TRUE MOTION ESTIMATION — THEORY, APPLICATION, AND IMPLEMENTATION

Yen-Kuang Chen

A DISSERTATION

PRESENTED TO THE FACULTY

OF PRINCETON UNIVERSITY

IN CANDIDACY FOR THE DEGREE

OF DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE

BY THE DEPARTMENT OF

ELECTRICAL ENGINEERING

November 1998

Abstract

This thesis offers an integrated perspective of the theory, applications, and implementation of *true motion estimation*. Taking the pictures of 3D real-world scene generates sequences of video images. When an object in the three-dimensional real world moves, there are corresponding changes in the brightness—or luminance intensity—of its two-dimensional image. The physical three-dimensional motion projected onto the two-dimensional image space is referred to as "true motion." The ability to track true motion by observing changes in luminance intensity is critical to many video applications. This thesis explores techniques that track such motion and shows how these techniques can be used in many important applications.

On the theoretical side, three fundamental issues are explored: (1) the intensity-conservation principle, (2) basic matching- and gradient- measurement, and (3) four levels of constraints for motion-consistency (i.e., block-, object-, neighborhood-, temporal-). Various existing and future true-motion-estimation algorithms can be constructed from these theoretical bases. Based on the theoretical development, we have built a true motion tracker (TMT) using a *neighborhood relaxation* formulation.

From an application perspective, the TMT successfully captured true motion vectors in our experiments for many video applications. For example, in *MPEG video compression*, the use of true motion vectors on individual macroblocks can optimize the bit rate for residual information and motion information. The TMT also offers significant improvement in motion-compensated *spatial- and temporal- video interpolation*, e.g., frame-rate up-conversion and interlaced-to-progressive scan conversion. Another piece of evidence

that demonstrates the effectiveness of the TMT is its successful application to *object motion estimation* and *video-object segmentation*, both of which are vital preprocessing steps for object-based video processing in MPEG-4 and MPEG-7 applications.

In regard to implementation, although the proposed TMT is computation-demanding and control-intensive, we have an effective system design of the TMT. We tackle the great challenge by (1) partitioning the TMT into two parts—computationally intensive and control-intensive and (2) supporting both parts with a multimedia architecture consisting of a core-processor and a processing array. The first computation-demanding part of the TMT is efficiently conducted on the processing array, and the other control-intensive part of the TMT is easily executed on the core-processor.

Acknowledgements

I would like to express my sincere appreciation to Professor Sun-Yuan Kung, my advisor, for his extensive and invaluable guidance, support, and encouragement, which helped me accomplish the doctorate degree and prepared me to accomplish more life goals in the future. In addition, I could never express enough gratitude to his wife, Mrs. Se-Wei Kung, for her hearfelt concern.

I would like to thank Professor Michael Orchard and Dr. Huifang Sun for their precious time spent on the review of the work and for their valuable suggestions and comments. This work has benefited from many stimulating discussions with John Chi-Hong Ju, Dr. Yun-Ting Lin, and Anthony Vetro.

I would like to thank Michael Dorn, David Driscoll, Tailoong Hsu, Milton Leebaw, Chihpin Tu, and Ivy Yip, who helped me improve my English during the study and provided valuable feedback that greatly improved the clarity of the work.

I would like to thank my uncle and aunt, Mr. & Mrs. Suei-Ho & July Chang, my cousins, Dr. & Mrs. Min-Tsong & Michelle Chang and Mr. & Mrs. Peter Min-Yau & Sherry Chang, for their ardent support during my study aboard. The support of this work by *George Van Ness Lothrop Fellowship* is also acknowledged.

Most of all, I would like to thank my parents, Mr. & Mrs. Hwang-Huei & Mei-Ching Chen for the constant support and encouragement I needed to survive graduate study.

Yen-Kuang Chen

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This work is on digital video processing and is concerned specifically with true motion estimation. There have been two major revolutions in television history. The first occurred a half century ago in 1954 when the first color TV signals were broadcast. Today, black-and-white TV signals have disappeared entirely from the airwaves. The second revolution is now imminent. By the end of 1998, *digital TV* signals* will be broadcast on the air [6]. By the end of 2006, traditional *analog TV* signals† will have disappeared from the airwaves just as completely as black-and-white signals have now. Digital TV is more than just theater-quality entertainment at home; it also allows many multimedia applications and services to be introduced. The digital video processing technology discussed in this thesis is closely linked to the second and imminent revolution.

While there are various research topics in the field of digital video processing, this work focuses primarily on motion estimation techniques. Video processing differs from image processing in that most objects in the video move. Understanding how objects move helps us to transmit, store, understand, and manipulate video in an efficient way. Algorithmic development and architectural implementation of motion estimation techniques have been major research topics in multimedia for years.

*A digital video signal is in a discrete digital coded number form suitable for digital storage and manipulation.

†An analog Video signal is in a continuously varying voltage form.

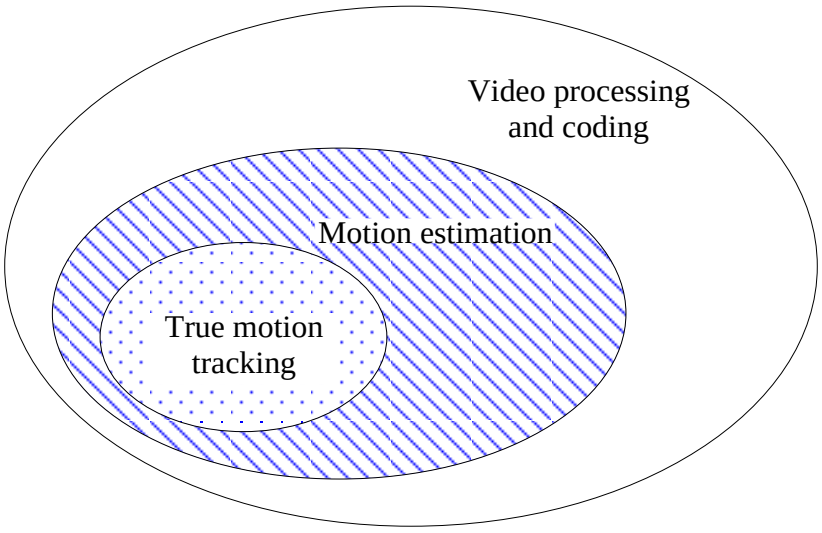


Figure 1.1: The scope of this work is about a digital video processing technique. Among various research topics in the digital video, this thesis explores the challenges of extracting "true" motion in video images in order to obtain a better picture quality and a better manipulation of video objects.

This thesis examines both methods for extracting true motion and applications of true motion estimation. While there are various motion estimation techniques, extracting true motion in video images delivers pictures of superior quality and increases the ease with which video objects can be manipulated. Specifically, one contribution of this work is a true motion tracker (TMT) based on a neighborhood relaxation formulation. Motion fields estimated by our TMT are closer to true motion than motion fields estimated by conventional minimal-residue block-matching algorithm (as adopted in MPEG test models). We demonstrate that a dependable TMT paves the way for many follow-up implementations.

1.1 Motion Estimation Algorithms in Video

There are two kinds of motion estimation algorithms: the first identifies the true motion of a pixel (or a block) between video frames, and the second removes temporal redundancies between video frames.

1. **Tracking the true motion:** The first kind of motion estimation algorithms aims to accurately track the true motion of objects/features in video sequences. Video sequences are generated by projecting a 3D real world onto a series of 2D images (e.g., using CCD). When objects in the 3D real world move, the brightness (pixel[‡] intensity) of the 2D images change correspondingly. The 2D motion projected from the movement of a point in the 3D real world is referred to as the "true motion" (as shown in Figure 1.2). For example, Figure 1.3(a) and (b) show two consecutive frames of a ball moving upright and Figure 1.3(c) shows the corresponding true motion of the ball. Computer vision, the goal of which is to identify the unknown environment via the moving camera, is one of the many potential applications of true motion.

2. **Removing temporal redundancy:** The second kind of motion estimation algorithm aims to remove temporal redundancy in video compression. In motion pictures, similar scenes exist between a frame and its previous frame. In order to minimize the amount of information to be transmitted, block-based video coding standards (such as MPEG and H.263) encode the displaced difference block instead of the original block (see Figure 1.4). For example, a block in the current frame is similar to a displaced block in the previous frame in Figure 1.5. The residue (difference) is coded together with the motion vector. Since the actual compression ratio depends on the removal of temporal redundancy, conventional block-matching algorithms use minimal-residue as the criterion to find the motion vectors [45].

Although the minimal-residue motion estimation algorithms are good at removing temporal redundancy, they are not sufficient for finding the true motion vector, as clarified by the following example. In Figure 1.5, two motion vectors produce the minimal residue, but one of the two motion vectors is not the true motion vector. In this case, the non-uniqueness

[‡]A tiny chunk of an image that has been converted to a digital word. There are typically a constant number of pixels per line, ranging from a few hundred to a couple thousand. Pixel is short for PIXture (picture) ELement.

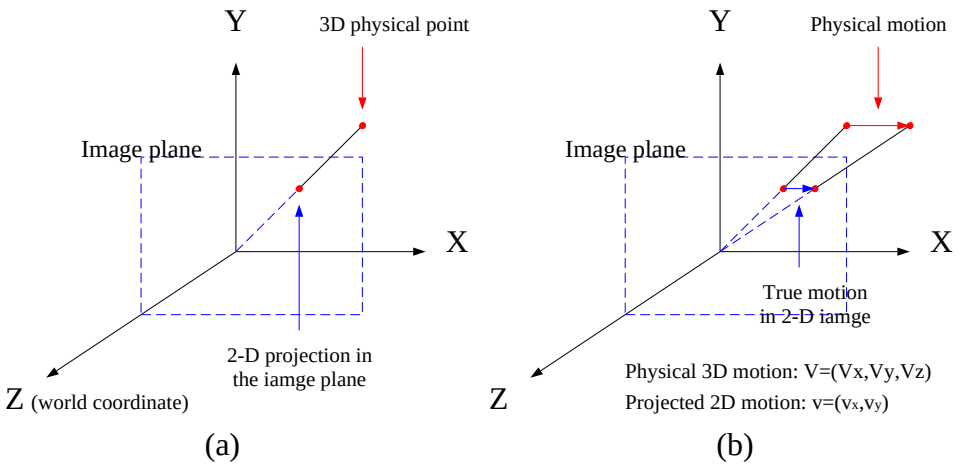


Figure 1.2: (a) A 2D image comes from projection of a 3D real world. Here, we assume a pinhole camera is used. (b) The 2D projection of the movement of a point in the 3D real world is referred as the "true motion."

of the motion vectors that can produce the minimal residue of a block contributes to their difference. The motion estimation algorithm for removing temporal redundancy is happy with finding any of the two motion vectors. However, the motion estimation for tracking the true motion is targeted at finding the only one. In general, motion vectors for the minimal residue, though good for the redundancy removal, may not actually be true motion.

1.2 Classes in True Motion Trackers

Table 1.1 shows a brief summary of motion estimation algorithms and their techniques. Despite this wide variety of approaches, algorithms for computing motion flow can be divided into the following classes:

Matching-based techniques: These operate by matching specific "features" (e.g., small blocks of images) from one frame to the next. The matching criterion is usually a normalized correlation measure.

Gradient-based techniques: These are also known as "differential" techniques. They estimate motion vectors from the derivatives of image intensity over space and time,

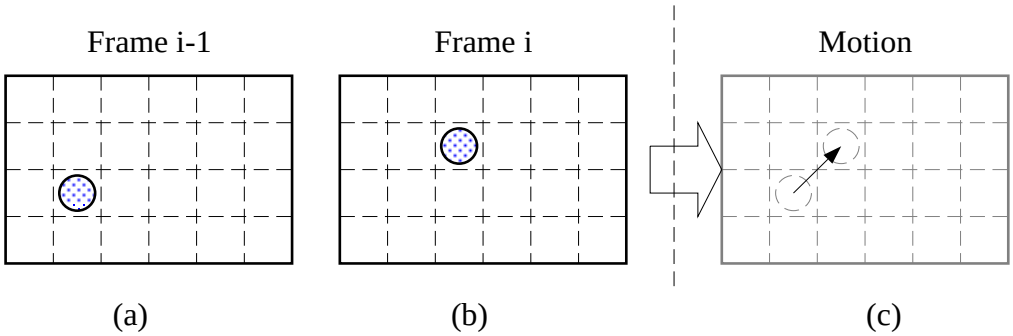


Figure 1.3: (a)(b) show two consecutive frames of a ball moving upright and (c) shows the true motion—the physical motion—in the 2D images.

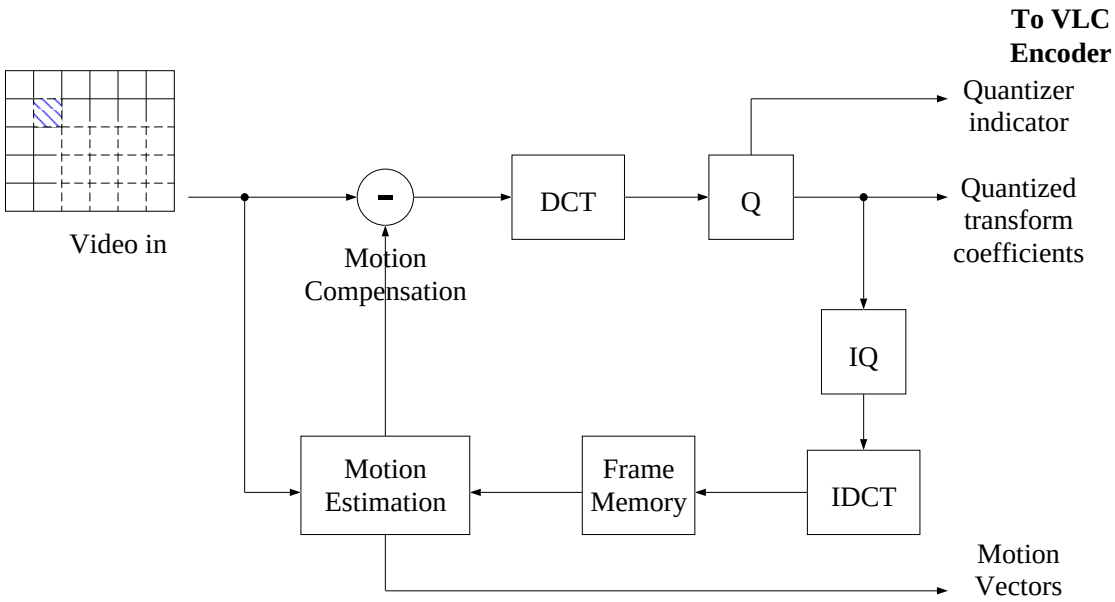


Figure 1.4: A generic MPEG-1 and MPEG-2 encoder structure is shown here. There are two basic components in video coding. The first one is the discrete cosine transform (DCT) which removes spatial redundancy in a static picture. Another one is motion estimation which removes temporal redundancy between two consecutive pictures. When the encoder receives the video, motion estimation and motion compensation first remove the similar parts between two frames. Then, DCT and quantization (Q) remove the similar parts in the texture. These quantizer indicators, quantized transform coefficients, and motion vectors are sent to a variable length encoder (VLC) for final compression. Note that the better the motion estimation, the less the work to be done in the DCT. That is to say, the better the motion estimation, the better the compression performance.

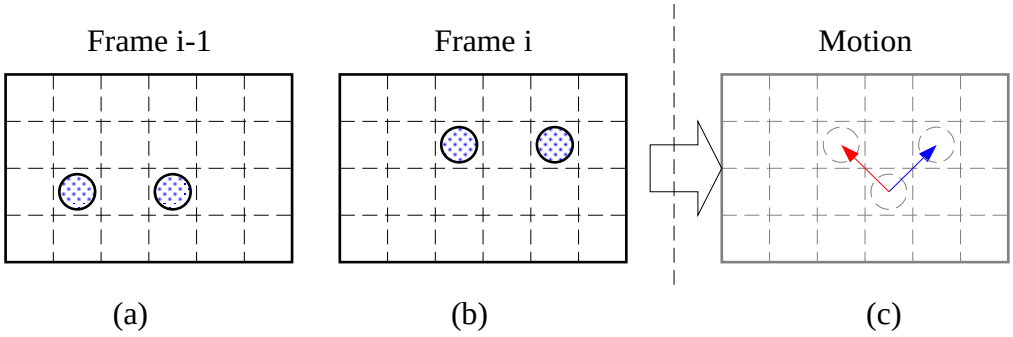


Figure 1.5: (a)(b) show two consecutive frames of two balls moving upright and (c) shows that there are two possible motion vectors that can produce the minimal residue and remove temporal redundancy in the 2D images. One of the two motion vectors is true motion while the other is not.

by considering the total temporal derivative of a conserved quantity.

Frequency-domain or filter-based techniques: These approaches are based on spatio-temporal filters, which are velocity-sensitive. They are typically considering the motion problem in the frequency domain.

In [89], Simoncelli studies image velocity from computer vision as well as biological modeling. He also compares various approaches to velocity estimation and learns that many of the solutions are remarkably similar and their origins can be viewed as filtering operations. The unification of theories regarding gradient-based and filter-based techniques bridges a long standing gap between the two techniques. In [7], Anandan presents a framework which provides a unifying perspective for correlation-matching and gradient-based techniques.

This work shows a unification theory of matching-based techniques and gradient-based techniques. We start discussing and describing some approaches of matching-based techniques and gradient-based techniques. In doing this, we have two goals in mind. The first is to introduce a set of representative solutions to the image velocity problem. In order to understand a set of basic properties that are desirable in a velocity estimation system, we consider approaches that are derived from different motivations. The second goal is to

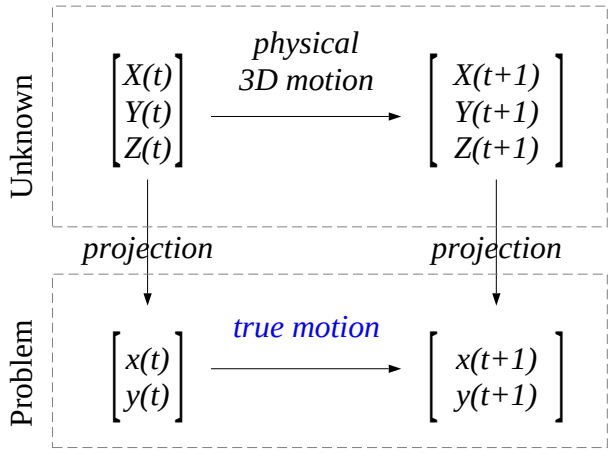


Figure 1.6: Given the 2D images, the goal of our true motion tracker is to find the 2D true motion from the image intensity changes.

summarize a number of basic features in true motion estimation and to pave the way toward the construction of a TMT that is at once both simple and powerful.

1.3 Theoretical Foundation

As mentioned before, a video sequence is generated by projecting a 3D real-world moving scene onto a series of 2D images. 2D motion projected from the movement of a point in the 3D real world is referred to as true motion[§] (shown in Figure 1.6). (Object motion is a collective decision based on all the true motion vectors belonging to the feature blocks of the same object.) The goal of this technology is to identify the 2D true motion from two or more 2D images.

[§]There is some information in 3D which is unavailable in 2D images, for example, occluded parts and depth information on each pixel. Some unknown information will affect the information gathering processes and the follow-up applications. Without the INTER-object depth information, objects with different sizes may look the same. In this case, they may have different motion in 3D, but may have the same 2D image motion. If the application of the motion tracker is video compression, then the loss of such depth information is insignificant. If the application of the motion tracker is motion-based video-object segmentation, the loss of such information will diminish the ability to distinguish different moving objects. Without the INTRA-object depth information, two pixels that have the same 3D motion, may look different in 2D motion. Therefore, the motion difference may create noise in the image processing. In addition to developing noise-immune post-processing techniques so as to ignore the noise, one may get around this problem by introducing an appropriate or approximate object model in the motion tracking processes (see Section 1.3.2).

1.3.1 Intensity Conservation Principle

One of the most important and fundamental assumptions made in motion estimation is that the intensity is conserved over time, as explained below: A pixel $[x(t), y(t)]^T$ in the image corresponds to a 3D point $[X(t), Y(t), Z(t)]^T$ in the real world. If the 3D point can be seen throughout the tracking period, the corresponding pixel is assumed to have constant brightness and color. Considering monochromatic video, the 2D image intensity of the projection of the 3D point is conserved over time, that is,

$$I(x(t), y(t), t) = I \quad \forall t \tag{1.1}$$

where $I(x, y, t)$ is the intensity of the pixel $\vec{p} = [x(t), y(t)]^T$ at time t and is equal to a constant I over time. Eq. (1.1) is fundamental to most motion estimation algorithms[¶]. Based on this foundation, there are two main classes in motion tracking:

1. **Matching-Based Measurement:**

   In matching-based approaches, pixel intensities in one frame are compared with pixel intensities in another frame. Smaller differences imply a better match. Based on the differences of pixel intensities, true motion vectors can be determined. Let $\vec{v} = [v_x, v_y]^T \equiv [x(t+1) - x(t), y(t+1) - y(t)]^T$ represent the motion of pixel $\vec{p}$. Because

   $$\begin{aligned} I(x(t), y(t), t) &= I \\ &= I(x(t+1), y(t+1), t+1) \end{aligned}$$

   we have

   $$I(x(t) + v_x, y(t) + v_y, t+1) - I(x(t), y(t), t) = 0 \tag{1.2}$$

[¶]Note: this equation may not hold true in the some conditions, for example, (1) object occlusions and reappearance (see Section 2.2), (2) non-motion brightness changes, lighting changes (see Section 7.1), (3) camera acquisition errors, spatial aliasing, and temporal aliasing.

2. **Gradient-Based Measurement:**

   Gradient-based approaches measure the spatio-temporal derivatives of pixel intensities and determine true motion vectors based on the "normal components" of velocity. Following the assumption *intensity conservation over time*, then the total derivative of the image intensity function with respect to time (assuming $I(x,y,t)$ is differentialable on x,y,t) should be zero at each position in the image and at every time:

   $$\frac{\partial I}{\partial x}\frac{dx}{dt}+\frac{\partial I}{\partial y}\frac{dy}{dt}+\frac{\partial I}{\partial t}=0$$

   where $\vec{v}=[v_x,v_y]^T\equiv[\frac{dx}{dt},\frac{dy}{dt}]^T$ is the motion of pixel $\vec{p}$. Namely,

   $$\frac{\partial I}{\partial x}v_x+\frac{\partial I}{\partial y}v_y+\frac{\partial I}{\partial t}=0 \tag{1.3}$$

Eq. (1.2) is the basis for measuring the displaced frame difference for block-matching motion estimation algorithms, and Eq. (1.3) is the basis for measuring the "normal components" of velocity for optical flow techniques [8]. Both of the equations can be characterized using the same parameters, denoted as v_x and v_y. In short, we can use the following generic expression to represent both Eq. (1.2) and Eq. (1.3):

$$s(v_x,v_y)=0 \tag{1.4}$$

Since there are two unknown components of $\vec{v}$ constrained by only one linear equation, Eq. (1.4) by itself does not offer a unique solution, as shown in Figure 1.7. There are ambiguities in determining the true velocity. Namely, an isolated pixel has no information.

1.3.2 Consistency Constraints in Motion Fields

In this section, we integrate basic velocity measurement to produce a motion field. Categories of the motion tracking algorithms depend largely upon the constraints chosen for motion consistency.

	Matching-based	Gradient-based
Overview	[36]	[8]
Feature selection	[7, 34]	
Frequency domain		[57, 89]
Spatial correlation	[11, 29, 30, 109]	[43, 86, 100, 107]
Temporal correlation	[11, 29, 30, 109]	[43]
Variable block size	[9, 35, 63, 87]	
Pixel/block subsampling	[11, 72]	
Multiresolution	[11, 27, 68, 101, 109, 112]	[43]
Motion vector multiscale	[35, 58, 85, 90]	[55, 76]
Motion consistency	[80]	
Rigidity-constrained	[34]	[74]
Probability model	[7]	[89]
Rate-distortion optimized	[13, 14]	

Table 1.1: Examples of motion estimation algorithms. (Numbers refer to the bibliography.)

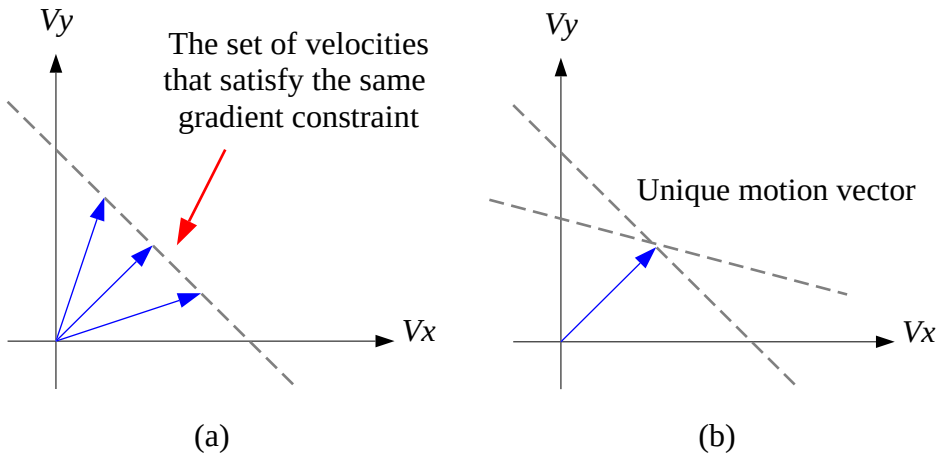


Figure 1.7: (a) A single linear equation in gradient-based motion tracking approaches cannot determine a unique motion vector. (b) If the pixels in a small region have the same motion vectors, then it is more likely that we can find a unique motion vector by two or more linear equations.

The Beginning of Time.

Generally speaking, two "$s(v_{x_i}, v_{y_i}) = 0$" equations can yield a unique solution. If two or more pixels are known to have the same motion, then it becomes possible for us to determine the motion from two or more "$s(v_{x_i}, v_{y_i}) = 0$" equations. A larger number of pixels allows us to obtain a more robust estimation of motion.

Pixels contained in the same moving object move in a consistent manner in a video sequence. Assuming translational motion only, the blocks associated with the same object should share exactly the same motion. Even for the most general motion, there should at least be a good degree of motion similarities between the neighboring blocks. Therefore, the motion vector can be more accurately estimated if the motion trend of an entire neighborhood is considered, as opposed to that of a single feature point.

Optical Flow Technique.

In [8], Barron, Fleet, and Beauchemin present a number of gradient-based techniques, regularly called optical flow techniques. Although there are some differences in those techniques, many optical flow techniques have two basic processing stages.

In the first stage, "normal components" of velocity, such as spatio-temporal derivatives, are measured, for example, see Eq. (1.3):

$$s(v_x, v_y) = \frac{\partial I}{\partial x} v_x + \frac{\partial I}{\partial y} v_y + \frac{\partial I}{\partial t}$$

In the second stage, those basic velocity measurements are integrated to produce a motion field, which involves assumptions about the smoothness of the underlying flow field. For example:

$$\min \int_{\mathbf{R}} s(v_x, v_y)^2 + \lambda(\|\nabla v_x\|^2 + \|\nabla v_y\|^2) d\vec{p}$$

where the magnitude of λ reflects the influence of the smoothness term and $\mathbf{R}$ is the region that we would like to track the motion of in the video image.

Three spatial constraints and one temporal constraint are used to identify the motion of a pixel or a region:

Block-Constraint:

Let B stand for a block of pixels in the video image and assume that the motion of the block is purely translational, i.e.,

$$\begin{bmatrix} v_{x_i} \\ v_{y_i} \end{bmatrix} = \begin{bmatrix} v_x^* \\ v_y^* \end{bmatrix} \quad \forall \vec{p}_i \in B$$

we then have

$$\sum_{\vec{p}_i \in B} \|s(v_x^*, v_y^*)\| = 0$$

which implies that

$$\vec{v^*} \in \arg\left\{ \min_{\vec{v}} \left\{ \sum_{\vec{p}_i \in B} \|s(v_x^*, v_y^*)\| \right\} \right\}$$

If we assume there is one unique minimum, then

$$\begin{bmatrix} v_x^* \\ v_y^* \end{bmatrix} = \arg\left\{ \min_{\vec{v}} \left\{ \sum_{\vec{p}_i \in B} \|s(v_x, v_y)\| \right\} \right\} \tag{1.5}$$

where the $\|\cdot\|$ can be 1-norm or 2-norm. Matching-based techniques locate the minimum by testing all the possible motion-vector candidates. To reduce the computational complexity, matching-based techniques often use 1-norm. On the other hand, gradient-based approaches usually use gradient-descent techniques. To ensure the formulation is differentiable, gradient-based approaches often use 2-norm.

Block-Matching Motion Estimation Algorithm Technique.

The basic idea of the BMA is to locate a displaced candidate block that is most similar to the current block, within the search area in the previous frame. Various similarity-measurement criteria have been presented for block matching. The most popular one is the sum of the absolute differences (SAD) of a block of pixels. The motion vector is determined by the least SAD for all possible displacements within a search area, as the following:

$$\vec{v} = \arg\left\{\min_{v_x, v_y}\left\{\mathrm{SAD}(B, v_x, v_y)\right\}\right\} \tag{1.6}$$

where

$$\mathrm{SAD}(B, v_x, v_y) \equiv \sum_{\vec{p}\in B} |I(x,y,t) - I(x+v_x, y+v_y, t+1)| \tag{1.7}$$

The exhaustive search leads to the absolute minimum of the prediction error, as shown in Figure 1.8.

Neighborhood-Constraint:

The chance that the motion of a block is purely translational is fairly large when the size of the block is small. We can find translational regions $\{R_1, R_2, \ldots, R_n\}$ within an object whose shape is arbitrary and within an arbitrary motion region. Since the motion is purely translational in each region,

$$\begin{bmatrix} v_{x_i} \\ v_{y_i} \end{bmatrix} = \begin{bmatrix} v^*_{x_j} \\ v^*_{y_j} \end{bmatrix} \quad \forall \vec{p}_i \in R_j$$

That is,

$$\sum_{\vec{p}_i \in R_j} \|s(v^*_{x_i}, v^*_{y_i})\| = 0$$

Because pixels contained in the same moving object move in a consistent manner, there should be a good degree of motion smoothness between the neighboring regions. That is,

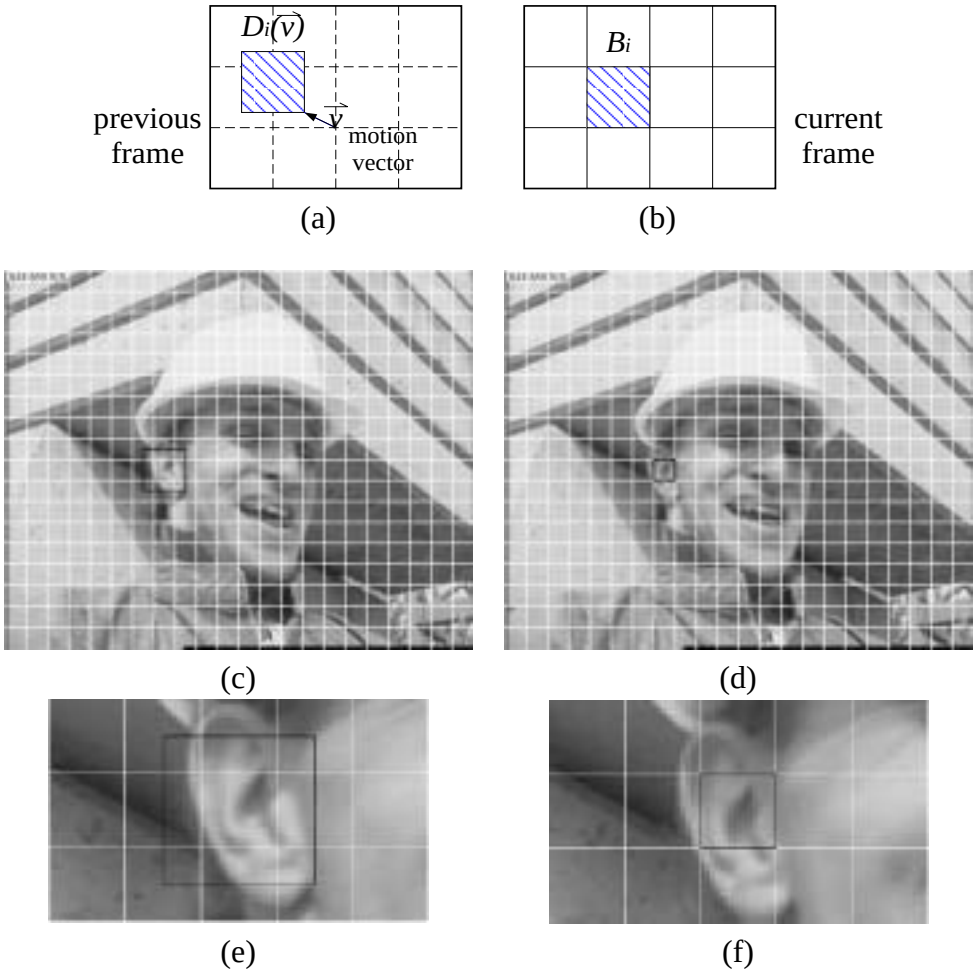


Figure 1.8: Block-matching motion estimation algorithms find the motion vector ($\vec{v}$) of the current block (B_i) by finding the best-matching displaced block ($D_i(\vec{v})$) in the previous frame. For example, (c) and (d) show the 137th frame and 138th frame of the "foreman" sequence. The current frame is divided into non-overlapping blocks, as shown in (d). To find a motion vector of a block (as shown in (f)) in the current frame by finding the best-matching displaced block in the corresponding search window (as shown in (e)) in the previous frame. The displacement vector which produces the minimal matching-error is the motion vector (as shown in Figure 1.9).

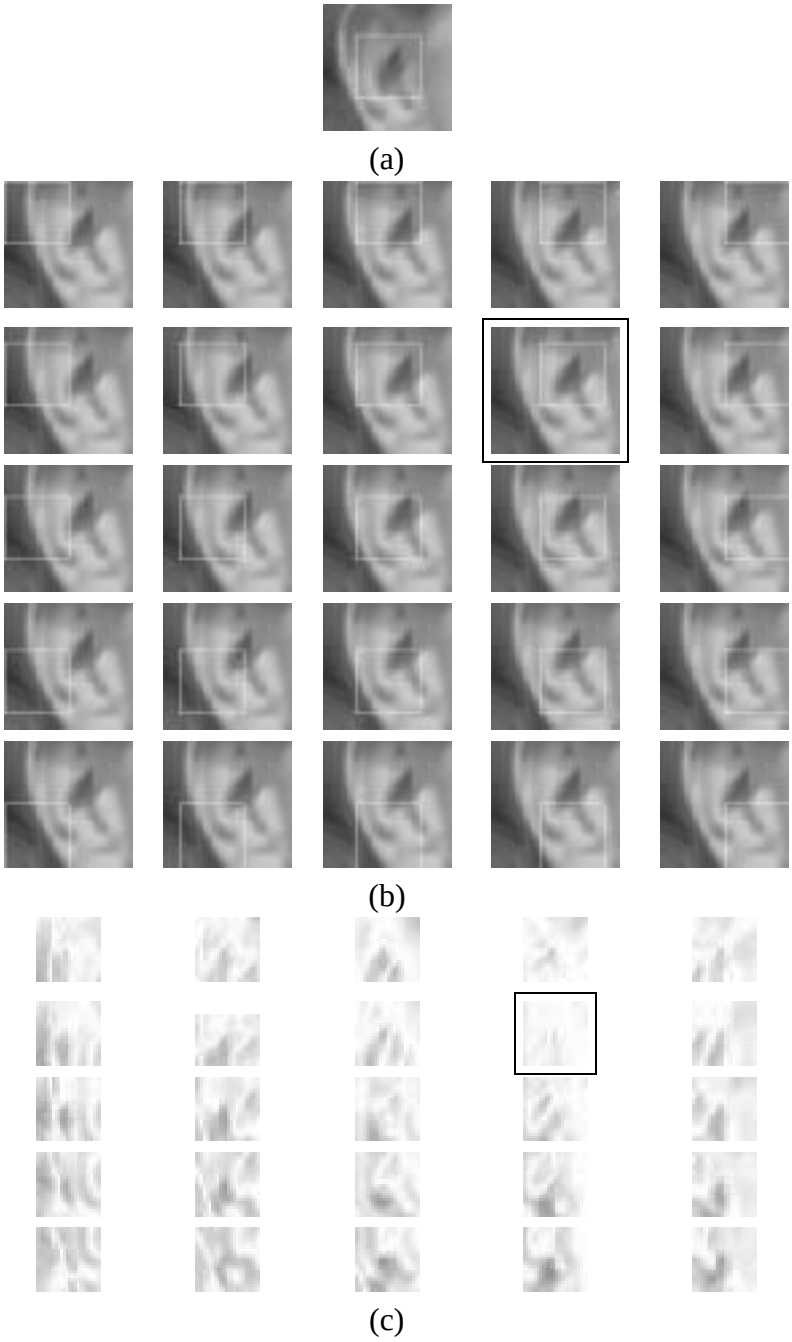


Figure 1.9: (a) shows the current block. (b) shows different displaced blocks in the previous frame. (c) shows the corresponding residues (matching errors). The displacement (upper-right) that finds the best-matching block (as marked) is the motion vector.

$\vec{v}_j \approx \vec{v}_k$. Then, it is clear that

$$motion = \arg\min_{\{\vec{v}_j\}} \left\{ \sum_{R_j \in N} \left(\sum_{\vec{p}_i \in R_j} \|s(v_{x_j}, v_{y_j})\| \right) + \lambda \sum_{R_j \in N} \|\nabla \vec{v}_j\| \right\} \tag{1.8}$$

where the neighborhood N is the union of the n disjoint regions (i.e., $N = R_1 \cup R_2 \cup \cdots \cup R_n$) and $\lambda \sum \|\nabla \vec{v}_j\|$ is for the motion smoothness in the neighborhood.

Neighborhood Relaxation Technique.

The true motion field is piecewise continuous. In Section 2.3, we show how the motion of a feature block is determined by examining the directions of all its neighboring blocks. (On the other hand, the minimum SAD of a block of pixels is used to determine the motion vector of the block in BMAs.) This allows a chance that a singular and erroneous motion vector may be corrected by its surrounding motion vectors (just like median filtering). Since the neighboring blocks may not have uniform motion vectors, a neighborhood relaxation formulation is used to allow for some local variations of motion vectors among neighboring blocks:

$$\begin{aligned} motion\ of\ B_{i,j} \quad = \quad & \arg\min_{\vec{v}} \Big\{ \mathrm{SAD}(B_{i,j}, \vec{v}) + \\ & \sum_{B_{k,l} \in N(B_{i,j})} W(B_{k,l}, B_{i,j}) \min_{\vec{\delta}} \{\mathrm{SAD}(B_{k,l}, \vec{v} + \vec{\delta})\} \Big\} \end{aligned}$$

where $B_{i,j}$ is a block of pixels for which we would like to determine the motion, $N(B_{i,j})$ is the set of neighboring blocks of $B_{i,j}$, $W(B_{k,l}, B_{i,j})$ is the weighting factor for different neighbors, and a small $\vec{\delta}$ is incorporated to allow for some local variations of motion vectors among neighboring blocks [20].

Object-Constraint:

Because 2D video is the projection of 3D scenes onto images, all pixels that belong to a 3D object follow the object motion. If we know the structure, the motion model, and the

location of an object O in the video, then we can determine the pixel motion for all pixels contained within that object. Namely,

$$\begin{bmatrix} x_i(t+1) \\ y_i(t+1) \end{bmatrix} = \text{Model}\left(\begin{bmatrix} x_i(t) \\ y_i(t) \end{bmatrix}, \text{ motion parameters} \right)$$

holds for all $\vec{p}_i \in O$. Then, again,

$$motion = \arg\left\{ \min_{motion} \left\{ \sum_{\vec{p}_i \in O} \|s(v_{x_i}, v_{y_i})\| \right\} \right\} \tag{1.9}$$

There are various object structure models (such as, 2D rigid body, 3D rigid body, plastic) and a variety of motion models (such as, translation, rotation, deformation). Some of them are very powerful but they are also computationally intensive.

2D Affine Matching Technique.

One of the most commonly used motion models to describe the motion of an object is the 2D affine model:

$$\begin{bmatrix} x_i(t+1) \\ y_i(t+1) \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_i(t) \\ y_i(t) \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

which can model 2D objects in translational/rotational motion and 3D objects in translational motion. Under this assumption, Eq. (1.9) becomes

$$\begin{aligned} \{a_{ij}, b_k\} \quad = \quad & \arg\Big\{ \min_{\{a_{ij}, b_k\}} \sum_{\vec{p}_i \in O} \|I(a_{11}x_i(t) + a_{12}y_i(t) + b_1, \\ & a_{21}x_i(t) + a_{22}y_i(t) + b_2, t+1) - I(x_i(t), y_i(t), t)\| \Big\} \end{aligned}$$

Since there are six unknown parameters, at least six reference points are required in order to generate a unique solution.

2D Affine Gradient-Based Technique.

In a manner similar to the previous approach, the 2D affine motion model can be used with gradient-based measurement as well:

$$\{a_{ij}, b_k\} = \arg\Big\{ \min_{\{a_{ij}, b_k\}} \sum_{\vec{p}_i \in O} \| \frac{\partial I}{\partial x}((a_{11}-1)x_i(t) + a_{12}y_i(t) + b_1) + \frac{\partial I}{\partial y}(a_{21}x_i(t) + (a_{22}-1)y_i(t) + b_2) + \frac{\partial I}{\partial t}) \| \Big\}$$

Object-Constrained Feature Corresponding Technique.

In [34], Dreschler and Nagel present an object tracking method in stationary TV sequences. Frames from such an image sequence can be separated into stationary and non-stationary parts. Whereas the stationary parts consist of static foreground and background, the non-stationary parts are divided further into sub-images corresponding to one or more moving objects, for example, a car or a pedestrian. In Dreschler and Nagel's method, prominent features of non-stationary objects on each frame are first selected. Then, the tracking of the features is formulated as the correspondence problem, i.e., the search for a suitable match between features from two different image frames[‖]. To increase the accuracy of the tracking results, a graph-based matching approach, which constrains the movement of a set of features, is used to solve this feature correspondence problem.

Rigidity-Constrained Optical Flow Technique.

In [74], Mendelsohn, Simoncelli, and Bajcsy present another algorithm for estimating motion flow. Unlike traditional optical flow approaches that impose smoothness constraints on the flow field, this algorithm assumes that the camera is aimed at a rigid object such that there should be a consistent

[‖]Such a problem occurs not only during the evaluation of temporal image sequences, but also in the disparity determination required for binocular stereo-vision.

relationship between pixel velocities. Therefore, they assume smoothness on the inverse depth map.

Temporal-Constraint:

In a short period of time, the motion of a pixel over a small number of frames can be considered as constant. Let T represent a period of time in the video sequence and assume that the motion of the pixel is constant,

$$\begin{bmatrix} v_x(t) \\ v_y(t) \end{bmatrix} = \begin{bmatrix} v_x^* \\ v_y^* \end{bmatrix} \quad \forall t \in T$$

Then, it is clear that

$$\sum_{t \in T} \|s(v_x^*, v_y^*)\| = 0$$

It implies that

$$\vec{v^*} \in \arg\left\{ \min_{\vec{v}} \left\{ \sum_{t \in T} \|s(v_x^*, v_y^*)\| \right\} \right\}$$

If there is one unique minimum, then

$$\begin{bmatrix} v_x^* \\ v_y^* \end{bmatrix} = \arg\left\{ \min_{\vec{v}} \left\{ \sum_{t \in T} \|s(v_x, v_y)\| \right\} \right\}$$

Spatial/Temporal Correlation Technique.

