular patch is split into hierarchical triangulation. This triangulation is represented a sequence of "1" and "0" where each "1" represents a split. Each split is associated with three motion vectors that represent the motion of the three nodes.

The structured hierarchical mesh is a technique borrowed from 3-D modeling for mesh construction. Fine-to-coarse and coarse-to-fine are two approaches used to define the hierarchical mesh representation. The former generates a fine mesh, removes selected nodes from a full-resolution mesh and generates the hierarchical mesh. In the latter, a coarse-to-fine mesh, the number of nodes and triangles increases gradually based on a specified threshold function. The adding of nodes is based on successive triangulation to minimize the mesh coding.

C. Memory Serialization Unit

The memory serialization unit is used to serialize the memory writing. Write requests are issued by the parallel affine unit architectures. Fig. 25 shows the structure of the memory serialization unit for two parallel affine units. This structure can be extended to any number of parallel units. The A/D_i are the

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 11:59:48 EDT from IEEE Xplore. Restrictions apply.

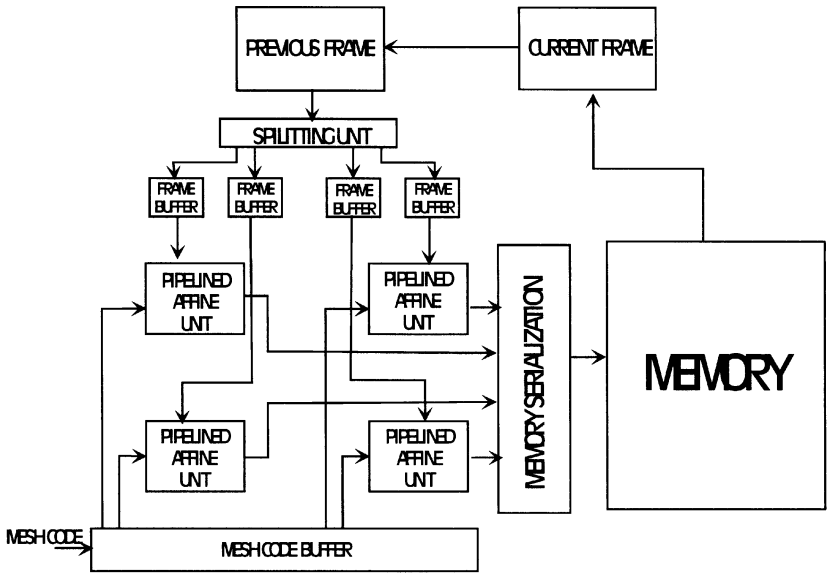


Fig. 20. Block diagram for the mesh-based motion compensation architecture.

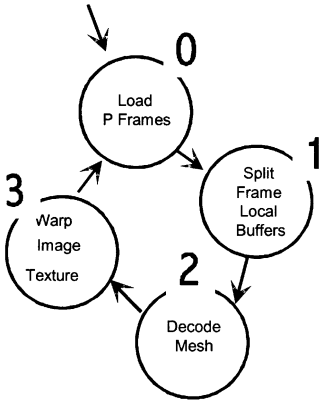


Fig. 21. Motion compensation FSM.

memory address and data for the predicted frame. They are generated by the affine unit i. W_i is the data valid signal, which indicates data needed to be written to the memory. E_i is a control line that serializes the parallel data to be transferred to the main memory. The prototype implements an 8-b address and 8-b data with a 16-b A/D_i bus. Fig. 26 shows the timing diagram for the memory serialization unit. It considers four parallel affine units. The writing process is scheduled among the four units using the E_i signal where $i = 0, 1, 2, 3$. Notice that the memory write control signal MW is raised every other clock cycle, which allows the use of slow on-chip cache. Using QCIF format and 15 frames/s $188 \times 172 \times 2 \times 15$ memory write cycles/s are needed. The minimum frequency will be 1.031 MHz.

VI. Performance Analysis

To measure the performance of the proposed mesh-based video representation and the motion estimation architecture, several experiments have been conducted. The architecture has been prototyped in C++ to study the visual performance and then it has been prototyped on silicon to study the operations performance.

A. Response Time of the Motion Estimation

The response time is one of the most important parameters for the online video applications. It is defined as the latency of the motion estimation algorithm. The response time is highly dependent on the details of the building blocks. To evaluate the performance of the proposed motion estimation algorithm, the computational cost is measured as a number of blocks processed. Figs. 27 and 28 show the performance of the motion estimation algorithm using four different algorithms: 1) The three steps motion algorithm with 8×8 block (TSS-8); 2) the fixed 2-D mesh using 16-pels resolution (Q-1-16); 3) the fixed 2-D mesh using 32 pels resolution (Q-1-32); and 4) the motion estimation using the proposed hierarchical adaptive structure mesh with two levels of details (Q-2-32). Note that the computational cost considers only the maximum number of the block because the performance of the proposed algorithm depends on the content of the video sequence.

Fig. 27 shows the performance of the algorithms using QCIF, 256×256 and CIF video format. Fig. 28 shows the performance of the algorithms using 512×512, CCIR-601, and HDTV video format. It shows about 90% savings using the mesh-based motion estimation technique compared to the block matching techniques. It shows that at a certain mesh size, as the coarse mesh resolution increases the computational efforts decreases because of the reduction in the number of the mesh nodes. Fig. 29 shows the performance saving of using different video formats compared to the TSS-8 algorithm. It shows that the saving is a function of the size of the video-frame. It is shown that the smaller size of the video-frame results in more saving than the larger size. Notice that the HDTV achieves about 14% more saving than the QCIF using the Q-2-32 algorithm.

Fig. 30 shows the number of addition operations using the scan line and the new multiplication-free algorithm for different frame sizes. In this evaluation, one multiplication operation is performed using 11 addition operations. It shows that the new algorithm outperform the scan-line algorithm. It is low power

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 11:59:48 EDT from IEEE Xplore. Restrictions apply.