The true motion field is also piecewise continuous in the temporal domain. That is, motion fields are not only piecewise continuous in spatial domain (2D) but also piecewise continuous in temporal domain (1D). In [29, 109], de Haan *et al.* and Xie *et al.* introduce motion estimation algorithms that exploit motion vector correlations between temporal adjacent blocks and spatial adjacent blocks. The initial search area can be reduced by exploiting these correlations.

	Matching-based	Gradient-based
Block constraint	[36, 101]	[8, 76]
Neighbor constraint	[20]	
Object constraint	[34]	[74]
Temporal constraint	[29, 109]	

Table 1.2: By combining matching-based or gradient-based measurement with the block constraint, the object constraint, the neighborhood constraint, or the temporal constraint, we define a variety of motion tracking algorithms. (Numbers refer to the bibliography.)

Different Motion Estimation Algorithms in One Generic Equation.

By combining either matching-based measurement or the gradient-based measurement with the block constraint, the object constraint, the neighborhood constraint, or the temporal constraint, we define a variety of motion tracking algorithms (see Table 1.2). For example, block-matching algorithms (BMAs) are based on the minimization of the matching error (see Eq. (1.2)) of a block of pixels (see Eq. (1.5)).

All of the constraint equations can be characterized by the same form of integration. In short, we can use the following generic expression to represent all of them:

$$\{\vec{v}_i\} = \arg\left\{\min_{\{\vec{v}_i\}}\left\{\sum \|s(\vec{p}_i,\vec{v}_i)\|\right\}\right\} \tag{1.10}$$

where $\{\vec{v}_i\}$ satisfy certain constraints (e.g., block-wise translational motion, 2D affine motion).

1.3.3 Correctness, Precision, and Accurate in Motion Estimation

Before discussing "better" true motion tracking algorithms, some terms must first be defined:

1. **Correctness:** An estimated motion vector $\vec{u}$ is correct when it is close (within a certain range) to the true motion vector $\vec{v}^*$. The correctness of an estimated motion field

is the ratio of the correctly estimated motion vectors to the total estimated motion vectors.

2. **Precision:** The precision of the estimated motion field is inversely proportional to the estimation error between the estimated motion field and the true motion field.

3. **Accurate:** An estimated motion field is accurate when it is correct and precise.

In video processing, subjective visual quality is one of the most important issues. We find that the correctness of the motion field is highly correlated to subjective visual quality (e.g., in non-residue motion compensation). The first goal of this work is to develop a high-correctness true motion tracker.

1.3.4 Tradeoffs in Different Measurement

In this work, we first focus on implementing a core true-motion tracker using matching-based measurement (gradient-based measurement is used in Chapter 4).

Motion estimation techniques that use matching-based measurement are adopted widely by video compression communities for two important reasons: they are easy to implement and are efficient at removing redundancy. Matching-based techniques are often derived using the sum of the absolute differences (SADs) and locate the minimum SAD by testing all the possible motion-vector candidates. The required operations are *simple and regular* (integer additions and subtractions) and are easy to implement in ASICs (e.g., systolic arrays). In addition, in many video compression standards, removing temporal redundancy is more important than finding the true motion vector. (As mentioned in Section 1.1, the motion vector for minimal residue may not be the true motion.) Finding the minimum SAD by testing all the possible motion-vector candidates is an efficient way to remove redundancy.

On the other hand, gradient-based techniques (using gradient-based measurement) are usually adopted by computer vision communities because finding the true motion vector is

more important than removing temporal redundancy. Based on an analytical formulation using the spatial and temporal derivatives of image intensity (using floating-point multiplications and divisions), gradient-based techniques can precisely determine *sub-pel* motion vectors. This is difficult for matching-based techniques to do.

However, gradient-based approaches often perform poorly in highly textured regions [89] and fast motion regions [76] (see Section 4.3.1). Practical gradient-based algorithms use finite differences to approximate the derivatives. Finite differences approximate the derivatives well at slow motion regions, but unfortunately the approximation degrades rapidly with increasing motion speed [76]. When the initial position is too far away from the solution, it is likely that gradient-based approaches converge to a local minimum and produce undesirable results. That is, in high motion regions, they could find motion vectors inaccurately.

There are two reasons for focusing on the implementation of a core true-motion tracker based on matching-based techniques, and for why we use gradient-based techniques in Chapter 4: (1) We would like to demonstrate that matching-based techniques can not only remove redundancy but can also find true motion vectors when proper constraints are given. (2) If proper initial motion vectors are given, the gradient-based techniques can be accurate in high motion regions. In this context, the initial motion vectors can be provided by the matching-based true motion tracker so as to avoid complex computations, such as floating-point multiplications and divisions.

1.3.5 Tradeoffs in Different Constraints

In this work, our true motion tracker is built upon neighborhood relaxation. In Section 1.3.2, we present three spatial constraints and a temporal constraint used to integrate basic velocity measurement for producing a motion field. In this section, we discuss the tradeoffs of different constraints and explain the reason for choosing the neighborhood constraint.

The block constraint has several advantages: (1) the assumption behind this constraint is very simple; (2) it covers the most common cases; (3) its implementation is also very easy. Therefore, it is widely used in most video compression standards [3, 51].

The block constraint requires a hard decision in determining the block size. The smaller the block, the greater the chance that more than one "minimum" exists. (The high number of possible candidates makes it very hard to determine which is the true one.) On the other hand, the larger the block, the greater the chance that the block does not follows purely translational motion (e.g., the blocks can contain two or more moving objects.)

The advantage of using the object constraint is two-fold: (1) it is powerful in modeling the real world; (2) it covers most cases in video.

However, the object constraint is very complex and slow in implementation. Before the object constraint can be applied to integrate basic velocity measurement, some critical questions must be answered:

1. **Choice of object model:** Applying complicated object models to simple objects is acceptable, but it is more expensive in terms of computation. Therefore, it is important to choose the best (simple and accurate) model for an object. For example, from a far distance a book is a 2D object, and a human head from a distance is a 3D cylinder.

2. **Choice of motion model:** In order to reduce computation time and increase tracking stability, the best motion model for the object must be selected. For example, the translational motion model is simpler than the general affine model.

3. **Object location in the image:** We must determine whether a pixel belongs to an object or lies outside of it. The more accurate the object boundary, the more accurate the motion estimation.

4. **Initial parameter estimation:** Most object/motion models have a high number of

parameters; the better initial estimation of the parameters would result in more satisfactory final tracking results.

Several difficulties are encountered when applying the object constraint to motion tracking. It is generally agreed to be an ill-posed problem, because theoretically the motion models can only be applied to regions homogeneous with respect to the movement, which means that a preliminary stage of segmentation is required. Since the object motion is not known, no criteria are available to carry out the segmentation [34, 37, 43, 74]. (Many approaches to object motion estimation problems get around this difficulty by basing initial object segmentation on block motion vectors as a preprocessing stage.)

An advantage of the temporal constraint is that the temporal constraint and the spatial constraints can supplement each other (see Section 2.4). Applying the temporal constraint without any spatial constraint is unique. Although theoretically speaking, two independent equations can solve two unknown variables, we need much more than three frames to have two independent equations in the real world. Therefore, it is preferable that the length of the period T is greater than three. However, when the length of the period T is too large, it is unlikely that the motion vector of the pixel remains constant.

A disadvantage of the temporal constraint is the cost (processing delay and memory size). Three or more frames of the video must be stored before the motion vectors can be determined. This results in a processing delay, and requires more memory space than motion estimation algorithms based on two frames.

The neighborhood constraint is a compromise between between block constraint and object constraints. For example, its implementations are much simpler and faster than the object constraint, but not as simple as the block constraint. It is also more applicable to a broader range of situations than the block constraint is, but not as broad as the object constraint. In the neighborhood constraint, finding small piecewise-translational regions within a moving object is easy. Neither object model nor motion model is required for motion tracking. However, the neighborhood constraint suffers the same problems with the

block constraint: (1) how to decide the size of the neighborhood, and (2) how to determine whether the neighborhood belongs to the same moving object (see Section 5.3.4).

In this work, we focus on the neighborhood constraint. We would like to develop a fundamental motion tracker applicable to a number of tracking problems, even to object motion tracking. Therefore, our core true-motion tracking algorithm does not assume any object information.

1.4 Contribution and Organization of Dissertation

In this work, we discuss fundamentals of true motion estimation, application-specific implementations, and system design of a true motion tracker.

One contribution the thesis makes to the field of digital video processing is our baseline TMT, which uses block-matching and neighborhood relaxation techniques. The proposed true motion tracker is based solely on the sum of the absolute differences (SADs)—the simplest method in matching-based techniques. Simple and regular computations are used. The neighborhood relaxation formulation allows for the correction of a singular and erroneous motion vector by utilizing its surrounding motion vector information (just like median filtering). Our method is designed for tracking more flexible affine-type motions, such as rotation, zooming, shearing, etc.

Applications highlight the importance of the true motion tracker. This work discusses the theory of true motion tracking with respect to its many useful applications and demonstrates that true motion tracking is effective in achieving lower bit-rates in video coding and higher quality in video interpolation. This work evaluates the wide variety of applications of true motion tracking.

This thesis also addresses the TMT system design and implementation. The effective implementation of the TMT on programmable parallel architectures shows another promising aspect of the proposed TMT.

Part I: Theory of True Motion Tracker

Figure 1.10 depicts the organization of this work and the relationship between different sections and chapters. In Section 1.3, we discuss the theoretical foundation of true motion tracking: the intensity conservation principle, two basic means of measurement (matching-based and gradient-based), and four motion-consistency constraints (block-, object-, neighborhood-, and temporal-).

These two basic techniques and four different constraints can form many different kinds of motion estimation algorithms, as shown in Table 1.2. Among the wide variety of motion estimation algorithms, we shall place our focus on the neighborhood-matching motion estimator. In addition, Chapter 4 breaks new ground by integrating matching-based and gradient-based techniques.

In order to improve the correctness and precision of the true motion tracker, we recapitulate five rules (block-sizing, traceability, trackability, and consistency) from some basic observations of true motion tracking (cf. Table 2.1). In Section 2.1, the block-sizing rule, which allows a decision regarding the right block size to be made, is discussed. Basically, we prefer a large block size but wish to avoid the error of having different moving objects in a single block. Therefore, in Section 2.3, we demonstrate that the spatial-consistency rule can lead to a spatial neighborhood relaxation formulation, which can incorporate more spatial neighborhood information. This formulation is our baseline TMT—the key component in Chapters 3, 4, and 5.

In Section 2.2, the traceability and trackability rule presents the method for and importance of identifying and ruling out untraceable and untrackable blocks. This rule is used in Chapter 5. In Section 2.4, the temporal-consistency rule leads to a temporal relaxation scheme, which can incorporate more temporal information (similar to the spatial-consistency rule leading to a spatial relaxation scheme). The frame-rate up-conversion scheme discussed in Chapter 3 uses the temporal relaxation formulation.

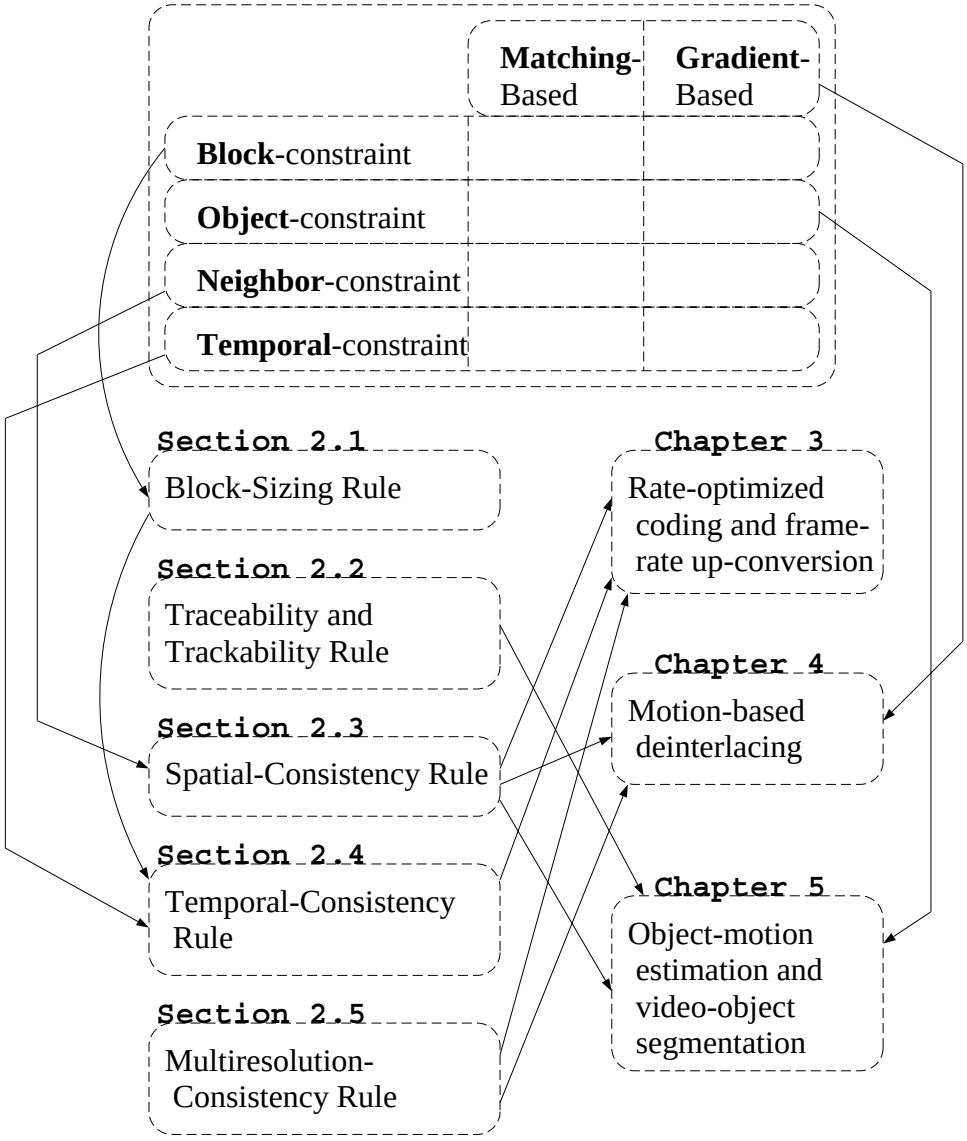


Figure 1.10: The organization of this work and the correlation between different sections and chapters. Section 1.3 shows two basic measurement techniques and four different constraints, which can be applied to many different kinds of motion estimation algorithms. In Chapter 2, we present a set of rules for "better" true motion trackers. Based on these, we build up our true motion tracker and three different application-specific true motion trackers as used in Chapters 3, 4, and 5.

As shown in Section 2.5, the multiresolution-consistency rule leads to a motion tracking scheme using multiple resolutions. In addition, similar to Chapter 4, Section 2.5 also explores how the precision of the tracking results can be increased using a motion vector refinement scheme, in which small changes on the estimated motion vectors are allowed. Moreover, this work is perhaps the first research that presents a multiresolution motion estimation scheme in conjunction with neighborhood relaxation.

Part II: True Motion Tracker and Applications

This work discusses true motion tracking with respect to its many useful applications. True motion estimation in video sequences has many useful applications, such as:

1. Video compression: includes efficient coding, rate optimized motion vector coding, subjective picture quality (less block effects), object-based video coding, object-based global motion compensation, and so on.

2. Video spatio-temporal interpolation: includes field rate conversion applications, interlaced-to-progressive scan conversion, enhancement of motion pictures, synthesis, and so forth.

3. Video analysis and understanding: includes object motion estimation (including recovering the camera motion relative to the scene), video object segmentation (including determining the shape of a moving object), 3D video object reconstruction (monoscopic or stereoscopic), machine vision for security, transportation, and medical purposes, etc.

For each of the above applications, different degrees of accuracy and resolution in the computed motion flow are required. Moreover, different applications can afford different amounts of computational time. As a result, different applications exploit different techniques.

We design the TMTs with the applications in mind. Although our baseline TMT is generally reliable, it still suffers from some of the problems that occur in the natural scene, such as, homogeneous regions, object occlusion and reappearance, etc. We do not tackle the problems in the baseline TMT. Instead, we tackle the problems based on the *applications* since different applications require different degrees of accuracy and resolution in the motion fields and can afford different computational times. The advantages of this approach are (1) the baseline TMT is simple and (2) the application-specific TMT is efficient.

This is perhaps the first thesis research that demonstrates that the TMT can be successfully applied to video compression (as shown in Chapter 3). According to MPEG standards, the motion fields are encoded differentially. Therefore, the minimal-residue block-matching algorithm (BMA) can optimize the residual coding but cannot optimize the bit rate for motion information coding. Using the true motion on individual blocks can also optimize the bit rate for residual and motion information. In addition, since the TMT provides true motion vectors for encoding, the blocking artifacts are decreased and, hence, the pictures look better subjectively.

As another contribution, Chapter 3 also shows that the TMT offers significant improvement over the minimal-residue BMA in motion-compensated frame-rate up-conversion using decoded motion vectors. Motion-compensated interpolation differs from the ordinary interpolation in that the pixels in different frames are properly skewed according to the movements of the object. The more accurate the motion estimation, the better the motion-compensated interpolation.

Another contribution is that we demonstrate in Chapter 4 that the proposed TMT can be applied to the spatio-temporal interpolation of video data. Using the interlaced-to-progressive scan conversion as an example in the spatio-temporal interpolation of video data, we extend the basic TMT to an integration of the matching-based technique and the gradient-based technique. The application-specific TMT for the interlaced-to-progressive scan conversion differs from the application-specific TMT discussed in Chapter 3. The

more accurate the motion estimation, the better the motion-compensated interpolation. Therefore, we have a high-precision application-specific TMT, which takes much more computational time. The matching-based motion estimation can find "full-pel" motion vectors accurately. Afterwards, the gradient-based technique may be adopted to find "sub-pel" motion vectors more precisely and easily, based on those full-pel motion vectors.

As the last application-related contribution, we demonstrate in Chapter 5 that a modification of the TMT can be successfully applied to motion-based video-object segmentation. For object-based coding and segmentation applications, the major emphasis is not placed on the number of feature blocks to be tracked (i.e., quantity) but on the reliability of the feature blocks we choose to track (i.e., quality). The goal of application-specific TMT in this chapter is to find motion vectors of the features for object-based motion tracking, in which (1) any region of an object contains a good number of blocks, whose motion vectors exhibit certain consistency; (2) only true motion vectors for a small number of blocks per region are needed. This means that we can afford to be more selective in our choice of feature blocks. Therefore, one of the natural steps is to eliminate those unreliable or unnecessary feature blocks. We propose a new tracking procedure: (1) At the outset, we disqualify some of the reference blocks that are considered to be unreliable to track (intensity singularities). (2) We adopt a multi-candidate pre-screening to provide some robustness in selecting motion candidates. (3) We have a motion candidate post-screening to screen out possible errors in tracking the blocks on object boundaries (object occlusion and reappearance).

Part III: Effective System Design and Implementation of the True Motion Tracker

This discussion would be incomplete if the subject of the system design and implementation of the TMT were omitted. Because conventional BMAs for motion estimation are computationally demanding, to design an efficient implementation of the BMA has been a challenge for years. Although the proposed TMT is more computationally intensive and control-intensive than conventional BMAs, an effective system implementation in Chapter 6 demonstrates another promising aspect of the TMT.

First, the TMT is implemented on a programmable parallel architecture consisting of a core processor and an array processing accelerator. Driven by novel algorithmic features of multimedia applications and advances in VLSI technologies, many architecture platforms for new media-processors have been proposed in order to provide high performance and flexibility. Recent architectural designs can be divided into internal (core processor) and external (accelerator) designs. Some algorithmic components can be implemented using a programmable core-processor while others must rely on hardware accelerators. Therefore, we implement the TMT using architecture that integrates a core-processor and an array processing accelerator.

Second, using an optimal operation and scheduling scheme, we process the regular and computationally intensive components of the TMT on the processing array. Systematic methodology capable of partitioning and compiling algorithms is vital to achieving the maximum performance of the parallel and pipelined architecture (like systolic design methods). Because the gap between processor speed and memory speed is getting larger and larger, the memory/communication bandwidth is a bottleneck in many systems. An effective operation placement and scheduling scheme must deliver an efficient memory usage. Particularly, memory access localities (data reusability) must be exploited. Another important contribution made here is an algebraic multiprojection methodology for operation placement and scheduling, which can manipulate an algorithm with high-dimensional data reusability and provides high memory-usage efficiency.

To summarize, the architecture platform and the operation placement and scheduling scheme are useful for our development of the TMT system implementation. First, the TMT algorithm is partitioned into two parts—a regular and computationally intensive part versus a complex but less computation-demanding part. The regular and computationally intensive part will be assigned to the processing array to exploit the accelerator's high performance, while the complex but less computation-demanding part will be handled by the core-processor due to its high flexibility.

Chapter 2

Useful Rules for True Motion Tracking

Hundreds of motion tracking (motion estimation) algorithms have been developed for machine vision, video coding, image processing, etc. Some of them are quite similar in the basic techniques but are different in computational formulation. Each of the techniques brings a new point of view and a new set of tools to tackle the problems untackled by other algorithms. However, each of them may probably fail to handle situations that occur in natural sceneries. In this chapter, we study the ground common to all motion tracking algorithms and establish the foundation for our true motion tracker (TMT).

2.1 Choose the Right Block

In this section, we discuss the relationship between the block size in the block-constraint and the correctness and precision of the estimated motion field.

Block-Sizing Rule: *Use large blocks to improve correctness, but not so large as to hurt correctness.*

Block-Sizing Observation: In a video sequence, the pixels of the same moving object are expected to move in a consistent way. The pixels associated with the same object should share a good degree of motion similarities. Consider only translational motion in block-constraint matching-based techniques:

Rules	Functions and Advantages
Block-sizing rule	Use large blocks to improve correctness, but not so large as to hurt correctness
Traceability and trackability rule	Locate and weed out (1) untextured regions, and (2) object boundary, occlusion, and reappearance regions to improve correctness
Spatial-consistency rule	Use spatial smoothness to improve correctness; use spatial neighborhood relaxation to improve precision
Temporal-consistency rule	Use temporal smoothness to improve correctness; use temporal neighborhood relaxation to improve precision
Multiresolution-consistency rule	Use multiresolution relaxation to reduce computation; use motion vector refinement to improve precision

Table 2.1: Rules for accurate motion tracking.

1. Within a same moving object, when the block is larger and larger, it becomes increasingly unlikely to find a matched displaced block using a displacement vector that is not the true motion vector.

2. When the block size is excessively larger than a consistent moving object, finding the true motion for any point in the block becomes difficult.

For simplicity, we use a two-dimensional (one spatial dimension and one temporal dimensional) signal $f(n,t)$ to explain these phenomena. We define the score function for matching-based techniques as

$$s(n,\Delta) \equiv |f(n+\Delta,t+1)-f(n,t)| \tag{2.1}$$

and the block-constraint motion estimation algorithm as

$$d = \arg\left\{\min_{\Delta}\left\{\sum_{n=1}^{k} s(n,\Delta)\right\}\right\}$$

where k is the block size, Δ is the displacement, and d is the estimated motion vector.

1. We consider that blocks are always contained within a same moving object. Assuming (without losing generality) that the whole $f(n,t)$ is shifted by a true displacement d_0 into $f(n,t+1)$ at time $t+1$, i.e., then

$$f(n,t) = f(n+d_0,t+1) \quad \forall n \tag{2.2}$$

To find a matched displaced block using a displacement vector d that is not the true motion vector d_0 means that

$$\sum_{n=1}^{k} s(n,d) = 0$$

i.e., for all $n \in \{1,\ldots,k\}$,

$$\begin{aligned} f(n,t) &= f(n+d,t+1) \qquad \text{(from Eq. (2.1))} \\ &= f(n+d-d_0,t) \qquad \text{(from Eq. (2.2))} \end{aligned}$$

which means that a block of signals are identical to another block of signals. As the block size is larger and larger, a match between the features of two blocks becomes increasingly harder. That is, it is increasingly unlikely that this block will be confused with other blocks.

Another way of stating this phenomenon is this: as the block size k becomes larger, it becomes easier to distinguish the true displacement d_0 from the displacement d by the "min$\{\sum s(n,\Delta)\}$" criterion.

2. We consider that a block contains two or more moving objects. For simplicity, we assume that the block size is k and the block contains two different moving blocks—one is from $n=1$ to $n=k'$ and the other one is from $n=k'+1$ to $n=k$ ($k'<k$). Different parts of the $f(n,t)$ are shifted by different displacements as the following:

$$\begin{aligned} f(n+d_1,t+1) &= f(n,t) \quad \forall\, 1 \le n \le k' \\ f(n+d_2,t+1) &= f(n,t) \quad \forall\, k'+1 \le n \le k \end{aligned}$$

(a) In order to have matched displaced block using d_1, i.e.,

$$\sum_{n=1}^{k} s(n, d_1) = 0$$

we must have

$$f(n+d_1, t+1) = f(n+d_2, t+1) \quad \forall\, k'+1 \le n \le k$$

which means that a block of signals (from $n = k'+1+d_1$ to $n = k+d_1$) is identical to another block of signals (from $n = k'+1+d_2$ to $n = k+d_2$).

(b) Similarly, in order to have matched displaced block using d_2, we must have

$$f(n+d_1, t+1) = f(n+d_2, t+1) \quad \forall\, 1 \le n \le k'$$

which means that a block of signals (from $n = 1+d_1$ to $n = k'+d_1$) is identical to another block of signals (from $n = 1+d_2$ to $n = k'+d_2$).

In both cases, as the block size ($k-k'$ or k') becomes larger, a match between the features of two blocks becomes increasingly harder. That it, it is increasingly unlikely to find a matched displaced block using the true motion vectors of the objects due to the matching error introduced by different moving objects.

By assuming that the image signal is *separable*, the problem of finding the unknown displacement d from $\{f(n,t)\}$ and $\{f(n,t+1)\}$ is similar to our problem of finding the motion vector (v_x, v_y) from $\{I(x,y,t)\}$ and $\{I(x,y,t+1)\}$. The larger the block size used to find the unknown true displacement, d_0, from the "$\min\{\sum s(n,\Delta)\}$" criterion, the higher the ability to distinguish the true displacement d_0 from other displacements. And so, we conclude that the larger the block, the greater the chance to find the unique and correct estimation. We also showed that it is difficult to have a correctly estimated motion vector when the block size is too large.

We assume that the true motion field is piecewise continuous in the spatial domain. Given this premise, the motion vector can be more dependably estimated if the global

motion trend of an entire neighborhood is considered, as opposed to considering one feature block itself [36, 63]. This enhances the chance that a singular and erroneous motion vector may be corrected by its surrounding motion vectors [107]. For example, a tracker fails to track its central block because of noise, but successfully tracks its surrounding blocks. With the smoothness constraint in the neighborhood, the true motion of the central block could be recovered.

Making the block (neighborhood) size larger increases the correctness when the block size is smaller than the translational moving region of an object. A region moves translationally under three circumstances. (1) A region is contained within an object that is moving translationally related to the camera. (2) A region is contained within an object that is moving non-translationally, but the movement is minute and hence a small piece of the object moves as if translationally. (3) A region is contained within a stationary background when the camera is panning or not moving. For most coding standards, the block size is 16×16. From the actual video scene (with picture CIF 352×288 or larger), such a block size is usually smaller than most of the translational moving region. Generally, the motion vector obtained from a larger block is more noise-resistant than that obtained from a smaller block. So, in this work, we use a neighborhood larger than the conventional block size.

On the other hand, when the block size is larger than a translational moving region of an object, it is unlikely to find a matched displaced block using the true motion vectors of the objects due to the matching error introduced by different moving objects. Clearly, choosing a correct block size is critical.

In order to reduce the estimation error introduced when the block contains different objects on the block size, a weighting scheme is introduced that puts more emphasis on the center of a block and less emphasis on the periphery of a block. In general, when a block contains different objects, the center of the block contains a single object while the periphery of the block contains other objects. In this case, the weighting scheme can reduce

the damage and so enable us to use a larger block.

Neighborhood-Sensitive Gradient-Based Technique.

Since the motion fields are piecewise continuous in the spatial domain, several algorithms have put the neighborhood influences into the optimization criteria, just like SNAKE [53]. In [100], Tomasi and Kanade present an algorithm that minimizes the sum of squared intensity differences between a past and a current window. In addition to modeling the changes as a more complex transformation, such as an affine map, the algorithm emphasizes the central area of the window as the following:

$$\min \int_{\mathbf{R}} w(\vec{p}) s(\vec{p}, \vec{v})^2 d\vec{p}$$

where $w(\cdot)$ is a Gaussian-like function, which puts higher weights on the center than on the boundary.

Neighborhood-Sensitive Block-Matching Technique.

Instead of considering each feature block individually, we determine the motion of a feature block (say, $B_{i,j}$) by moving all its neighboring blocks ($N(B_{i,j})$) along with it in the same direction. A score function is introduced as the following [100]:

$$\begin{aligned} \text{score}(B_{i,j}, \vec{v}) &= \text{SAD}(B_{i,j}, \vec{v}) + \\ &\qquad \sum_{B_{k,l} \in N(B_{i,j})} (W(B_{k,l}, B_{i,j}) \, \text{SAD}(B_{k,l}, \vec{v})) \qquad (2.3) \\ &= \text{image force (external energy)} + \\ &\qquad \text{constraint forces (internal energy)} \end{aligned}$$

where $W(B_{k,l}, B_{i,j})$ is the weighting function. The final estimated motion vector is the minimal-score displacement vector:

$$\textit{motion of } B_{i,j} = \arg\min_{\vec{v}}\{\text{score}(B_{i,j}, \vec{v})\}$$

where $\vec{v}$ should be one of the possible candidates recorded by the multi-candidate pre-screening.

The central block's residue in the score function is called *image force*, which is similar to the external energy function of SNAKE [53]. In addition, the neighbors' residue in the score function is called *constraint forces*, which reflect the influence of neighbors and correspond to the internal energy function of SNAKE.

The above approach will be inadequate for non-translational motion, such as object rotating, zooming, and approaching [86]. For example, in Figure 2.3(b), an object is rotating counterclockwise. Because Eq. (2.3) assumes that the neighboring blocks will move in the same translational motion, this equation may not adequately model the rotational motion.

Instead of choosing a fixed block size (e.g., in video coding), the quad-tree-structured technique shows an adaptive scheme for choosing different block sizes. The multi-scale technique uses different block sizes for different levels of estimation precision.

Quad-Tree-Structured Technique.

Because the information details are not uniformly distributed in the spatial domain, Seferidis and Ghanbari [87] use the quad-tree structured spatial decomposition. Fine structures are important in detailed ares, whereas coarse structures are sufficient in uniform regions.

Multi-Scale Technique.

Dufaux and Moscheni report that large range displacements are correctly estimated on large-scale structures and short range displacement are accurately estimated on small-scale structures [36]. They introduce a locally adaptive mesh refinement procedure. This allows both from-fine-to-coarse and from-coarse-to-fine motion field refinements. While the more classical strategy of using only coarse-to-fine refinement allows more accurate motion fields on edges of moving objects, the strategy of using fine-to-coarse refinement allows more accurate motion fields on homogeneous regions.

Hierarchical Fast Motion Estimation Technique.

Assume that the motion vector obtained from a larger block size provides a good initial estimate for motion vectors associated with smaller blocks that are contained by the larger block. The hierarchical methods, which use the same image size but different block sizes at each level, also restrict the number of search locations at large-scale motion vectors at first, and then refine the predicted motion vector later [9, 35].

Multi-Scale Optical Flow Technique.

Practical algorithms use simple finite differences to approximate the derivatives. In [76], Moulin, Krishnamurthy, and Woods notice that finite differences approximate the derivative very well at slow motion; unfortunately the approximation degrades rapidly with increasing motion speed. Furthermore, gradient approaches often perform poorly in highly textured regions of an image unless the pre-filter is excellent [89]. Hence, a coarse-to-fine multi-scale optical flow approach is proposed.

However, their method did not perform well with fast and complex motions (e.g., the football and Susie sequences), because of the over-smoothness parameter caused by the hierarchical estimator. The multi-scale algorithm suffers from (1) propagation of estimation errors from the coarse level of the pyramid to the finer level of the pyramid, and (2) the inability to recover from these errors.

In short, it is important to use large blocks to improve correctness, but not so large as to hurt correctness. We have shown a few adaptive techniques that can be used to tackle the critical decision. In Sections 2.3 and 2.4, we use a relaxation formulation (more computation), in which some biased noise becomes unbiased noise, to accommodate non-constant motion. We are using more computation to improve the precision given the assurance of the correctness even though there is still a risk of losing the correctness.

2.2 Locate and Weed out Untraceable and Untrackable Regions

There are two cases when a block should be weeded out from the list of correctly motion-tracked blocks:

1. Untraceable: If a block can be perfectly motion-compensated from the other frames (1) when the true motion vector is given, and (2) when many motion vectors that are not true motion are given to the block, then it is extremely challenging to distinguish the true motion vector from other vectors. In this case, the block is called "untraceable."

2. Untrackable: If a block cannot be well motion-compensated when the true motion vector is given, then it is arduous to identify the true motion vector using the intensity-conservation principle. In this case, the block is called "untrackable."

In both cases, the correctness of the estimated motion vector becomes uncertain and so we present the following traceability and trackability rule for higher tracking correctness:

Traceability and Trackability Rule: *For higher tracking correctness, locate and weed out (1) untextured regions and (2) the object boundary, occlusion, and reappearance.*

Observation of the Homogeneous Region Untraceability: The information details are not uniformly distributed in the spatial domain—some parts of the image have more details, some have less. An untextured region keeps its image intensity constant regardless of the motion, and hence it is extremely challenging to determine how (or even whether) it moves. Consider only translational motion in block-constraint matching-based techniques:

- When the block is within a same moving object and is homogeneous (untextured), it is likely to find a matched displaced block using a displacement vector that is not the true motion vector.

Following our explanation of the previous observation, we have two signals $f_1(n,t)$ and $f_2(n,t)$ and assume (without losing generality) that $\{f_1(n,t)\}$ is more homogeneous than $\{f_2(n,t)\}$. In general, it is easier to find two pixels with the same intensity in the homogeneous region. In this case, a match between the features of two blocks becomes easier. That is, it is harder to distinguish the true displacement from other displacements in $f_1(n)$ than in $f_2(n)$. We conclude that it is harder to find the true motion when the block is more homogeneous.

We can also look at this phenomenon from another perspective. In gradient-based techniques, we first measure the spatial gradient ($\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$) and the temporal gradient ($\frac{\partial I}{\partial x}$). In an untextured regions, the spatial gradient is equal to zero. In this case, the basic measurement Eq. (1.4) becomes

$$s(v_x, v_y) = 0v_x + 0v_y + \frac{\partial I}{\partial t} = \frac{\partial I}{\partial t}$$

which means that the basic measurement is independent of the motion vector. Therefore, determining the true motion vector become extremely difficult.

Since we are considering techniques that only estimate the motion of small regions, the problem of finding multiple matched blocks will occur whenever the image brightness pattern is uniform in the region being considered. Often there is not enough information in a region to determine the motion flow. This is referred to as the "blank wall" problem.

Similarly, if we are looking at a pattern composed of stripes, then we cannot determine how fast (or whether) it is moving along the direction of the stripes. If the pattern slides along the direction of the stripes, the image intensity pattern will not change. Again the pattern need only be striped within the region being considered. This problem is typically known in the literature as the "aperture" problem.

For confidence in tracking correctness, it is important to identify and rule out the untextured region (see our implementation in Section 5.3.1.)

Directional-Confidence Technique.

Choosing a motion vector to describe the motion in a blank region constitutes an over-commitment to a solution. In general, there will be many areas of the image with insufficient information at a particular scale for the local determination of displacements. In [7], Anandan advocates the use of "confidence" or "reliability" measures to indicate whether or not to accept a motion vector for further processing, and also to indicate the degree to which the motion vector can be trusted.

Since the image displacement is a vector quantity, it is possible that different directional components of the displacement may be locally computable with different degrees of reliability. For instance, it is clear that in a homogeneous area of the image no component of the displacement can be reliably estimated. At a point along a line (or an edge), the component perpendicular to

the line can be reliably computed, while the component parallel to the line may be ambiguous. At a point of high curvature along an image contour it may be possible to completely and reliably determine the motion vector based purely on local information. These observations suggest that the confidence measure should be directionally selective—i.e., that it should associate different confidences with the different directional components of the motion vector.

Observation of the Boundary and Occlusion Untrackability: Most algorithms for estimating image motion fields rely on the assumption that the image intensity corresponding to a point remains constant as that point moves. Severe violations occur at boundaries and occlusion regions. For example, a block located at the object boundaries contains regions moving in at least two different directions. A single motion vector compensates one or more of these regions incorrectly [80]. As another instance, as shown in Figure 2.1, due to the object occlusion, the pixel intensity in the arrow head is not constant.

Because tracking the blocks on the object boundary, occlusion, and reappearance regions generally produces incorrect motion vectors, it is important to identify and rule out the object occlusion, reappearance, and boundary for confidence in tracking correctness (see Section 5.3.4).

Consistency Post-Screening Technique for Untrackable Blocks.

The energy of the residue can reveal how well a block is matched from the previous frame. From this point, it is possible to determine whether the block is tracked well or not. For example, in most video coding standards, there is an INTRA/INTER coding mode decision for each macroblock in the P-frame (or the B-frame) [3, 23, 45, 75]. The INTER is used when the motion

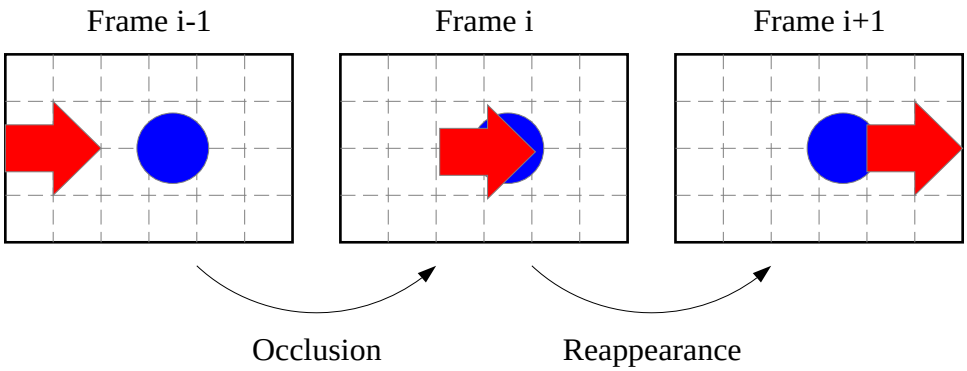


Figure 2.1: There are 3 consecutive frames of an arrow moving rightward. The arrow head occludes behind the ball in frame i while the arrow head reappear in frame $i+1$.

estimation/compensation can find a good match for the block. On the other hand, when there is a block on the object boundary, the occlusion region, or the reappearance region, it is difficult to find a matched block from the previous frame. Hence, the INTRA mode is used for that macroblock. Based on the same idea, in Section 5.3.4, we have a motion candidate post-screening step to screen out possible errors in tracking the blocks on object boundaries. The post-screening step disqualifies a block from the trackable feature block list when the residue is larger than a predetermined threshold.

2.3 Spatial Neighborhood Relaxation

To remedy the problem of choosing the block size as mentioned in Section 2.1, we propose a neighborhood relaxation formulation. Almost all the block-matching motion estimation algorithms (BMAs) assume that a single translational motion vector is sufficient to describe the motion in each macroblock. When we choose a larger block size, this assumption is frequently violated in natural scenes (for example, rotating, expanding, or contracting motions, or the motion of non-rigid objects—fluids or deforming elastic materials). In such cases, a single velocity vector is not sufficient to describe the motion of a

block. The problem is the *size* of the block. A neighborhood relaxation formulation enables us to use a large block without assuming that the whole block is purely translational.

Spatial-Consistency Rule: *Use spatial smoothness to improve correctness; use spatial neighborhood relaxation to improve precision.*

Spatial Neighborhood Relaxation Observation: Assuming that the 2D image intensity of a 3D point projection is conserved over time, then

$$I(x,y,t) = I(x+v_x(x,y), y+v_y(x,y), t+1) \tag{2.4}$$

where $I(x,y,t)$ is the intensity of the pixel $\vec{p} = [x(t), y(t)]^T$ at time t.

1. We assume that the motion vector of every pixel of a block B follows the same motion vector:

$$\begin{cases} [v_x(x,y)] = v_x^* \\ [v_y(x,y)] = v_y^* \end{cases} \qquad \vec{p} \in B$$

where $[x]$ means rounding x to the nearest integer. Then, the weighted sum of displaced frame difference is equal to zero:

$$\sum_{\vec{p} \in B} w(x,y) \, |I(x,y,t) - I(x+v_x^*, y+v_y^*, t+1)|^m = 0 \tag{2.5}$$

where $w(x,y) > 0$ and $m \geq 1$. The $w(x,y)$ is the weighting factor for different pixels and will be explained more thoroughly in Section 2.3.1. We set $m = 1$ for simplicity in the implementation. Therefore, most of the BMAs determine the motion vectors as the following [3, 51]* (cf. Eq. (1.6)):

$$\vec{v} = \arg \left\{ \min_{v_x, v_y} \{ \text{WSAD}(B, v_x, v_y) \} \right\} \tag{2.6}$$

*Nevertheless, Eq. (2.5) can only imply

$$\vec{v}^* \in \arg \left\{ \min_{v_x, v_y} \{ \sum_{\vec{p} \in B} w(x,y) \, |I(x,y,t) - I(x+v_x, y+v_y, t+1)| \} \right\}$$

Only if there is one unique minimum, can we have the equal sign as shown in Eq. (2.6). We will address this issue in Section 5.3.2.

where WSAD (the weighted sum of the absolute differences) is defined as below:

$$\text{WSAD}(B, v_x, v_y) \equiv \sum_{\vec{p} \in B} w(x,y)\ |I(x,y,t) - I(x+v_x, y+v_y, t+1)|$$

2. The assumption that $[v_x(x,y)]$ and $[v_y(x,y)]$ are constant is valid on limited conditions, such as the purely translational motion of a rigid object. It is possible that there is a pixel $\vec{p} \in B$ so that $[v_x(x,y)] - v_x^* = \delta_x \neq 0$ and $[v_y(x,y)] - v_y^* = \delta_y \neq 0$. To be more accurate, Eq. (2.5) should be written as:

$$\sum_{\vec{p} \in B} w(x,y)\ |I(x,y,t) - I(x+v_x^* + \delta_x(x,y), y+v_y^* + \delta_y(x,y), t+1)| = 0 \quad (2.7)$$

Let $B = \{B_{1,1}, B_{1,2}, \cdots, B_{n,n}\}$ and $B_{i,j}$ be the center block, as shown in Figure 2.2 and set

$$\delta_x(x,y) = \delta_y(x,y) = 0 \qquad \forall \vec{p} \in B_{i,j}$$

When the neighborhood is associated with the same object, each block shares similar motion vectors:

$$\begin{cases} \delta_x(x,y) = \delta_x^{(k,l)} & |\delta_x^{(k,l)}| \leq 1 \\ \delta_y(x,y) = \delta_y^{(k,l)} & |\delta_y^{(k,l)}| \leq 1 \end{cases} \quad \forall \vec{p} \in B_{k,l} \qquad (2.8)$$

where $B_{k,l}$ is the neighboring blocks of $B_{i,j}$. From Eq. (2.7) and Eq. (2.8), we have

$$\sum_{\vec{p} \in B_{i,j}} \text{WSAD}(B_{i,j}, v_x^*, v_y^*) + \sum_{B_{k,l} \neq B_{i,j}} \left\{ \min_{-1 \leq \delta_x, \delta_y \leq 1} \left\{ \text{WSAD}(B_{k,l}, v_x^* + \delta_x, v_y^* + \delta_y) \right\} \right\} = 0 \qquad (2.9)$$

3. For simplicity in implementation, we calculate the un-weighted sum of the absolute difference instead of the weighted sum of the absolute difference. Let $w(x,y) = 1 \ \ \forall \vec{p} \in B_{i,j}$ and $w(x,y) = W(B_{i,j}, B_{k,l}) \ \ \forall \vec{p} \in B_{k,l}$. Then, Eq. (2.9)

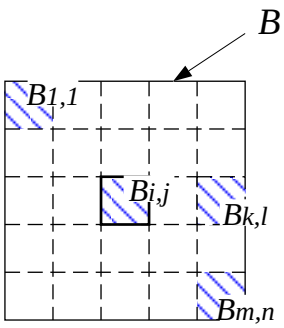


Figure 2.2: Neighborhood blocks in the B. $B_{i,j}$ is the center block and $B_{1,1}$, $B_{k,l}$, $B_{m,n}$ are the neighbors of $B_{i,j}$.

equals to

$$\mathrm{SAD}(B_{i,j},\vec{v}^*)+\sum_{B_{k,l}\in N(B_{i,j})} W(B_{k,l},B_{i,j})\ \min_{\vec{\delta}}\{\mathrm{SAD}(B_{k,l},\vec{v}^*+\vec{\delta})\}=0 \quad (2.10)$$

where $\vec{v}^*=[v_x^*,v_y^*]^T$, $\vec{\delta}=[\delta_x,\delta_y]^T$, and SAD (the sum of the absolute differences) is defined as below:

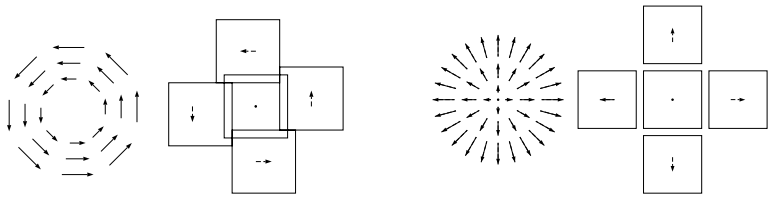
$$\mathrm{SAD}(B,v_x,v_y)\equiv\sum_{\vec{p}\in B}|I(x,y,t)-I(x+v_x,y+v_y,t+1)|$$

We divide a region into a number of small blocks so that each of them can be easily described by a single motion vector. The integration of these blocks can accurately track the true motion. The neighborhood relaxation formulation is the following:

$$\mathrm{score}(B_{i,j},\vec{v}) \;=\; \mathrm{SAD}(B_{i,j},\vec{v})+\sum_{B_{k,l}\in N(B_{i,j})} W(B_{k,l},B_{i,j})\ \min_{\vec{\delta}}\{\mathrm{SAD}(B_{k,l},\vec{v}+\vec{\delta})\} \quad (2.11)$$

where $B_{i,j}$ means a block of pixels whose motion we would like to determine, $N(B_{i,j})$ is the set of neighboring blocks of $B_{i,j}$, and $W(B_{k,l},B_{i,j})$ is the weighting factor for different neighbors. A small $\vec{\delta}$ is incorporated to allow some local variations of motion vectors among neighboring blocks. Some biased noise becomes unbiased noise after this variation process. The motion vector is obtained as

$$\textit{motion of } B_{i,j}=\arg\min_{\vec{v}}\{\mathrm{score}(B_{i,j},\vec{v})\}$$



(a) (b)

Figure 2.3: Neighborhood relaxation will consider the global trend in object motion and provide some flexibility to accommodate non-translational motion. Local variations $\vec{\delta}$ among neighboring blocks, cf. Eq. (2.11), are included in order to accommodate other (i.e., non-translational) affine motions such as (a) rotation, and (b) zooming/approaching.

As shown in Figure 2.3, the variation afforded to every block can be used to fine-tune the motions to accommodate the non-translational effect. This in principle can track more flexible affine-type motions, such as rotating, zooming, shearing, etc.

2.3.1 Spatial-Dependent Neighborhood Weighting Factors

In Eq. (2.11), not all neighboring blocks have the same weighting factors. In fact, they can be made to be dependent on several factors, such as the **distance** to the central block, the **confidence** of the blocks, the color/texture **similarity** between $B_{i,j}$ and $B_{k,l}$, etc. In terms of distance, the weighting function could be a Gaussian-like function, putting higher emphasis in the central area. In terms of the confidence, low-confidence blocks should be guided by their higher confidence neighbors [7]. In general, a block with a larger variance can be tracked more confidently. (An obvious example is that homogeneous blocks do not yield a high confidence.) Therefore, the weighting function must take into account the block variance.

In this discussion, we assume that all the blocks in B come from a same moving object. However, Eq. (2.10) may not hold true when $B_{i,j}$ is close to object boundaries (this is elaborated in Section 5.3.4). Because different objects usually have different color/texture characteristics, we set $W(B_{k,l}, B_{i,j})$ to be proportional to the color/texture similarity between $B_{i,j}$ and $B_{k,l}$. This will reduce the weights of the neighborhoods that contain different

objects.

2.4 Temporal Neighborhood Relaxation

In the previous formulations, only the intensity information in the frame t and the information in the frame $t+1$ are used. It is natural to extend this work by using multiple frames.

Temporal-Consistency Rule: *Use temporal smoothness to improve correctness; use temporal neighborhood relaxation to improve precision.*

Temporal Neighborhood Relaxation Observation: True motion field is also piecewise continuous in the temporal domain. The motion estimation will be more accurate by exploiting multiple-frame information. The basic motion tracking formulation, Eq. (1.10), is extended to the following:

$$\{\vec{v}_i(t)\} = \arg\left\{\min_{\{\vec{v}_i(t)\}}\left\{\sum_t\sum_i \|s(\vec{p}_i,\vec{v}_i(t),t)\|\right\}\right\} \tag{2.12}$$

where $\{\vec{v}_i(t)\}_i$ satisfy certain spatial constraints and $\{\vec{v}_i(t)\}_t$ satisfy certain dynamic models, such as one of the following:

1. **Linear model:** $\vec{v}_i(t) = \vec{v}_i(t-1) = \vec{v}_i(t+1)$
2. **Neighborhood relaxation model:** $\vec{v}_i(t) = \vec{v}_i(t-1) + \vec{\delta}_1 = \vec{v}_i(t+1) + \vec{\delta}_{-1}$
3. **Recursive dynamic model:** $\vec{v}_i(t+1) = \mathit{filter}(\vec{v}_i(t), \vec{v}_i(t-1), \cdots, \vec{v}_i(t-q))$ where q is the order of the filter.

In principle, exploiting multiple-frame information can increase the correctness of the motion estimation. The motion of a feature block is determined by examining the directions of all its temporal neighbors. This allows the chance that a singular and erroneous motion vector may be corrected by its surrounding motion vectors (just like spatial neighborhood).

In addition to the spatial neighborhood relaxation formulation, we also present the temporal neighborhood relaxation formulation to improve precision.

In a short period of time T, the motion of a pixel should be similar, i.e., $\vec{v}^*(t) \approx \vec{v}^*(t') \quad \forall t - t' \in T$ where $\vec{v}^*(t)$ is the true motion vector from the frame t to the frame $t+1$. Let

$$\vec{v}^*(t) = \vec{v}^*(t') + \vec{\delta}$$

and define the new sum of the absolute differences with time as a parameter:

$$\mathrm{SAD}(B_{i,j}, \vec{v}, t) \equiv \sum_{\vec{p} \in B_{i,j}} |I(\vec{p}, t) - I(\vec{p} + \vec{v}, t+1)|$$

where $B_{i,j}$ is a pure-translational moving region. We know that $\mathrm{SAD}(B_{i,j}, \vec{v}^*(t), t) \approx 0$ and $\mathrm{SAD}(B_{i,j}, \vec{v}^*(t) + \delta, t') \approx 0$, and hence

$$\sum_{\Delta_t \in T} \mathrm{SAD}(B_{i,j}, \vec{v}^*(t) + \vec{\delta}, t + \Delta_t) \approx 0$$

After that, we define our spatial and temporal neighborhood relaxation formulation as the following:

$$\begin{aligned} \mathrm{score}(B_{i,j}, \vec{v}(t)) \quad = \quad & \mathrm{SAD}(B_{i,j}, \vec{v}(t), t) \\ & + \sum_{B_{k,l} \in N(B_{i,j})} W(B_{k,l}, B_{i,j}) \min_{\vec{\delta}} \{\mathrm{SAD}(B_{k,l}, \vec{v}(t) + \vec{\delta}, t)\} \\ & + \sum_{\Delta_t \in T} W_t(\Delta_t) \min_{\vec{\delta}} \{\mathrm{SAD}(B_{i,j}, \vec{v}(t) + \vec{\delta}, t + \Delta_t)\} \end{aligned} \tag{2.13}$$

where the T is the temporal neighborhood and $W_t(\Delta_t)$ is the weighting factor in the temporal neighborhood. The motion vector for block $B_{i,j}$ from frame t to frame $t+1$ is obtained as:

$$\textit{motion of } B_{i,j} = \arg\min_{\vec{v}} \{\mathrm{score}(B_{i,j}, \vec{v})\}$$

In Chapter 3, we use this model in our frame-rate up-conversion motion estimation algorithm.

2.5 Multiresolution Motion Estimation with Neighborhood Relaxation

In the previous two sections, we used the neighborhood relaxation scheme (more computation) to improve the precision of the block-matching true motion tracker. In this section, we use a motion-vector refinement scheme (extra computation), in which small changes to the estimated motion vectors are allowed, to increase the precision of correct motion vectors given the assurance of the correctness (although there is still a risk of losing the correctness).

Similar to the true motion tracker shown in Chapter 4, this section uses a refinement scheme to increase the precision of correct motion vectors. Different from the tracking method shown in Chapter 4, the method reduces the computational complexity by a multiresolution scheme.

Multiresolution-Consistency Rule: *Use multiresolution relaxation to reduce computation; use motion vector refinement to improve precision.*

The use of multiple resolutions in the recognition process is *computationally* and conceptually interesting. In the analysis of signals, it is often useful to observe a signal in successive approximations. For instance, in pattern recognition applications the vision system attempts to classify an object from a coarse approximation. If the classification does not succeed, additional details are added such that a more accurate view of the object is obtained. This process can be continued until the object has been recognized.

From a given image, multiresolution analysis consists of a set of images having different levels of details, i.e., more or less energy in the high frequencies. We obtain images of lower resolution by successive low-pass filtering, followed by down-sampling. We denote the reduction in resolution by a mapping function:

$$R : \mathbf{I}^{(k)} \rightarrow \mathbf{I}^{(k+1)}$$

where the image $\mathbf{I}^{(k)}$ is a set of pixels $\{I^{(k)}(x,y)\}$ and $\mathbf{I}^{(0)}$ is the original image. Most of the time, the filter is half-band and the down-sampling is done with a factor of two in each direction (horizontal and vertical).

Let $s^{(k)}(\vec{p}_i,\vec{v}_i)$ be defined over the image at level *k*, then Eq. (2.12) can be extended to the following:

$$\{\vec{v}_i^{(k)}\} = \arg\left\{ \min_{\{\vec{v}_i\}} \left\{ \sum_{\vec{p}_i^{(k)}} \|s^{(k)}(\vec{p}_i^{(k)},\vec{v}_i)\| \right\} \right\} \tag{2.14}$$

where $s^{(k)}(\vec{p}_i^{(k)},\vec{v}_i)$ is similar to the definition in Section 1.3.1 but is defined over the $\mathbf{I}^{(k)}$ in this section.

Observation of Motion Field Inheritance of Multiresolution Processes: We have a set of multiresolution images $\left\{\mathbf{I}^{(k)}\right\}$ using a linear transformation R, which is a half-band filter followed by a 2:1 down-sampling, i.e., $R\left(\left\{I^{(k)}(\vec{p}_i^{(k)},t)\right\}\right) = \left\{I^{(k+1)}(\vec{p}_i^{(k+1)},t)\right\}$. Assume that we have the ability to filter, subsample, and quantize the motion vectors, then the motion fields of various levels usually follow the same multiresolution relationship, i.e., $R\left(\{\vec{v}_i^{(k)}\}\right) = \{\vec{v}_i^{(k+1)}\}$.

1. We have the ability to filter, subsample, and quantize the motion vectors. When the motion field $\{\vec{v}_i^{(k)}\}$ is band-limited, then $R\left(\left\{I^{(k)}(\vec{p}_i^{(k)}+\vec{v}_i^{(k)},t+1)\right\}\right) = \left\{I^{(k+1)}(\vec{p}_i^{(k+1)}+\hat{v}_i^{(k+1)},t+1)\right\}$ where we define

$$R\left(\{\vec{v}_i^{(k)}\}\right) = \{\hat{v}_i^{(k+1)}\} \tag{2.15}$$

We have

$$\begin{aligned}
& R\left(\left\{I^{(k)}(\vec{p}_i^{(k)},t) - I^{(k)}(\vec{p}_i^{(k)}+\vec{v}_i^{(k)},t+1)\right\}\right) \\
= \;& R\left(\left\{I^{(k)}(\vec{p}_i^{(k)},t)\right\}\right) - R\left(\left\{I^{(k)}(\vec{p}_i^{(k)}+\vec{v}_i^{(k)},t+1)\right\}\right) \\
= \;& \left\{I^{(k+1)}(\vec{p}_i^{(k+1)},t)\right\} - \left\{I^{(k+1)}(\vec{p}_i^{(k+1)}+\hat{v}_i^{(k+1)},t+1)\right\} \\
= \;& \left\{I^{(k+1)}(\vec{p}_i^{(k+1)},t) - I^{(k+1)}(\vec{p}_i^{(k+1)}+\hat{v}_i^{(k+1)},t+1)\right\}
\end{aligned}$$

2. Because the true motion vector can minimize the displaced frame difference, we define the estimated motion vectors as

$$\{\vec{v}_i^{(k)}\} = \arg\min_{\{\tilde{v}_i^{(k)}\}} \left\{ \mathrm{DFD}(\mathbf{I}^{(k)}, \{v_i^{(k)}\}) \right\} \tag{2.16}$$

where the displaced frame difference (DFD) is defined as

$$\mathrm{DFD}(\mathbf{I}^{(k)}, \{v_i^{(k)}\}) \equiv \left\| \left\{ I^{(k)}(\vec{p}_i^{(k)}, t) - I^{(k)}(\vec{p}_i^{(k)} + \tilde{v}_i^{(k)}, t+1) \right\} \right\|$$

Assume before and after R , the minimum displaced frame difference is **unique** before and after R and **the position of the minimum does not change**. Namely, when $\mathrm{DFD}(\mathbf{I}^{(k)}, \{\vec{v}_i^{(k)}\})$ is the minimum,

$$\mathrm{DFD}(\mathbf{I}^{(k+1)}, \{\hat{v}_i^{(k+1)}\}) \text{ is the minimum.} \tag{2.17}$$

From Eq. (2.16) and Eq. (2.17),

$$\{\hat{v}_i^{(k+1)}\} = \{\vec{v}_i^{(k+1)}\} \tag{2.18}$$

3. From Eq. (2.15) and Eq. (2.18), we conclude that $R\left(\{\vec{v}_i^{(k)}\}\right) = \{\vec{v}_i^{(k+1)}\}$, i.e., the motion fields of various levels usually follow the same multiresolution relationship.

4. There are two cases where $R\left(\{\vec{v}_i^{(k)}\}\right) \neq \{\vec{v}_i^{(k+1)}\}$:

   (a) The position of the minimum will change before and after R , perhaps because of an aliasing introduced by the spatial sampling. A one-dimensional difference signal $\{1,-1,1,-1,\cdots\}$ will become $\{0,0,\cdots\}$ after the 2:1 resolution reduction. While the sum of the absolute differences is $1+1+1+1+\cdots$ before the resolution reduction, the sum of the absolute differences is $0+0+\cdots$ after the resolution reduction. In this case, the position of the minimum may change.

(b) When (1) there are multiple minimal SADs in the $\mathbf{I}^{(k)}$ images or (2) there is an aliasing after the resolution reduction, there are multiple minimal SADs in the $\mathbf{I}^{(k+1)}$ images. In these cases, the statement that DFD($\mathbf{I}^{(k+1)}, \{\hat{v}_i^{(k+1)}\}$) is minimum is not equivalent to the statement that $\{\hat{v}_i^{(k+1)}\}$ are the estimated motion vectors of the $\mathbf{I}^{(k+1)}$ images.

Observation: Under the multiresolution framework, we assume

1. $\mathbf{I}^{(k)}$ is band-limited to ω_k.
2. If $\mathbf{I}^{(k)}$ is band-limited to ω_k, then the motion field $\{\vec{v}_i^{(k)}\}$ is also band-limited to ω_k.
3. If the image $\mathbf{I}^{(k)}$ has noise, the motion vectors $\{\vec{v}_i^{(k)}\}$ have noise.

and we make the following statements:

1. If the image $\mathbf{I}^{(k)}$ has noise, the estimation of the motion vectors $\{\vec{v}_i^{(k+1)}\}$ has higher correctness than that of $\{\vec{v}_i^{(k)}\}$.
2. If the image $\mathbf{I}^{(k)}(t)$ has some subtle motion, the estimation of the motion vectors $\{\vec{v}_i^{(k+1)}\}$ are less precise than that of $\{\vec{v}_i^{(k)}\}$.

We define the following symbols before our discussion: $\hat{v}_i$ is the estimated motion vector, $\vec{v}_i$ is the true motion vector, and $\tilde{v}_i$ is the motion vector noise. We know that

$$\hat{v}_i \equiv \vec{v}_i + \tilde{v}_i$$

We divide the motion signals into two sets. One is low-pass band signals (which are band-limited from 0 to $\omega_k/2$), denoted as $\hat{v}_{iL}$, $\vec{v}_{iL}$, and $\tilde{v}_{iL}$. The second one is high-pass band signals (which are band-limited from $\omega_k/2$ to ω_k), denoted as $\hat{v}_{iH}$, $\vec{v}_{iH}$, and $\tilde{v}_{iH}$. We know that

$$\hat{v}_i \equiv \hat{v}_{iL} + \hat{v}_{iH}, \quad \vec{v}_i \equiv \vec{v}_{iL} + \vec{v}_{iH}, \quad \tilde{v}_i \equiv \tilde{v}_{iL} + \tilde{v}_{iH}$$

Because the lower resolution image is obtained by a low-pass filter followed by a 2:1 down-sampling, $\mathbf{I}^{(k+1)}$ is band-limited to $\omega_k/2 = \omega_{k+1}$. Therefore, the high-frequency (from $\omega_k/2$ to ω_k) signals in $\mathbf{I}^{(k)}$ will be eliminated. That is,

$$\begin{aligned} \{\hat{v}_i^{(k+1)}\} &= R\left(\{\hat{v}_i^{(k)}\}\right) \\ &= R\left(\{\vec{v}_{iL}^{(k)} + \tilde{v}_{iL}^{(k)}\}\right) \end{aligned}$$

Now, we have the following:

1. The high-frequency noise in the $\mathbf{I}^{(k)}$ creates the high-frequency noise in the $\{\tilde{v}_{iH}^{(k)}\}$. During the resolution reduction, the high-frequency components (large noisy motion) are eliminated, and the estimation of the motion vectors $\{\vec{v}_i^{(k+1)}\}$ has higher correctness than that of $\{\vec{v}_i^{(k)}\}$.

2. Because the high-frequency components $\{\vec{v}_{iH}^{(k)}\}$ are filtered, if the image $\mathbf{I}^{(k)}(t)$ has some subtle motion, the motion vectors $\{\vec{v}_i^{(k+1)}\}$ are less precise than $\{\vec{v}_i^{(k)}\}$.

This multiresolution technique simplifies tracking of large motions while reducing the amount of necessary computation. When there are three different resolutions, the finest resolution is 16 times larger than the coarsest resolution. Therefore, the search area used at the coarsest resolution is 16 times smaller. Thus, the computational complexity is dramatically reduced.

Although we show that the estimated motion field obtained from the third level image can have higher correctness than the estimated motion field taken directly from the original image, the motion vector obtained from the coarsest resolution is 4 times coarser in scale. As a result, local refinement in the finer resolution is required for higher precision. One method for tackling both correctness and precision is the hierarchical updating scheme. In this scheme, (1) coarse resolution images are used for coarse scale motion vectors, and (2) finer resolution images are used to refine the motion vector candidates in a fine scale (see Figure 2.4).

Multiresolution Technique with Different Image Sizes for Previous Frame.

Reducing the number of search positions and the number of pixels in residual calculation can also reduce computation. The multiresolution motion estimation algorithms rely on the technique of predicting an approximate large-scale motion vector in a coarse-resolution video and refining the estimated motion vector in a multiresolution fashion to achieve the motion vector in the finer resolution. The size of the image is smaller at a coarser level (i.e., of a pyramid form). Since A block at the coarser level represents a larger region than a block with the same number of pixels at the finer level, a smaller search area can be used at coarser levels. In addition, multiresolution motion estimation algorithms also reduce the number of pixels in residual calculation. These algorithms can be further divided into two groups: constant block size and variable block size.

1. In [68, 101], the same block size is used at each level. If the image size is reduced to half as the level becomes more coarse, one block at a coarser level covers four corresponding blocks at the next finer level. In this way, the motion vector of the coarser-level block is either directly used as the initial estimate for the four corresponding finer-level blocks [68] or interpolated to obtain four motion vectors of the finer level [101].
2. In [112], different block sizes are employed at each level to maintain a one-to-one correspondence between blocks in different levels. As a result, the motion vector of each block can directly be used as an initial estimate for the corresponding block at the finer level.

Multiresolution Technique with Same Image Size for Previous Frame.

In [27], instead of reducing the number of search locations, the multiresolution method trades the number of search locations for better estimation quality. This method uses different image resolutions with the same image size of a pyramid form. Since the same image size is used at each level, the number of possible motion candidates is the same at each level. The block size is not the same at each level and is reduced by half as the level becomes coarser. A block at the coarser level represents the same region as that at the finer level. Then, in the coarsest level, a set of motion candidates is selected from the maximum motion candidate set using a full search with fewer pixels in residual calculation. In each of the finer levels, the motion candidate set is further screened. At the last level, only a single motion vector is selected.

Multi-scale Optical-Flow Technique.

Based on a 3D Markov model for motion vector fields, Kim and Woods use an extended Kalman filter as a pel-recursive estimator [55]. The three dimensions consist of the two spatial dimensions plus a scale (coarse-to-fine) dimension.

Multiresolution-Spatio-Temporal Correlation Technique.

Based on spatio-temporal correlation integrating with a multiresolution scheme, a fast motion estimation, which is about 2 orders of magnitude faster than full search motion estimation algorithms, is introduced by Chalidabhongse and Kuo [11].

We also present a novel true motion tracker based on successive refinement of motion-vector candidates on images of different resolutions. Although many similar multiresolution approaches have been proposed, there is no neighborhood relaxation in conventional

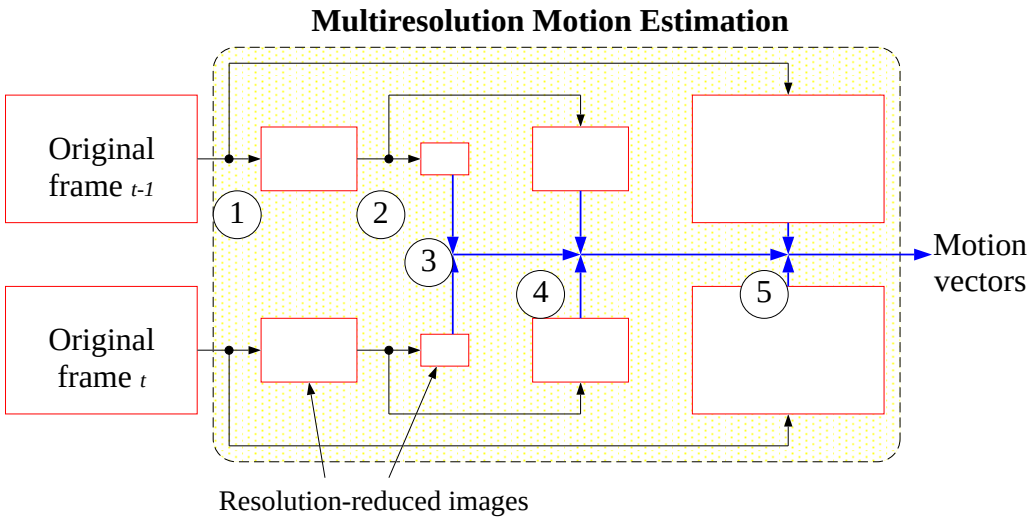


Figure 2.4: The 3-level multiresolution motion estimation scheme has five steps: (1) and (2) reduce the image size into a quarter along both directions. (3) performs a full-search motion estimation in the coarse scale. (4) and (5) refine the motion-vector candidates in the fine scale.

multiresolution motion estimation algorithms. Our method is the first one that integrates multiresolution with neighborhood relaxation.

Step 1 *The algorithm starts with a search on the images of the most-coarse resolution.*

The third level images are divided into subblocks of 8×8 pixels, as shown in Figure 2.5. (Each of the subblocks can search in the $\pm 3 \times \pm 3$ positions when the original search distance is ± 15.) The SADs are denoted as

$$\mathrm{SAD}_8^{(k)}(i,j,\vec{v}) \equiv \sum_{x=0}^{7}\sum_{y=0}^{7} |I^{(k)}(8i+x,8j+y,t) - I^{(k)}(8i+x+v_x,8j+y+v_y,t-1)|$$

where $k=2$ and $\vec{v}=[v_x,v_y]^T$. Here, (i,j) in $\mathrm{SAD}_8^{(k)}(i,j,\vec{v})$ is the subblock coordinate, i.e., a pixel at $[x,y]^T$ is in the subblock at $[\lfloor x/8 \rfloor, \lfloor y/8 \rfloor]^T$. (When the picture size is 352×288 pixels, there are 44×36 subblocks.)

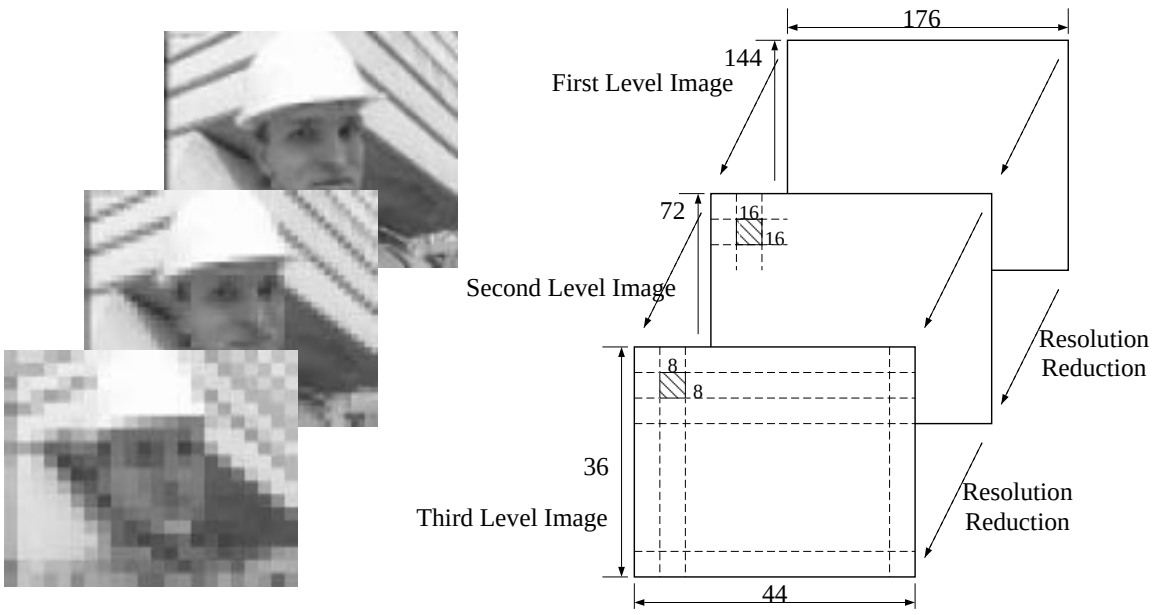


Figure 2.5: The first level images are the images of original resolution. The second level images are the images of a quarter resolution of the first level. (A pixel in second level corresponds to the low-pass filtering of four pixels in the corresponding position.) The third level images are a quarter of the second level.

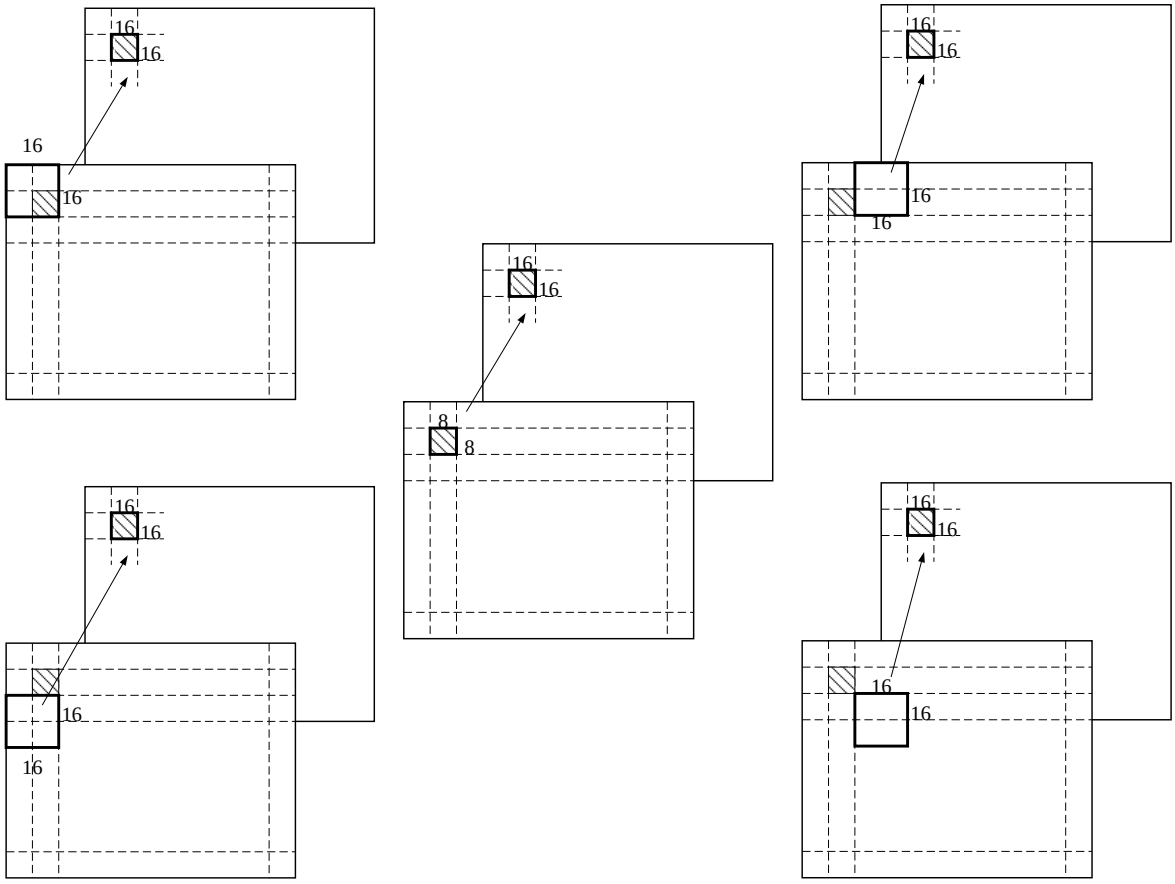


Figure 2.6: Each macroblock in the second-level image is covered by one macroblock in the third-level image and one subblock in the third-level image. Hence, a macroblock on this level will at least inherit the two motion-vector candidates (one from the subblock and one from the macroblock) from the third level as the base motion vectors. In order to capture the motion vectors of the neighborhood, we also take the motion vectors from three nearest-neighbor macroblocks (in the third-level image) into account. Therefore, a macroblock on this level will inherit the *five* motion-vector candidates (one from the subblock and four from the macroblocks) from the third level as the base motion vectors.

Without too much computational overhead, the SADs of non-overlapping macroblocks (of 16×16 pixels) then can be computed as

$$\begin{aligned} \mathrm{SAD}_{16}^{(k)}(i,j,\vec{v}) &\equiv \sum_{x=0}^{15}\sum_{y=0}^{15} |I^{(k)}(16i+x,16j+y,t) - \\ &\qquad I^{(k)}(16i+x+v_x,16j+y+v_y,t-1)| \\ &= \sum_{\Delta i=0}^{1}\sum_{\Delta j=0}^{1} \{\mathrm{SAD}_{8}^{(k)}(2i+\Delta i,2j+\Delta j,\vec{v})\} \end{aligned}$$

Here, (i,j) in $\mathrm{SAD}_{16}^{(k)}(i,j,\vec{v})$ is the macroblock coordinate, i.e., a pixel at $[x,y]^T$ is in the macroblock at $[\lfloor x/16 \rfloor, \lfloor y/16 \rfloor]^T$. (When the picture size is 352×288 pixels, there are 22×18 macroblocks.)

In conventional multiresolution motion estimation algorithms, only one of either $2 \times \arg\min_{\vec{v}}\{\mathrm{SAD}_{8}^{(k)}(\vec{v})\}$ or $2 \times \arg\min_{\vec{v}}\{\mathrm{SAD}_{16}^{(k)}(\vec{v})\}$ (but not both) is used as the motion candidate for the refinement in the second level images. (There is a multiplicand of 2 because a motion vector in level-$(k-1)$ is twice as large as the same motion vector in level-*k*.) We observe that the motion vector for the macroblock (from SAD_{16}) is better at capturing the global common motion when the macroblock is inside a moving object. On the other hand, the motion vector for the subblock (from SAD_{8}) is better at capturing its own true motion when the macroblock covers two or more moving objects. Hence, we select the motion vectors that carry minimal SADs (either for subblocks or for macroblocks) as the candidates, as shown here:

$$\begin{aligned} V^{(k-1)}(i,j) &= \{2 \times \arg \min_{\vec{v} \in V^{(k)}(\lfloor \frac{i}{2} \rfloor, \lfloor \frac{j}{2} \rfloor))\}} \{\mathrm{SAD}_{8}^{(k)}(i,j,\vec{v})\}, \\ &\qquad 2 \times \arg \min_{\vec{v} \in V^{(k)}(\lfloor \frac{i-1}{2} \rfloor, \lfloor \frac{j-1}{2} \rfloor))\}} \{\mathrm{SAD}_{16}^{(k)}(\left\lfloor \frac{i-1}{2} \right\rfloor, \left\lfloor \frac{j-1}{2} \right\rfloor, \vec{v})\}, \\ &\qquad 2 \times \arg \min_{\vec{v} \in V^{(k)}(\lfloor \frac{i-1}{2} \rfloor, \lfloor \frac{j+1}{2} \rfloor)} \{\mathrm{SAD}_{16}^{(k)}(\left\lfloor \frac{i-1}{2} \right\rfloor, \left\lfloor \frac{j+1}{2} \right\rfloor, \vec{v})\}, \end{aligned}$$

$$2 \times \arg \min_{\vec{v} \in V^{(k)}(\left[\frac{i+1}{2}\right], \left[\frac{j-1}{2}\right])} \{\mathrm{SAD}_{16}^{(k)}(\left[\frac{i+1}{2}\right], \left[\frac{j-1}{2}\right], \vec{v})\},$$

$$2 \times \arg \min_{\vec{v} \in V^{(k)}(\left[\frac{i+1}{2}\right], \left[\frac{j+1}{2}\right])} \{\mathrm{SAD}_{16}^{(k)}(\left[\frac{i+1}{2}\right], \left[\frac{j+1}{2}\right], \vec{v})\}\}$$

where V denotes the set of motion vector candidates and $V^{(3)} = \{\pm 3 \times \pm 3\}$ when the original search range is ± 15.

Step 2 *The motion vector candidates are refined on the images of the finer resolution.*

As shown in Figure 2.6, a macroblock on this second level will inherit the five motion-vector candidates (one from subblock and four from macroblocks) from the third level as the base motion vector candidates. Then, the subblocks will search in the $\pm 1 \times \pm 1$ window around these five motion vectors. The motion vectors that carry minimal SADs are selected (either for the subblocks of 8×8 pixels or the macroblocks of 16×16 pixels) as the motion candidates for the first level.

Step 3 *In the final step of this method, only macroblocks of the finest resolution require motion estimation.*

Again, a macroblock on this level will inherit five motion vectors from the second level as the base motion vectors, and then search in the $\pm 1 \times \pm 1$ window around these five motion vectors for the final true motion vector. The motion vector that carries minimal SAD is selected.

The resolution reduction also alleviates the computational burden of calculating the local variations $\vec{\delta}$ because the local variations disappear after the resolution reductions. Moreover, the successive refinement procedure in the finer resolution serves the same purpose of the iterative procedure in the common neighborhood relaxations, which we omit in Eq. (2.11).

Chapter 3

Application in Compression: Rate-Optimized Video Compression and Frame-Rate Up-Conversion

Video compression can make use of the true motion tracker in various shapes, such as rate-optimized motion vector coding, object-based video coding, and object-based global motion compensation [19, 24, 52]. In this chapter, we demonstrate that *the proposed true motion tracker (TMT) can provide higher coding efficiency and better subjective visual quality than conventional minimal-residue block-matching algorithms.*

Video compression plays an important role in many multimedia applications, from video-conferencing and video-phone to video games. The key to achieving compression is to remove temporal and spatial redundancies in video images. Block-matching motion estimation algorithms (BMAs) have been widely exploited in various international video compression standards to remove temporal redundancy.

For differentially encoded motion vectors, we observe that a piecewise continuous motion field reduces the bit-rate. Hence, we propose a rate-optimized motion estimation algorithm based on the neighborhood relaxation TMT. The unique features of this algorithm come from two parts: (1) we incorporate the number of bits for encoding motion vectors

into the minimization criterion, and (2) instead of counting the actual number of bits for motion vectors, we approximate the number of bits by the residues of the neighborhood.

This algorithm has two advantages: (1) its bit-rate is lower than the original coder using full-search motion-estimation algorithms, and (2) its computational complexity is lower than the computational complexity of the rate-distortion optimized method.

In addition, we present a motion-compensated frame-rate up-conversion scheme using the decoded motion. Such use of the decoded motion can save computation on the decoder side. The more accurate the motion information is, the better the performance of frame-rate up-conversion will turn out to be. Hence, using the true motion vectors for the compression results in a better picture quality of frame-rate up-conversion than using the motion vectors estimated by the minimal-residue block-matching algorithms (BMA).

3.1 Rate-Distortion Optimized Motion Estimation

In most video compression algorithms, a tradeoff between picture quality and compression ratio (and computational cost) is unavoidable. Generally speaking, the lower the compression ratio, the better the picture quality. Considerable efforts have been made to develop new (better) algorithms that can

- *achieve higher picture quality with the same number of bits*, i.e., minimize the total distortion of intra-frames (D_I) and inter-frames (D_i)

  $$D_{total} = D_I + D_i$$

  under a bit rate constraint:

  $$\begin{aligned} B_{total} &= \sum B_I + \sum B_i \\ &= \sum B_I + \sum (B_{i\_res} + B_{i\_mv}) \\ &\leq B_{constraint} \end{aligned}$$

where B_I stands for the number of bits for intra-frames, B_i stands for the number of bits for inter-frames, B_{i_res} stands for the number of bits for inter-frame residues, and B_{i_mv} stands for the number of bits for inter-frame motion vectors.

- *achieve the same picture quality with fewer bits*,

  $$B_{total} = \sum B_I + \sum (B_{i\_res} + B_{i\_mv})$$

  under the same

  $$D_{total} = D_I + D_i.$$

This section discusses the rate-distortion optimized motion estimation algorithm, which can ascertain best tradeoff points in the rate-distortion curves.

Minimal-residue criterion.

In high-quality video compression applications (e.g., video broadcasting), quantization scales are usually low (see Figure 1.4). And so, the number of bits for residues B_{i_res} dominates the total number of bits B_{total}. The following was generally believed (until around 1996): the less the residue is, the fewer the bits for residues B_{i_res} will be, and, thus, the less the total bit rate B_{total} will turn out to be. Hence, the minimal-residue criterion is still widely used in BMAs as:

$$\textit{motion vector} = \arg\min_{\vec{v}} \{\mathrm{SAD}(\vec{v})\} \tag{3.1}$$

Namely, the motion vector for this block is the displacement vector that carries the minimal-SAD (the sum of the absolute differences, see Eq. (1.7), Section 1.3.2).

Among numerous motion estimation algorithms [11, 36], the full-search methods, where the residues of all possible displaced candidates within the search area in the previous frame are compared, give the best solution in terms of estimation error.

However, the full-search BMAs are flawed because they:

1. usually do not produce the *true motion field*, physical motion, which could produce better subjective picture quality (see Section 1.1).

2. cannot generally produce the *optimal bit rate* for low bit-rate video coding standards (as we will explain in a moment).

3. are *computationally* expensive for a practical real-time multimedia application (see Chapter 6).

Minimal rate-distortion criterion.

The total number of bits B_{total} also includes the number of bits of coding motion vectors B_{i_mv}. In some coding standards (e.g. H.261, H.263, MPEG-1, MPEG-2, MPEG-4, which encode the motion vectors differentially within a slice [3, 51]), the following is not always true: the less the residue is, the less the bit rate will be. Consequently, those conventional block-matching algorithms (BMAs), which treat the motion estimation problem as an optimization problem on residue only, could suffer from a high price on the differential coding of motion vectors [14].

Figure 3.1 shows the bit requirement of the motion-vector difference in the MPEG standards and the H.263 standard. The smaller the difference, the fewer the bits required. A rate-distortion optimized algorithm or a rate-optimized motion estimation algorithm should take into account the total number of bits when determining the motion vectors:

$$\begin{aligned}\{\vec{v}_i\}_{i=1}^{n} \;=\; & \arg\min_{\{\vec{v}_i\}}\{bits(\text{residue}(B_1,\vec{v}_1),Q_1)+bits(\vec{v}_1) \\ & + bits(\text{residue}(B_2,\vec{v}_2),Q_2)+bits(\Delta\vec{v}_2) \\ & + \cdots \\ & + bits(\text{residue}(B_n,\vec{v}_n),Q_n)+bits(\Delta\vec{v}_n)\} \end{aligned} \tag{3.2}$$

where B_i stands for the block i, $\vec{v}_i$ is the motion vector of block i, $\Delta\vec{v}_i = \vec{v}_i - \vec{v}_{i-1}$*, residue$(B_i,\vec{v}_i)$ represents the residue of block i, Q_i stands for the quantization parameters

*In fact, $\Delta\vec{v}_i \equiv \vec{v}_i - $ *prediction of* $\vec{v}_i$ [3, 51]. In this work, we assume *prediction of* $\vec{v}_i = \vec{v}_{i-1}$ for simplicity.