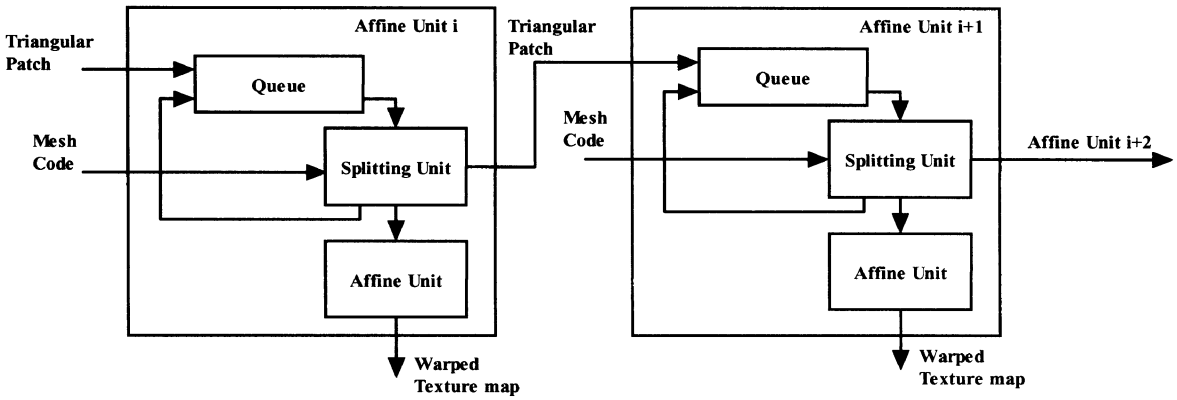


Fig. 22. Block diagram for scalable affine unit.

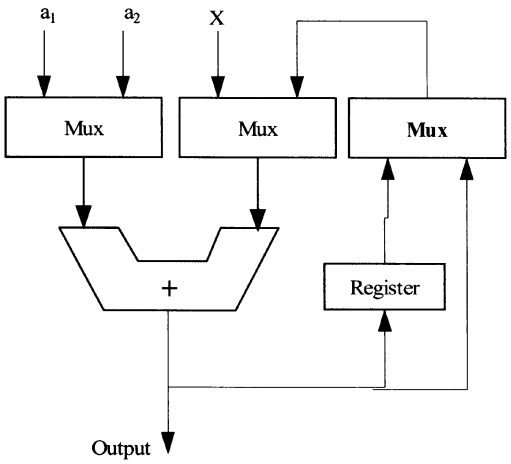


Fig. 23. Patch warping unit.

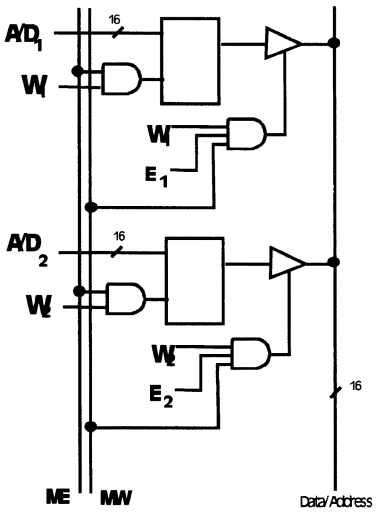


Fig. 25. Memory serialization unit.

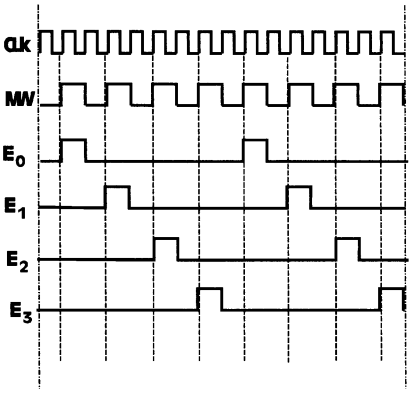


Fig. 26. Timing diagram for the memory serialization unit.

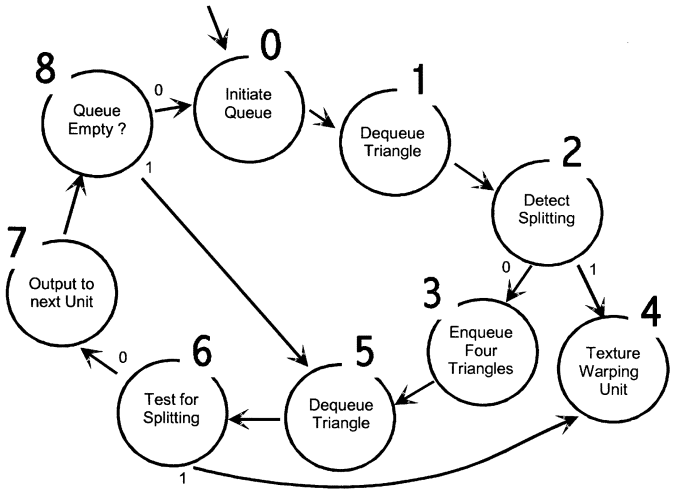


Fig. 24. Affine unit FSM.

technique since it needs less computational effort. Moreover, it reduces the latency of the operation due to the savings in the number of operations. Fig. 31 shows the percentage saving of the computation cost. It is noticed that the saving increases when the size of the mesh patches decreases due to increase in the number of the patches.

B. Mesh Code Size

The size of the mesh code defines the required bandwidth. The generated code size outperforms the currently used mesh coding techniques. To compare the proposed technique with the currently used techniques, the same benchmark, as in [33], is used. Fig. 32 shows the number of bits that is used to code the mesh nodes using (512×512 frame size). It compares the proposed technique with the 6 b/node coding, the predictive coding proposed in [20] and the Benoit approach [33]. Fig. 32 shows

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 11:59:48 EDT from IEEE Xplore. Restrictions apply.

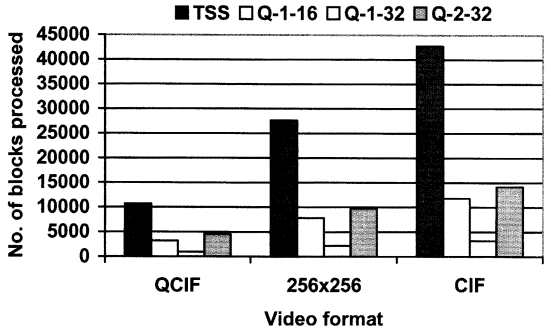


Fig. 27. Performance of the TSS, Q-1-16, Q-1-32, and Q-2-32 motion estimation algorithms using QCIF, 256 × 256 and CIF formats.