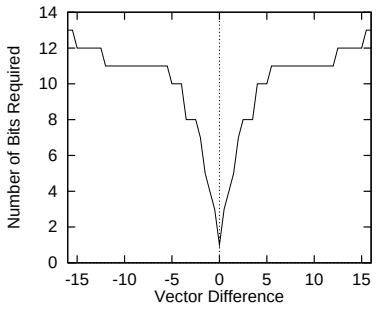


Figure 3.1: Variable length coding in motion vector difference used in the MPEG standards and the H.263 standard.

of B_i, and $bits(\text{residue}(B_i, \vec{v}_i), Q_i)$ is the number of bits required for this residue.

In [14], Chen and Willson formulated the motion estimation problem as a shortest-path finding problem, minimizing the number of bits for texture and for motion as given in Eq. (3.2). They used Viterbi-type dynamic programming to determine the optimal motion vectors. Calculating the number of bits for a residue is computationally expensive (such an undertaking includes a DCT transform, a quantization, a run-length coding, and a variable length coding). This technique does achieve good bit rates. In [13], Chen, Villasenor, and Park presented an alternative motion estimation algorithm that considers rate-distortion trade-offs in a multiresolution framework. Their algorithm reduces the computational complexity but increases the bit rates.

3.2 Neighborhood-Relaxation Motion Estimation for Rate Optimization

Since a piecewise continuous motion field is so attractive in reducing the bit rate for differentially encoded motion vectors, this section presents a neighborhood relaxation formulation for the rate-optimized motion estimation.

Eq. (3.2) can be written as

$$\textit{motion of } B_i \approx \arg\min_{\vec{v}} \{bits(\text{residue}(B_i, \vec{v}), Q_i) + bits(\Delta\vec{v})\}$$

by assuming that the motion vectors of other blocks are fixed. Because expressing mathematically the bit costs for different residues and $\Delta\vec{v}$ is extraordinarily difficult, the above is first simplified into the following approximation

$$\begin{aligned} \textit{motion of } B_i &\approx \arg\min_{\vec{v}} \left\{ \frac{\alpha_1}{Q_i} \mathrm{SAD}_i(\vec{v}) + \alpha_2 ||\Delta\vec{v}|| \right\} \\ &\approx \arg\min_{\vec{v}} \{ \mathrm{SAD}_i(\vec{v}) + \beta ||\Delta\vec{v}|| \} \end{aligned} \tag{3.3}$$

because (1) the $bits(\text{residue}(B_i, \vec{v}), Q_i)$ increases when $SAD_i(\vec{v})$ increases and Q_i decreases; (2) $bits(\Delta\vec{v})$ increases when $SAD_i(\vec{v})$ and $||\Delta\vec{v}||$ increases. The $\beta = \alpha_2 Q_i / \alpha_1$.

First, we assume that the rate-optimized motion vector of B_i's neighbor produces close-to-minimal SAD. Say, B_j is the neighbor of B_i and $\vec{v}_j^*$ is the known optimal motion vector of B_i:

$$\vec{v}_j^* = \arg\min_{\vec{v}} \{ \mathrm{SAD}(B_j, \vec{v}) \} \tag{3.4}$$

Second, we assume that $\mathrm{SAD}(B_j, \vec{v})$ increases linearly as $\vec{v}$ deviates from $\vec{v}_j^*$ according to

$$\mathrm{SAD}(B_j, \vec{v}) \approx \mathrm{SAD}(B_j, \vec{v}_j^*) + \gamma ||\vec{v} - \vec{v}_j^*|| \tag{3.5}$$

or

$$||\Delta\vec{v}|| = ||\vec{v} - \vec{v}_j^*|| \approx \gamma^{-1} (\mathrm{SAD}(B_j, \vec{v}) - \mathrm{SAD}(B_j, \vec{v}_j^*)) \tag{3.6}$$

Substituting Eq. (3.6) into Eq. (3.3), we have

$$\textit{motion of } B_i \approx \arg\min_{\vec{v}} \{ \mathrm{SAD}(B_i, \vec{v}) + \mu \sum_{B_j \in N(B_i)} \left(\mathrm{SAD}(B_j, \vec{v}) - \mathrm{SAD}(B_j, \vec{v}_j^*) \right) \} \tag{3.7}$$

where $N(B_i)$ means the neighbor blocks of B_i ($\mu = \beta/\gamma = \alpha_2 Q_i / \alpha_1 \gamma$).

Here we use an idea commonly adopted in the relaxation method: we let $\mathrm{SAD}(B_j, \vec{v}_j^*)$ remain constant during the block i updating of the neighborhood relaxation. Therefore, the $\mathrm{SAD}(B_j, \vec{v}_j^*)$ can be dropped from Eq. (3.7), resulting in

$$\textit{motion of } B_i \approx \arg\min_{\vec{v}} \{ \mathrm{SAD}(B_i, \vec{v}) + \mu \sum_{B_j \in N(B_i)} \mathrm{SAD}(B_j, \vec{v}) \} \tag{3.8}$$

If a motion vector can reduce the residue of the center block and reduce the residues of the neighbors as well, then this motion vector will be selected for the encoder. That is, when two motion vectors produce similar residues, the one closer to the neighbors' motion will be selected. The motion field produced by this method is smoother than that produced by the minimal-residue criterion (see Eq. (3.1)).

The above approach is inadequate for modeling non-translational motion, such as object rotation, zooming, and approaching [86]. For example, an object is rotating counterclockwise in Figure 2.3(b). In this case, the neighbor blocks have small motion differences. Because Eq. (3.8) assumes that the neighboring blocks will move in the same translational motion, this approach may not adequately model the rotational motion. Since the neighboring blocks may not have uniform motion vectors, a further relaxation on the neighboring motion vectors is introduced (see Eq. (2.11), Section 2.3):

$$\textit{motion of } B_i \quad = \quad \arg\min_{\vec{v}}\{\mathrm{SAD}(B_i,\vec{v}) + \sum_{B_j \in N(B_i)} (\mu_{i,j} \times \mathrm{SAD}(B_j,\vec{v}+\vec{\delta})\} \tag{3.9}$$

where a small $\vec{\delta}$ is incorporated to allow local variations of motion vectors among neighboring blocks, which comes from the non-translational motions, and $\mu_{i,j}$ is the weighting factor for different neighboring blocks[†]. As illustrated in Figure 2.3, this can in principle track more flexible motions, such as rotation, zooming, shearing, etc.

3.2.1 Coding Efficiency of Neighborhood-Relaxation Motion Estimation

We incorporated Eq. (3.9) into the baseline H.263 video codec provided by Telenor R&D [96].

The motion vectors found by the original minimal-residue-based approach and by our neighborhood relaxation method are shown in Figure 3.2[‡]. The motion field of our method

[†]The shorter the distance between B_i and B_j, the larger the $\mu_{i,j}$. The larger the Q_i, the larger the $\mu_{i,j}$. In practice, we use the 4 nearest neighbors with $\mu_{i,j} \in [0.05, 0.40]$.

[‡]A block of 16×16 pixels is used as a macroblock for motion vector estimation. Only forward prediction

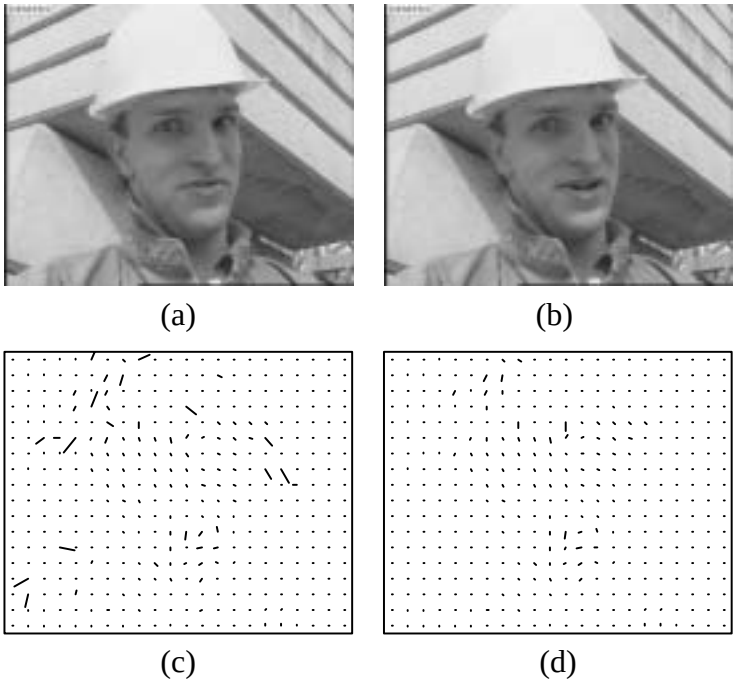


Figure 3.2: Comparison between the motion estimation algorithm using the minimal-residue criterion and using our neighborhood relaxation formulation. (a)(b) show the 105th frame and the 108th frame of the "foreman" sequence. (c) shows the motion vectors found by the original approach which is based on the minimal residue criterion. (d) shows the motion vectors found by our neighborhood relaxation method.

is smoother and, consequently, the number of bits for coding motion vectors is smaller. Using a fixed quantization parameter, our method can achieve *13.9%* bit-rate reductions (25.4% bit-rate reductions in coding motion vectors) as well as a *higher* (+0.02 dB) signal-to-noise ratio (SNR) in coding the 108th frame of the "foreman" sequence.

In the full-search motion estimation on the minimal SAD, two possible situations will produce a small SAD: (1) tracking after the true motion, and (2) tracking after noise. In general, the larger the residue, the lower the SNR after the quantized DCT coding. However, even when two residual blocks have the same SAD, some differences still exist in their SNRs and bit counts [75]. The DCT-based quantization tends to truncate high-frequency components. When two residual blocks have the same SAD, the one that has predominately higher frequency components will have a lower SNR. In the full-search BMA on the minimal SAD, occasionally some small SADs are the result of tracking after the noise effect. Those blocks usually have more high-frequency components and hence have lower SNRs [13, 66]. Our true motion tracker is deliberately made impervious to noise. Therefore, our TMT could give even higher SNR after judicious bit-rate saving.

Figure 3.3 shows the rate-distortion curves for the H.263 QCIF test sequences [1][§]. Clearly, our proposed neighborhood relaxation motion estimation algorithm performs as well as (if not better than) the original minimal-residue motion estimation algorithm. When the quantization step is coarse, the cost of residue coding is relatively smaller and thus the cost of coding the motion vectors becomes dominant. In this case, our method results in a better picture quality and a lower bit rate. (As illustrated in the lower-left corner of Figure 3.3(f), the improvement increases as the quantization step grows.)

Figure 3.4 demonstrates the visual quality difference between two motion estimation algorithms. Because the motion field estimated by the neighborhood relaxation TMT is much smoother than the motion field estimated by the minimal-residue BMA, the block

is implemented in the experiments. The maximum horizontal and vertical search displacement is ± 16.

[§]QCIF: 176 pixels $\times$ 144 pixels

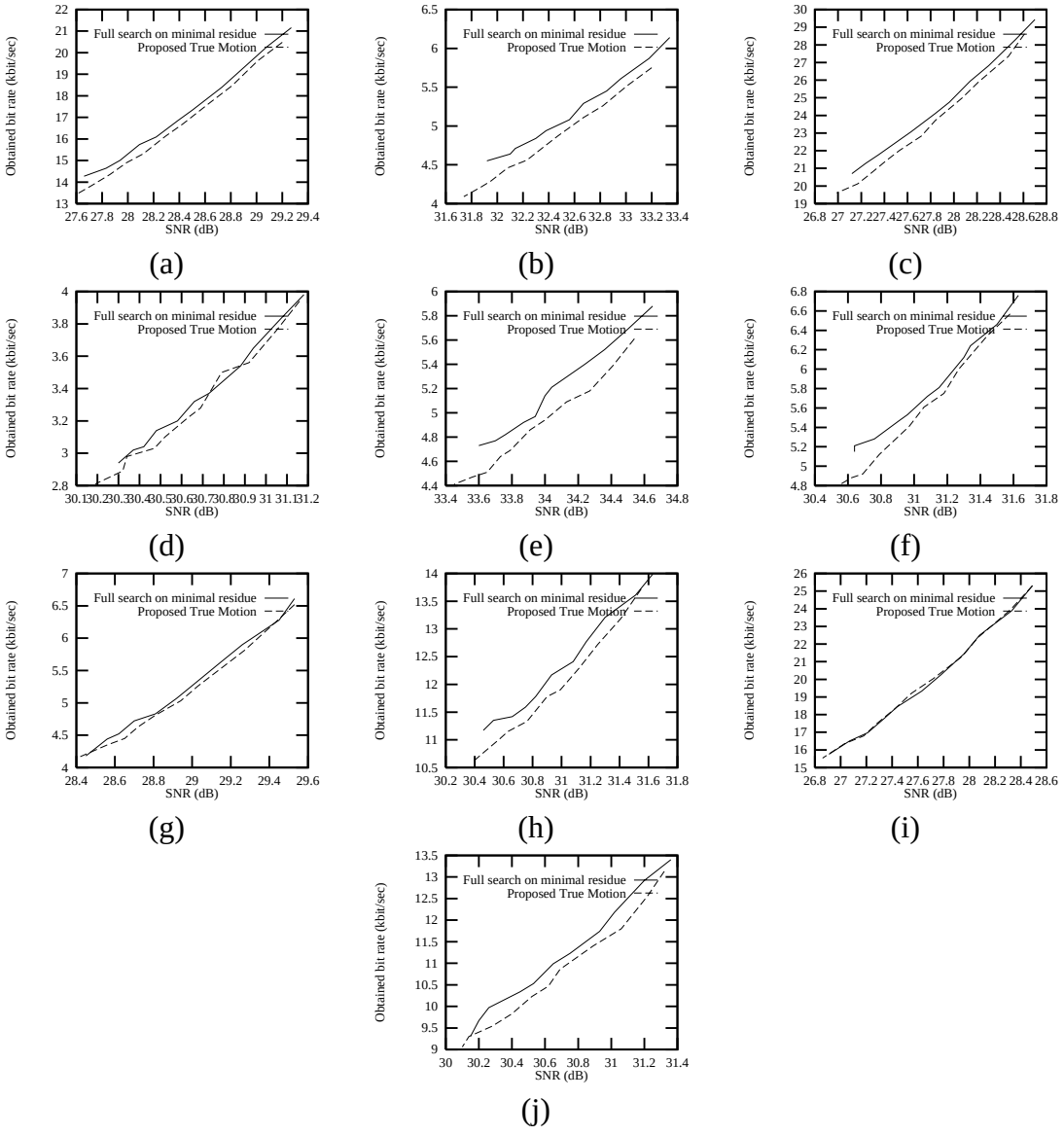


Figure 3.3: The rate-distortion curves for all the H.263 sequences. (a) shows the rate-distortion curve for the carphone sequence (average over the 380 frames). (b) claire (490 frames). (c) foreman (300 frames). (d) grandma (860 frames). (e) miss-am (150 frames). (f) mthr-dotr (300 frames). (g) salesman (445 frames). (h) suzie (150 frames). Because there is a scene change in the 60th frame of the trevor sequence, we divide the simulation of the trevor into two parts. (i) first part of the trevor (60 frames) and (j) second part of the trevor (90 frames).

artifacts are less noticeable.

3.2.2 Coding Efficiency of Multiresolution Motion Estimation

We also incorporated the multiresolution tracking algorithm described in Section 2.5 into the baseline H.263 video codec provided by Telenor R&D. Motion estimation is known to be the main bottleneck in real-time encoding applications, and the search for an effective motion estimation algorithm has been a challenging problem for years. The multiresolution algorithm described in Section 2.5 not only reduces the bit-rate but also reduces computational complexity (7 times faster than the conventional full-search block-matching algorithm).

Figure 3.5(a) shows the motion vectors found by the multiresolution method without neighborhood relaxation and our multiresolution motion estimation. These results are based on the 105th frame and the 108th frame of the "foreman" sequence (as shown in Figure 3.2). The motion field of our method is smoother than that of the full-search BMA. As a result, the number of bits for coding motion vectors is smaller. Using a fixed quantization parameter, our method can achieve *10.7%* bit-rate reductions (21.3% bit-rate reductions in coding motion vectors) as well as a *higher* (+0.01 dB) SNR in coding the 108th frame of the "foreman" sequence.

Note that Figure 3.5(b) also shows the motion vectors found by the multiresolution method *without* neighborhood relaxation. Similar to the results given by our TMT, this method also produces smoother motion field than the original full-search method. Thus, it lowers the bit rate by 7.2% (it reduces the bits for motion vectors by 25.5%). Alas, it *degrades* the SNR by -0.06 dB.

Figure 3.6 further shows that our new algorithm is also better than the previous multiresolution algorithm. The H.263 test sequence library can be categorized into the following classes: (1) Low spatial detail and a medium amount of movement (e.g., miss-am, mthr-dotr). (2) Medium spatial detail and a medium amount of movement (e.g., carphone,

foreman, suzie, second part of trevor). (3) High spatial detail and a small amount of movement (e.g., claire, grandma, salesman). (4) High spatial detail and a large amount of local movement (e.g., first part of trevor). When there is high spatial detail and a small amount of movement, there are more bits for texture and fewer bits for motion vectors. Therefore, potential for motion vector savings is less. As shown in Figure 3.6 (b), (d), and (g), three algorithms perform almost the same. On the other hand, (a), (c), (e), (f), (h), (i), and (j) show significant improvement (from 0.2 dB up to 1.5 dB) from the proposed algorithm to the conventional multiresolution search algorithm. In all the test sequences except (e), the differences between the proposed algorithm and the full-search BMA are within 0.1 dB range. *In sequence (e), the proposed algorithm shows 0.2 dB better than the full-search method.*

3.3 Frame-Rate Up-Conversion

Frame-rate up-conversion has attracted much attention in recent years. To accomplish acceptable coding results at low bit-rates, most codecs reduce the temporal resolution, but often at the expense of having to bring the received video back up to its original frame rate.

The frame-rate up-conversion problem can be formulated as the following: to find $\{I(2t+1)\}$ given $\{I(2t)\}$ (see Figure 3.7). The schemes for this frame-rate up-conversion problem can be categorized as:

1. **No interpolation:**

   Typically, the last decoded frame is repeated until a new one is received, i.e., $I(2t+1) = I(2t)$. This simple solution creates jerkiness in areas of large and complex motion, especially when the time delay between transmitted frames is relatively long.

2. **Non-motion-compensated interpolation:**

   Linear interpolation averages the pixel values between two transmitted frames to re-create the missing ones, i.e., $I(2t+1) = \{I(2t) + I(2t+2)\}/2$. This scheme leads to

blurriness and ghost artifacts in areas of motion.

3. **Motion-compensated interpolation:**
   Motion-compensated schemes interpolate the pixel values along the motion trajectory between two frames. The motion-compensated interpolation provides the best solution in various up-sampling applications [10, 47, 54, 99] (especially, for moving objects).

In the next section, we present a frame-rate up-conversion method that "motion-compensated" interpolates the missing frames by utilizing decoded motion vectors. Our method has the following unique features:

1. Our approach is based on the decoded motion vectors and thus does not require a separate motion estimation just for the interpolation. Most previous methods are not designed for a coding environment and require the high-quality motion estimation, which is computationally expensive. For example, in [99], the receiver needs to perform a separate motion estimation just for the interpolation.

2. Our approach does not require proprietary information to be sent and hence can work with most video coding standards. As mentioned before, blocks located at the object boundaries contain regions moving in at least two different directions. A single motion vector compensates one or more of these regions incorrectly. In [54], the proposed algorithm considers multiple motion vectors for a single block so as to provide better picture qualities. However, this scheme requires extra motion information to be sent. In [47], the motion-compensated interpolation scheme is based on an object-based interpretation of the video. The interpolation scheme can use the decoded motion and segmentation information without refinement. (This is attributable to the fact that the object-based representation is close to the real world.) However, a proprietary codec is used.

3. In order to have better performance, we use the TMT (see Eq. (3.9)) at the encoder instead of the original minimal-residue BMAs (see Eq. (3.1)). The more accurate the motion information, the better the performance of frame-rate up-conversion.

3.4 Motion-Compensated Interpolation Using Transmitted Motion Vectors

In this section, we present our motion-compensated interpolation scheme for frame-rate up-conversion, which uses the decoded motion vectors (cf. Figure 3.8). The proposed interpolation scheme is based on the following premise: As shown in Figure 3.9, if block B_i moves $\vec{v}_i$ from frame F_{t-1} to frame F_{t+1}, then it is likely that block B_i moves $\vec{v}_i/2$ from frame F_{t-1} to frame F_t, i.e.,

$$I(\vec{p}-\vec{v}_i,t-1) = I(\vec{p}-\frac{\vec{v}_i}{2},t) = I(\vec{p},t+1) \quad \forall \vec{p} \in B_i$$

Basic formulation.

The basic technique of motion-based frame-rate up-conversion is to interpolate frame F_t based on frame F_{t-1}, frame F_{t+1}, and block motion vectors $\{\vec{v}_i\}$ as the following:

$$\tilde{I}(\vec{p}-\frac{\vec{v}_i}{2},t) = \frac{1}{2}\Big\{I(\vec{p}-\vec{v}_i,t-1)+I(\vec{p},t+1)\Big\} \quad \forall \vec{p} \in B_i \tag{3.10}$$

The pixel intensity at frame F_t is reconstructed as the average of the intensity of the corresponding pixels at frame F_{t-1} and at frame F_{t+1}, in order to reduce the reconstruction noise.

Motion estimation on the encoder side.

There are a number of choices of motion estimation algorithms on the encoder side.

1. The minimal residue BMAs as widely adopted in most coding standards, see Eq. (3.1).

2. The more accurate the motion estimation ($\vec{v}_i/2$), the smaller the reconstruction error ($\sum||\tilde{I}(\vec{p},t)-I(\vec{p},t)||$), i.e., the higher the quality of the motion-based interpolation. Therefore, one possible technique of frame-rate up-conversion using transmitted motion is to encode F_{t-1}, F_{t+1}, $\cdots$ with $\{2\hat{v}_i\}$ where $\hat{v}_i$ is the motion vector of B_i from F_t to F_{t+1}. In this case, the reconstruction error will be minimized, but the rate-distortion curves may not be optimized.

3. Another possible motion estimation algorithm is to use the neighborhood relaxation TMT (see Eq. (3.9)). We show that the neighborhood relaxation TMT captures the true movement of the block more accurately than the minimal-residue BMAs do. It is likely that $\vec{v}_i/2$ using Eq. (3.9) is more accurate than $\vec{v}_i/2$ using Eq. (3.1) and hence produces better frame-rate up-conversion results. In addition, we have just shown that it provides better rate-distortion curves.

4. Another possible motion estimation algorithm is to use the spatial and temporal neighborhood relaxation formulation (see Eq. (2.13), Section 2.4):

$$\begin{aligned} score(B_{i,j},\vec{v}) &= \mathrm{SAD}(B_{i,j},0,\vec{v},t,t-2) \\ &\quad + \sum_{B_{k,l}\in N(B_{i,j})} W(B_{k,l},B_{i,j}) \min_{\vec{\delta}}\{\mathrm{SAD}(B_{k,l},0,\vec{v}+\vec{\delta},t,t-2)\} \\ &\quad + W_t\Big\{\min_{\vec{\delta}}\{\mathrm{SAD}(B_{i,j},0,\frac{\vec{v}}{2}+\vec{\delta},t,t-1)\} \\ &\qquad + \min_{\vec{\delta}}\{\mathrm{SAD}(B_{i,j},\frac{\vec{v}}{2}+\vec{\delta},\vec{v},t-1,t-2)\}\Big\} \end{aligned} \tag{3.11}$$

where the SAD is defined as

$$\mathrm{SAD}(B_{i,j},\vec{v}_1,\vec{v}_2,t_1,t_2) \equiv \sum_{\vec{p}\in B_{i,j}} |I(\vec{p}+\vec{v}_1,t_1)-I(\vec{p}+\vec{v}_2,t_2)|$$

When we find the motion vector for block $B_{i,j}$ from frame $t-2$ to frame t as:

$$\textit{motion of } B_{i,j} = \arg\min_{\vec{v}}\{score(B_{i,j},\vec{v})\}$$

the first term of Eq. (3.11) minimizes the residue coding for $B_{i,j}$ itself, the second term minimizes the motion vector coding for $\vec{v}_{i,j}$, and the third term minimizes the motion interpolation error. This formulation reduces reconstruction errors at the cost of sacrificing coding efficiencies. Nevertheless, both of the coding efficiency and the reconstruction error of this formulation are better than or equal to conventional motion estimation algorithms (e.g., full-search BMAs).

Uncovered and occluded regions.

In Eq. (3.10), it is assumed that a pixel can be seen in all frames. However, in reality, a pixel may be occluded or uncovered (see Figure 2.1), i.e.,

$$I(\vec{p}-\vec{v}_i,t-1) = I(\vec{p}-\frac{\vec{v}_i}{2},t) \neq I(\vec{p},t+1)$$

or

$$I(\vec{p}-\vec{v}_i,t-1) \neq I(\vec{p}-\frac{\vec{v}_i}{2},t) = I(\vec{p},t+1)$$

Therefore, in addition to the motion interpolation formulation shown in Eq. (3.10), we use the following heuristics to interpolate the uncovered and the occluded regions:

1. **Uncovered region:** We identify a block B_i as an uncovered region, when it can be seen in F_t and F_{t+1} but not in F_{t-1}. When a block B_i in F_{t+1} is coded as the INTRA block[¶], it usually implies there is no matched displacement block in F_{t-1}. That is, B_i

[¶]To code a particular macroblock in a P- or B- video-object-plane (VOP), there are many coding-mode choices as specified by the MPEG-4 standard [3]. For one, the encoder must choose between INTRA and INTER coding [23]. The INTER mode uses motion compensation. First, the block matching motion estimation is used, for a current block, to find the best matched block. A motion compensated difference block (residual block) is formed by subtracting the pixel values of the predicted block from that of the current block point by point. Texture coding is then performed on the difference block. On the contrary, in the INTRA mode, texture coding is performed on the original texture of the macroblock. No motion compensation is required in this mode. Because most of the blocks in the current frame are similar to a predicted block in the existing frames, the residue of the block subtracted by a similar predicted block is usually small. In this case, a small number of bits can encode the residual block. Most of the time, coding the motion vector as well as coding the residue costs fewer bits than coding the texture of the block. In some cases (e.g., no textural information in the block, no matched displacement block), coding the difference (residue) block and the motion vector may require more bits than coding the original texture of the current block. In these cases, INTRA mode is chosen to encode the block. Here, when the INTRA mode is chosen for B_i, we assume that there is no matched displacement block for it.

is in the uncovered region (from F_{t-1} to F_{t+1}). Since we have no information about the uncovered region from F_t to F_{t+1}, we use a heuristic: a pixel is in the uncovered region if it (1) belongs to the corresponding location of an INTRA block B_i and (2) has not been motion compensated by other blocks. Hence, we have the following formulation:

$$\tilde{I}(\vec{p},t) = I(\vec{p},t+1) \quad \forall \vec{p} \in B_i \tag{3.12}$$

Note that since the block is INTRA coded, there is no motion information about the block. In this heuristic, we assume the occluded and uncovered regions are stationary ($\vec{v} = 0$). The reason is that object occlusion and reappearance often happen in the background. And, it is most likely that the background has no motion. Hence, zero motion vectors are used for the occluded and uncovered regions.

2. **Occluded region:** Similar to the uncovered region, we identify a block B_i as an occluded region, when it can be seen in F_{t-1} and F_t but not in F_{t+1}. All the blocks that are not in the occluded region can be motion compensated by the Eq. (3.10) and Eq. (3.12). When $\tilde{I}(\vec{p},t)$ has not been assigned any value by Eq. (3.10) and Eq. (3.12), it usually is in the occluded region. As a result,

$$\tilde{I}(\vec{p},t) = I(\vec{p},t-1) \tag{3.13}$$

To reduce the block artifacts.

When given frame F_{t-m} and frame F_{t+n}, our method to reconstruct the frame F_t can be summarized as follows:

$$\begin{aligned} I(\vec{p},t) \quad = \quad & \sum_{\{B_i\}} \sum_{\vec{p} \in B_i} w(\vec{p}, \vec{p}_i + \frac{n\vec{v}_i}{m+n}) \Big(\frac{n}{m+n} I(\vec{p} - \frac{m\vec{v}_i}{m+n}, t-m) + \\ & \frac{m}{m+n} I(\vec{p} + \frac{n\vec{v}_i}{m+n}, t+n) \Big) \end{aligned}$$

$$
\begin{aligned}
W(\vec{p},t) &= \sum_{\{B_i\}} \sum_{\vec{p}\in B_i} w(\vec{p},\vec{p}_i + \frac{n\vec{v}_i}{m+n}) \\
\tilde{I}(\vec{p},t) &= \begin{cases} I(\vec{p},t)/W(\vec{p},t) & \text{if} \quad W(\vec{p},t) \neq 0 \\ I(\vec{p},t-1) & \text{if} \quad W(\vec{p},t) = 0 \end{cases}
\end{aligned} \tag{3.14}
$$

where $\vec{v}_i$ is the movement of B_i from frame F_{t-m} to frame F_{t+n}, $\vec{p}_i$ is the pixel coordinate of the upper-left corner of B_i, and $w(\vec{p},\vec{p}_i)$ is the window function.

With respect to window function there are two two choices:

1. The conventional non-overlapped block motion compensation scheme:

   We set $w(\vec{p},\vec{p}_i) = 1$ when $\vec{p}$ is inside the B_i ($0 \leq$ the x-component and y-component of $\vec{p} - \vec{p}_i \leq 7$ when the block size is 8×8). We set $w(\vec{p},\vec{p}_i) = 1$ when $\vec{p}$ is outside the B_i. Because the motion tracker determines the motion vector $\vec{v}_i$ of B_i, the motion compensation is only applied to the B_i. If there is a small motion difference between $B_{i,j}$ and its neighbors, the conventional non-overlapped block motion compensation scheme will create noticeable block artifacts around the block boundary.

2. The overlapped block motion compensation scheme (OBMC):

   In the overlapped block motion compensation scheme, a pixel in $B_{i,j}$ will be motion-compensated by three motion vectors—one is the motion vector of $B_{i,j}$ and two are the motion vectors of the neighbor blocks of $B_{i,j}$. The *weighted* motion compensation scheme will (1) put more weights on the pixels that are closer to the block center and, (2) put fewer weights on the pixels that are on the block boundary, and (3) put even fewer weights on the pixels that are on the boundary of the neighbor blocks, as shown in Figure 3.10. This scheme smoothes out motion vector differences gradually from the pixels in this block to the pixels in the next block. Hence, the block artifacts will be reduced.

In order to reduce the block artifacts, we make the weighting function $w(\vec{p},\vec{p}_i)$ similar to

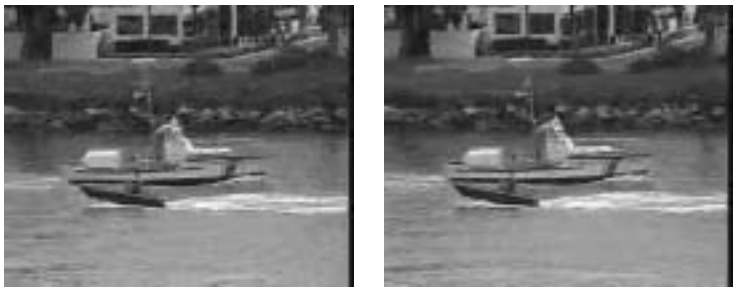
the coefficients defined for the overlapped block motion compensation scheme. In a huge purely-translational moving region, where all the motion vectors are the same, there is no difference between the conventional non-overlapped block motion compensation scheme and the overlapped block motion compensation scheme.

In addition, the weighting function can take into account the *confidence* of the tracking of the block. In general, the less the residue of a block, the higher the *probability of correctly tracking* the block (see Section 7.1). If the true motion tracker fails to track a block $B_{i,j}$ but can track its surrounding blocks, reducing the weighting factors of the block can relatively increase the weighting factors of its surrounding blocks, which have higher correctness. That is, we can propagate the motion information from high-confidence block to low-confidence block so as to have better reconstruction pictures.

3.4.1 Performance Comparison in Frame-Rate Up-Conversion

Figure 3.7 shows our performance comparison scheme in the frame-rate up-conversion using transmitted true motion. Table 3.1 shows the performance of our motion-based frame-rate up-conversion using the neighborhood relaxation true motion tracker (see Eq. (3.9)) compared with the performance of the motion-based frame-rate up-conversion using the original minimal-residue motion estimation (see Eq. (3.1)). Our simulation results show that our true motion estimation algorithm performs about 0.15dB–0.3dB better than the minimal-residue motion estimation method.

Figure 3.11 shows the frame-by-frame performance comparison of frame-rate up-conversion approaches. When there is almost no movement in the video, the difference between the two motion estimation algorithms is minimal. When there is some huge movement in the video, our method can show significant improvement. (Section 3.2.1 shows a similar result in video coding—the larger the motion, the greater the improvement gained by our proposed method.)



(a) (b)

Figure 3.4: The comparison between the proposed rate-optimized motion estimation algorithm and the original minimal-residue motion estimation algorithm. (a) shows the decoded 76th frame of the coastguard sequence using the original minimal-residue motion estimation algorithm. (b) shows the decoded 76th frame of the coastguard sequence using the proposed rate-optimized motion estimation algorithm. Because the motion field estimated by the neighborhood relaxation TMT is much smoother than the motion field estimated by the minimal-residue BMA, the block artifacts are less noticeable (especially, in the background).

Sequences	Original BMA	Our Method	SNR Improvement
akiyo	42.71 dB	42.87 dB	0.16 dB
coastguard	30.47 dB	30.62 dB	0.15 dB
container	40.47 dB	40.77 dB	0.30 dB
foreman	28.30 dB	28.63 dB	0.33 dB
hall_monitor	36.02 dB	36.15 dB	0.13 dB
mother_daughter	39.68 dB	39.93 dB	0.25 dB
news	32.78 dB	33.08 dB	0.30 dB
silent	34.01 dB	34.33 dB	0.32 dB
stefan	21.06 dB	21.20 dB	0.14 dB

Table 3.1: Comparison of different motion-based frame-rate up-conversion schemes. Our true motion tracker performs about 0.15dB–0.3dB SNR better than the minimal-residue motion estimation method.

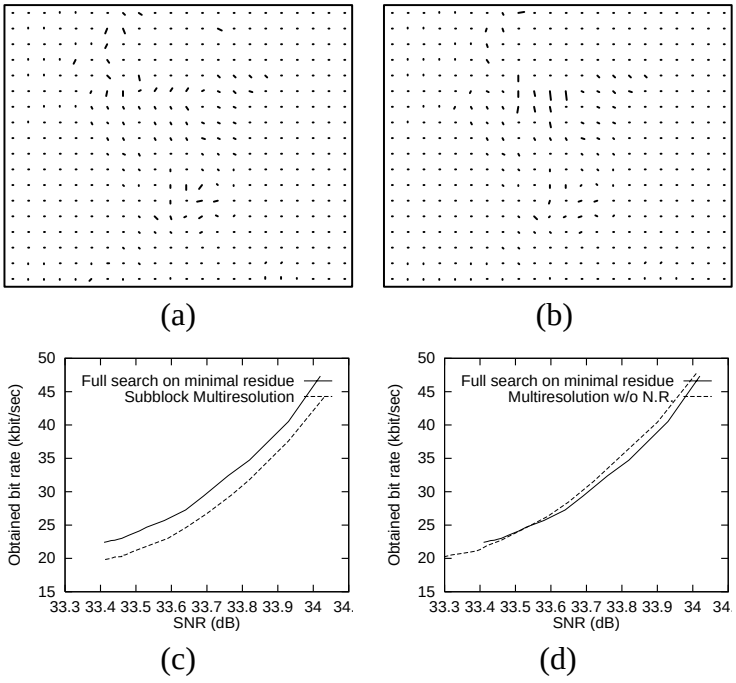


Figure 3.5: The simulation result based on the 105th frame and the 108th frame of the "foreman" sequence as shown in Figure 3.2. (a) shows the motion vectors found by our multiresolution search method *with* neighborhood relaxation. The motion field is smoother than the motion field produced by the minimal-residue criterion shown in Figure 3.2 and, as a result, the bits for coding motion vectors is fewer. (b) shows the motion vectors found by the multiresolution approach *without* neighborhood relaxation. (c)&(d) show the rate-distortion curve for our method, the original full-search method, and the multiresolution method without neighborhood relaxation. It is clear that our method could give better quality and better bit-rate.

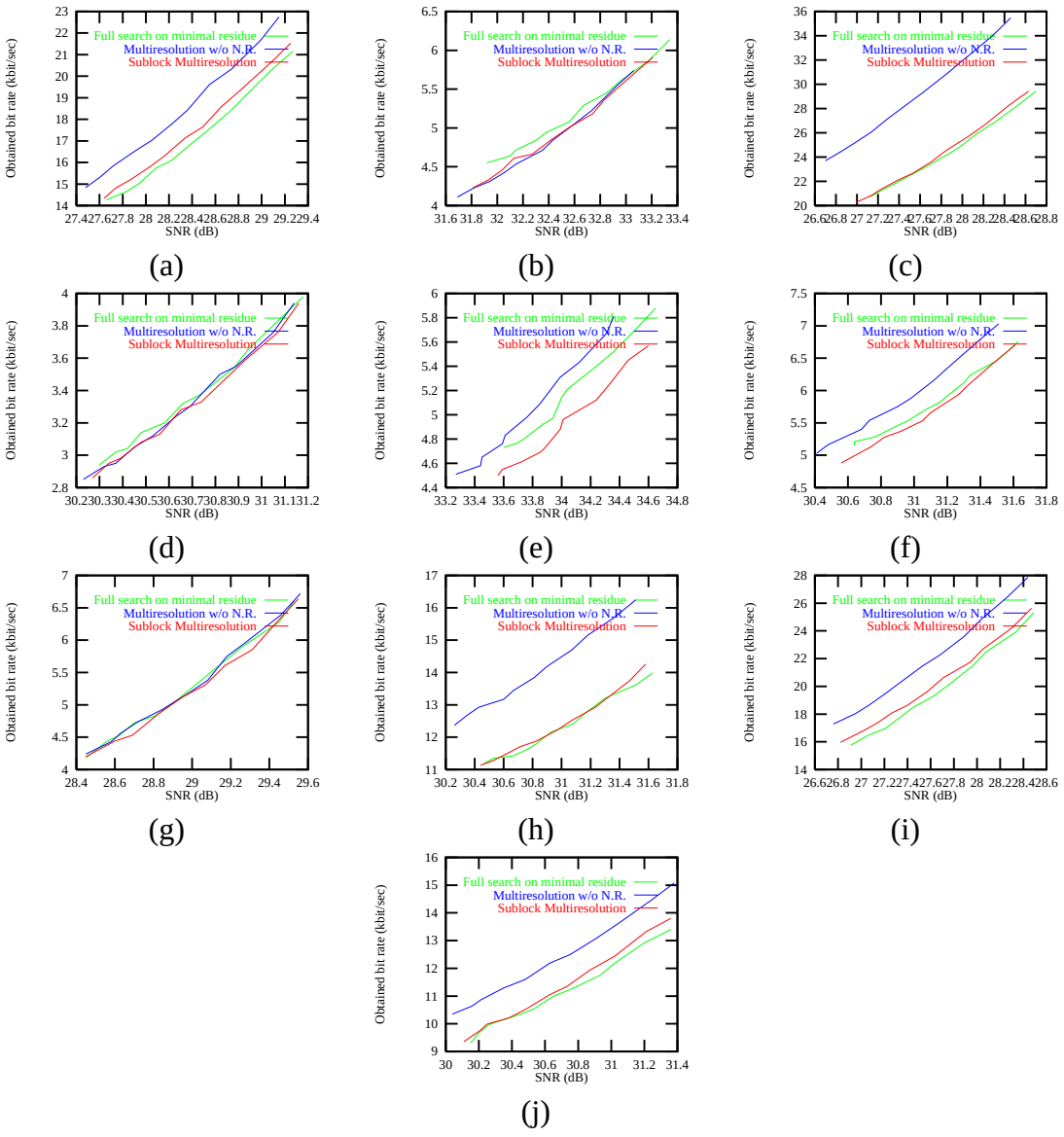


Figure 3.6: The rate-distortion curves for all the H.263 sequences. (a) shows the rate-distortion curve for the carphone sequence (average over the 380 frames). (b) claire (490 frames). (c) foreman (300 frames). (d) grandma (860 frames). (e) miss-am (150 frames). (f) mthr-dotr (300 frames). (g) salesman (445 frames). (h) suzie (150 frames). Because there is a scene change in the 60th frame of the trevor sequence, we divide the simulation of the trevor into two parts. (i) first part of the trevor (60 frames) and (j) second part of the trevor (90 frames).

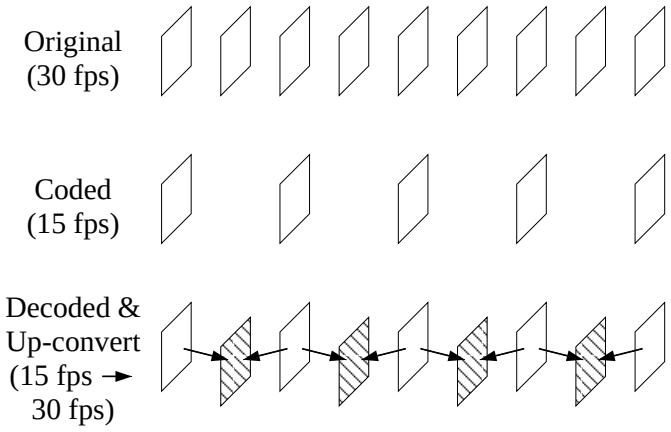


Figure 3.7: Our approach toward comparing the performance of the frame-rate up-conversion scheme using transmitted true motion. We drop one of every two frames in the original sequence ($\{I(t)\} \rightarrow \{I(2t)\}$), then encode the sequence using a standard video codec ($\{I(2t)\} \rightarrow \{\tilde{I}(2t)\}$). While we decode the sequence, we also interpolate the missing frames ($\{\tilde{I}(2t)\} \rightarrow \{\tilde{I}(2t+1)\}$). We compare the skipped original frame ($\{I(2t+1)\}$) and the interpolated reconstructed frame ($\{\tilde{I}(2t+1)\}$) for the performance measurement.

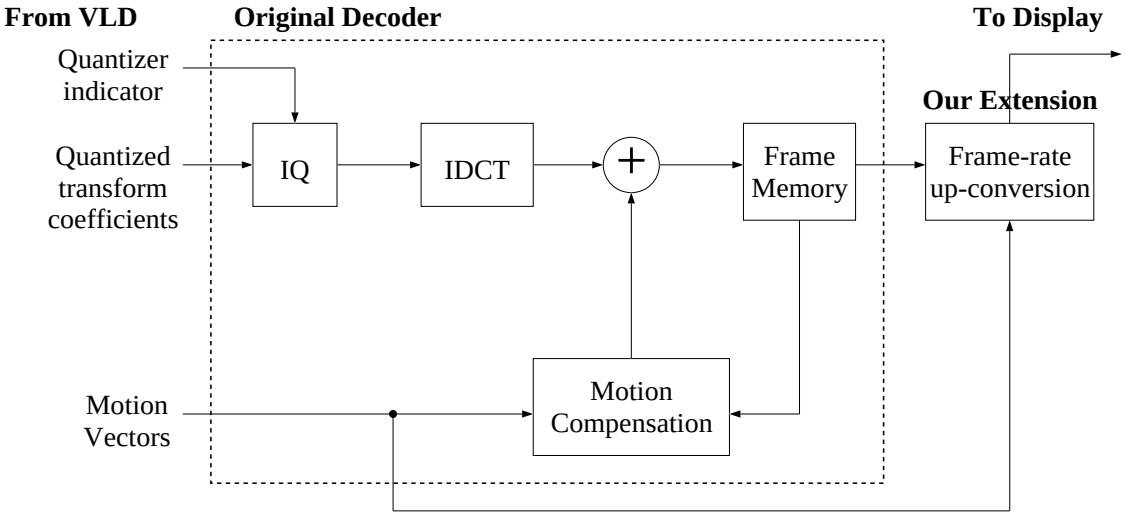


Figure 3.8: Our frame-rate up-conversion scheme, which uses the decoded motion vectors. After a variable-length coding decoder (VLD), a basic MPEG decoder includes an inverse quantization (IQ), an inverse discrete cosine transform (IDCT), and a motion compensation (cf. Figure 1.4). Our frame-rate up-conversion scheme scheme uses the decoded motion vectors for motion-compensated interpolation.