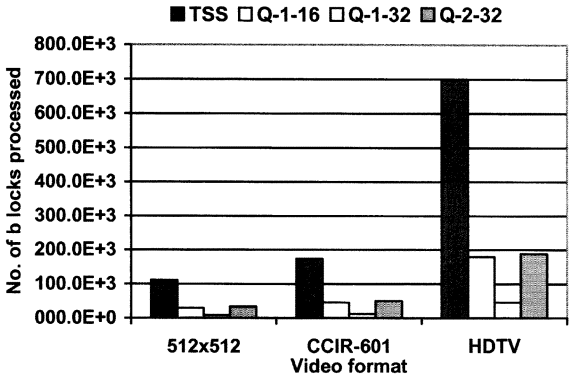


Fig. 28. Performance of the TSS, Q-16, Q-1-32, and Q-2-32 motion estimation algorithms using 512 × 512, CCIR-601, and HDTV formats.

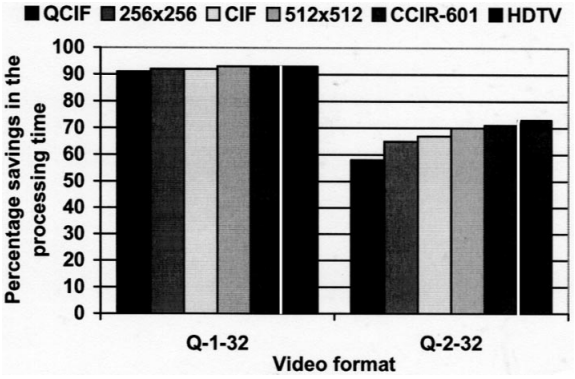


Fig. 29. Percentage savings in the processing time for the Q-1-32 and Q-2-32 motion estimation algorithms using QCIF, 256 × 256, CIF, 512 × 512, CCIR-601, and HDTV formats.

the number of nodes versus the number of coding bits. In this figure the proposed technique outperforms the other methods since the coding of the splitting operation can code more than one node using one bit about 60% gain can be achieved over the other methods. This amount can be used to enhance the quantization or increase the number of node per mesh, which results in a better quality.

C. Visual Quality Size

The visual quality is determined as the statistical peak signal to noise ratio (PSNR), which is evaluated as the following:

$$PSNR = 10 \times \log \left[\frac{255^2}{\frac{1}{N_v N_h} \sum_{m=1}^{N_v} \sum_{n=1}^{N_h} (c_{m,n} - \tilde{c}_{m,n})^2} \right]$$

where $c_{m,n}$ are pels of the original video object plan (VOP) at location (m, n) and $\tilde{c}_{m,n}$ are pels of the reconstructed VOP.

To study the visual quality of the proposed architecture, a C++ prototype that generates the proposed 2-D mesh and the motion vectors has been developed. The motion compensation algorithm warps the triangular patches (it maps the texture) to predict the frame. It uses the affine transformation, which is defined as $X = a_{i1}x + a_{i2}y + a_{i3}$ and $Y = a_{i4}x + a_{i5}y + a_{i6}$. The affine transformation can be calculated after the constants $a_{i1} - a_{i6}$ are evaluated.

Fig. 33 shows the PSNR for the first 90 frames of the QCIF carphone video sequence. It shows the PSNR using the 2-D mesh with coarse size of 32 pels with 1, 2, and 4 mesh levels (Q-1-32, Q-2-32, and Q-4-32). Fig. 34 shows the rate distortion for the carphone sequence using the 2-D mesh with coarse size of 32 pels with 1 and 4 mesh levels (Q-1-32 and Q-4-32). Fig. 35 shows the PSNR for the first 30 frames of the QCIF Claire video sequence. The figure shows the PSNR using the 2-D mesh with coarse size of 16, 32, and 64 pels with one mesh level (Q-1-16, Q-1-32, and Q-1-64). Figs. 33–41 indicate clearly the superiority of the proposed algorithm. Moreover, it shows the increase in the mesh level will enhance the PSNR, since more information can be used for motion compensation and smaller initial resolution will contribute to the performance enhancement since more information will be known. The rate-distortion curves shows that the proposed algorithm is better than the regular mesh approach.

In State 2, three mesh nodes are loaded from the queue and stored in the register array as shown in Fig. 15(a). In State 3, a check if the motion vector is equal to 15 will take place. Then in State 4, the motion vectors will be estimated and placed in the register array, as shown in Fig. 15(b). Note that State 4 will be executed only for patches in the level 0. In State 5, a check for splitting will take place and three new nodes may be placed in the register array as in State 7. In State 8, the motion vectors will be estimated for the new three nodes. In State 9, the nodes will be pushed to the queue in the proper node order. Fig. 41 shows the queue after splitting the first patch. In State 6, a check for an empty queue condition will take place and the system may start at state 0.

The PSNR of the foreman, carphone, and salesman QCIF sequence is presented in Figs. 38–41. The PSNR of the small mesh size outperforms the large size of the mesh patch since it benefits from the extra local information.

VII. Architecture Prototype

The architecture has been prototyped using 0.6-μm CMOS technology with three layers of metal. The design methodology

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 11:59:48 EDT from IEEE Xplore. Restrictions apply.

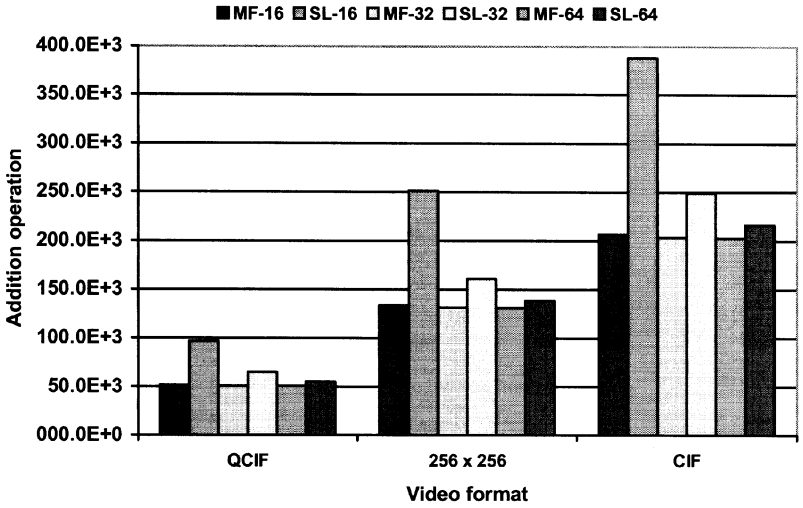


Fig. 30. Performance of the multiplication free MF and the scan line SL for 16, 32, and 64 mesh resolution using QCIF, 256 × 256, and CIF formats.

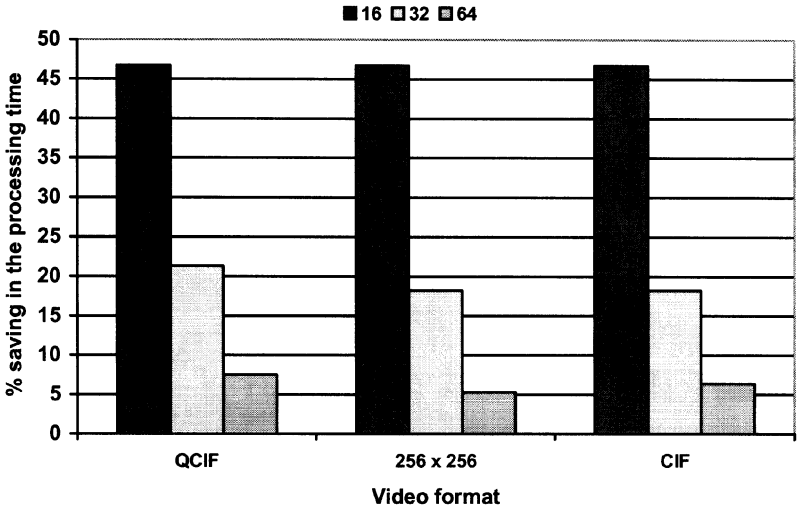


Fig. 31. Percentage of performance saving for the multiplication free MF compared to the scan line SL for 16, 32, and 64 mesh resolution using QCIF, 256 × 256, and CIF formats.

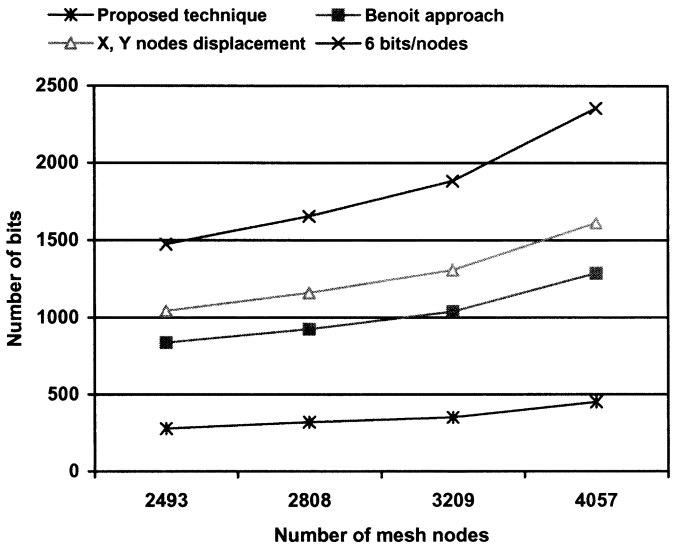


Fig. 32. Number of bits versus number of nodes.

used in the prototyping is shown in Fig. 39. Synopsys tools are used for the "front-end" steps, which capture and simulate the desired behavior of the architecture, while Cadence tools are used for the "back-end" steps.

The behavior of the architecture is modeled using Verilog and is synthesized, simulated and analyzed using Synopsys. Then, Cadence is used for schematic capture and modification using gate-level Verilog code to generate the final layout.

The technology and symbol libraries used in both Synopsys and Cadence for synthesis and simulation are generated based on the standard-cell library. This guarantees the synthesized design has simulation results close to the final layout.

Transferring the design from Synopsys to Cadence is accomplished by generating a Verilog netlist, which creates symbol and schematic views for all levels for the design hierarchy. Power and ground pads are then added to the top-level schematic.

Two groups of simulations are used, one tests the correctness of the design at different design steps, the other measures different performance parameters for the proposed architecture.

Many verification simulation steps are involved in the design flow. The first one uses Verilog simulation tools to verify the correctness of the architecture's behavior. The second is applied to confirm the functionality of the extracted gate level circuit.

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 11:59:48 EDT from IEEE Xplore. Restrictions apply.

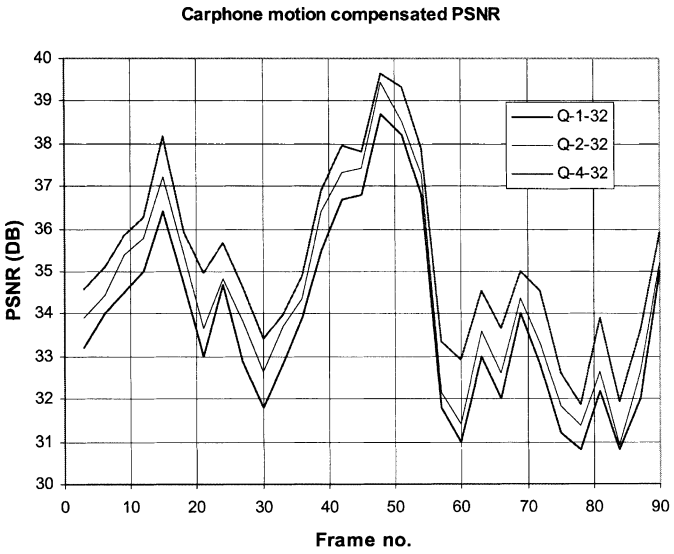


Fig. 33. PSNR for Q-1-32, Q-2-32, and Q-4-32 motion compensation algorithm using carphone QCIF format.

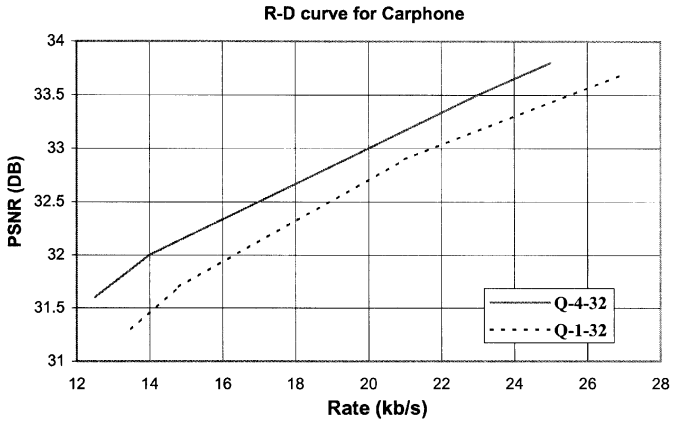


Fig. 34. Rate distortion for Q-1-32 and Q-4-32 using carphone QCIF format.