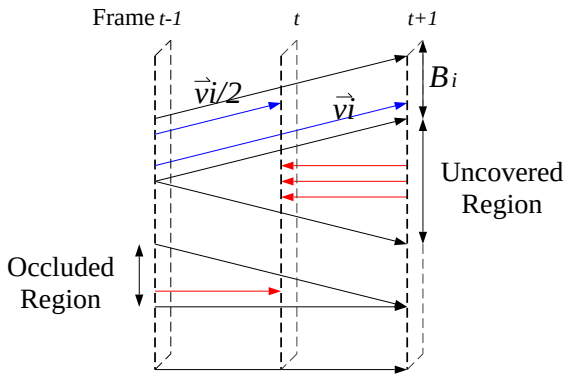


Figure 3.9: The proposed motion interpolation scheme. If block B_i moves $\vec{v}_i$ from frame F_{t-1} to frame F_{t+1}, then it is likely that block B_i moves $\vec{v}_i/2$ from frame F_{t-1} to frame F_t. We use the $\vec{v}_i/2$ to interpolate the missing pixels. When there is an occluded region in the next frame, we use the pixels in the previous frame for interpolation. When there is an uncovered region in the previous frame, we use the pixels in the next frame for interpolation.

Significant improvement may not be shown by the average numbers in Table 3.1. Figure 3.12 shows an example in the foreman sequence. When the foreman turns his head, the lamination of his face changes. That is, the rudimentary assumption of the intensity conversation is not valid in this scene. This condition is extraordinarily difficult for motion trackers (see Section 7.1). Therefore, the motion vectors estimated by a minimal-residue criterion have many errors. Because our true motion tracker is more reliable, the SNR improvement in this example is 1.5 dB.

In summary, we have demonstrated that the neighborhood relaxation TMT can be successfully applied to video compression. Specifically, in this chapter, we used it as a cost-effective rate-optimized motion estimation algorithm. Using the true motion vectors for video coding has the following advantages:

1. It optimizes the bit rate for residual and motion information. It is our observation that the coding cost is reduced when the block motion vectors resemble the true motion of the video.

2. It reduces the block artifacts and subjectively provides better pictures.

0	0	0	0	1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8	0	0	0	0
0	0	0	0	1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8	0	0	0	0
0	0	0	0	1/8	1/8	2/8	2/8	2/8	2/8	1/8	1/8	0	0	0	0
0	0	0	0	2/8	2/8	2/8	2/8	2/8	2/8	2/8	2/8	0	0	0	0
1/8	1/8	1/8	2/8	**4/8**	**5/8**	**5/8**	**5/8**	**5/8**	**5/8**	**5/8**	**4/8**	2/8	1/8	1/8	1/8
1/8	1/8	1/8	2/8	**5/8**	**5/8**	**5/8**	**5/8**	**5/8**	**5/8**	**5/8**	**5/8**	2/8	1/8	1/8	1/8
1/8	1/8	2/8	2/8	**5/8**	**5/8**	**6/8**	**6/8**	**6/8**	**6/8**	**5/8**	**5/8**	2/8	2/8	1/8	1/8
1/8	1/8	2/8	2/8	**5/8**	**5/8**	**6/8**	**6/8**	**6/8**	**6/8**	**5/8**	**5/8**	2/8	2/8	1/8	1/8
1/8	1/8	2/8	2/8	**5/8**	**5/8**	**6/8**	**6/8**	**6/8**	**6/8**	**5/8**	**5/8**	2/8	2/8	1/8	1/8
1/8	1/8	2/8	2/8	**5/8**	**5/8**	**6/8**	**6/8**	**6/8**	**6/8**	**5/8**	**5/8**	2/8	2/8	1/8	1/8
1/8	1/8	1/8	2/8	**5/8**	**5/8**	**5/8**	**5/8**	**5/8**	**5/8**	**5/8**	**5/8**	2/8	1/8	1/8	1/8
1/8	1/8	1/8	2/8	**4/8**	**5/8**	**5/8**	**5/8**	**5/8**	**5/8**	**5/8**	**4/8**	2/8	1/8	1/8	1/8
0	0	0	0	2/8	2/8	2/8	2/8	2/8	2/8	2/8	2/8	0	0	0	0
0	0	0	0	1/8	1/8	2/8	2/8	2/8	2/8	1/8	1/8	0	0	0	0
0	0	0	0	1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8	0	0	0	0
0	0	0	0	1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8	0	0	0	0

Figure 3.10: Weighting coefficients in the overlapped block motion compensation scheme when the block size is 8×8 [3, 51]. A pixel in $B_{i,j}$ will be motion-compensated by three motion vectors—one is the motion vector of $B_{i,j}$ and two are the motion vectors of the neighbor blocks of $B_{i,j}$. The *weighted* motion compensation scheme will (1) put more weights on the pixels that are closer to the block center and, (2) put fewer weights on the pixels that are on the block boundary, and (3) put even fewer weights on the pixels that are on the boundary of the neighbor blocks. The highlighted center is the location of the block itself on which the weighting function puts most emphasis. This scheme smoothes out motion vector differences gradually from the pixels in this block to the pixels in the next block.

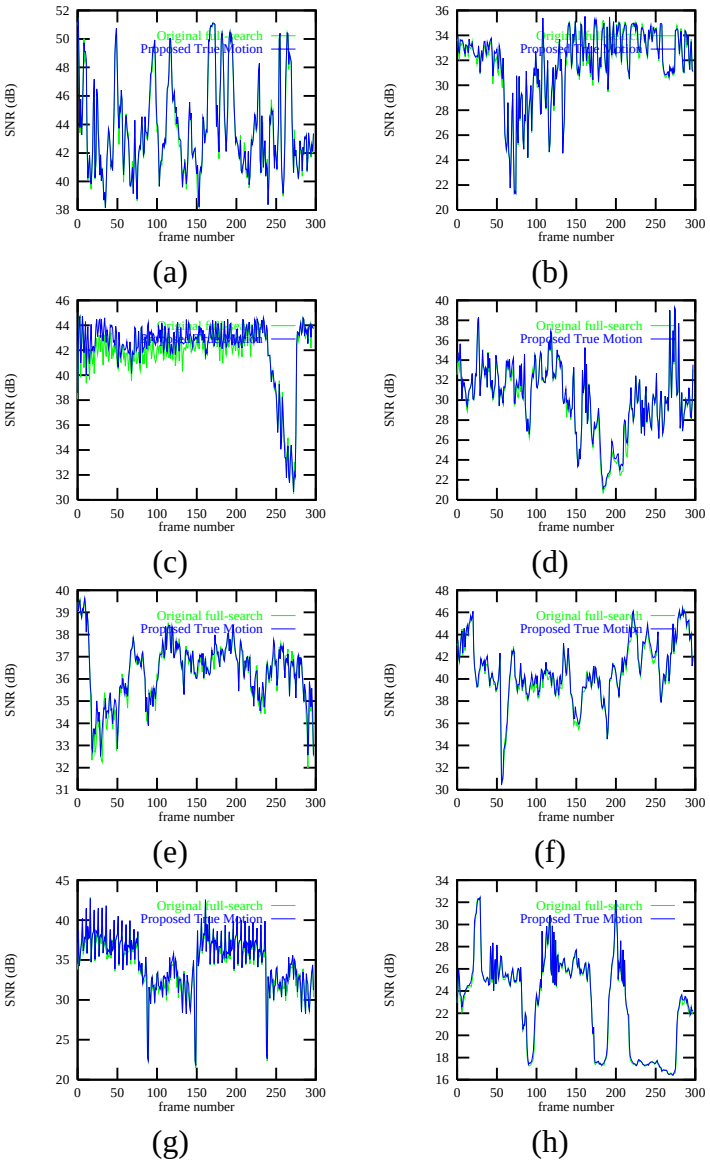


Figure 3.11: Frame-by-frame performance difference of the frame-rate up-conversion scheme between original BMA and the proposed true motion estimation using the (a) akiyo, (b) coastguard, (c) container, (d) foreman, (e) hall monitor, (f) mother and daughter, (g) news, and (h) stefan sequence.

(a) (b) (c)

(d) (e)

Figure 3.12: Simulation results for visual comparison. (a) the 138th frame of the "foreman" sequence. (b) the 139th frame, which is not coded in the bit-stream. (c) the 140th frame. (d) the reconstructed 139th frame (28.31 dB) using transmitted motion vectors which are generated from full-search motion estimation. (e) the reconstructed 139th frame (29.81 dB) using transmitted motion vectors which are generated from the proposed true motion estimation.

3. It offers significant improvement in motion-compensated frame-rate up-conversion over the minimal-residue BMA. The more accurate the motion estimation, the better the performance of frame-rate up-conversion. If we would like to perform the frame-rate up-conversion using the compression domain information (i.e., we do not want to have the motion estimation again at the decoder side), then encoding the video using the "true" motion vectors provides great improvement.

In addition, we incorporated the neighborhood relaxation in a multiresolution framework to reduce the computational complexity. Motion estimation is known to be the main bottleneck in real-time encoding applications, and the search for an effective motion estimation algorithm has been a challenging problem for years. The computational complexity required by our algorithm is around 17 times less than that required by the full-search BMA. The overall speedup of the whole video coding using this fast motion estimation algorithm is about 6.6. Moreover, in 9 out of our 10 benchmark simulations, the performance of the full search algorithm and that of our multiresolution method are about the same. In 1 out of our 10 benchmark simulations, our method has noticeable improvement.

Conventional multiresolution algorithms use only information from coarser levels to refine the motion vector in finer levels, and do not exploit spatial correlations of the motion vectors. Although the computational complexity of a conventional multiresolution algorithm is less than our method[‖], its motion vectors could come from tracking after a local minimum. Consequently, simulation shows that its coding efficiency is inferior to either the full search BMAs or our method.

[‖] A multiresolution algorithm without neighborhood relaxation is about 30% less complex than our multiresolution algorithm with neighborhood relaxation.

Chapter 4

Application in Spatio-Temporal Interpolation: Interlaced-to-Progressive Scan Conversion

True motion vectors play an important role in various spatio-temporal interpolation applications, including frame-rate or field-rate conversion applications, interlaced-to-progressive scan conversion, enhancement of motion pictures, and synthesis [22, 30, 40]. This chapter demonstrates a motion-compensated interlaced-to-progressive scan conversion (deinterlacing) method*.

Our deinterlacing scheme has two innovations. First, we increase the accuracy in motion estimation. Second, we reduce the errors in interpolation. While in previous chapters, our true motion trackers (TMTs) find motion vectors of *full-pel* resolution, the TMT in this chapter finds motion vectors of *sub-pel* resolution.

Our high-precision TMT vertically integrates two parts: (1) a matching-based motion tracker as the basis (as shown in Chapter 2) and (2) a gradient-based motion vector refiner.

*In *interlaced scan* video, multiple video fields are put together to obtain a complete image. Each *field* covers the entire space of an image, but each line skips the places where lines from other fields will be displayed. Unlike interlaced video, *progressive scan* video makes one pass down the display screen to produce a complete image.

In general, gradient-based approaches are accurate in finding motion vectors of sub-pel resolution. When the initial position is too far away from the solution, it is more likely that the gradient-based approaches converge to a local minimum and produce undesirable results. That is, in high motion regions, gradient-based approaches could be inaccurate in finding motion vectors [76]. But, if proper initial motion vectors are given, the gradient-based techniques can be accurate in the high motion region. In this context, the initial motion vectors can be provided by the matching-based TMT. Our simulation results demonstrate that a deinterlacing scheme using our true motion tracker produces a better performance than the same scheme using the original minimal-residue block-matching algorithm does.

4.1 Interlaced-to-Progressive Scan Conversion

Interlaced-to-progressive scan conversion techniques have been widely studied in various shapes for a number of reasons. Historically, the interlaced scanning scheme has been introduced to offer a compromise between picture quality and required bandwidth. However, the interlaced scanning scheme, which has been widely employed in current television systems, creates some uncomfortable visual artifacts—an edge flicker, an inter-line flicker, and a line-crawling. Figure 4.1 demonstrates the well-known "comb-effect" in the interlaced video. To reduce the visual affects of those artifacts, interlaced-to-progressive scan conversion is often mandated.

While conventional televisions use the interlaced scanning scheme, current computer monitors use the progressive scanning scheme. In order to display interlaced video (e.g., decoded MPEG-2 video), an interlaced-to-progressive scan conversion is required.

Because of different television standards, format conversion is necessary to interchange video programs globally. For example, although the US Federal Communications Commission (FCC) set the next generation digital television standard in 1996, the specific video

Figure 4.1: (a), (b), and (c) show the 138th, 139th, and the 140th field of the foreman sequence, respectively. (d) shows the frame in which the 138th field is put on top of the 137th field directly. (e) shows the frame in which the 139th field is put on top of the 138th field. (f) shows the frame in which the 140th field is put on top of the 139th field. Since the background is almost static, the quality of this part is good. On the contrary, because the head is moving, a lot of obvious “comb” effects appear.

formats to be used for digital broadcast television will be the subject of voluntary industry standards [41]. For consumers to perceive various video formats[†], format conversion is also required. The general interlaced-to-progressive scan conversion problem has been studied extensively from a video format conversion point of view.

Furthermore, interlaced video makes the MPEG-2 encoding less efficient and produces more artifacts. For the *advanced layered coding scheme*, one preprocessing step is to perform an interlaced-to-progressive scan conversion [33].

4.2 Motion-Compensated Interlaced-to-Progressive Scan Conversion

As shown in Figure 4.2, the interlaced-to-progressive scan conversion problem can be formulated as the following: to find $\{I(m,2n+((t-1) \bmod 2),t)\}$ given $\{I(m,2n+(t \bmod 2),t)\}$, where m,n is the horizontal and vertical pixel coordinate. That is, it is essentially a problem of interpolating the missing lines in every received field. Solutions to the problem have been classified as:

1. **Intraframe techniques:**

   Intraframe techniques interpolate the missing line based on the scanned lines immediately before and after the missing line. One example is the "line-averaging", which replaces a missing line by averaging the two lines adjacent to it, i.e., $I(m,2n,2t+1) = \{I(m,2n-1,2t+1)+I(m,2n+1,2t+1)\}/2$ or $I(m,2n+1,2t) = \{I(m,2n,2t)+I(m,2n+2,2t)\}/2$. And, there are other improved intraframe methods that use more complicated filers or edge information [64]. However, such techniques cannot predict information that is lost from the current field but does appear

[†]The HDTV video formats include 720 lines $\times$ 1280 pixels per line format at 24, 30, and 60 frames per second progressively scanned and a 1080 lines $\times$ 1920 pixels per line format at 24 and 30 frames per second progressively scanned and 60 fields per second interlaced scanned. In November 1998, CBS broadcast 1080i HDTV programs while ABC broadcast 720P HDTV programs [6].

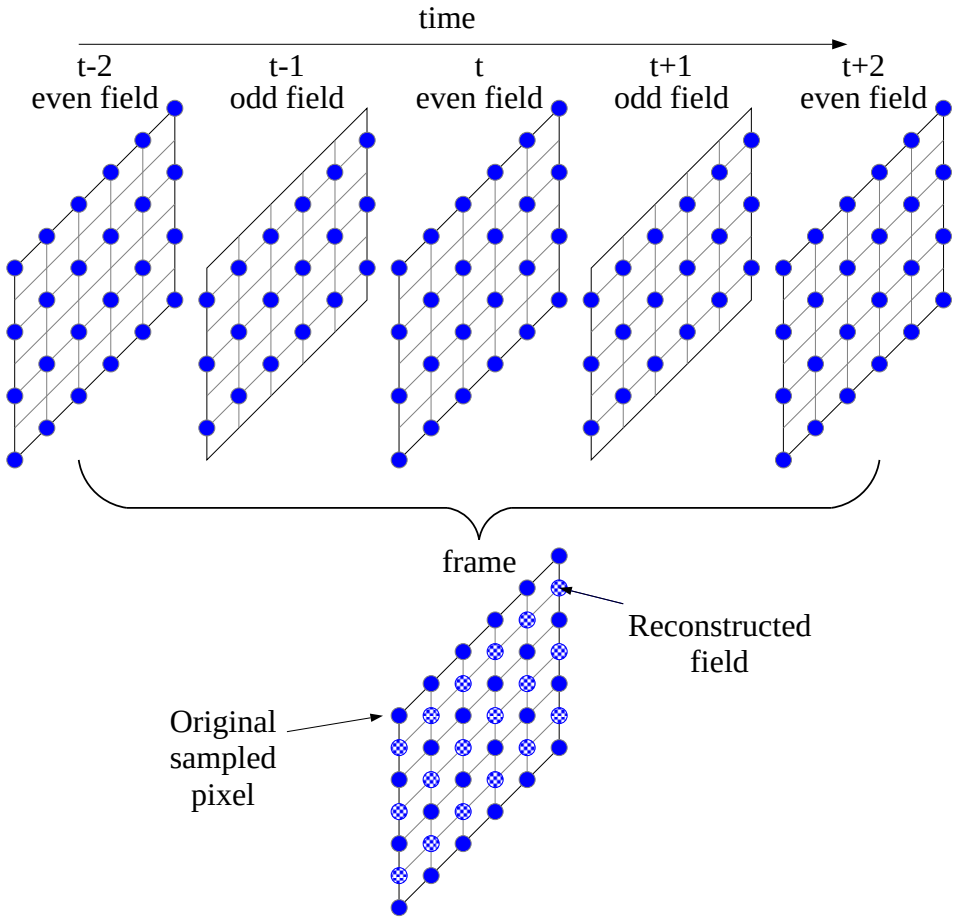


Figure 4.2: The interlaced-to-progressive scan conversion problem can be formulated as the following: to find $\{I(m, 2n + ((t-1) \bmod 2), t)\}$ given $\{I(m, 2n + (t \bmod 2), t)\}$. That is, it is essentially a problem of interpolating the missing lines in every received field.

in neighboring fields.

2. **Non-motion-compensated interframe techniques:**

   Interframe techniques take into account the pixels in the previous fields (or future fields) in the interpolation procedure. For example, one simple and widely-adopted method is the field-staying scheme, which lets $I(m, 2n+((t-1) \bmod 2), t) = I(m, 2n+((t-1) \bmod 2), t-1)$. In general, non-motion-compensated approaches, which apply linear or nonlinear filters, are fine for stationary objects. But, non-motion-compensated approaches result in severe artifacts for moving objects, as shown in Figure 4.1.

3. **Motion-compensated interframe techniques:**

   For moving objects, motion compensation should be used in order to achieve a higher picture-quality. Some motion compensated deinterlacing techniques have been proposed [28, 62, 104, 106]. It has been shown that the motion compensated deinterlacing methods are better than the interframe methods and non-motion-compensated interframe methods.

   There are three variations in the motion-compensated deinterlacing methods:

   (a) **Integer-pixel motion compensation:** If every pixel has only full-pel movement, then a missing pixel can be reconstructed by an existing pixel without any interpolation. This method is computationally easy because only full-pel motion estimation is required. For example, we can treat this interlaced-to-progressive scan conversion problem as two field-rate up-conversion problems—(1) to find $\{I(m, 2n, 2t+1)\}$ given $\{I(m, 2n, 2t), I(m, 2n, 2t+2)\}$ and (2) to find $\{I(m, 2n+1, 2t)\}$ given $\{I(m, 2n, 2t-1), I(m, 2n+1, 2t+1)\}$—in which we can use the method presented in Chapter 3. When the region has no motion and/or no texture, this method performs well. However, the assumption

that every pixel has only full-pel movement is often invalid and hence reduces the practical use of this method.

(b) **Interpolation using the generalized sampling theorem:** If a pixel has sub-pel movement, then an existing pixel from another field will move to the current field on a non-grid point. As shown in Figure 4.3, in order to reconstruct all the grid-point pixels, a generalized sampling theorem is used [104]. In spite of the effectiveness of the method, a perfect reconstruction requires the use of pixels from many lines, in which the motion vectors may not be constant. In addition, when the region has an odd-pixel vertical-movement, it is difficult to reconstruct the missing pixels.

(c) **Recursive deinterlacing:** If it is possible to have a perfectly deinterlaced previous frame in a memory, the previous frame can be used to deinterlace the current input field easily with the Nyquist sampling theorem, as shown in Figure 4.4. The new deinterlaced field is written in the memory so as to deinterlace the next incoming field [106]. However, the interpolation defects in the previous deinterlaced picture can be easily propagated. For example, when the motion estimator uses the defective deinterlaced image to produce inaccurate motion vectors, the deinterlaced image of the current frame would also be incorrect. In order to prevent the error propagation, many adaptive methods, e.g., a median function and a clip function, are proposed [28].

The method proposed in [62] is unique. Similar to the previous methods, this method uses the deinterlaced previous frames to deinterlace the current field. It is encouraging that the next frames are also deinterlaced (by the line-averaging method) to deinterlace the current field. Since most of the frame information is available for the motion estimation processing, the motion estimation will be more correct.

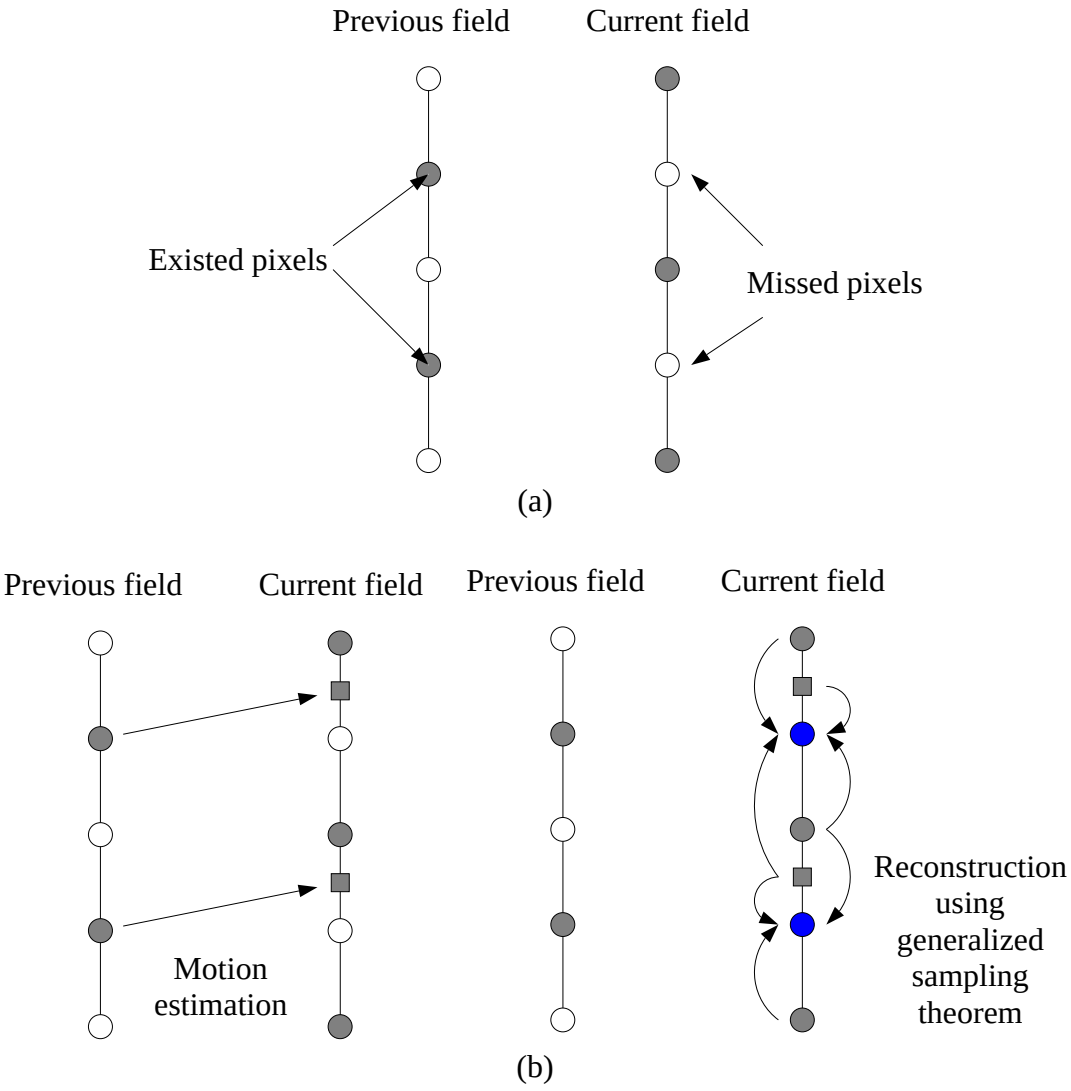


Figure 4.3: Interlaced-to-progressive deinterlacing methods using the generalized sampling theorem. (a) shows the previous field and the current field. (b) A motion estimator determines the pixel positions (the gray squares) in the current field that correspond to the pixels in the previous field. Note that the more accurate the motion estimation, the better the pixels to be reconstructed. Therefore, it is important to have a true motion tracker. (c) The generalized sampling theorem (see Section 4.3.2) is used to reconstruct the missed grid pixels (the dark circle) in the current frame.

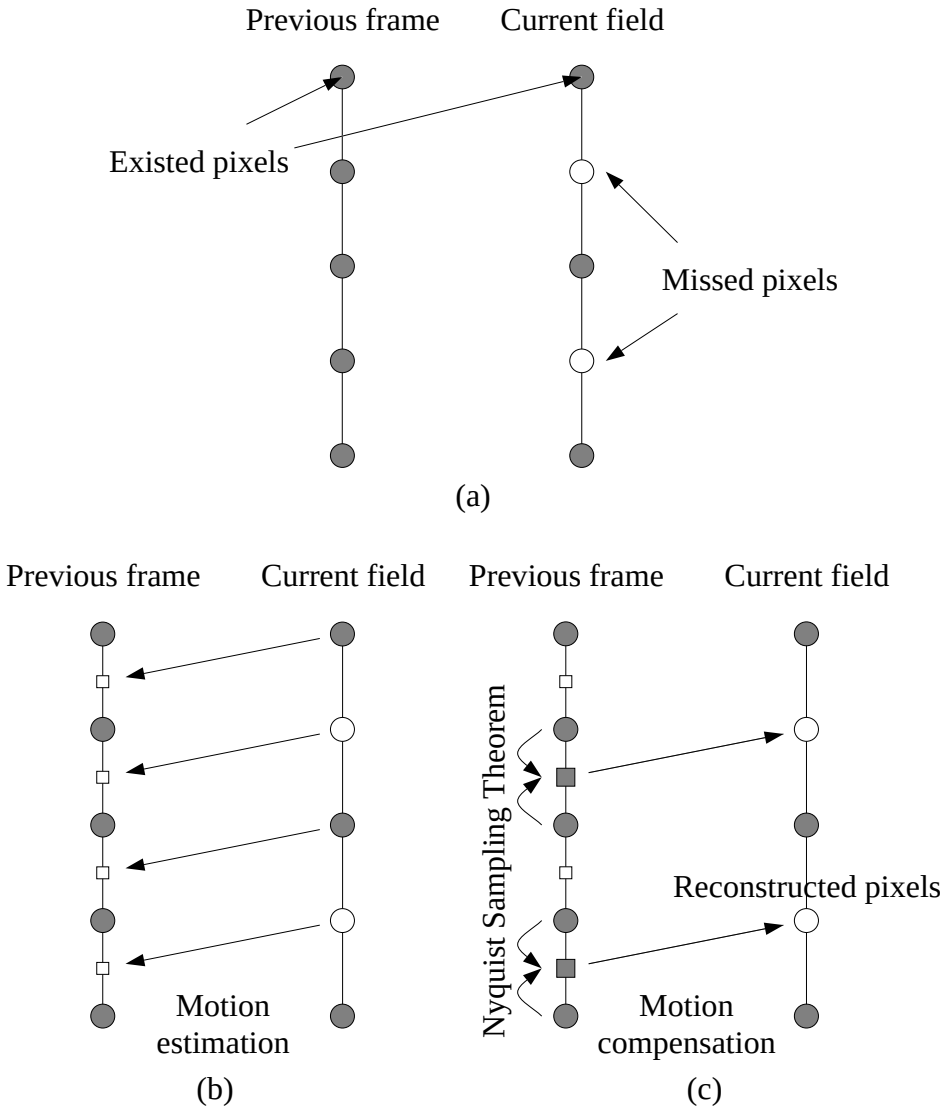


Figure 4.4: Recursive deinterlacing methods deinterlace the video without using the generalized sampling theorem. (a) assumes that we have a deinterlaced previous frame in a memory. The previous frame is used to deinterlace the current field. (b) A motion estimator determines the pixel positions in the previous frame that correspond to the pixels in the current field. It is important to have a true motion tracker because the more accurate the motion estimation, the better the pixels to be reconstructed. (c) Since the non-grid pixels (the gray squares) in the previous frame can be reconstructed easily using the Nyquist sampling theorem, the missed pixels in the current field can be motion compensated using the non-grid pixels. The new deinterlaced field is written in the memory so as to deinterlace the next incoming field recursively.

In this chapter, we present a new method based on accurate motion estimation/compensation and the generalized sampling/reconstruction theorem. As shown in Figure 4.5, our interlaced-to-progressive scan conversion procedure contains two parts: (1) finding $\{I(m+\Delta_x, 2n+\Delta_y, 2t)\}$ given $\{I(m, 2n+1, 2t-1)\}$ and $\{I(m, 2n+1, 2t+1)\}$ and (2) finding $\{I(m, 2n+1, 2t)\}$ given $\{I(m, 2n, 2t)\}$ and $\{I(m+\Delta_x, 2n+\Delta_y, 2t)\}$. The first step requires a motion-based compensation (see Section 4.3.1) while the second step requires a generalized sampling theorem (see Section 4.3.2).

There are five unique features in our method:

1. Our TMT provides higher resolution for applying the generalized sampling theorem. In [28], de Haan and Bellers use a 3D recursive-search block matcher to estimate the motion up to quarter-pel resolution [30]. In [62], the motion estimation is up to half-pel resolution. Our high-precision TMT vertically integrates two parts: (1) a matching-based motion tracker as the basis and (2) a gradient-based motion vector refiner (see Section 4.3.1). Gradient-based approaches are accurate in finding motion vectors of less than full-pel resolution.

2. Our method uses non-casual information. On the other hand, most of the previous methods use casual information (they never use the next fields [28, 104, 106]), as shown in Figure 4.3. In our method, the motion estimation is performed on the previous field and the next field. By assuming that the motion of a block is linear over a small time period, we can linearly interpolate the motion vectors related to the current field. In addition, because we have the information from the previous and next fields, we bi-directionally interpolate the non-grid-point pixels of the current field for higher precision.

3. We do *not* use odd fields for even field motion estimation or vice versa (see Section 4.3.3). Most of the previous methods perform the motion estimation using the previous fields related to the current field [28, 62, 104, 106], no matter if the previous

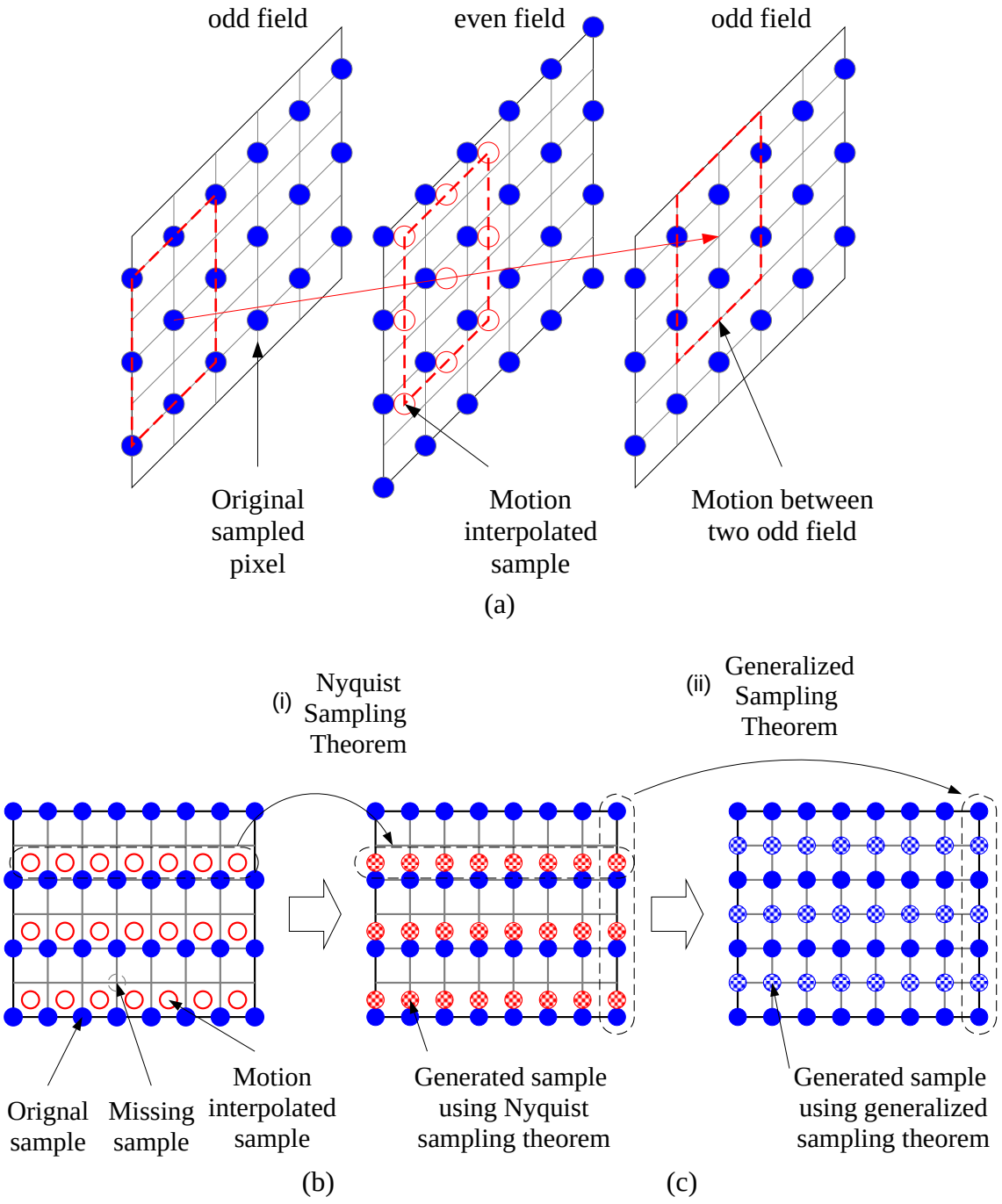


Figure 4.5: Our interlaced-to-progressive scan conversion approach contains two parts: (a) Finding $\{I(m+\Delta_x, 2n+\Delta_y, 2t)\}$ given $\{I(m, 2n+1, 2t-1)\}$ and $\{I(m, 2n+1, 2t+1)\}$. (b) Finding $\{I(m, 2n+1, 2t)\}$ given $\{I(m, 2n, 2t)\}$ and $\{I(m+\Delta_x, 2n+\Delta_y, 2t)\}$. This part contains two steps (i) finding $\{I(m, 2n+\Delta_y, 2t)\}$ given $\{I(m+\Delta_x, 2n+\Delta_y, 2t)\}$ and (ii) finding $\{I(m, 2n, 2t)\}$ given $\{I(m, 2n+\Delta_y, 2t)\}$.

field is even or odd. But, we use previous or next *odd* fields for the motion estimation of the current *odd* field. We use previous or next *even* fields for the motion estimation of the current *even* field. Most of the time, pixels in the odd field will stay at the odd field (e.g., non-motion background, horizontal panning regions). Only when there is an odd-pixel vertical movement, will a pixel in the odd field move to the even field. However, when there is an odd-pixel vertical movement, the lost information in the current odd field is also lost in the previous even field. This means that it is unnecessary to track the motion of the pixels that move from an odd field to an even field. Therefore, using previous or next *odd* fields for the motion estimation of the current *odd* field or using previous, or next *even* fields for the motion estimation of the current *even* field is good enough!

4. In our method, we do not use any deinterlaced frames for motion estimation or motion interpolation. Some of the previous methods use the previous and/or future deinterlaced frames in order to make the motion estimation easier and the sample reconstruction simpler [28, 62, 106]. Since the interpolation defects in the previous/future deinterlaced picture can be too easily propagated to the current frame, our method does not use any deinterlaced frames for motion estimation or motion interpolation.

5. Our method adaptively combines the line-averaging technique and the motion compensated deinterlacing technique. Based on the position of the motion compensated sampled point, we have different weightings assigned to it. When the motion compensated sampled point has the same position as the missed pixel (e.g., non-motion region), it has the highest reliability. When the motion compensated sampled point has the same position as the pre-existing pixel, it has the lowest reliability. In addition, the confidence of the motion vector also influences the weighting of the motion compensated sampled point. (See Section 3.4 and Section 4.3.3; when the residue of

the motion vector is larger, it is unlikely that the motion vector is reliable.)

4.3 Proposed Deinterlacing Algorithm

In this section, we present our deinterlacing algorithm based on the accurate TMT and the generalized sampling/reconstruction theorem. As shown in Figure 4.5, our deinterlacing approach contains two parts: (1) finding $\{I(m+\Delta_x, 2n+\Delta_y, 2t)\}$ given $\{I(m, 2n+1, 2t-1)\}$ and $\{I(m, 2n+1, 2t+1)\}$ and (2) finding $\{I(m, 2n+1, 2t)\}$ given $\{I(m, 2n, 2t)\}$ and $\{I(m+\Delta_x, 2n+\Delta_y, 2t)\}$. The first step requires a motion-based compensation (see Section 4.3.1) while the second step requires a generalized sampling theorem (see Section 4.3.2).

4.3.1 Integrating Matching-Based and Gradient-Based Motion Estimation

In the proposed deinterlacing approach, the first step is to determine the $\{I(m+\Delta_x, 2n+\Delta_y, 2t)\}$. In order to have accurate Δ_x and Δ_y, the TMT in this application requires higher accuracy than the TMTs presented in the previous chapters. Hence, we present a high-precision TMT, which vertically integrates the matching-based technique and the gradient-based technique.

Our matching-based true motion tracker using a neighborhood relaxation formulation (see Chapter 2) is considered to be reliable. However, the TMT can only find full-pel motion vectors. That is, the precision of the motion vectors estimated by the matching-based TMT cannot be smaller than an integer. On the other hand, the precision of the motion vectors estimated by gradient-based techniques can be fractional. Therefore, gradient-based techniques must be exploited in this high-precision motion estimation.

We do not use gradient-based techniques in the first place because the gradient-based techniques may not be dependable in the high motion region. Let

$$s(v_x, v_y) \equiv I(x_0+v_x, y_0+v_y, t_0+1) - I(x_0, y_0, t_0)$$

Using Taylor's expansion, we have

$$s(v_x, v_y) = s(v_{x0}, v_{y0}) + (v_x - v_{x0})\frac{\partial s}{\partial v_x} + (v_y - v_{y0})\frac{\partial s}{\partial v_y} + E(\Delta^2)$$

where $E(\Delta^2)$ means second and higher order terms. When (v_x, v_y) resembles the true motion, $s(v_x, v_y) = 0$ (from the rudimentary assumption—intensity conservation over time, see Eq. (1.1)), i.e.,

$$s(v_{x0}, v_{y0}) + (v_x - v_{x0})\frac{\partial s}{\partial v_x} + (v_y - v_{y0})\frac{\partial s}{\partial v_y} + E(\Delta^2) = 0 \tag{4.1}$$

Consider the following two situations:

1. When $v_x \approx 0$, $v_y \approx 0$, we choose $v_{x0} = v_{y0} = 0$ so that $E(\Delta^2) \approx 0$. Therefore,

   $$s(0,0) + v_x\frac{\partial s}{\partial v_x} + v_y\frac{\partial s}{\partial v_y} = 0$$

   This is equivalent to

   $$I(x_0, y_0, t_0 + 1) - I(x_0, y_0, t_0) + v_x\frac{\partial I}{\partial x}\Big|_{(x_0, y_0, t_0+1)} + v_y\frac{\partial I}{\partial y}\Big|_{(x_0, y_0, t_0+1)} = 0$$

   that is, the foundation for gradient-based techniques (see Eq. (1.3)).

2. On the other hand, when $|v_x| \gg 0$ or $|v_y| \gg 0$, choosing $v_{x0} = v_{y0} = 0$ may lead to a large error $E(\Delta^2)$. A large error $E(\Delta^2)$ means that

   $$s(0,0) + v_x\frac{\partial s}{\partial v_x} + v_y\frac{\partial s}{\partial v_y} \neq 0$$

   which is equivalent to

   $$I(x_0, y_0, t_0 + 1) - I(x_0, y_0, t_0) + v_x\frac{\partial I}{\partial x}\Big|_{(x_0, y_0, t_0+1)} + v_y\frac{\partial I}{\partial y}\Big|_{(x_0, y_0, t_0+1)} \neq 0$$

   that is, the foundation for gradient-based techniques is *invalid*. Thus, the gradient-based techniques may not be dependable in the high motion region.

We proposed a motion tracking algorithm that consists of (1) finding the coarse motion (v_{x0}, v_{y0}) by our matching-based true motion tracker and (2) finding the detailed motion by a gradient-based motion estimator. This makes sense because the gradient-based techniques can be dependable in the high motion region if proper initial motion vectors are given (i.e., (v_{x0}, v_{y0})). That is,

$$I(x_0 + v_{x0}, y_0 + v_{y0}, t_0 + 1) - I(x_0, y_0, t_0) + (v_x - v_{x0}) \frac{\partial I}{\partial x}\Big|_{(x_0+v_{x0}, y_0+v_{y0}, t_0+1)} + (v_y - v_{y0}) \frac{\partial I}{\partial y}\Big|_{(x_0+v_{x0}, y_0+v_{y0}, t_0+1)} = 0 \tag{4.2}$$

Note that this two-step algorithm will suffer from propagating estimation errors from the coarse-resolution motion field to the fine-resolution motion field, and may not be able to recover from the errors [76, 89]. The correctness of the matching-based true motion tracker is critical.

4.3.2 Generalized Sampling Theorem

The *Sampling Theorem* is well known as the following: If $f(t)$ is a 1D function having a Fourier transform $F(\omega)$ such that $F(\omega) = 0$ for $|\omega| \geq \omega_0 = \pi/T_s$ (*a band-limited signal*), and is sampled at the points $t_n = nT_s$ (*Nyquist rate*), then $f(t)$ can be reconstructed exactly from its samples $\{f(nT_s)\}$ as follows:

$$f(t) = \sum_{n=-\infty}^{\infty} f(nT_s) \frac{\sin[\omega_0(t - nT_s)]}{[\omega_0(t - nT_s)]} = \sum_{n=-\infty}^{\infty} f(nT_s) \operatorname{sinc}\left(\frac{t - nT_s}{T_s}\right) \tag{4.3}$$

where $\operatorname{sinc}(0) = 1$ and $\operatorname{sinc}(x) = \sin(\pi x)/\pi x$ whenever $x \neq 0$.

Since the debut of the original sampling theorem, it has been generalized into various extensions [81]. One *Generalized Sampling Theorem* is the following: If $f(t)$ is band-limited to $\omega_0 = \pi/T_s$, and is sampled at $1/m$ the Nyquist rate but in each sampling interval not one but m samples are used (bunched samples) then $f(t)$ can be reconstructed exactly from its samples $\{f(mnT_s + \Delta T_k) \mid 0 < \Delta T_k < mT_s,\ k = 1, \cdots, m,\ \Delta T_i \neq \Delta T_j\ \forall i \neq j\}$.

In this work, we are particularly interested in $m = 2$ and we have derived a closed form solution in the following. Given

1. $f(t)$ is band-limited to $\omega_0 = \pi/T_s$, and

2. in each sampling interval, there are 2 samples $\{f(2nT_s), f(2nT_s + \Delta T)\}$ (where $0 < \Delta T < 2T_s$),

we would like to reconstruct the $\{f((2n+1)T_s)\}$. After that, $f(t)$ can be reconstructed exactly using Eq. (4.3).

From the original sampling theorem,

$$\begin{aligned} f(2iT_s + \Delta T) &= \sum_n f(nT_s)\,\mathrm{sinc}\left(\frac{2iT_s + \Delta T - nT_s}{T_s}\right) \\ &= \sum_n f(2nT_s)\,\mathrm{sinc}\left(\frac{2iT_s + \Delta T - 2nT_s}{T_s}\right) + \\ &\quad \sum_n f((2n+1)T_s)\,\mathrm{sinc}\left(\frac{2iT_s + \Delta T - (2n+1)T_s}{T_s}\right) \end{aligned}$$

This implies

$$\begin{aligned} f(2iT_s + \Delta T) &- \sum_n f(2nT_s)\,\mathrm{sinc}\left(\frac{2iT_s + \Delta T - 2nT_s}{T_s}\right) \\ &= \sum_n f((2n+1)T_s)\,\mathrm{sinc}\left(\frac{2iT_s + \Delta T - (2n+1)T_s}{T_s}\right) \quad \forall i \end{aligned}$$

These equations can be written more concisely in a matrix form as

$$\hat{\mathbf{f}} - \mathbf{h} = \mathbf{S}\mathbf{f}$$

where $\hat{\mathbf{f}}$ is the $k \times 1$ column vector of known k samples, $\mathbf{h}$ is the $k \times 1$ column vector with components given by

$$h_i = \sum_n f(2nT_s)\,\frac{\sin[\omega_0 \Delta T]}{\omega_0(2iT_s + \Delta T - 2nT_s)}$$

and which depends only on the known samples, and $\mathbf{S}$ denotes the $k \times k$ matrix with entries

$$S_{ij} = \frac{\sin[\omega_0(\Delta T - T_s)]}{\omega_0(2iT_s + \Delta T - (2j+1)T_s)}.$$

The vector $\mathbf{f}$ can be found as the solution to the system of equations as:

$$\mathbf{f} = \mathbf{S}^{-1}(\hat{\mathbf{f}} - \mathbf{h}) \tag{4.4}$$

In practice, we find that

$$S_{ij} = \frac{\sin[\omega_0(\Delta T - T_s)]}{\omega_0(2iT_s + \Delta T - (2j+1)T_s)} \approx 0 \quad \forall |i-j| \geq 1$$

Therefore,

$$f((2i+1)T_s) \approx \left\{ f(2iT_s + \Delta T) - \sum_n f(2nT_s) \frac{\sin[\omega_0 \Delta T]}{\omega_0(2iT_s + \Delta T - 2nT_s)} \right\} \frac{\omega_0(\Delta T - T_s)}{\sin[\omega_0(\Delta T - T_s)]} \tag{4.5}$$

And, it can be estimated by an iterative procedure (e.g., [42]) as:

$$f^{(l)}((2i+1)T_s) = f^{(l-1)}((2i+1)T_s) + \left\{ f(2iT_s + \Delta T) - \sum_n f(2nT_s) \frac{\sin[\omega_0 \Delta T]}{\omega_0(2iT_s + \Delta T - 2nT_s)} - \sum_n f^{(l-1)}((2n+1)T_s) \frac{\sin[\omega_0(\Delta T - T_s)]}{\omega_0(2iT_s + \Delta T - (2n+1)T_s)} \right\} \frac{\omega_0(\Delta T - T_s)}{\sin[\omega_0(\Delta T - T_s)]} \tag{4.6}$$

where $f^{(0)}((2n+1)T_s)$ could be 0 or $\{f(2nT_s) + f((2n+2)T_s)\}/2$.

4.3.3 Our Interlaced-to-Progressive Scan Conversion Algorithm

Step 1: Motion-Compensation of Non-Grid Samples.

The first step of this proposed deinterlacing approach is to perform the motion-compensation and obtain a set of missing samples on the non-grid points $\{I(m+\Delta_x, 2n+\Delta_y, 2t)\}$, as shown in Figure 4.5(a). There are many ways to obtain the missing samples, such as:

1. We can find $I(m+v_x, 2n+1+v_y, 2t) = I(m, 2n+1, 2t+1) = I(m+2v_x, 2n+1+2v_y, 2t-1)$ from the motion estimation between field $(2t-1)$ and field $(2t+1)$.

2. We can find $I(m+v_x, 2n+v_y, 2t) = I(m, 2n, 2t+2)$ from the motion estimation between field $(2t)$ and field $(2t+2)$.

3. We can find $I(m+v_x, 2n+v_y, 2t) = I(m, 2n, 2t-2)$ from the motion estimation between field $(2t)$ and field $(2t-2)$.

Step 2: Reconstruction of Grid Samples.

Once a new set of motion compensated samples has been reconstructed, we are able to use them to determine the set of samples that lie on the sampling grid for the missing field, as shown in Figure 4.5(b). Even though the sampling and reconstruction problem is two-dimensional, we assume that the signal is separable[‡]. So, if we are given $\{I(m,2n,2t), I(m+\Delta_x, 2n+\Delta_y, 2t)\}$, it actually takes two steps to find $\{I(m,2n+1,2t)\}$:

1. **Given** $\{I(m+\Delta_x, 2n+\Delta_y, 2t)\}$**, to find** $\{I(m,2n+\Delta_y,2t)\}$**:**

   Because we have enough horizontal samples at the Nyquist rate, we only apply Eq. (4.3):

   $$I(x,2y+\Delta_y,2t) = \sum_m I(m+\Delta_x, 2y+\Delta_y, 2t)\,\mathrm{sinc}(x-m-\Delta_x) \tag{4.7}$$

   (see Figure 4.5(b-i))

2. **Given** $\{I(m,2n,2t), I(m,2n+\Delta_y,2t)\}$**, to find** $\{I(m,2n+1,2t)\}$**:**

   Since $0 < \Delta_y < 2$, we have to use the generalized sampling theorem (see Eq. (4.6)) as:

   $$\begin{aligned} I^{(l)}(x,2y+1,2t) &= I^{(l-1)}(x,2y+1,2t) + \\ &\left\{ I(x,2y+\Delta_y,2t) - \sum_n I(x,2n,2t)\,\mathrm{sinc}(2y+\Delta_y-2n) - \right. \\ &\left. \sum_n I^{(l-1)}(x,2n+1,2t)\,\mathrm{sinc}(2y+\Delta_y-2n-1) \right\} \frac{1}{\mathrm{sinc}(\Delta_y-1)} \end{aligned} \tag{4.8}$$

   (see Figure 4.5(b-ii))

[‡]In the image domain, the sampling and reconstruction problem is two-dimensional. Here, we assume that the signal is *separable*, i.e., the reconstruction of the signal is as the following:

$$I(x,y) = \sum_m \sum_n I(m,n)\,\mathrm{sinc}(x-m)\,\mathrm{sinc}(y-n)$$

Prediction of Information Never Seen.

The above method generally works well except in the following two special cases:

1. **Object Occlusion and Reappearance:** As we mentioned earlier, object occlusion and reappearance make the motion estimation and the motion-based spatial and temporal interpolation more difficult. In this motion-compensated deinterlacing problem, we ignore the motion vectors at the object occlusion and reappearance region and use the intrafield information (e.g., using the line-averaging technique).

2. **Field Motion Singularity:** There is a difficult situation for our method—an object is moving upward/downward at $(2n+1)$ pixels per field. ($\Delta_y = 0$.) Under this situation, multiple fields do not provide more information than a single field does. This means that we should use the intrafield information (e.g., using the line-averaging technique).

Reduction of the Block Artifacts.

Using field f_{2t} as the current field, our method can be summarized as follows:

$$
\begin{aligned}
\Delta(i,2j+1) &= \sum_x \sum_y \sum_{\{B_k\}} w(x,y,B_k)\, \delta(i,x+v_{xi})\, \delta(2j+1,y+v_{yi}) \\
&\quad \Big\{ (I(x,y,2t-1)+I(x+2v_{xi},y+2v_{yi},2t+1))/2 - \\
&\quad \sum_m \sum_n I(m,2n,2t)\, \mathrm{sinc}(x+v_{xi}-m)\, \mathrm{sinc}(y+v_{yi}-2n) - \\
&\quad \sum_m \sum_n \tilde{I}^{(l)}(m,2n+1,2t)\, \mathrm{sinc}(x+v_{xi}-m)\, \mathrm{sinc}(y+v_{yi}-2n-1) \Big\} \\
W(i,2j+1) &= \sum_x \sum_y \sum_{\{B_k\}} w(x,y,B_k)\, \delta(i,x+v_{xi})\, \delta(2j+1,y+v_{yi}) \\
&\quad \mathrm{sinc}(x+v_{xi}-i)\, \mathrm{sinc}(y+v_{yi}-2j-1) \\
\tilde{I}^{(l+1)}(i,2j+1,2t) &= \tilde{I}^{(l)}(i,2j+1,2t) + \\
&\quad \begin{cases} \Delta(i,2j+1)/W(i,2j+1) & \text{if} \quad W(i,2j+1) \neq 0 \\ 0 & \text{if} \quad W(i,2j+1) = 0 \end{cases}
\end{aligned}
\tag{4.9}
$$

where $2\vec{v}_i$ is the movement of B_k from field f_{2t-1} to field f_{2t+1}, $\delta(a,b) = 1$ when $|a-b| < 1$ and $\delta(a,b) = 0$; otherwise, $w(x,y,B_k)$ is the window function, and the initial value of $\tilde{I}^{(0)}(i,2j+1,2t)$ is from the line-averaging technique using the field f_{2t}. That is,

$$\tilde{I}^{(0)}(i,2j+1,2t) = \sum_n I(i,2n,2t)\,\mathrm{sinc}\left(\frac{2n-2j-1}{2}\right)$$

In order to reduce the block artifacts, we choose a weighting function $w(x,y,B_k)$ that is similar to the coefficients defined for the overlapped block motion compensation scheme (OBMC), as shown in Figure 3.10. In this weighting scheme, a pixel in $B_{i,j}$ will be motion-compensated by three motion vectors—one is the motion vector of $B_{i,j}$ and two are the motion vectors of the neighbor blocks of $B_{i,j}$. This scheme smoothes out motion vector differences gradually from the pixels in this block to the pixels in the next block. Hence, the block artifacts will be reduced.

Moreover, the weighting function can also depend on the tracking confidence of the block (see Section 3.4). When the confidence of tracking a block is low, we can relatively increase the effect of the intraframe technique by decreasing the weight of the motion interpolation.

4.4 Performance Comparison of Deinterlacing Schemes

Figure 4.6 demonstrates our approach toward measuring the performance of an interlaced-to-progressive scan conversion. The approach consists of the following three steps. (1) A progressive video benchmark $\{I(m,n,t)\}$ is transformed into an interlaced video $\{I(m,2n+(t \bmod 2),t)\}$ by dropping every other field in each frame. (2) We reconstruct the progressive video $\{\tilde{I}(m,n,t)\}$ by interlaced-to-progressive scan conversion techniques. (3) The error between the original video $\{I(m,n,t)\}$ and the reconstructed progressive video $\{\tilde{I}(m,n,t)\}$ is measured. The better the interlaced-to-progressive conversion, the smaller the error between $\{I(m,n,t)\}$ and $\{\tilde{I}(m,n,t)\}$.

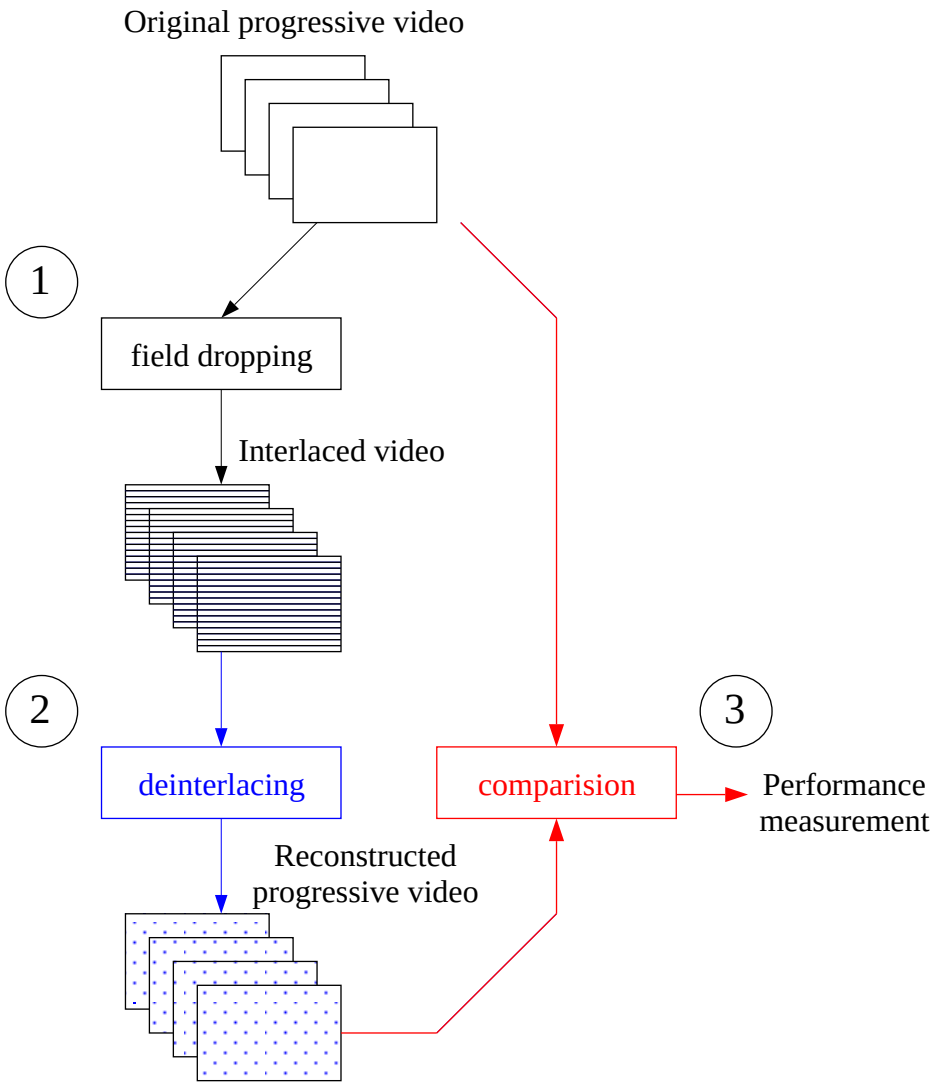


Figure 4.6: Flow chart of the proposed approach for performance comparison in interlaced-to-progressive scan conversion. (1) We transform a progressive video benchmark $\{I(m,n,t)\}$ into an interlaced video $\{I(m,2n+(t \bmod 2),t)\}$ by dropping every other field in each frame. (2) We reconstruct the progressive video $\{\tilde{I}(m,n,t)\}$ by an interlaced-to-progressive scan conversion technique. (3) We measure the performance of the technique by comparing the original video $\{I(m,n,t)\}$ and the reconstructed progressive video $\{\tilde{I}(m,n,t)\}$. The better the interlaced-to-progressive conversion, the smaller the error between $\{I(m,n,t)\}$ and $\{\tilde{I}(m,n,t)\}$.

Sequences	Field-Staying	Line-Averaging	Motion-Based Field-Rate Up-Convert		
			Original BMA	Our TMT	+GST
akiyo	42.05	39.70	43.72	43.86	43.86
coastguard	26.85	28.39	33.28	33.42	33.54
container	40.53	27.90	44.08	44.35	45.08
foreman	26.76	30.53	30.69	30.98	31.70
hall_monitor	36.07	29.61	38.95	39.04	39.12
mother_daughter	38.97	36.28	42.14	42.46	42.96
news	33.15	34.00	35.71	35.84	36.54
stefan	20.12	27.41	23.88	24.07	25.66

Table 4.1: Performance of different deinterlacing approaches. Numbers are dB in SNR.

Table 4.1 shows the performance of five different interlaced-to-progressive scan conversion approaches. The field-staying scheme is better than the line-averaging scheme in the sequences that have almost no movement (e.g., container). That is, the line-averaging scheme (an intra-field prediction scheme) cannot accurately predict the pixels that are never seen. On the other hand, when the sequences have some high motion activities, the field-staying scheme is not as good as the line-averaging scheme (e.g., foreman and stefan). That is, when the motion information is not used the inter-field method cannot provide an accurate prediction. Table 4.1 also reveals that there is a 0.2 dB SNR difference between using the minimal-residue BMA and using our true motion tracker.

Our motion-based interlaced-to-progressive scan conversion, using the true motion estimation, is the best in seven out of the eight test sequences. (It fails when the movement in the scene is intense.) In average, our deinterlacing approach, which utilizes the proposed true motion tracker and the generalized sampling theorem, is 5 dB SNR better than the line-averaging method. In [62], the best deinterlacing method is only 4 dB SNR better than the line-averaging method[§].

[§]During our evaluation, we assumed that the camera is without such filter. If the camera has an optical per-filter, then the methods may not be evaluated correctly.

Figure 4.7 shows the frame-by-frame performance comparison of deinterlacing approaches. Figure 4.8 shows an example in the foreman sequence for visual comparison. The reconstructed frame using the field-staying method has very clear comb effects in the moving region (e.g., the head). The reconstructed frame using the line-averaging method looks blurry in the non-moving regions (e.g., the building). The reconstructed frame using our approach is better both in the moving and non-moving regions.

In summary, in this chapter, we vertically integrated the matching-based technique and gradient-based technique into our TMT for higher accuracy and higher precision. An accurate and precise motion tracker is essential for interlaced-to-progressive scan conversion. Our TMT together with a generalized sampling theorem provides a fundamental tool for conversion of various video formats.

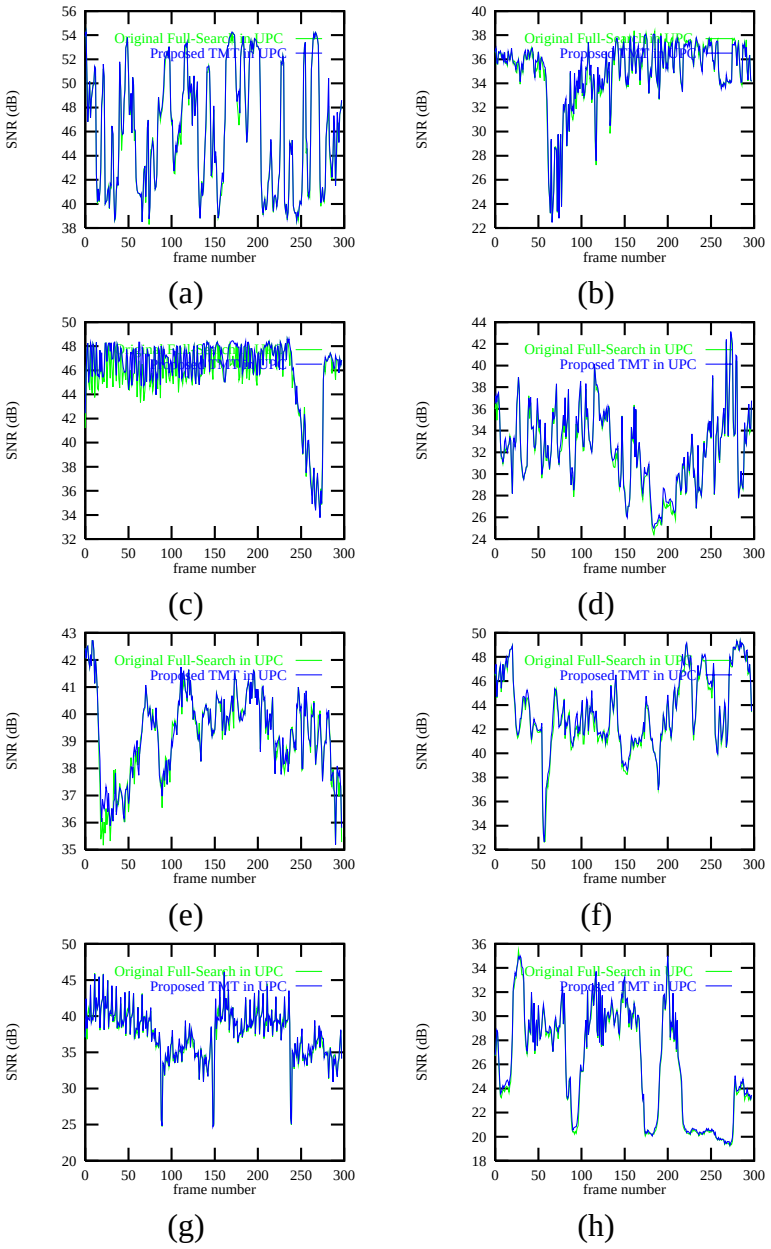


Figure 4.7: Frame-by-frame performance difference of deinterlacing schemes with the proposed true motion tracker and with the original minimal-residue block-matching algorithm using the (a) akiyo, (b) coastguard, (c) container, (d) foreman, (e) hall monitor, (f) mother and daughter, (g) news, and (h) stefan sequence.

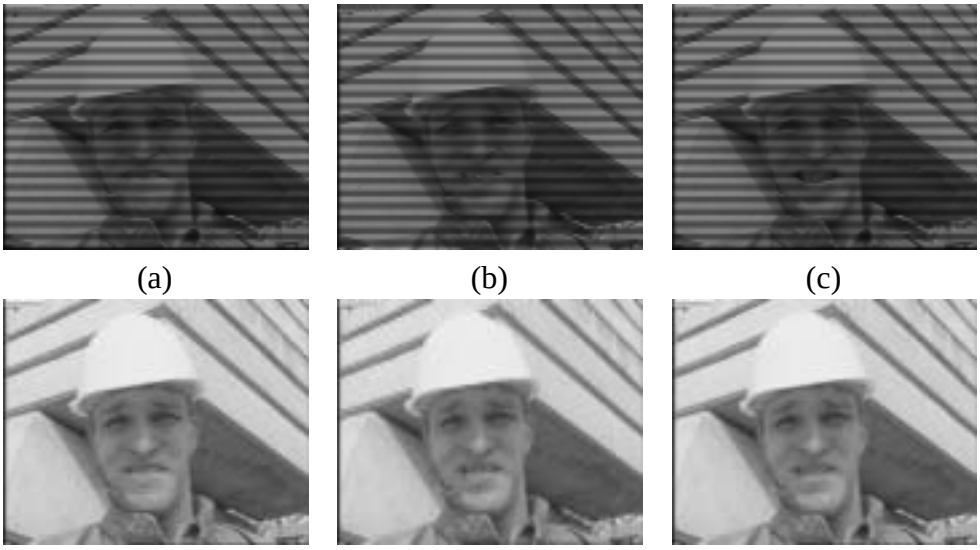


Figure 4.8: Simulation results for visual comparison for different deinterlacing approaches. (a) the 40th field of the "foreman" sequence. (b) the 41st field. (c) the 42nd frame. (d) the reconstructed 41st frame (32.36 dB) that directly combined the 40th field and the 41st field. Comb effects are very clear in the head region. (e) the reconstructed 41st frame (31.52 dB) using line-averaging of the 41st field. The image looks blurry in the non-moving regions. (f) the reconstructed 41st frame (37.45 dB) using the frame-rate up-conversion method with TMT as proposed in Chapter 3.

Chapter 5

Application in Motion Analysis and Understanding: Object-Motion Estimation and Motion-Based Video-Object Segmentation

Motion analysis and understanding is one of the key components in computer/machine vision. Relevant applications include object motion estimation, camera motion estimation, video object segmentation, 3D video object reconstruction [15, 34, 56, 61, 70, 69]. In this chapter, we present a true motion tracker for object-motion estimation and motion-based video-object segmentation.

As mentioned earlier, for each of the above applications, different degrees of accuracy and resolution in true motion tracking are required. As a result, different techniques are used for different applications. Although the neighborhood-relaxation true motion tracker is often dependable, it cannot accurately track the motion in homogeneous/occlusion regions. In Chapters 3 and 4, we used an INTRA mode detection scheme to identify occluded regions. In this chapter, we present a pre-selection technique and a post-screening

technique to tackle the tracking problems in both the homogeneous and the object occlusion regions. Specifically, these techniques are presented for object-motion estimation and motion-based video-object segmentation.

5.1 Manipulation of Video Object—A New Trend in MPEG Multimedia

Video-object segmentation is an essential task of many video applications, for instance, object-oriented video coding, object-based retrieval, and video-object manipulation in composite video.

The H.261, H.263, MPEG-1, and MPEG-2 video compression standards are based on the same framework: they use block-based motion compensation and DCT-based texture coding. Generally speaking, these techniques work very well and are eminently practical. However, two shortcomings are observed. First, the block-based coding of motion and texture induces block artifacts in reconstructed video. Second, the techniques cannot address several new functionalities identified in MPEG-4 and MPEG-7 standards, especially for those functionalities that have the content-based interactivity [2, 4, 88].

MPEG-4 is built on the successes of three fields—digital video technologies, interactive applications, and the Internet—and will provide the standardized technological elements that enable the integration of production, distribution, and content access paradigms of the three fields. As shown in Figure 5.1, MPEG-4 achieves these goals by providing a set of standardized tools to:

1. represent units of audio and visual content, called “audio/visual objects”;
2. compose these objects together to create compound audiovisual objects (e.g., an audiovisual scene);
3. multiplex and synchronize the data associated with audiovisual objects so that they

can be transported over networks; and

4. interact with the audiovisual scene generated at the receiver's end.

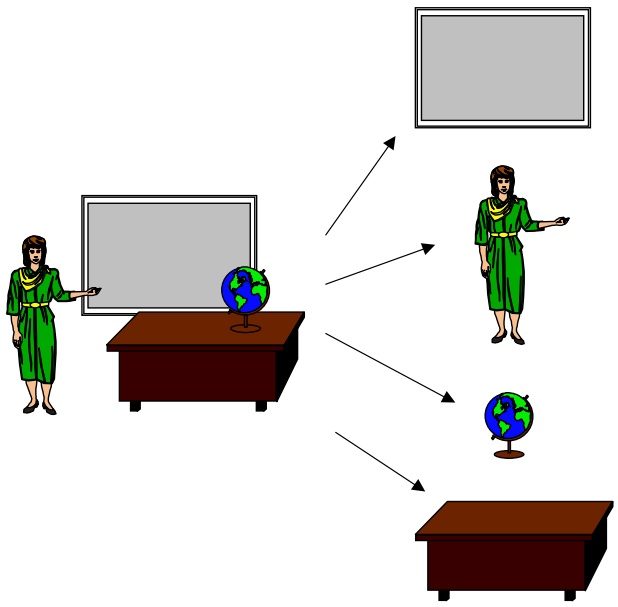
As shown in Figure 5.2, a key to the success of MPEG-4 is the segmentation of video-objects, which is the topic we now turn to.

5.2 Motion-Based Video Object Segmentation

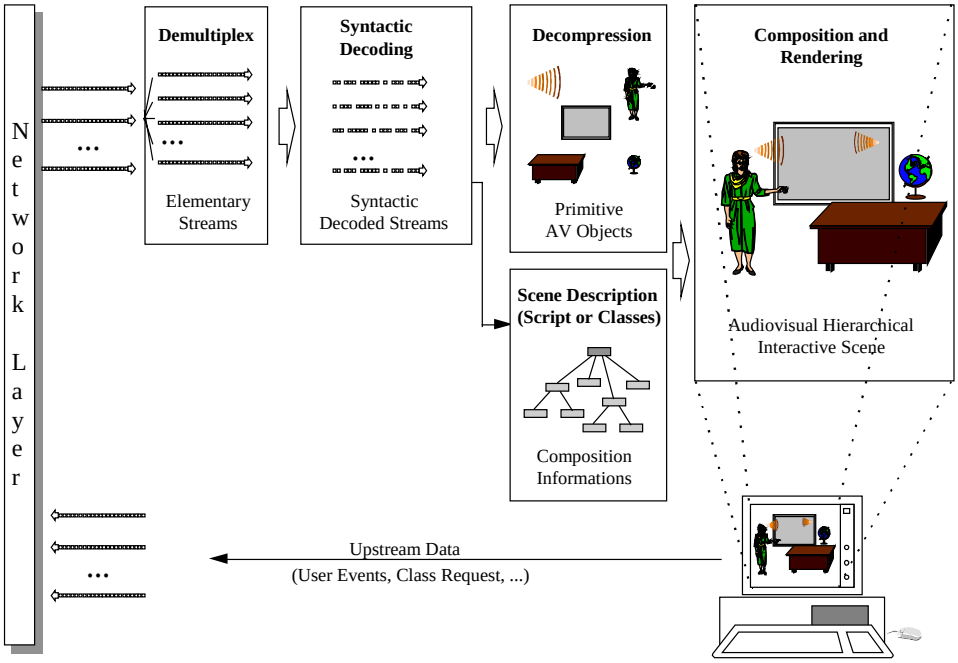
Video-object segmentation algorithms separate a video image into several different parts, such as people, trees, houses, and cars. Depending on the information used, three kinds of approaches to the segmentation of video objects can be adopted.

1. **Texture-based techniques:** Several static cues—such as edge/boundary, color, and texture—have been used for segmenting still images into different object regions. Because different objects have different color and texture, the segmentation can be achieved using the texture information. However, one object can consist of a number of different texture regions. Texture segmentation techniques often generate too many regions (with no correspondence to real objects). Hence, these techniques are most suited for sequences with simple and limited textures.

2. **Motion-based techniques:** The temporal segmentation has a great potential for segmenting video into different object regions because there is a rich body of temporal information, such as object motion [77]. Because different objects have different motion information, their segmentations can also be based on motion. Although one object can also consist of a number of different moving regions (e.g., human faces), many objects in the real scene are rigid objects (e.g., backgrounds). The motion-based segmentation over an image is generally much more homogeneous than the texture segmentations. The major pitfall of such schemes is that some parts of the image may be homogeneous, making it difficult to identify the motion, and thereby making it difficult to segment.



(a)



(b)

Figure 5.1: (a) One of the major differences between the MPEG-4 standard and the previous MPEG standards (MPEG-1 and MPEG-2) is that a MPEG-4 encoder separately encodes different video-objects. (b) Then, a MPEG-4 decoder decompresses different video-objects and composes the video scene together *based on viewers' option*.

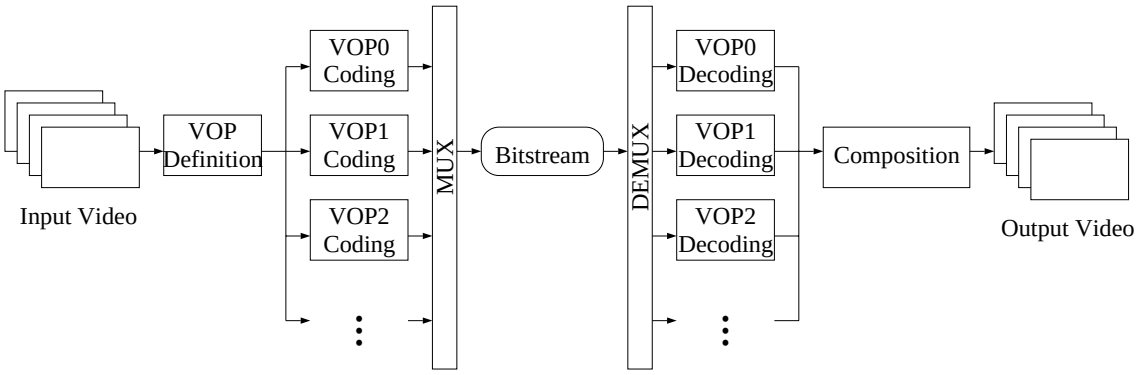


Figure 5.2: Basic MPEG-4 encoder and decoder structure. Similar to the previous MPEG-1 and MPEG-2 standards, MPEG-4 only specifies the syntax of the bit-stream, leaving opportunities for improvement at the encoder and decoder implementation. One of the keys to success at the encoder implementation is to generate proper video object planes (VOPs).

3. **Multicue-based techniques:** To overcome the hardships of the previous two kinds of techniques, the segmentation can be accomplished using multiple spatio-temporal information, such as combining texture and motion segmentation. In general, these techniques provide higher quality, but take much more computation than the previous two kinds of techniques.

As shown in Figure 5.3, our video-object segmentation method uses motion information as the major characteristic for distinguishing different moving objects and then for segmenting the scene into object regions [15, 69]. In our method, we have three steps: (1) initial motion tracking of feature blocks, (2) the initial feature block clustering, and (3) the object motion estimation and the final motion-based segmentation. The more accurate the object motion estimation, the more accurate the video-object segmentation. The more accurate the initial block motion tracking, the more accurate the object motion tracking. Therefore, in this chapter, we focus on a true motion estimation algorithm for the initial motion tracking of feature blocks, which is fundamental for object motion estimation and for the proposed video-object segmentation scheme.

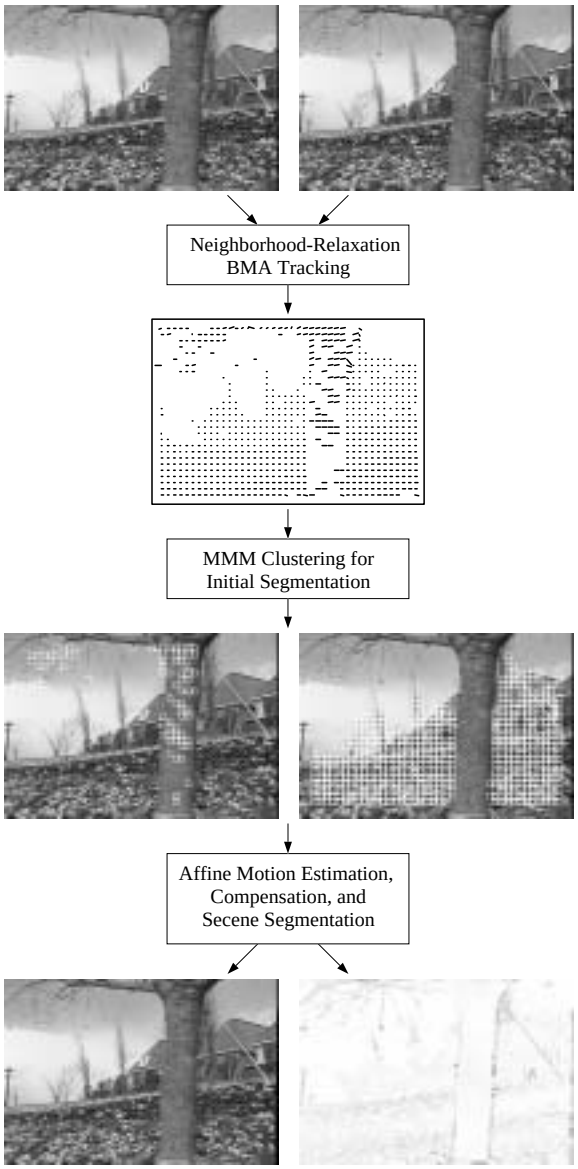


Figure 5.3: A flow chart of a motion-based segmentation by the multi-module minimization (MMM) clustering method. First, several frames of a video sequence are fed into the true motion tracker. After the tracking of moving features, a multi-module minimization neural network is used to separate the tracked features into several clusters. To model the object motion, we use affine motion whose parameters are estimated from the tracked feature blocks. Finally, affine motion estimation and affine motion compensation tests for each clusters are applied so that the image is segmented.

5.3 Block Motion Tracking for Object Motion Estimation

The goal of the proposed TMT is to find the true motion vectors for object motion estimation. The success of the proposed TMT relies on two conditions: selection of reliable feature blocks and accuracy of the tracking method.

The novelty of our feature tracker comes from three parts: (1) pre-selection of feature blocks (Section 5.3.1), (2) multi-candidate pre-screening of motion vectors (Section 5.3.2), and (3) spatial neighborhood relaxation (Section 5.3.4).

The proposed scheme has the following advantages: (1) it saves considerable computation because only reliable and important feature blocks will be tracked, (2) it screens out wrong motion vector candidates, (3) it is resilient to spatial spotty noises, and thus (4) it produces accurate true motion fields.

5.3.1 Feature Block Pre-Selection

Accurate motion estimation of a block in the presence of ambiguous regions is an extremely difficult task. In an untextured region whose intensity is constant, a block of pixels may look identical to another block of pixels. Choosing a vector to describe the motion in a homogeneous region is an over-commitment and could also be misleading: the later stages of analysis will not be able to distinguish reliable vectors from unreliable ones. As shown in Section 2.2, some researchers have advocated the use of "confidence" or "reliability" measures to indicate which motion flow vectors can be trusted [7, 100].

Observation: For object-based coding and segmentation applications, the major emphasis is not placed on the number of feature blocks to be tracked (i.e., quantity) but on the reliability of the feature blocks we choose to track (i.e., quality). This means that we can afford to be more selective in our choice of feature blocks.

Because we can afford to be more selective in our choice of feature blocks, a natural first step is to eliminate those unreliable or unnecessary feature blocks. For example, if

a block does not contain any prominent texture feature, then it is likely to be confused with its adjacent blocks because of image noise. It is advisable to exclude such a feature block from the pool of "valid" feature blocks, saving the tracking efforts and improving the tracking results.

The first step in our true motion tracker for object motion estimation is to avoid tracking such homogeneous blocks—we pre-select the feature block by taking into account the variance of the blocks. Our scheme divides a frame into blocks $\{B_{i,j}\}$ (with 8×8 pixels or 16×16 pixels), just like conventional block-matching algorithms (BMAs). However, our scheme, unlike BMAs, will not track all of the blocks because many of them do not contain sufficient prominent features, and thus are unreliable to track. If the block's variance of intensity is small, the block is considered to be of low-confidence and will be disqualified. In other words, only blocks with a variance exceeding a certain threshold will be considered.

5.3.2 Multi-Candidate Pre-Screening

After the prominent feature blocks are properly identified, the main component of this work lies in a novel technique to determine the true motion vectors.

The minimal-residue criterion does not necessarily deliver the true motion vector. As mentioned earlier in Section 2.3, Eq. (2.5) can only imply that in the noise free condition, true motion vector can deliver the minimal SAD (the sum of the absolute differences) as:

$$\vec{v}^* \in \arg\left\{\min_{v_x,v_y}\left\{\sum |I(x,y,t) - I(x+v_x,y+v_y,t+1)|\right\}\right\}$$

However, multiple minimum residues may exist. Furthermore, when some noise appears, the true motion vector does not always yield the minimal residue.

Observation: Although the minimal residue criterion often misses the true motion vector:

1. The true motion vector, while not the absolute minimum, is likely to be one of the *multiple minima*, according to the residue criterion.

2. The true motion vector is the absolute minimum if the score criterion can be modified properly.

In order to keep track of all possible true motions, which may not be the absolute minimum, we propose a multi-candidate pre-screening scheme in this section. In order to determine the true motion vectors, we propose a neighborhood relaxation score function in Section 5.3.3 (also discussed in Section 2.3).

In the true motion tracking for object motion estimation, a multi-candidate pre-screening scheme becomes necessary for maintaining a more inclusive record of possible motion vectors, and preventing (1) the true motion vector from being eliminated and (2) the wrong motion vectors from being accepted at an early stage. Since the true motion vector may not be the absolute minimum, we should make a final list among several *minima*.

At the outset, two kinds of thresholds can be used to qualify or disqualify the candidates. Let $V(i,j)$ denote the set of motion vector candidates for block $B_{i,j}$.

1. One threshold is the *lower residue threshold*, $\underline{\mathrm{R}_{th}}$. As long as the residue is less than this threshold, the motion vector will be automatically **accepted** as a possible candidate, cf. Figure 5.4(a). That is,

$$\vec{v} \in V(i,j) \quad \forall\, \mathrm{SAD}(B_{i,j},\vec{v}) \leq \underline{\mathrm{R}_{th}}$$

2. In contrast, we also propose an *upper residue threshold*, $\overline{\mathrm{R}_{th}}$. If the residue exceeds this threshold, the motion vector will be automatically **eliminated** from the candidate pool, see Figure 5.4(b). That is,

$$\vec{v} \notin V(i,j) \quad \forall\, \mathrm{SAD}(B_{i,j},\vec{v}) > \overline{\mathrm{R}_{th}}$$

After this initial stage, we now adopt a slightly more sophisticated scheme to select the final candidates. Since we intend to provide some robustness in selecting candidates, a

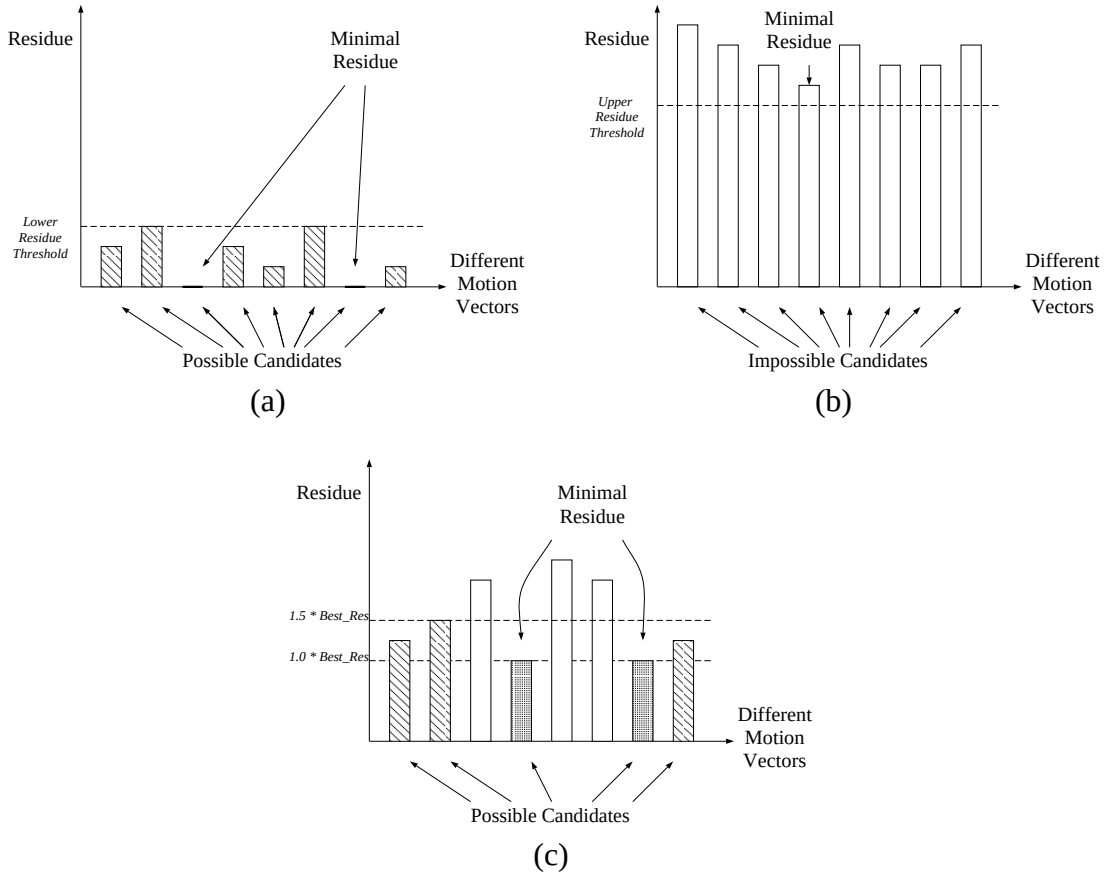


Figure 5.4: Multi-candidate pre-screening contains three parts. At the outset, two kinds of thresholds can be used to qualify or disqualify the candidates. (a) One is the *lower residue threshold*, $\underline{R_{th}}$. As long as the residue is less than this threshold, the motion vector will be automatically **accepted** as a possible candidate. (b) In contrast, we also propose an *upper residue threshold*, $\overline{R_{th}}$. If the residue exceeds this threshold, the motion vector will be automatically **eliminated** from the candidate pool. (c) After this initial stage, we now adopt a slightly more sophisticated scheme to select the final candidates. Since we intend to provide some robustness in selecting candidates, a certain noise margin must be allowed so as to admit a proper number of possible candidates. In order to maintain some kind of statistical consistency from block to block, we apply the same level of relative tolerance to all blocks.

certain noise margin must be allowed so as to admit a proper number of possible candidates, see Figure 5.4(c). In order to maintain some kind of statistical consistency from block to block, we apply the same level of *relative tolerance* to all blocks. The motion vectors yielding no more than η folds of the minimal SAD will be admitted into the candidate pool*, i.e.,

$$\vec{v} \in V(i,j) \quad \text{iff} \quad \text{SAD}(B_{i,j},\vec{v}) \leq \text{R}_{th}(B_{i,j}) \tag{5.1}$$

where we define the $\text{R}_{th}(B_{i,j})$ as η folds of the minimal SAD but no less than the lower residue threshold and no more than the upper residue threshold:

$$\text{R}_{th}(B_{i,j}) \equiv \min\left\{\max\left\{\eta \times \min_{\vec{v}}\{\text{SAD}(B_{i,j},\vec{v})\}, \underline{\text{R}_{th}}\right\}, \overline{\text{R}_{th}}\right\}$$

As mentioned in the trackability rule (Section 2.2), for higher tracking correctness, it is helpful to identify and rule out the untrackable block, which either is in the occluded or reappeared regions or contains two moving objects. When a block B is in the occluded or reappeared regions or contains two moving objects, finding a matched displacement block in the previous frame is difficult. That is, it is possible that the $\text{SAD}(B,\vec{v})$ for every motion vector candidate is larger than the upper residue threshold. Therefore, the motion vector candidate set will become null after applying this multi-candidate pre-screening technique. This scheme helps us in ruling out those untrackable blocks.