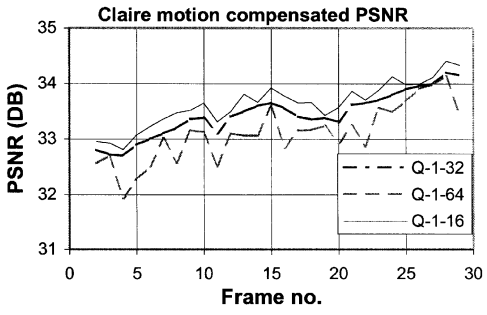


Fig. 35. PSNR for Q-1-16, Q-1-32, and Q-4-64 motion compensation algorithm using Claire QCIF format.

The third ensures that the gate-level schematic, which has been imported and edited in Cadence, is equivalent to the proposed architecture. These are followed by the design rule checking (DRC) and the layout versus schematic (LVS), which are done with Cadence Diva. Once LVS is done, it is assumed that the layout is functionally correct.

The architecture has been prototyped using 0.6-μm CMOS technology with three layers of metal. The layout of prototype

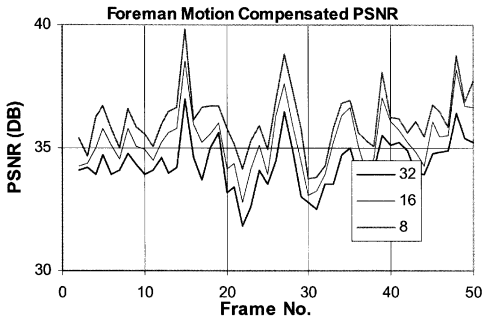


Fig. 36. PSNR for mesh size 8, 16, and 32 using foreman QCIF format.

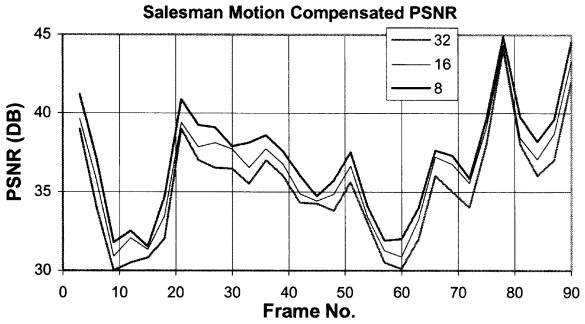


Fig. 37. PSNR for mesh size 8, 16, and 32 using salesman QCIF format.

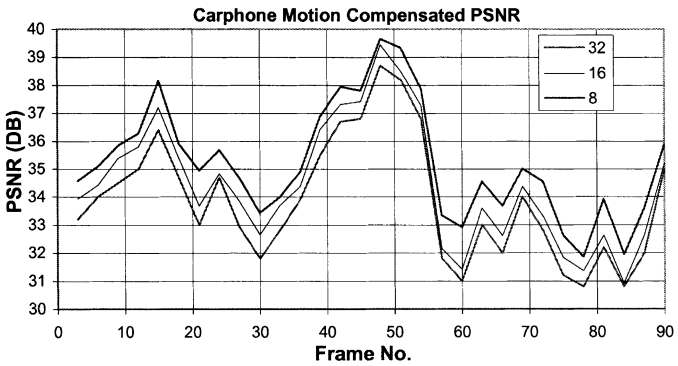


Fig. 38. PSNR for mesh size 8, 16, and 32 using carphone QCIF format.

of the architecture has a total area of $9807.20 \times 7683.96\ \mu m^2$ for the motion estimation architecture and $4807.20 \times 4183.96\ \mu m^2$ for the motion compensation architecture. The power and delay is estimated using HSPICE simulation. According to the Synopsys extraction and SPICE simulation, the motion estimation architecture is expected to consume power less than 0.6 W. The delay and power consumption for different building blocks in the motion estimation architecture are shown in Table I. The delay for the motion compensation architecture is 58.9 ns while the power consumption is 562.45 mW. The delay and power consumption for different building blocks are shown in Table II.

VIII. Conclusion

This paper presents a VLSI architecture for mesh-based motion tracking. The new architecture uses a novel hierarchical mesh coding technique. It optimizes the mesh coding using an adaptive structured mesh to capture the contents of the video

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 11:59:48 EDT from IEEE Xplore. Restrictions apply.