5.3.3 Neighborhood Relaxation True Motion Tracker

After the feature block pre-selection and multi-candidate pre-screening, we apply the neighborhood relaxation formulation to identify the true motion vector from the motion vector candidate set $V(i,j)$. Instead of considering each feature block individually, we determine the motion of a feature block (say, $B_{i,j}$) by moving all its neighboring blocks ($N(B_{i,j})$) along with it in a similar direction. This allows a chance that a singular and erroneous motion vector may be corrected by its surrounding motion vectors (just like

*In our statistics, when $\eta = 1.5$, about 90% of true motion vectors are kept.

median filtering). The true motion vector is the absolute minimum after the modification of the score criterion. That is, (see Eq. (2.11)).

$$\vec{v}^*_{i,j} = \arg\left\{ \min_{\vec{v} \in V(i,j)} \{\text{score}(B_{i,j}, \vec{v})\} \right\}$$

where we define the score function as

$$\text{score}(B_{i,j}, \vec{v}) = \text{SAD}(B_{i,j}, \vec{v}) + \sum_{B_{k,l} \in N(B_{i,j})} W(B_{k,l}, B_{i,j}) \min_{\vec{\delta}} \{\text{SAD}(B_{k,l}, \vec{v} + \vec{\delta})\} \quad (5.2)$$

$B_{i,j}$ represents the block of pixels whose motion we would like to determine. $N(B_{i,j})$ is the set of neighboring blocks of $B_{i,j}$. $W(B_{k,l}, B_{i,j})$ is the weighting factor for different neighbors. A small $\vec{\delta}$ is incorporated to allow some local variations of motion vectors among neighboring blocks.

5.3.4 Consistency Post-Screening

As mentioned earlier in Section 2.1 and Section 2.3, an estimation error occurs when the neighborhood contains two moving regions. In this section, we present a consistency post-screening technique to weed out possible neighborhoods that contain two moving objects.

Observation: When $\vec{v}^*_{i,j}$ is the true motion vector of $B_{i,j}$, and $\vec{v}^*_{k,l}$ is the true motion vector of $B_{k,l}$, we know that the residues of the true motion vectors are less than the residue thresholds:

$$\text{SAD}(B_{i,j}, \vec{v}^*_{i,j}) < \text{R}_{th}(B_{i,j}) \quad (5.3)$$

and

$$\text{SAD}(B_{k,l}, \vec{v}^*_{k,l}) < \text{R}_{th}(B_{k,l}) \quad (5.4)$$

Consider the following two situations:

1. $B_{k,l}$ and $B_{i,j}$ are in a single object:

   Since they belong to the same object, $\vec{v}^*_{i,j}$ and $\vec{v}^*_{k,l}$ should be similar to each other, i.e.,

   $$\vec{v}^*_{k,l} = \vec{v}^*_{i,j} + \vec{\delta} \tag{5.5}$$

   From Eq. (5.4) and Eq. (5.5), we know that the SAD of a block using the relaxed true motion vector of the neighbor block is less than the residue threshold:

   $$\mathrm{SAD}(B_{k,l}, \vec{v}^*_{i,j} + \vec{\delta}) < \mathrm{R}_{th}(B_{k,l}) \tag{5.6}$$

   Therefore, from Eq. (5.3) and Eq. (5.6), we know that the score of the neighborhood relaxation formulation (the weighted sum of the SADs) should be less than the weighted sum of the residue-thresholds:

   $$\mathrm{SAD}(B_{i,j}, \vec{v}^*_{i,j}) + W(B_{k,l}, B_{i,j}) \times \mathrm{SAD}(B_{k,l}, \vec{v}^*_{i,j} + \vec{\delta}) < \mathrm{R}_{th}(B_{i,j}) + W(B_{k,l}, B_{i,j}) \times \mathrm{R}_{th}(B_{k,l}) \tag{5.7}$$

2. $B_{k,l}$ and $B_{i,j}$ are *not* in a single object:

   Assuming that the true motion vector $\vec{v}^*_{i,j}$ is much different from the true motion vector $\vec{v}^*_{k,l}$, we know that the SAD of a block using the relaxed true motion vector of the neighbor block is greater than the residue threshold:

   $$\mathrm{SAD}(B_{k,l}, \vec{v}^*_{i,j} + \vec{\delta}) \gg \mathrm{R}_{th}(B_{k,l})$$

   Therefore, it is likely that the score of the neighborhood relaxation formulation (the weighted sum of the SADs) is greater than the weighted sum of the residue-thresholds:

   $$\mathrm{SAD}(B_{i,j}, \vec{v}^*_{i,j}) + W(B_{k,l}, B_{i,j}) \times \mathrm{SAD}(B_{k,l}, \vec{v}^*_{i,j} + \vec{\delta}) > \mathrm{R}_{th}(B_{i,j}) + W(B_{k,l}, B_{i,j}) \times \mathrm{R}_{th}(B_{k,l}) \tag{5.8}$$

From this observation, our post-screening step disqualifies a block from the feature block list when the score of the estimation motion vector is greater than the weighted sum

of the residue-thresholds, i.e.,

$$\mathrm{SAD}(B_{i,j},\vec{v}^{*}_{i,j}) + \sum_{B_{k,l}\in N(B_{i,j})} W(B_{k,l},B_{i,j}) \times \min_{\vec{\delta}}\{\mathrm{SAD}(B_{k,l},\vec{v}^{*}_{i,j}+\vec{\delta})\} > \mathrm{R}_{th}(B_{i,j}) + \sum_{B_{k,l}\in N(B_{i,j})} W(B_{k,l},B_{i,j}) \times \mathrm{R}_{th}(B_{k,l})$$

This rules out the tracking of the neighborhood that contains two moving objects for higher tracking correctness.

5.3.5 Background Removal

Generally, we are not interested in non-moving background objects. (If the background is of some interest, it can be labelled and tracked as an object.) For example, in our motion-based video-object segmentation, before initially clustering the feature blocks, a preprocessing step that removes the feature blocks belonging to the background can facilitate the clustering process. As discussed in [69], after the background blocks are removed, the remaining feature blocks corresponding to the foreground objects tend to have much better *center dominance*, which is critical to the success of object motion classification by clustering the feature blocks in the principal component domain.

Our tracker has a background-removal capability. In [56], the method detects and then separates moving parts from the background by assuming that the camera never moves. Here we use a different method, in which we assume that the background takes up the largest area in the frame and thus the dominant motion vector involved in the scene is due to the camera movement [63]. When the majority of motion vectors with a relaxation term is extracted, the corresponding blocks are regarded as background blocks. Because the relaxation term allows neighboring blocks to have a similar but not exactly the same motion, this can accommodate most of the camera motions (panning, tilting, translating).

5.4 Performance Comparison in Feature Block Tracking

5.4.1 Qualitatively

Figure 5.5 depicts two consecutive frames ((a)(b)) and the corresponding blocks tracked by our method ((c)(d)). The true motion field is illustrated in Figure 5.5(e). Figure 5.5(f) shows the motion vectors estimated by the minimal-residue block-matching algorithm (BMA). The conventional full-search BMA obviously performs poorest, with spurious vectors scattered around in homogeneous regions. Significant improvement is observed (Figure 5.5(g)) by applying the feature block pre-screening to trim down the unreliable blocks. However, there are still noticeable tracking errors on the object boundaries (e.g., above and below the left-book). Figure 5.5(h) shows the motion vectors estimated by our approach. The neighborhood relaxation shows clearly improvement in tracking of the blocks in the same region, e.g., the lower-left corner. The multi-candidate pre-screening has succeeded in eliminating many wrong motion vectors. Still, we have observed minor tracking errors on points along a long edge (e.g., below the left-book).

A possible solution to avoiding tracking errors on points along a long edge is to adopt Anandan's scheme [7], in which the weighting factor (in $W(\cdot)$ of Eq. (5.2)) takes into account the cross-correlation between vertical/horizontal components of the motion vector and image features. Its purpose is that the motion of a block that has a low confidence in vertical movement can be guided by its neighbors that have a higher confidence in vertical movement.

Figure 5.6 shows the comparison of tracking results using the "coastguard" sequence. Figure 5.7 shows the comparison of tracking results using the "foreman" sequence. It is clear that the estimated motion field estimated by our approach is much closer to the true motion. Most of the tracking errors are removed after the feature block pre-selection, the consistency post-screening, and the background-removal.

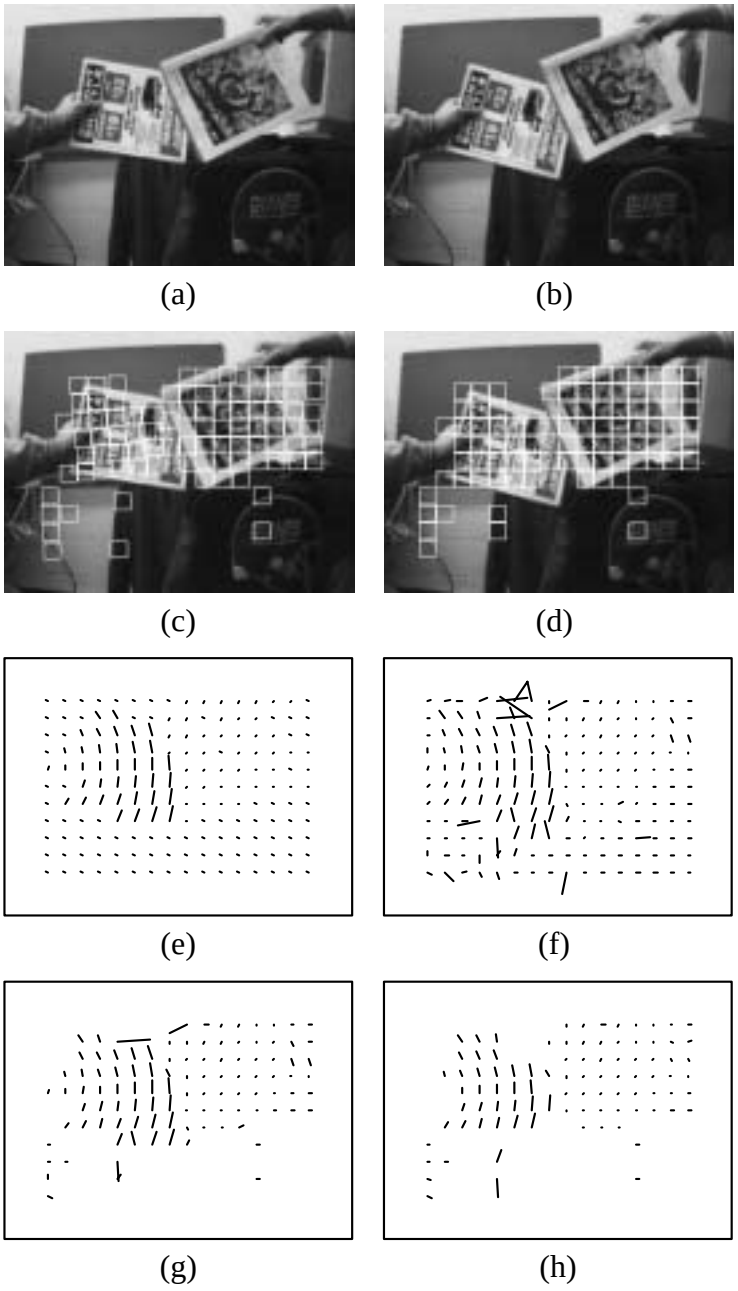


Figure 5.5: (a) and (b) show 2 consecutive frames of 2 rotating books amid a panning background, (c) and (d) show the feature blocks tracked by our approach. (e) shows the true motion field corresponding to (a) and (b). (f) the motion vectors estimated by conventional full-search block-matching algorithms (BMAs). (g) the motion vectors estimated by conventional BMA with the feature block pre-selection (a variance threshold applied). (h) the motion vectors estimated by our approach, corresponding to (c) and (d).

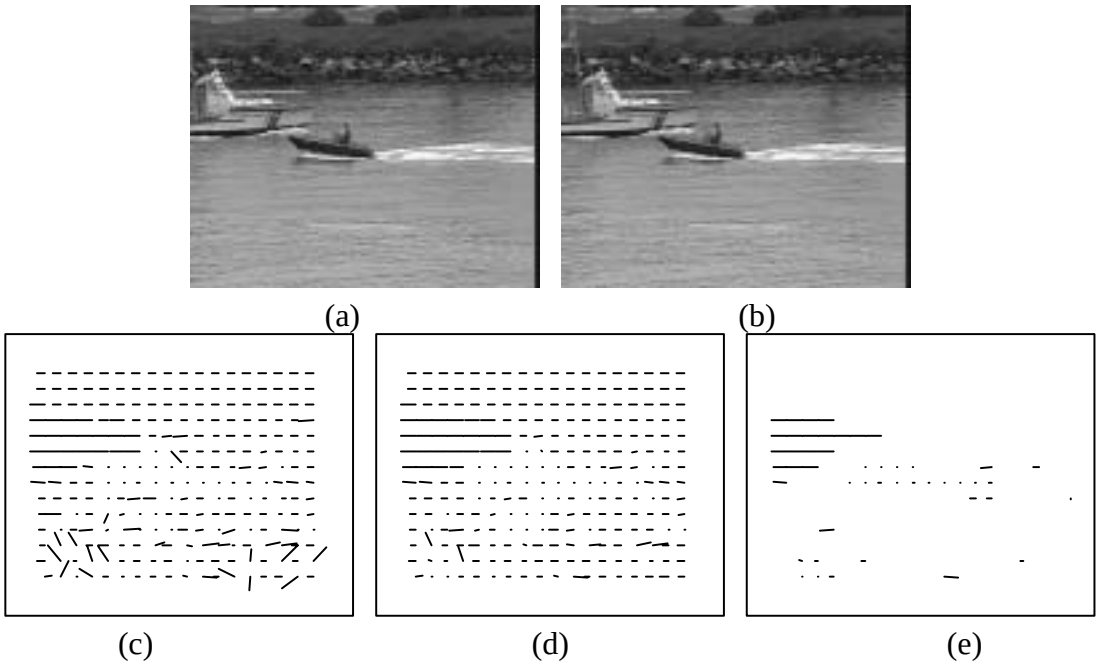


Figure 5.6: (a) and (b) show the 30th and 34th frames of the “coastguard” sequence. The background is moving rightward in order to track on the lower boat. The upper boat is moving fast rightward. (c) shows the motion vectors estimated by conventional full-search BMAs. There are some errors in the water. (d) shows the motion vectors estimated by our neighborhood relaxation method. The motion field is much closer to the true motion. (e) shows the motion vectors estimated by our approach with the feature block pre-selection, the consistency post-screening, and the background removed. Most of the tracking errors are removed. It is clear from the motion field that there are two moving objects in this picture: the lower boat is moving slowly while the upper boat is moving quickly.

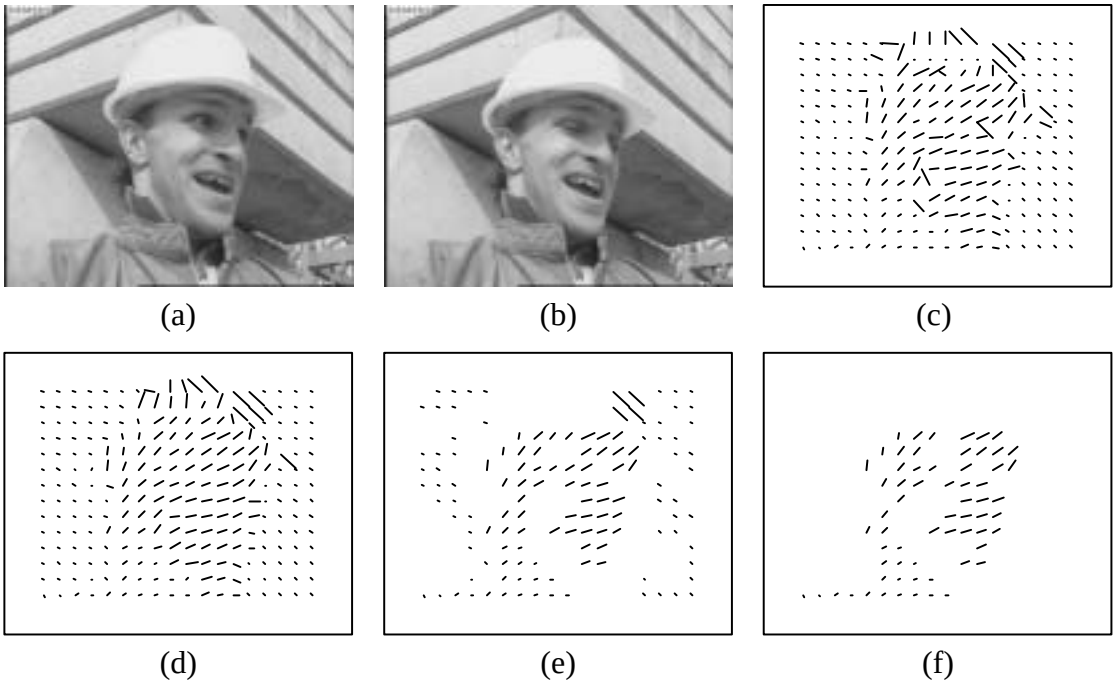


Figure 5.7: (a) and (b) show the 132th and 136th frames of the “foreman” sequence. He is moving his head backward. (c) shows the motion vectors estimated by conventional full-search BMAs. There are some errors around his head. (d) shows the motion vectors estimated by our neighborhood relaxation method. The motion field is much closer to the true motion. (e) shows the motion vectors estimated by our neighborhood relaxation method with feature block pre-selection and consistency post-screening. Some blocks of the low confidence are trimmed. The motion field is even closer to the true motion. (f) shows the motion vectors estimated by our approach with feature block pre-selection, consistency post-screening, and background removed. Most of the tracking errors are removed.

The tracked results can be subsequently used for motion-based video-object segmentation. In [69], we apply a principal component coordinate transformation on the *feature track matrix* (formed by the tracked feature blocks) for separating image layers or moving objects. As shown in Figure 5.8, our method has the following steps: (1) a TMT captures the true motion vectors, (2) feature blocks contained within different moving objects form separate clusters on the principal component space, and (3) the frame can thus be divided into different layers (segments) characterized by a consistent motion. Figure 5.9 and Figure 5.10 show the segmentation results using our TMT.

5.4.2 Quantitatively

Figure 5.11 illustrates the scheme that is used to measure the quality or "trueness" of the feature-block motion estimation for the object motion estimation. (1) Motion estimation is performed on the unsegmented original frames to yield the translational motion vectors ($\{[v_{xi}, v_{yi}]^T\}$). (2) Affine motion parameters are found using the translational motion vectors and the video-object-plane ($\mathrm{VOP}_j(t)$). (3) The affine motion parameters ($\{a_{ij}, b_i\}$) and the video-object-plane at frame t ($\mathrm{VOP}_j(t)$) are used to predict the video-object-plane at frame $t+1$ ($\hat{\mathrm{VOP}}_j(t+1)$). (4) By comparing $\mathrm{VOP}_j(t+1)$ and $\hat{\mathrm{VOP}}_j(t+1)$, we know the performance of the motion estimation for individual blocks.

Table 5.1 shows the comparison of object-motion estimation using different block-motion estimation algorithms. We use standard test sequences and video-object segmentations provided by the MPEG-4 committee [3]. Because we use the 2D affine model as our object motion model, we limit our performance analysis to the video objects that can be described by the affine model (e.g., static background, translational moving objects, and moving background). In our simulations, the block size is 16×16, the neighborhood includes the nearest four blocks (see Figure 2.3), and the neighborhood weighting factor is a constant 0.3.

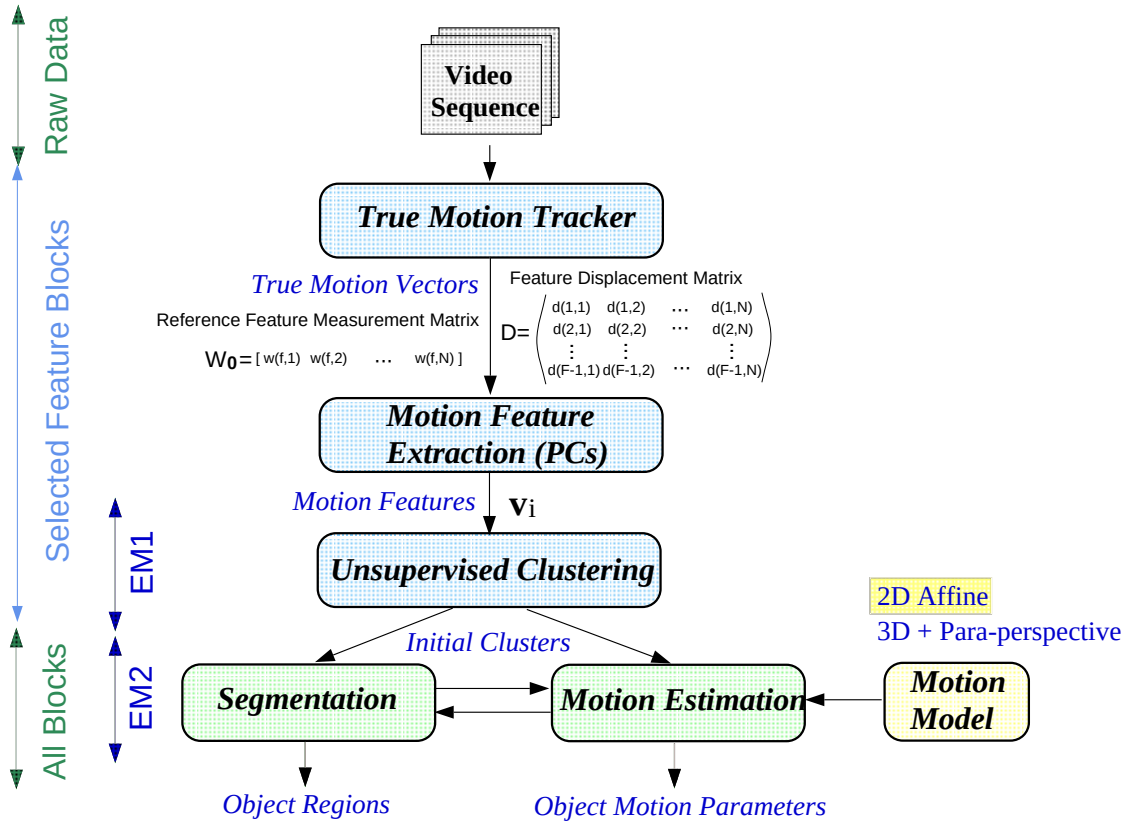


Figure 5.8: Flow chart of a motion-based video-object segmentation algorithm proposed in [69]. Primitive local motion attributes are the true motion vectors of a set of feature blocks determined by a TMT. The feature for each block is extracted by a principal component (PC) analysis. After that, the feature blocks are clustered into motion clusters. Finally, all the blocks in the scene are processed. The expectation-maximization (EM) algorithm is adopted in the last two steps.

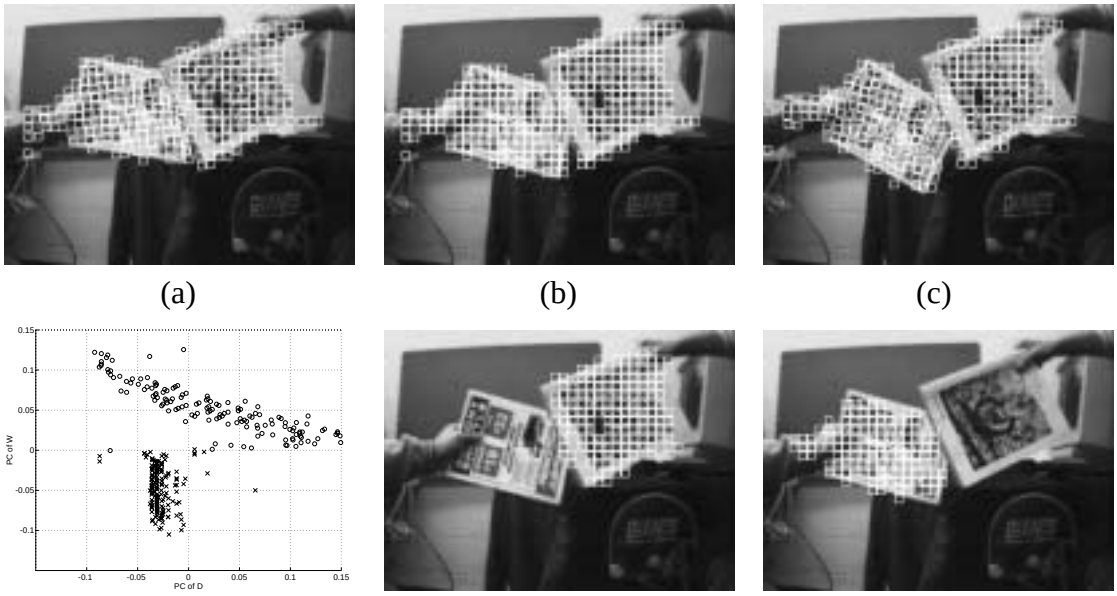


Figure 5.9: Tracking and clustering results of the “2-Books” sequence using our TMT with the motion-based segmentation algorithm proposed in [69] (see Figure 5.8). (a)(b)(c) show feature blocks tracked through 3 frames. (Background blocks are removed.) (d) depicts clusters of motion features in the feature space. Two very distinctive object clusters are presented in the feature space. Motion features of blocks from the *left book* are marked in ’o’ and those from the *right book* are marked in ’x’. (e) and (f) show the corresponding feature blocks for each object marked on the image after the unsupervised feature block clustering.

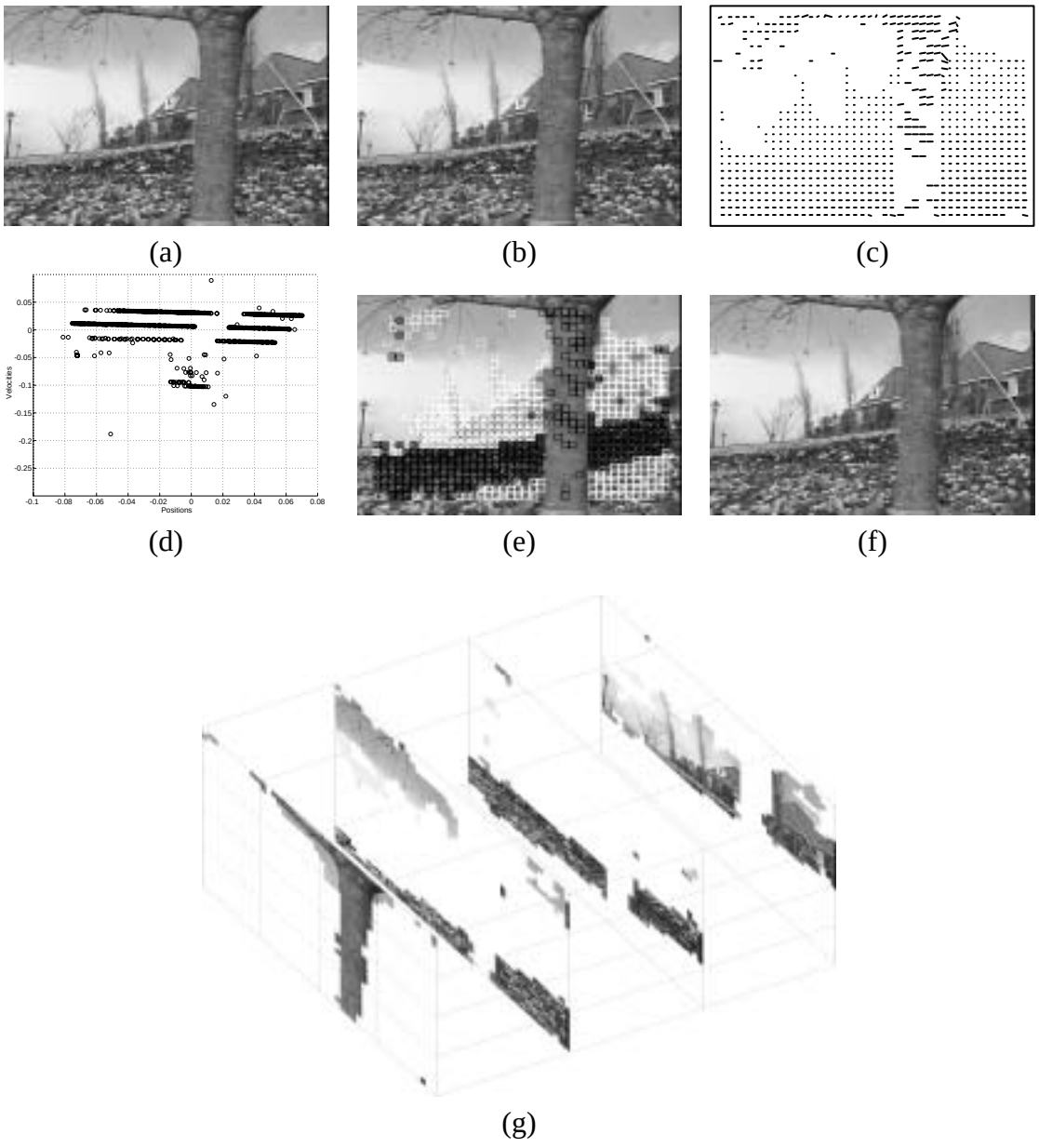


Figure 5.10: The segmentation result (layer extraction) on the "flower garden" sequence [69] (see Figure 5.8). (a)(b) show original frames. (c) shows the tracked motion field. (d) shows the feature blocks in the principal component domain. Four distinctive and parallel layers are prominently displayed in the feature space. From top to bottom: house-sky, back-garden, front-garden, and tree. (e) Feature blocks are classified into four clusters. (g) shows four-layer segmentation of the entire frame. (f) shows the object-motion compensated frame (without residue).

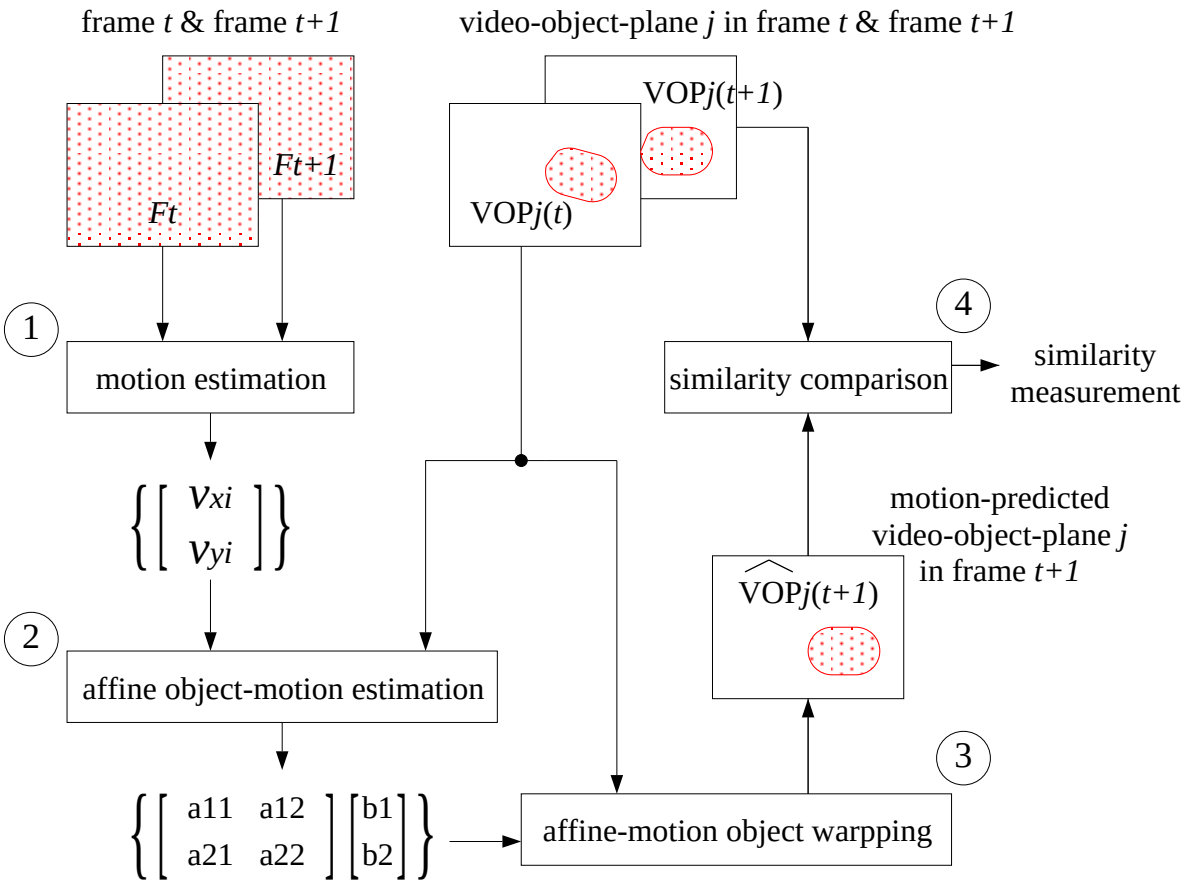


Figure 5.11: A measurement of quality or "trueness" in feature-block motion estimation for the object motion estimation. (1) Motion estimation is performed on the unsegmented original frames for translational motion vectors ($\{[v_{xi}, v_{yi}]^T\}$). (2) Affine motion parameters are found using the translational motion vectors and the video-object-plane ($\text{VOP}_j(t)$). (3) The affine motion parameters $\left(\left\{\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}\right\}\right)$ and the video-object-plane at frame t ($\text{VOP}_j(t)$) are used to predict the video-object-plane at frame $t+1$ ($\hat{\text{VOP}}_j(t+1)$). (4) By comparing $\text{VOP}_j(t+1)$ and $\hat{\text{VOP}}_j(t+1)$, we measure the performance of the motion estimation for individual blocks.

Video objects	Picture size	Original minimal-residue BMA	Our true motion tracker	Improvement (dB)
akiyo 0	CIF	39.67	39.67	0.00
(static background)	QCIF	39.45	39.45	0.00
coastguard 0	CIF	20.14	20.48	0.34
(water)	QCIF	21.71	22.68	0.97
coastguard 1	CIF	16.51	17.07	0.56
(moving boat)	QCIF	16.06	15.63	-0.43
coastguard 3	CIF	25.16	25.64	0.48
(moving background)	QCIF	25.40	26.04	0.64
container 0	CIF	31.29	31.34	0.05
(water)	QCIF	31.06	31.50	0.46
container 1	CIF	21.75	21.79	0.04
(static ship)	QCIF	23.14	23.14	0.00
container 4	CIF	22.04	27.70	5.66
(static background)	QCIF	27.66	29.23	1.57
hall monitor 0	CIF	26.95	27.32	0.37
(static background)	QCIF	27.86	27.88	0.02
news 0	CIF	40.76	40.68	-0.08
(static background)	QCIF	40.92	41.16	0.24
stefan 1	CIF	19.13	19.50	0.37
(moving background)	QCIF	22.00	22.40	0.40

Table 5.1: Performance measurement of object-motion estimation using different block-motion estimation algorithms, which include (1) the original minimal-residue BMA and (2) our true motion tracker (see Figure 5.11). In the MPEG-4 standard test sequences, we use available video objects that can be described by a 2D affine model. It needs some improvement at two conditions when the video object is smaller than the TMT neighborhood and when the video object has “untextured” long edges (see Figure 5.5). On the average, our method is 0.3–0.5 dB superior to the original BMA.

On average, our true motion tracker (which includes feature-block preselection, multi-candidate prescreening, neighborhood relaxation, and consistency postscreening techniques) performs better than the original full-search BMA. In the test condition "container 4," our TMT demonstrates a great improvement over the original BMA. The video object is a static background, sky and some buildings. The sky region has no texture, and the blocks in this region cannot be tracked well. By using our feature-block prescreening technique, we can filter out these unreliable blocks and improve object-motion accuracy.

Our TMT will need some improvement in two cases: First, our method should check that the TMT neighborhood is not larger than the video object. In the test condition "coast-guard 1," the video object is a moving boat, which is relatively smaller than the size of the TMT neighborhood when the picture size is QCIF and the block size is 16×16. In this case, neighborhood relaxation introduces noise into the motion vectors. As a result, the object-motion estimation using our TMT is less accurate than the object-motion estimation using the BMA (see Section 2.1). On the other hand, (1) when the picture size is CIF and the block size is 16×16 or (2) when the picture size is QCIF and the block size is 8×8, the size of the neighborhood is smaller than the object and so we can see improvement in the SNR (0.56 dB/0.76 dB).

Second, our method should eliminate a block along a long edge from the feature-block list. In the test condition "news 0," performance degradation is the result of insufficiencies and side-effects of the feature-block preselection scheme. After the feature-block prescreening technique is used, some blocks that are considered untrackable are eliminated. The side-effect is that the number of feature blocks is reduced and thus the object-motion estimation is rendered more sensitive to noise from the block-motion estimation. If there is an error in the block-motion estimation, methods with fewer feature blocks perform worse than methods with more feature blocks. For example, when the object-motion estimation performance using the original BMA is measured with our feature-block preselection technique, the SNR is 40.64 dB, i.e., our feature-block preselection technique in the original

BMA degrades the performance by 0.12 dB. Although our neighborhood relaxation formulation can improve the correctness of the motion field, it cannot correctly track a block along a long edge (as shown in Section 5.4.1). In the future, it will be helpful to delete a block that has only directional variance.

To summarize, the techniques presented in this chapter allow us to extend our baseline true motion tracker to a true motion tracker for video-object motion estimation and motion-based video-object segmentation, important in both MPEG-4 and MPEG-7 standards. In previous chapters, some noticeable tracking failures occur at homogeneous regions and object boundaries. In the video-object segmentation and video-object motion-estimation applications, we have the option of selecting reliable motion vectors. The results show that our true motion tracker is superior to conventional approaches that do not use proper pre-selection or post-screening schemes.

Chapter 6

Effective System Design and Implementation of True Motion Tracker

This chapter shows another promising aspect of our true motion tracker (TMT). In previous chapters, applications drive the new design of the TMT. Conventional block-matching algorithms (BMAs) for motion estimation demand intensive computation. Our TMT requires even more computation and memory bandwidth. The discussion of this work would have been incomplete if we did not cover the subject of system design and implementation of the TMT.

BMAs have been widely employed in various international video compression standards to remove temporal redundancy. As shown in Figure 1.5, the basic idea of the BMA is to locate a displaced candidate block that is most similar to the current block, within the search area in the previous frame. Various similarity measurement criteria have been presented for block matching. The most popular one is the sum of the absolute differences (SAD) as

$$\mathrm{SAD}[u,v] = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} \mid s[i+u, j+v] - r[i,j] \mid \qquad -p \le u < p,\ -p \le v < p \tag{6.1}$$

where n is the block width and height, p is the absolute value of the maximum possible vertical and horizontal motion, $r[i,j]$ is the pixel intensity in the current block at coordinates

(i, j), $s[i+u, j+v]$ is the pixel intensity in the search area in the previous frame, and (u, v) represents the candidate displacement vector. The motion vector is determined by the least SAD$[u, v]$ for all possible displacements (u, v) within a search area, as shown here:

$$\text{Motion Vector} = \arg\min_{[u,v]}\{\text{SAD}[u, v]\} \tag{6.2}$$

The operations of a BMA are simple—additions and subtractions. However, BMAs are known to be the main bottleneck in real-time encoding applications. For example, 6.2×10^9 additions per second and 3.1 GB/sec external memory access would be required for a real-time MPEG-1 video coding, when there are 30 frames per second with frame size 352 pixels $\times$ 288 pixels and search range $\{-16, \cdots, +15\} \times \{-16, \cdots, +15\}$. The search for an effective implementation has been a challenging problem for years.

Our TMT demands even more computation and memory bandwidth. The goal of this section is to demonstrate an effective implementation of the TMT on programmable fine-grain parallel architectures.

6.1 Programmable Multimedia Signal Processors

Novel algorithmic features of multimedia applications and advances in VLSI technologies are driving forces behind the new multimedia signal processors. For example, the rapid progress in VLSI technology will soon reach more than 100 million transistors in a chip. This implies a tremendous amount of computing power for many multimedia applications. The silicon area required for implementing a specific function will decrease considerably, and a higher functionality can be realized on a single chip, for example, single-chip MPEG-2 encoders from NEC Corporation or Philips Electronics [5]. This trend leads to the integration of programmable processor cores, function-specific modules, and various system interfaces in order to enable high multimedia functionality at decreased system design costs.

Many architecture platforms have been proposed to provide high performance and flexibility. The overall architectural design can be divided into *internal* and *external* design spaces. The internal design focuses on *core processor* upgrade, while the external design focuses on *accelerators* that off-load tasks from the main core.