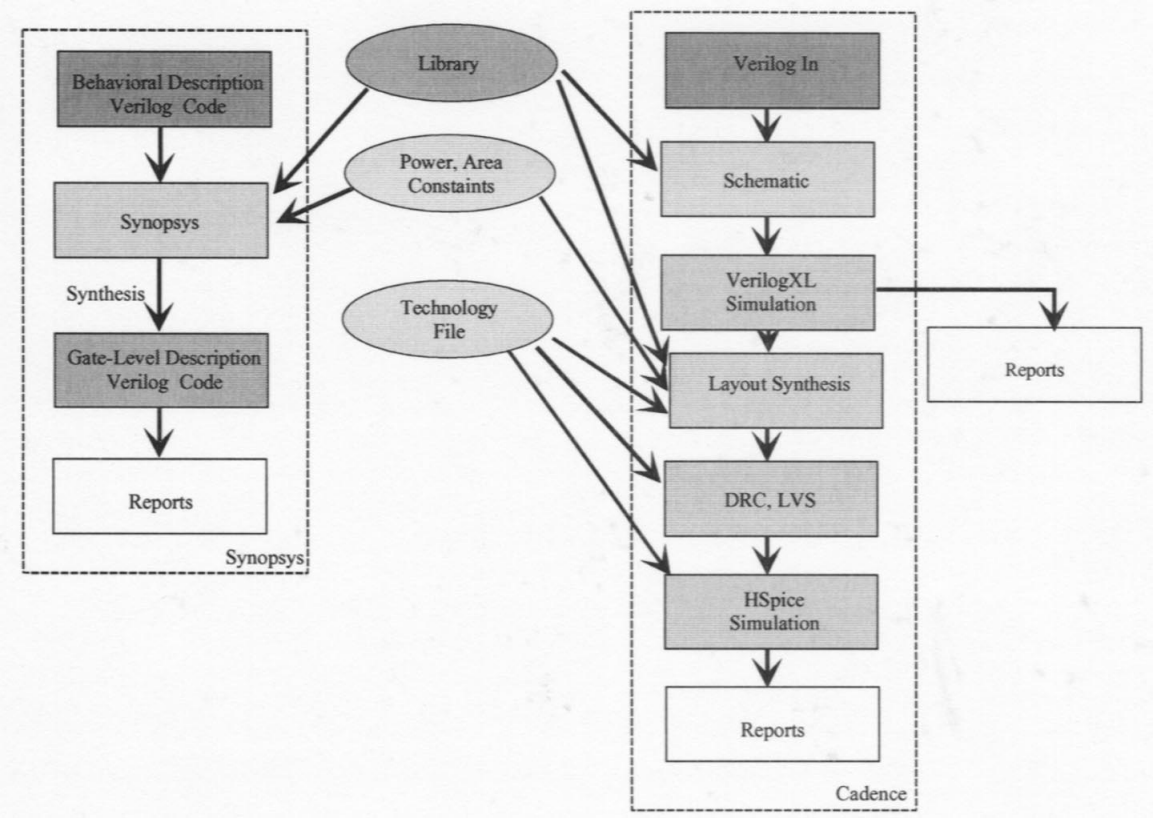


Fig. 39. Prototyping steps.

Nodes	X1	Y1	X2	Y2	X3	Y3	Mx1	My1	Mx2	My2	Mx3	My3	L	T
a,d,e	v	v	v	v	v	v	15	15	15	15	15	15	0	00
a,e,b	v	v	v	v	v	v	15	15	15	15	15	15	0	01
b,e,f	v	v	v	v	v	v	15	15	15	15	15	15	0	00
b,f,c	v	v	v	v	v	v	15	15	15	15	15	15	0	01
d,g,h	v	v	v	v	v	v	15	15	15	15	15	15	0	00
d,h,e	v	v	v	v	v	v	15	15	15	15	15	15	0	01
e,h,i	v	v	v	v	v	v	15	15	15	15	15	15	0	00
e,i,f	v	v	v	v	v	v	15	15	15	15	15	15	0	01

Fig. 40. Queue initialization for the mesh shown in Fig. 12 at the coarse level (l_0).

Nodes	X1	Y1	X2	Y2	X3	Y3	Mx1	My1	Mx2	My2	Mx3	My3	L	T
a,e,b	v	v	v	v	v	v	15	15	15	15	15	15	0	01
b,e,f	v	v	v	v	v	v	15	15	15	15	15	15	0	00
b,f,c	v	v	v	v	v	v	15	15	15	15	15	15	0	01
d,g,h	v	v	v	v	v	v	15	15	15	15	15	15	0	00
d,h,e	v	v	v	v	v	v	15	15	15	15	15	15	0	01
e,h,i	v	v	v	v	v	v	15	15	15	15	15	15	0	00
e,I,f	v	v	v	v	v	v	15	15	15	15	15	15	0	01
a,II,III	v	v	v	v	v	v	v	v	v	v	v	v	1	00
I,e,II	v	v	v	v	v	v	v	v	v	v	v	v	1	01
III,II,d	v	v	v	v	v	v	v	v	v	v	v	v	1	10
I,II,III	v	v	v	v	v	v	v	v	v	v	v	v	1	11

Fig. 41. Queue after the splitting in the finer level (l_1) for the mesh shown in Fig. 12.

sequence. The structured mesh offers a significant reduction in the mesh topology coding (about 60%).

The proposed architecture offers low power performance at the algorithmic level by: 1) decomposing the mesh into independent triangular patches that can be processed in parallel; 2) using a hierarchical structured triangulation technique, that simplify the processing unit; 3) using the three steps motion estimation algorithm to simplify the motion estimation of the mesh nodes; 4) using finite space analysis and a multiplication-free affine transformation, which significantly reduce the complexity of the

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 11:59:48 EDT from IEEE Xplore. Restrictions apply.

TABLE I
POWER AND DELAY DIFFERENT BUILDING BLOCK

Module	Delay (ns)	Power (mW)
Control Unit	6.3	47.14
Address Generator	13.2	29.16
TSS	11.7	209.68
MVU/Split Unit	18.9	123.6
Architecture	29.7	589.58

TABLE II
POWER AND DELAY DIFFERENT BUILDING BLOCK

Module	Delay (ns)	Power (mW)
Control Unit	5.6	16.73
Queue	26	4.17
Register	24.6	14.58
Affine Unit	28.3	123.45
Architecture	58.9	562.45

motion compensation architecture; and 5) pipelining the affine unit contributes a considerable power savings.

The architecture is prototyped using 0.6-μm CMOS technology with three layers of metal. The performance results of the prototype show that the architecture can be used for online applications and the power consumption shows that it is suitable for mobile applications.

ACKNOWLEDGMENT

The authors would like to thank the Natural Sciences and Engineering Research Council of Canada, the Canadian Foundation for Innovations, the University Research Grant Committee (URGC), Micronet, Alberta Innovation and Science—Research Excellence Envelope and the Canadian Microelectronics Corporation for supporting this research.

REFERENCES

[1] A. Nosratinia, "New Kernels for fast mesh-based motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, pp. 40–52, Jan. 2001.
[2] R. M. Haralick, "Automatic remote sensor image processing," in *Topics in Applied Physics*. New York: Springer-Verlag, 1976, vol. 11, Digital Picture Analysis, pp. 5–63.
[3] G. J. Holzmann, *Beyond Photography—The Digital Darkroom*. Englewood Cliffs, NJ: Prentice-Hall, 1988.
[4] G. Wolberg, *Digital Image Warping*. Los Almitos, CA: IEEE Comput. Soc. Press, 1990.
[5] H. Bruzewitz, "Motion compensation with triangles," in *Proc. 3rd Int. Conf. 64 kbit Coding of Moving Video*, Rotterdam, The Netherlands, Sept. 1990.
[6] G. J. Sullivan and R. L. Baker, "Motion compensation for video compression using control grid interpolation," in *Proc. IEEE ICASSP*, vol. 4, May 1991, pp. 2713–2716.
[7] Y. Nakaya and H. Harashima, "An iterative motion estimation method using triangular patches for motion compensation," in *Proc. SPIE Visual Communications and Image Processing*, vol. 1605, Nov. 1991, pp. 546–557.
[8] ——, "Motion compensation based on spatial transformations," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 4, pp. 339–356, June 1994.
[9] J. Nieweglowski, T. Campbell, and P. Haavisto, "A novel video coding scheme based on temporal prediction using digital image warping," *IEEE Trans. Consumer Electron.*, vol. 39, pp. 141–150, Aug. 1993.
[10] C. L. Huang and C. Y. Hsu, "A new motion compensation method for image sequence coding using hierarchical grid interpolation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 4, pp. 42–52, Feb. 1994.
[11] Y. Wang and O. Lee, "Active mesh—A feature seeking and tracking image sequence representation scheme," *IEEE Trans. Image Processing*, vol. 3, pp. 610–624, Sept. 1994.
[12] ——, "Use of two-dimensional deformable mesh structures for video coding, Part I—The synthesis problem: Mesh-based function approximation and mapping," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, pp. 636–646, Dec. 1996.
[13] Y. Wang, O. Lee, and A. Vetro, "Use of two-dimensional deformable mesh structures for video coding, Part II—The analysis problem and a region-based coder employing an active mesh representation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, pp. 647–659, Dec. 1996.
[14] Y. Wang and J. Osterman, "Evaluation of mesh-based motion estimation in h.263 like coders," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 8, pp. 243–252, June 1998.
[15] A. M. Tekalp, P. J. L. van Beek, C. Toklu, and B. Gunsel, "2-D mesh-based visual object representation for interactive synthetic/natural video," *Proc. IEEE*, vol. 86, pp. 1029–1051, June 1998.
[16] C. Toklu, A. Erdem, M. I. Sezan, and A. M. Tekalp, "Tracking motion and intensity variations using heirarchical 2-D mesh modeling for synthetic object transfiguration," *Graph. Models Image Process.*, vol. 58, pp. 553–573, Nov. 1996.
[17] A. Nosratinia, N. Mohsenian, M. T. Orchard, and B. Liu, "Interslice coding of magnetic resonance images using deformable triangular patches," in *Proc. IEEE ICIP*, vol. 2, Austin, TX, Nov. 1994, pp. 898–892.
[18] ——, "Interframe coding of magnetic resonance images," *IEEE Trans. Med. Imag.*, vol. 15, pp. 639–647, Oct. 1996.
[19] N. Mohsenian, A. Nosratinia, B. Liu, and M. T. Orchard, "Adaptive entropy constrained transform coding of maggetic resonance image sequences," *IEEE Trans. Nucl. Sci.*, vol. 42, pp. 2309–2316, Dec. 1995.
[20] O. Lee and Y. Wang, "Non-uniform image sampling and interpolation over deformed and its hierarchical extension," in *Proc. Int. Conf. Visual Communications and Image Processing*, vol. 3, 1995, pp. 289–300.
[21] W. Badawy and M. Bayoumi, "On minimizing hierarchical mesh coding overhead: (HASM) hierarchical adaptive structured mesh approach," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing, ICASSP*, Istanbul, Turkey, June 5–9, 2000, pp. 1923–1926.
[22] D. Terzopoulos and M. Vailescu, "Sampling and reconstruction using adaptive meshes," in *Proc. IEEE Int. Conf. Computer Vision and Pattern Recognition CVPR*, 1991, pp. 70–75.
[23] K. J. R. Liu *et al.*, "Algorithm-Based low power and high-performance multimedia signal processing," *Proc. IEEE*, vol. 86, no. 6, pp. 1155–1202, June 1998.
[24] F. Rocca and S. Zanoletti, "Bandwidth reduction via movement compensation on a model of the random video process," *IEEE Trans. Commun.*, vol. COM-20, pp. 960–965, Oct. 1972.
[25] S. Brofferio and F. Rocca, "Interframe redundancy reduction of video signals generated by translating objects," *IEEE Trans. Commun.*, vol. COM-25, pp. 448–455, Apr. 1977.
[26] G. Wolberg, *Digital Image Warping*. Los Alamitos, CA: Comput. Soc. Press, 1990.
[27] P. E. Eren, C. Toklu, and A. M. Tekalp, "Object-based manipulation and composition using 2-D meshes in VRML," in *Proc. IEEE Signal Processing 1st Workshop Multimedia Processing*, Princeton, NJ, June 1997, pp. 257–261.
[28] G. Seetharaman, "Image sequence analysis for three-dimensional perception of dynamic scenes," in *Handbook of Pattern Recognition and Image Processing: Computer Vision*, T. Y. Young, Ed. New York: Academic, 1994, pp. 361–403.
[29] G. Seetharaman and T. Y. Young, "A region correspondence approach to the recovery of 3-Dimensional motion and structure in dynamic scenes," in *Image Understanding in Unstructured Environment*, S.-S. Chen, Ed, Singapore: World Scientific, 1988, pp. 75–123.
[30] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion-compensated interframe coding for video conferencing," in *Proc. National Telecommun. Conf*, New Orleans, LA, 1981, pp. G.5.3.l–G.5.3.5.
[31] L. Chiariglione, "MPEG and multimedia communications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, pp. 5–18, Feb. 1997.
[32] H. Ghravi and M. Mile, "Block matching motion estimation algorithms: New results," *IEEE Trans. Circuits Syst. II*, vol. 37, pp. 649–651, May 1990.
[33] H. Benoit-Cattin, A. C. Planat, P. R. Joachimsmann, A. Baskurt, P. Clarysse, and I. E. Magnin, "On minimizing the active mesh coding," in *Proc. IEEE Int. Conf. Image Processing*, 1998, pp. 895–898.
[34] K. J. R. Liu *et al.*, "Algorithm-based low power and high-performance multimedia signal processing," *Proc. IEEE*, vol. 86, no. 6, pp. 1155–1202, June 1998.

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 11:59:48 EDT from IEEE Xplore. Restrictions apply.

[35] W. M. Badawy, G. Zhang, and M. A. Bayoumi, "VLSI architecture for hierarchical mesh-based motion estimation," presented at the *Proc. IEEE Workshop on Signal Processing Systems (SiPS)*, Taipei, Taiwan, R.O.C., Oct. 20–22, 1999.
[36] G. Wolberg, *Digital Image Warping*. Los Alamatos, CA: Comput. Soci. Press, 1990.
[37] W. Badawy and M. Bayoumi, "On minimizing hierarchical mesh coding overhead: (HASM) hierarchical adaptive structured mesh approach," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing*, Istanbul, Turkey, June 5–9, 2000, pp. 1923–1926.
[38] P. J. L. van Beek, A. M. Tekalp, and A. Puri, "2-D mesh geometry and motion compression for efficient object-based video representation," presented at the *Proc. Int. Conf. Image Processing*, Santa Barbara, CA, Oct. 1997.
[39] S. A. Khan and V. K. Madisetti, "System partitioning of MCM for low power," *IEEE Des. Test Comput.*, vol. 12, pp. 41–52, Spring 1995.
[40] A. P. Chandrakasan and R. W. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proc. IEEE*, vol. 83, pp. 498–523, Apr. 1995.
[41] H. K. Thapar and J. Cioffi, "A block processing method for designing high speed viterbi decoders," in *Proc. IEEE Communications Conf.*, June 1989, pp. 1096–1100.
[42] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-Power CMOS digital design," *IEEE J. Solid-State Circuits*, vol. 27, pp. 473–484, Apr. 1992.
[43] J. R. Jam and A. K. Jam, "Displacement measurement and its application in interframe image coding," *IEEE Trans. Commun.*, vol. COM-29, pp. 1799–1808, Dec. 1981.
[44] T. H. Meng, R. Broderson, and D. Messerschmitt, "Asynchronous design for programmable digital signal processors," *IEEE Trans. Signal Processing*, vol. 39, pp. 939–952, Apr. 1991.
[45] K. I. R. Liu, S.-F. Hsieh, and K. Yao, "Systolic block householder transformation for RLS algorithm with two-level pipelined implementation," *IEEE Trans. Acoust., Speech. Signal Processing*, vol. 40, pp. 946–958, Apr. 1992.
[46] C. L. Wang, "Bit-serial VLSI implementation of delayed LMS adaptive FIR filter," *IEEE Trans. Signal Processing*, vol. 42, pp. 2169–2175, Aug. 1994.
[47] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1989.



Wael Badawy (S'97–M'01) received the B.Sc. and M.Sc. degrees from the Department of Computer Science and Automatic Control Engineering, University of Alexandria, Alexandria, Egypt, in 1996 and 1997, and the M.S. and Ph.D. degrees from the Center for Advanced Computer Studies, University of Louisiana at Lafayette, in 1998 and 2000.

Currently, he is an Assistant Professor, Department of Electrical and Computer Engineering, University of Calgary, Calgary AB, Canada. He is the author and coauthor of more than 60 refereed journal/conference papers and about 20 technical reports. He edited three books in the area of system on chip. He is the Guest Editor for the special issue on System on Chip for Real-Time Applications in the *Canadian Journal on Electrical and Computer Engineering*. His research involves designing new models, techniques, algorithms, architectures and low power prototype for MPEG consumer products. His research interests are in the areas of VLSI architectures for video applications with low-bit rate applications, digital video processing, video library, watermarking, spatial information, low power design methodologies, microelectronics, and VLSI prototyping.

Dr. Badawy received the 1998 Upsilon Pi Epsilon Honor Society and IEEE Computer Society Award for Academic Excellence in Computer Disciplines, 2001 Micralyne Microsystems Design Award, and the 2002 Petro Canada Young Innovator Award. He is a technical reviewer for several IEEE journals and conferences. He is currently the Technical Chair for the 2002 International Workshop on SoC for real-time applications and a member of the IEEE-CAS Technical Committee on Communication, VLSI, and BioCAS.



Magdy A. Bayoumi (S'80–M'84–SM'87–F'99) received the B.Sc. and M.Sc. degrees in electrical engineering from Cairo University, Cairo, Egypt, in 1973 and 1977, the M.Sc. degree in computer engineering from Washington University in St. Louis, MO, in 1981, and the Ph.D. degree in electrical engineering from the University of Windsor, Windsor, ON, Canada, in 1984.

Currently, he is the Director of the Center for Advanced Computer Studies (CACS), Department Head of the Computer Science Department, the Edmiston Professor of Computer Engineering, and the Lamson Professor of Computer Science at the Center for Advanced Computer Studies, University of Louisiana at Lafayette, where he has been a faculty member since 1985. He has edited and coedited three books in the area of VLSI Signal Processing. He was an Associate Editor of the *Circuits and Devices Magazine* and is currently an Associate Editor of *Integration*, the *VLSI Journal*, and the *Journal of VLSI Signal Processing Systems*. He is a Regional Editor for the *VLSI Design Journal* and on the Advisory Board of the *Journal on Microelectronics Systems Integration*. His research interests include VLSI design methods and architectures, low power circuits and systems, digital signal processing architectures, parallel algorithm design, computer arithmetic, image and video signal processing, neural networks, and wideband network architectures.

Dr. Bayoumi received the University of Louisiana at Lafayette 1988 Researcher of the Year Award and the 1993 Distinguished Professor Award. He was an Associate Editor of the IEEE CIRCUITS AND DEVICES MAGAZINE, the IEEE TRANSACTIONS ON VLSI SYSTEMS, the IEEE TRANSACTIONS ON NEURAL NETWORKS, and the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: ANALOG AND DIGITAL SIGNAL PROCESSING. From 1991 to 1994, he served on the Distinguished Visitors Program for the IEEE Computer Society, and he is on the Distinguished Lecture program of the Circuits and Systems society. He was the Vice President for the technical activities of the IEEE Circuits and Systems Society. He was the Cochairman of the Workshop on Computer Architecture for Machine Perception, in 1993 and is a member of the Steering Committee of this workshop. He was the General Chairman of the 1994 MWSCAS and is a member of the Steering Committee of this symposium. He was the General Chairman for the 8th Great Lake Symposium on VLSI, in 1998. He has been on the Technical Program Committee for ISCAS for several years and he was the Publication Chair for ISCAS'99. He was also the General Chairman of the 2000 Workshop on Signal Processing Design and Implementation. He was a founding member of the VLSI Systems and Applications Technical Committee and was its Chairman. He is currently the Chairman of the Technical Committee on Circuits and Systems for Communication and the Technical Committee on Signal Processing Design and Implementation. He is a member of the Neural Network and the Multimedia Technology Technical Committees. Currently, he is the faculty advisor for the IEEE Computer student chapter at University of Louisiana at Lafayette.

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 11:59:48 EDT from IEEE Xplore. Restrictions apply.