Several alternatives are available for exploiting the parallelization potential of multimedia signal processing algorithms for programmable architectures (cf. Table 6.1):

1. Single Instruction Stream, Multiple Data Streams (external SIMD):

   Aiming at data parallelism, SIMD (Single Instruction Stream, Multiple Data Streams) architectures are characterized by several data paths executing the same operation on different data entities in parallel. While a high degree of parallelism can be achieved with little control overhead, data path utilization rapidly decreases for scalar program parts. In general, pure SIMD architectures are not an efficient solution for complex multimedia applications; they are best suited for algorithms with highly regular computation patterns. For example, Chromatic's Mpact 2, which can deliver 6 BOPS*, is a mixture of SIMD and VLIW [26].

2. Split-ALU (internal core-processor SIMD):

   Architectures featuring a split-ALU are based on a principle similar to SIMD: a number of lower-precision data items are processed in parallel on the same ALU. Figure 6.1 shows a possible implementation of the split-ALU concept. The advantage of this approach is its small incremental hardware cost provided a wide ALU is already available. Recent multimedia extensions of general-purpose processors are typically based on this principle, e.g., MAX-2 for HP's PA-RISC [65], VIS for SUN's UltraSparc [79], MMX for Intel's x86 [50].

3. Multiple Instruction Streams, Multiple Data Streams (external MIMD):

*BOPS: billion operations per second.

Processor (Reference)	Architecture	Size of datapath (bits)	Internal communi-cation	Internal data memory (KB)	Peak perform-ance (BOPS)	External memory bandwidth (MB/s)
Chromatic Mpact 2 [26]	VLIW/SIMD (6 ALUs)	72	792-bit cross-bar (18 GB/s)	4	6.0	1200
NEC V830R/ AV [93]	RISC core + a SIMD (split-ALU) processor	32/ 64	64-bit bus (1.6 GB/s)	16	2.0	600
Philips TriMedia [84]	VLIW (27 FUs)	32	32-bit bus (400 MB/s)	16	4.0	400
TI 'C62x [97]	VLIW (6 ALUs + 2 multipliers)	32/ 16	32-bit bus (800 MB/s)	64	1.6	800
TI 'C8x [98]	4 split-ALU DSPs + RISC core (MIMD)	32/ 32	crossbar (2.4 GB/s)	36	2.1	480

Table 6.1: List of some announced programmable multimedia processors. (1) All of them use massive parallelism (SIMD, split-ALU, MIMD, or VLIW) and pipelines. (2) In general, the size of the operands for the ALUs or functional units is less than 32 bits. Some of the ALUs are the split-ALUs that can operate on multiple sets of operands in one instruction. (3) They have high-speed and high-bandwidth internal communication channels (400 MB/s to 18 GB/s). (4) They all have high-speed on-chip data memory (register file, cache, RAM). (5) They can provide high computing power (1 BOPS to 6 BOPS). (6) Their external memory bandwidths are unusually high (400 MB/s to 1200 MB/s).

Task level as well as data level parallelism can be exploited by MIMD (Multiple Instruction Streams, Multiple Data Streams) architectures, which are characterized by a number of parallel data paths featuring individual control units [44]. Thus, MIMD processors offer the highest flexibility for algorithms to be executed in parallel. For example, TI's TMS320C80 [98], which can deliver 2 BOPS, is a MIMD processor. However, MIMD processors incur a high hardware cost for multiple control units as well as for a memory system delivering the sufficient bandwidth to supply all required instruction streams. Furthermore, synchronization difficulties and poor programmability have prevented MIMD processors from widespread use in multimedia applications so far.

4. Very Long Instruction Word (internal core-processor MIMD):

   Instruction level parallelism is targeted by VLIW (Very Long Instruction Word) architectures, which specify several operations within a single long instruction word to be executed concurrently on multiple functional units (Figure 6.2) [38]. In contrast to superscalar architectures, VLIW processors must rely on static instruction scheduling performed at the compilation time. The advantage is a simplified design since no hardware support for dynamic code reordering is required. For example, TMS320C6201 [97], a general-purpose programmable fixed-point DSP adopting a Very Long Instruction Word (VLIW) implementation, can deliver 1600 MIPS[†].

Simultaneously, another widely employed way of adapting programmable processors to special multimedia signal processing algorithm characteristics is to introduce specialized instructions for frequently recurring operations of higher complexity, e.g., a multiply-accumulate operation with saturation [78]. By replacing longer sequences of standard instructions, the use of specialized instructions may significantly reduce the instruction count,

[†]MIPS: million instructions per second.

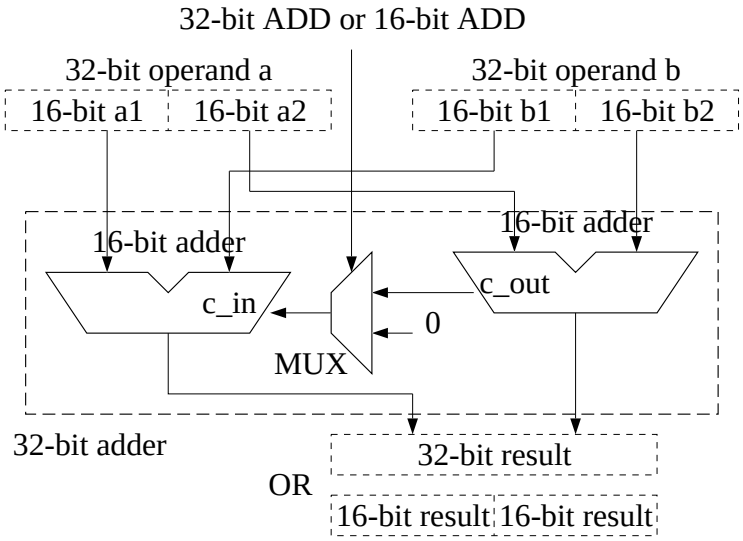


Figure 6.1: An example of the split-ALU implementation. A 32-bit adder can work as two 16-bit adders, which add two pairs of 16-bit operands. The only difference between the two functionalities of this adder is the carry propagation from the lower 16-bit adder to the upper 16-bit adder. Splitting this 32-bit adder into two 16-bit adders allows one single instruction to process multiple data. This data parallelism (also called subword parallelism) is quite similar to the SIMD architecture.

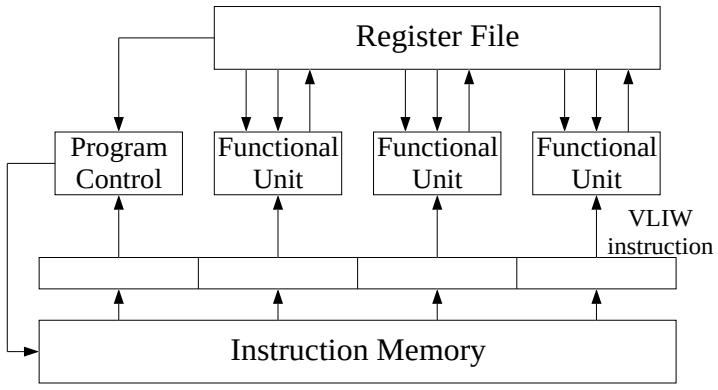


Figure 6.2: A generic VLIW architecture. A very-long-instruction-word architecture consists of multiple functional units (FUs). An issue of the VLIW instruction can activate multiple FUs to operate independently on multiple sets of operands.

```
; Assume ‘‘r0’’ holds 0
;        ‘‘r5’’ holds 255
      cmp   r1,r5     ; If r1 >= 255,
      bge   _max      ;    go to _max.
      mov   r5,r1     ; Clip to 255.
      br    _next
_max: cmp   r0,r1     ; If r1 <= 0,
      bge   _next     ;    go to _next.
      mov   r0,r1     ; Clip to 0.
_next:
```

(a)

```
      min3  r1,r5,r1  ; If r1 > 255, r1 = 255.
      max3  r0,r1,r1  ; If r1 < 0, r1 = 0.
```

(b)

Figure 6.3: Specialized instructions replace sequences of standard instructions: for example, the instruction stream for minimum maximum operations on the V810 (a) compared to the V830 (b). By introducing a single new instruction comprising a frequently executed sequence of standard instructions, the instruction count of multimedia code can be reduced significantly [78].

resulting in faster program execution (Figure 6.3). The design complexity required for implementing specialized instructions can usually be kept at modest levels; the decision about which instructions to implement depends on the probability of their use.

6.1.1 A High-Throughput Architectural Platform for Multimedia Application

In this section, we present an architecture platform for multimedia applications that builds upon earlier platforms.

Multimedia signal processor design is driven by algorithmic features in multimedia applications. From algorithmic perspectives, important characteristics of the multimedia signal processing algorithms can be summarized as following:

1. *Intensive computation for highly regular operations*

   Computation-intensive applications usually depend on a loop of instructions. There is a large amount of computations for highly regular operations. There is a great deal of parallelism on common operations, such as addition, subtraction, and multiplication. Therefore, parallels and pipelines should be exploited in the multimedia architecture (cf. Table 6.1).

2. *Intensive I/O or memory access*

   There is a large amount of I/O or memory access in multimedia applications. Hence, a multimedia signal processor should be able to support a high memory bandwidth (cf. Table 6.1). Because multimedia data operands have very frequent and very regular reusability, a good architecture should make good use of the data reusability.

3. *Frequent execution of small integer operands*

   In MPEG and other pixel-oriented algorithms, the data being operated on are small integers (such as, 8-bit or 16-bit), narrower than the existing integer data paths of microprocessors. Small processing elements or subword parallelism must be

exploited for higher efficiency, e.g., HP's PA-RISC [65], Intel's x86 [50], NEC's V830R/AV [93], SUN's UltraSparc [79], TI's C80 [98].

4. *High control complexity in less computationally intensive tasks*

   There are also some high control complexity tasks that are less time-consuming. It may be more efficient and economical to resort to software solutions for such tasks. Therefore, flexible RISC cores (master processors) are preferred, e.g., NEC's V830R/AV [93] and TI's C80 [98].

Multimedia signal processor design is also driven by available VLSI technologies. There are two important features in VLSI technologies:

1. *External memory is slow*

   There is a huge gap between memory speed and processor speed. Therefore, a high-speed on-chip data memory (register file, cache, RAM) is necessary to bridge the gap. For example, most of the announced programmable media processors (as listed in Table 6.1) use 16 KB to 64 KB on-chip data memory.

2. *Long-distance communication is slow*

   Because the feature size of the processing technology is getting smaller and smaller, more and more of the signal delay is on the wire than the transistor [73]. Long-distance and global (one to many) communication takes longer and is more expensive than local communication. Hence, for a sound design, it is important to make use of local communication channels and it is necessary to support local communication efficiently.

Conventional standard processors do not correspond well to those characteristics of multimedia signal processing algorithms. Therefore, special architectural approaches are necessary for multimedia processors to deliver the required high processing power with efficient use of hardware resources.

It is generally agreed that some multimedia signal processing functions can be implemented using programmable CPUs (software solutions) while others must rely on hardware accelerators (hardware solutions) [71]. A sound multimedia signal processing architecture style should be based on this principle.

Figure 6.4 shows a proposed architecture style for high-performance multimedia signal processing that is built upon some earlier platforms proposed by [44, 102, 103, 110]. It consists of array processors used as the hardware accelerator and RISC cores as the programmable processor. The programmable processor provides software solutions (which mean high flexibility) while the accelerator provides hardware solutions for high performance.

The processing array in the proposed architecture platform has the following unique features. (1) Every PU has its own local data memory/cache. The local caches have an external control protocol. For example, the program can ask the caches not to cache some part of the data [25]. (2) There is a local bus between two consecutive PUs. Hence, the PUs can talk to each other in two ways: (a) via the local bus between them, and (b) via the global communication channel, which may be a bus or a crossbar network.

These unique features provide two advantages. (1) The local data memory can provide very high data throughput. (2) The local communication can provide very high communication bandwidth between two consecutive PUs at an attractively low cost (in terms of area, power, and delay).

It is critical to note that multimedia signal processor designs are supported by algorithmic partitioning of multimedia applications. In order to have an effective execution, given a specific application, the algorithm is first manually or semi-automatically divided into two parts (cf. Figure 6.5):

1. *Computationally intensive and regular components,* for which a hardware solution is preferred.

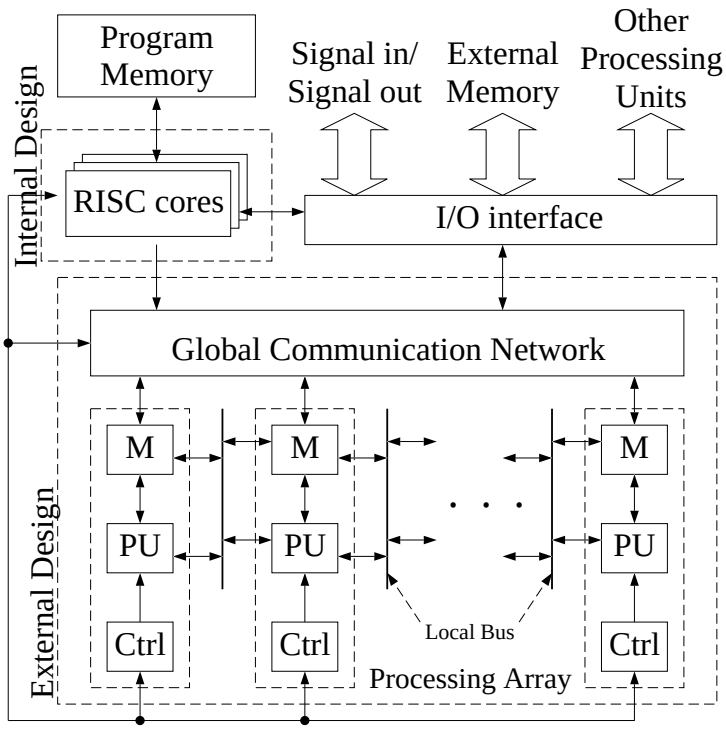


Figure 6.4: Proposed architectural style for high performance multimedia signal processing. There are two main components: (1) *processor arrays* to be used as the hardware accelerator for computationally intensive and regular components in an algorithm, and (2) *RISC cores* to be used as the programmable processor for complex but less computationally intensive components. M stands for the local memory. PU stands for the processing unit. Ctrl stands for the control unit.

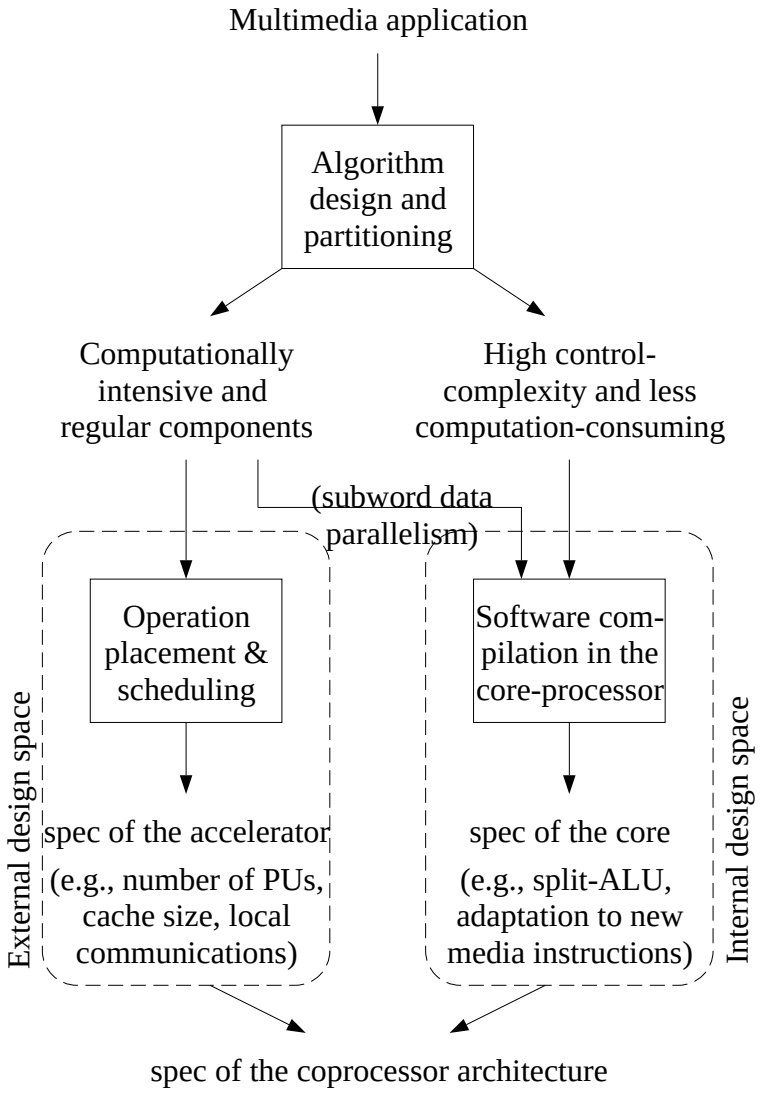


Figure 6.5: The proposed algorithm and architecture codesign approach for multimedia applications. In order to have an effective execution, given a specific application, the algorithm is first manually or semi-automatically divided into two parts: (1) computationally intensive and regular components, for which a hardware solution is preferred (e.g., motion estimation, DCT, IDCT), and (2) complex but less computationally intensive components, for which a software solution is preferred (e.g., VLC, VLD, rate control). From the results of the automatic operation placement and scheduling scheme, we can determine the spec of the accelerators, such as the number of PUs, the size of the datapath, the size of the local data memory. Combining the spec of the accelerators and the results of the core-processor adaptation, we can determine the final spec of the architecture.

2. *Complex but less computationally intensive components*, for which a software solution is preferred.

Computationally Intensive and Regular Components.

A systematic multimedia signal processing mapping method can facilitate design of processor arrays for computationally intensive and regular components. Since massive parallel and pipelined computing engines can provide high computational power for regular, intensive operations, various formal systematic procedures for systolic designs of many classes of algorithms have been proposed [60]. These transfer the computationally intensive, regular operations into simple processing elements, each with a fixed, data-independent function, along with a one- or two-dimensional nearest-neighbor communication pattern. These are the basic components of our design methodology for the multimedia signal processing system.

One major design objective is to make sure that the speed of the external memory keeps up with the speed of the processing engine. As shown in Section 6.1.2, the proposed approach is to fully exploit the frequent read-after-read data dependence (i.e., transmittent data) [16, 17]. By exploiting the locality, our allocation and scheduling reduces the communication-to-computation ratio, and hence reduces the amount of the external memory access/communication. The performance is enhanced, since the contention problem on the global communication network can be substantially alleviated.

In short, this architecture adopts systolic-type communication to speed up the computation since localized communication is faster. Moreover, this architecture reduces power consumption because it (1) segments global communication in local buses, (2) provides local, dedicated connection links, and (3) distributes control logics to individual PUs.

Complex but Less Computationally Intensive Components.

The complex but less computationally intensive components (e.g., controlling, data-dependent tasks) are supported by the software solution on RISC cores. Minor modification

to improved multimedia processing algorithms can be achieved by software updates. For example, different video coding standards can be implemented using the same hardware.

6.1.2 Systematic Operation Placement and Scheduling Method

In Appendix A, we present a systematic operation placement and scheduling method that can facilitate the design of processor arrays for computationally intensive and regular components. In this section, we briefly review the multimedia signal processing mapping method presented in Appendix A for readers' convenience.

In Section 6.1.1, we presented an architecture platform that can be configured to perform a variety of application-specific functionalities. The success of the proposed architectural platform depends on the efficient mapping of an application onto the target platform. For instance, to ensure an effective program, the cache locality is important because of the large speed gap between microprocessors and memory systems. It is also important to make use of local communication whenever possible, since it is cheaper, faster, and less power hungry than global communication.

Different data placement and operation scheduling would need different cache size requirement and global/local communication. We observe that although input dependence imposes no ordering constraints, input dependence does reveal the critical information on the data localities. To maximize the hit ratio of the caches, such information should be utilized for better data placement and operation scheduling by the parallel compilers. The proposed systematic code scheduling method has the following features:

1. Our multiprojection method deals with high-dimensional parallelism systematically. It can alleviate the burden of the programmer in coding and data partitioning.

2. It generates a fine-grain parallelism code that has low latency.

3. It exploits good temporary localities so that the utilization rate of caches is high.

4. It also exploits good spatial localities, which are good for new parallel architectures where localized communication is cheaper than global communication (cf. Figure 6.4).

6.2 Implementation of Block-Matching Motion Estimation Algorithm

Figure A.3 shows the pseudo code of the BMA of a single current block. We first concentrate on the first half, calculating the SADs (cf. Eq. (6.1)). Instead of viewing the BMA with only two-dimensional read-after-write data dependence, we consider that the BMA has four-dimensional read-after-read input dependence. Figure 6.6 shows the core in the 4D DG of the BMA for a current block. The operations of taking difference, taking absolute value, and accumulating residue are embedded in a four-dimensional space i, j, u, v. The following is the core in the 4D DG of the BMA:

Search window ($\vec{E}_1$)	$[1,0,-1,0]^T$	$D_4 = 0$
	$[0,1,0,-1]^T$	$D_4 = 0$
Current blocks ($\vec{E}_2$)	$[0,0,1,0]^T$	$D_4 = 0$
	$[0,0,0,1]^T$	$D_4 = 0$
Partial sum of SAD ($\vec{E}_3$)	$[1,0,0,0]^T$	$D_4 = 0$
	$[0,1,0,0]^T$	$D_4 = 0$

The indices i, j ($0 \leq i, j < n$) are the indices of the pixels in a current block. The u and v ($-p \leq u, v < p$) are the indices of the potential displacement vector. The actual DG would be a four-dimensional repeat of the same core.

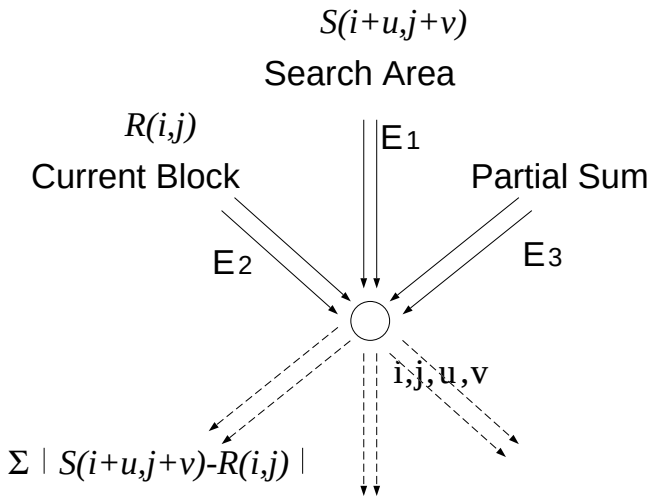


Figure 6.6: A core in the 4D DG of the BMA. There are $n \times n \times 2p \times 2p$ nodes in the DG. The node i, j, u, v represents the computation, `SAD[u, v] = SAD[u, v] + | s[i + u, j + v] - r[i, j] |`. $\vec{E}_1$ denotes the *read-after-read* data dependence of the search window. The $s[i+u, j+v]$ will be used repeatedly for (1) different i, j, (2) same $i+u$, and (3) same $j+v$. $\vec{E}_1$ is a two-dimensional reformable mesh. One possible choice is $[1, 0, -1, 0]^T$ and $[0, 1, 0, -1]^T$. The $r[i, j]$ will be used repeatedly for different u, v. Hence, $\vec{E}_2$, the data dependence of the current block, could be $[0, 0, 1, 0]^T$ and $[0, 0, 0, 1]^T$. The summation can be done in i-first order or j-first order. $\vec{E}_3$, which accumulates the difference, could be $[1, 0, 0, 0]^T$ and $[0, 1, 0, 0]^T$. The representation of the DG is not unique; most of the dependence edges can be redirected because of data transmittance. Although read-after-read data dependence is not "real" data dependence (does not affect the execution order of the operations), the read-after-read data dependence can identify memory and communication localities.

6.2.1 Multiprojecting the 4D DG of the BMA to a 1D SFG

From this 4D DG, we design a 1D SFG that can easily be implemented in the 1D processing array shown in Figure 6.2. First, we project the 4D DG along *v*, *u*, and *j* direction, using

$$\vec{d}_4 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad \vec{s}_4 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad P_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\vec{d}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \vec{s}_3 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \quad P_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\vec{d}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \vec{s}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad P_2 = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

To ensure processor availability, $M_4 = 2p$ and $M_3 = n$. Therefore,

$$\mathbf{A} = \mathbf{P_2 P_3 P_4} = [1,0,0,0] \tag{6.3}$$

$$\mathbf{S}^T = \vec{s}_2^T \mathbf{P_3 P_4} + M_3 \vec{s}_3^T \mathbf{P_4} + M_3 M_4 \vec{s}_4^T = [n+1,1,n,2pn] \tag{6.4}$$

Therefore, we have

Search window ($\vec{E}_1$)	Current blocks ($\vec{E}_2$)	Partial sum of SAD ($\vec{E}_3$)
1 ($D_1 = 1$)	0 ($D_1 = n$)	1 ($D_1 = 1+n$)
0 ($D_1 = 1-2pn$)	0 ($D_1 = 2pn$)	0 ($D_1 = 1$)

Because the edge $\vec{E}_1$, $[0,1,0,-1]^T$, has negative delay, we apply the redirection rule to it. Therefore, the new delay will be $(2pn-1)$. Because the edge $\vec{E}_3$, $[1,0,0,0]^T$, has too many units of delay, we apply the reformation rule to it so that the new delay would be 2 units. Note that the edge $\vec{E}_2$, $[0,0,0,1]^T$, and the edge $\vec{E}_2$, $[0,0,1,0]^T$, has the same

transmittent direction. In addition, the former is a multiple of the latter. Hence, the former can be eliminated. The final SFG becomes the following:

Search window ($\vec{E}_1$)	Current blocks ($\vec{E}_2$)	Partial sum of SAD ($\vec{E}_3$)
1 ($D_1 = 1$)	0 ($D_1 = n$)	1 ($D_1 = 2$)
0 ($D_1 = 2pn - 1$)		0 ($D_1 = 1$)

which can be visually seen in Figure 6.7.

6.2.2 Interpretation of the SFG

The search window data dependence, passed from PU_i to PU_{i+1}, has 1 unit of delay. As a result, we do not need a global broadcasting of the search window. Using local communication is faster and less power demanding.

In the mean time, the search window data dependence, passed from PU_i to itself, has $(2pn-1)$ units of delay. Consequently, we can expect that the same data will be reused after $(2pn-1)$ operations. Using a cache with size $(2pn-1)$ bytes is enough for this scheduling. For example, the cache size is 0.5 K-Bytes for $n = 16$ and $p = 16$. (Note that it is independent on the frame size.)

The reference data of the current block always stay in the same PU. There are 16 bytes for each PU. They can be put into either the cache or registers.

The summation of SAD, which is read-after-write data dependence, has two directions. The one which is inside PU_i itself has one unit of delay. That is, it is used immediately one after one to collect all of SAD in terms of loop j. The other one, which is passed from PU_i to PU_{i+1}, collects all of the partial sum of SAD in terms of loop i. Because it has two units of delay, the data passing is not synchronous. (The systolic implementation of the SFG is shown in Figure 6.8.)

The program can be divided into 4 parts:

1. First initialization loop, where there are no reference data of the current block and search window data in the PU, as shown in Figure 6.9.

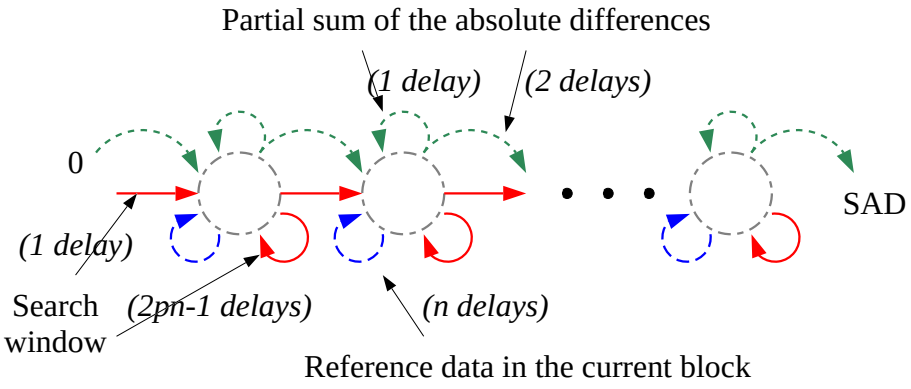


Figure 6.7: The SFG from multiprojecting the 4D DG of the BMA.

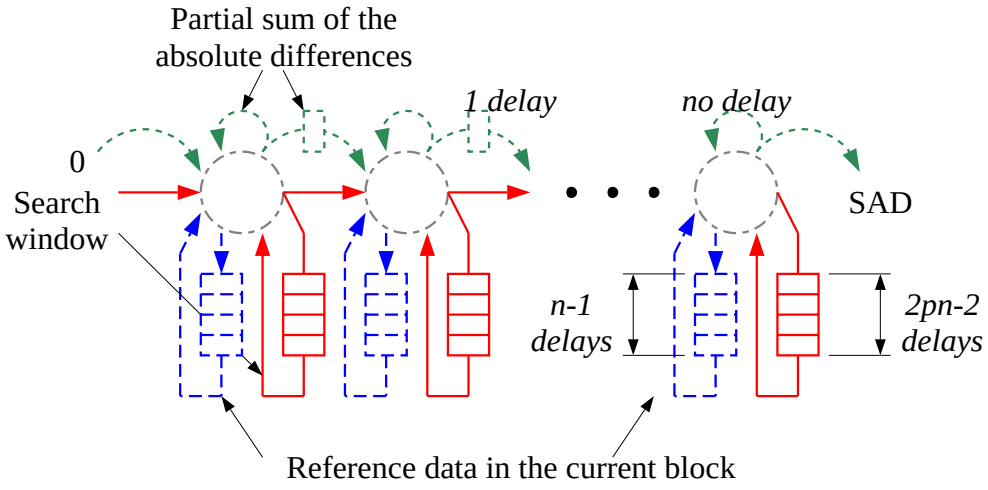


Figure 6.8: The systolic implementation of the SFG from multiprojecting the 4D DG of the BMA (cf. Figure 6.7).

PU_0	PU_1	PU_2	$\cdots$	PU_{n-1}
v=-p	- (idle)	- (idle)	-	- (idle)
u=-p	-	-	-	-
j=0	-	-	-	-
SAD[u,v]=0	-	-	-	-
get(s[i+u,j+v])	-	-	-	-
get(r[i,j])	-	-	-	-
SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	-	-	-	-
j=1	-	-	-	-
get(s[i+u,j+v])	-	-	-	-
get(r[i,j])	-	-	-	-
SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	-	-	-	-
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Figure 6.9: A "source-level" representation of the code assigned to the processor i ($0 \leq i < n$) during the initialization loops. Note that there is a `get(r[i,j])` from $j = 0$ to $j = n-1$ when $u = -p$. When there is a mark like get(r[i,j]), it denotes an external memory operation.

PU_0	PU_1	PU_2	$\cdots$	PU_{n-1}
send(SAD[u,v])	- (idle)	- (idle)	-	- (idle)
v=-p	- (idle)	-	-	-
u=-p+1	v=-p	-	-	-
j=0	u=-p	-	-	-
SAD[u,v]=0	j=0	-	-	-
get(s[i+u,j+v])	get(SAD[u,v])	-	-	-
send(s[i+u,j+v])	get(s[i+u,j+v])	-	-	-
SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	get(r[i,j])	-	-	-
j=1	SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	-	-	-
get(s[i+u,j+v])	j=1	-	-	-
send(s[i+u,j+v])	get(s[i+u,j+v])	-	-	-
SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	get(r[i,j])	-	-	-
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
v=-p	send(SAD[u,v])	-	-	-
u=-p+2	v=-p	-	-	-
j=0	u=-p+1	v=-p	-	-
SAD[u,v]=0	j=0	u=-p	-	-
get(s[i+u,j+v])	get(SAD[u,v])	j=0	-	-
send(s[i+u,j+v])	get(s[i+u,j+v])	get(SAD[u,v])	-	-
SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	send(s[i+u,j+v])	get(s[i+u,j+v])	-	-
j=1	SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	get(r[i,j])	-	-
get(s[i+u,j+v])	j=1	SAD[u,v]+=$\cdots$	-	-
send(s[i+u,j+v])	get(s[i+u,j+v])	j=1	-	-
SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	send(s[i+u,j+v])	get(s[i+u,j+v])	-	-
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Figure 6.10: A "source-level" representation of the code assigned to the processor i ($0 \leq i < n$) during the initialization loops. After $u > -p$, then `r[i,j]` can be loaded from the local cache. Also, because the next processor would require the data to be passed, the instruction `get(r[i,j])` is replaced by `send(s[i+u,j+v])`. When there is a mark like send(s[i+u,j+v]), it denotes a local bus transaction. Since the local buses are effectively used, there are only two external memory operations in each j loop in total.

PU_0	PU_1	PU_2	$\cdots$	PU_{n-1}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
v=-p+1	send(SAD[u,v])	$\vdots$	$\cdots$	$\cdots$
u=-p	v=-p+1	send(SAD[u,v])	$\cdots$	$\cdots$
j=0	u=-p	v=-p+1	$\cdots$	$\cdots$
SAD[u,v]=0	j=0	u=-p	$\cdots$	$\cdots$
SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	get(SAD[u,v])	j=0	$\cdots$	$\cdots$
j=1	SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	get(SAD[u,v])	$\cdots$	$\cdots$
SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	j=1	SAD[u,v]+=$\cdots$	$\cdots$	$\cdots$
j=2	SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	j=1	$\cdots$	$\cdots$
SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	j=2	SAD[u,v]+=$\cdots$	$\cdots$	$\cdots$
j=3	SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	j=2	$\cdots$	$\cdots$
$\vdots$	$\vdots$	SAD[u,v]+=$\cdots$	$\vdots$	$\vdots$
SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	j=n-3	$\vdots$	$\cdots$	$\cdots$
j=n-2	SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	j=n-3	$\cdots$	$\cdots$
SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	j=n-2	SAD[u,v]+=$\cdots$	$\cdots$	$\cdots$
j=n-1	SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	j=n-2	$\cdots$	$\cdots$
get(s[i+u,j+v])	j=n-1	SAD[u,v]+=$\cdots$	$\cdots$	$\cdots$
send(s[i+u,j+v])	get(s[i+u,j+v])	j=n-1	$\cdots$	$\cdots$
SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	send(s[i+u,j+v])	get(s[i+u,j+v])	$\cdots$	$\cdots$
send(SAD[u,v])	SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	send(s[i+u,j+v])	$\cdots$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Figure 6.11: A "source-level" representation of the code assigned to the processor i ($0 \leq i < n$) after the cache is full.

2. Initialization loops with cold caches, where there are reference data of the current block in the PU but there are no search window data in the PU, as shown in Figure 6.10.

3. Full-speed pipeline with few cache misses, where reference data and most of the search window data are in the cache. However, the very last search window datum is new (cf. Figure 6.11).

4. Full-speed pipeline with cache filled-up, where reference data and search window data are already in the cache (cf. Figure 6.12).

PU_i is one operation ahead of the PU_{i+1} in terms of j and one loop ahead in terms of u.

6.2.3 Implementation

For cache and communication localities, it is important to maximize the exploitation of read-after-read input dependence. Therefore, our multi-dimensional projection method for operation placement and scheduling is introduced.

Table 6.2 shows the comparison among several placement and scheduling results using 16 PUs. Our placement and scheduling result (obtained by multiprojecting the 4D DG of the BMA) reduces the amount of the external memory access by 103 times. With an 8-KB cache and 21 MB/sec local communication, the required bandwidth of the external memory access is more practical although this placement and scheduling takes more cycles (1.7%).

Moreover, when the data dependence between different reference blocks are taken into account, the dimension of the DG of the BMA is more than 4. Table 6.2 also shows that multiprojecting the 5D DG of the BMA reduces the amount of the external memory access by an additional 2.5 times, leading to approximately three orders of magnitude in reduction.

The proposed implementation of this computationally intensive and regular component of the BMA can achieve a speed-up ratio of 15.9, compared to a single processor implementation. After that, we concentrate on the second half of the BMA, determining the motion

Operation placement & scheduling	External memory access	Local communication per channel	Total cache size	Operations per block
Brute force without cache	3.1 GB/s	0	0	16384
Brute force with cache	64 MB/s	0	180 KB	16384
Ours from multiprojecting the 4D DG of the BMA	43 MB/s	33 MB/s	8 KB	16639
Ours from multiprojecting the 5D DG of the BMA	24 MB/s	27 MB/s	180 KB	16639

Table 6.2: Comparison between the operation placement and scheduling by the brute force method and our method (with frame size is 352×288, $p = 16$, $n = 16$, and 16 PUs). The parallelism is fully realized in the brute force method, and the number of operation cycles is minimized. However, the operation placement and scheduling can only work when an unusually high external memory bandwidth or a huge cache is provided. If the design does not exploit local communication and local caches, each pixel in the previous frame and the current frame will be read repeatedly $(2p)^2$ times. Hence, an extremely high external memory bandwidth is required. In order to capture the data reusability in the brute force design, each PU can use a local cache to store $(2p)$ lines of the previous frame and n pixels of current block. Consequently, the cache size is $(352$ (pixels/line) $\times$ 32 (lines/PU) + 16 (pixels/PU)$) \times 16$ PUs $=$ 180 KB, which is larger than the current state-of-the-art on-chip cache. Because the design does not use the local communication, each PU requests a copy of the previous frame independently, i.e., a pixel is read 16 times. Although the access to external memory is much less than that of the design without caches, it is still a large amount. In our designs, besides using a small compiler-directed cache [25, 48] to exploit the data reusability, the PUs use few cycles to exchange information via the local communication channels. Therefore, while using more cycles (1.7%), our designs use a small external memory bandwidth.

vector by the least SAD (cf. Eq. (6.2) and the pseudo code of the BMA of a single current block in Figure A.3). Since this part is control intensive but less computation-consuming (around 12.2 MOPS[‡]), it is easily supported by the software solution running on a RISC core.

6.3 Implementation of True Motion Tracking Algorithm

Because the true motion field is piecewise continuous, the motion of a feature block is determined by examining the directions of all its neighboring blocks. (Conventionally, the minimum SAD of a block of pixels is used to find the motion vector of the block in BMAs.) This allows a chance that a singular and erroneous motion vector may be corrected by its surrounding motion vectors (just like median filtering). Since the neighboring blocks may not have uniform motion vectors, a neighborhood relaxation formulation is used to allow some local variations of motion vectors among neighboring blocks:

$$\text{Score}(B_{x,y},\vec{v}) = \text{SAD}(B_{x,y},\vec{v}) + \sum_{B_{k,l}\in N(B_{x,y})} W(B_{k,l},B_{x,y}) \min_{\vec{\delta}}\{\text{SAD}(B_{k,l},\vec{v}+\vec{\delta})\}$$

where $B_{x,y}$ means a block of pixels whose motion we would like to determine, $N(B_{x,y})$ is the set of neighboring blocks of $B_{x,y}$, $W(B_{k,l},B_{x,y})$ is the weighting factor for different neighbors. A small $\vec{\delta}$ is incorporated to allow some local variations of motion vectors among neighboring blocks. The motion vector is obtained as

$$\textit{motion of } B_{x,y} = \arg\min_{\vec{v}}\{\text{Score}(B_{x,y},\vec{v})\}$$

This section demonstrates how to implement the TMT algorithm on the proposed architectural platform, which has a 16-PU processing array.

[‡]MOPS: million operations per second.

6.3.1 Algorithmic Partitioning of the True Motion Tracking Formulation

Although the formulation seems complicated, it can be divided into four steps as shown below. After that, each of them are regular for the hardware or software implementation.

Step 1.

We calculate the basic SADs as shown below:

$$\mathrm{SAD}[x,y,u,v] = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} | s[nx+i+u, ny+j+v] - r[nx+i, ny+j] | \tag{6.5}$$

where n is the block width and height, p is the absolute value of the maximum possible vertical and horizontal motion, indices x and y indicate the block position, $r[nx+i, ny+j]$ is the pixel intensity (pixel intensity) in the current block at coordinates (i, j), $s[nx+i+u, ny+j+v]$ is the pixel intensity in the search area in the previous frame, and (u,v) represents the candidate displacement vector.

For a frame with 352 pixels $\times$ 288 pixels, there are 22 blocks $\times$ 18 blocks because the block size is 16×16 ($n = 16$). That is, $0 \le x < 22$ and $0 \le y < 18$. As the search range p is 16, there are $32 \times 32 \times 16 \times 16 \times 2 = 524 \times 10^3$ additions for a block. For a P-frame, therefore, there are 208×10^6 additions, which must be finished within 1/30 of a second in a real-time application.

This step takes considerable computation and memory access. (In fact, it is the most computationally intensive part of the true motion tracker.) Fortunately, it is regular for parallel and pipeline processing. Section 6.2 shows an efficient implementation.

Step 2.

We calculate the minimum SADs after the $\vec{\delta}$-vibration:

$$\mathrm{mSAD}[x,y,u,v] = \min_{-1 \le \delta_u, \delta_v \le 1} \mathrm{SAD}[x,y,u+\delta_u, v+\delta_v] \tag{6.6}$$

where the vibration of the motion vector is limited within $\{-1,1\} \times \{-1,1\}$.

Each mSAD$[x,y,u,v]$ takes 9 operations (1 assignment and 8 comparisons) in Eq. (6.6). There are $32 \times 32 \times 22 \times 18$ such mSAD$[x,y,u,v]$ in a frame (within 1/30 seconds). Therefore, this step needs 109×10^6 operations per second.

Although this step takes less computation than the first step, a conventional programmable processor still has difficulty in supplying such a high computation demand. In Section 6.3.2, we will demonstrate how to implement this step on our processing array.

This computation requires a huge memory bandwidth. The second step reads the SAD array 109×10^6 times per second. There are also 97×10^6 read-after-write operations per second over the partial minimum. Without a good memory flow, the system design could be impractical (exorbitantly expensive to support a high memory bandwidth). Section 6.3.2 will address the memory flow design.

Step 3.

We calculate the Scores.

$$\begin{aligned} \text{Score}[x,y,u,v] &= \text{SAD}[x,y,u,v] + w\Big\{\text{mSAD}[x-1,y,u,v] + \text{mSAD}[x+1,y,u,v] \\ &\quad + \text{mSAD}[x,y-1,u,v] + \text{mSAD}[x,y+1,u,v]\Big\} \end{aligned} \tag{6.7}$$

where the neighborhood is the nearest four neighboring blocks. For simplicity, we make the $W(\cdot)$ depend only on the distance between the central block and the neighboring block [20]. Because these four neighbors are equi-distant from the central block, their weightings equal a constant w.

Each Score$[x,y,u,v]$ takes 5 operations in Eq. (6.7). There are $32 \times 32 \times 22 \times 18$ such Score$[x,y,u,v]$ in a frame (with 1/30 seconds). Therefore, this step needs 61×10^6 operations per second. Section 6.3.3 will demonstrate how to implement this step on our processing array.

Step 4.

We determine the motion vector by the least Score (cf. Eq. (6.2)):

$$\textit{motion of } B_{x,y} = \arg\min_{[u,v]}\{\text{Score}(x,y,u,v)\}$$

It takes 32×32 comparisons for each block. There are 406×10^3 comparisons per frame (1/30 seconds). Hence, this part is less computation-consuming (around 12 MOPS); it is easily supported by the software solution running on a RISC core.

6.3.2 Implementation of Calculating the mSAD

The 2D DG of calculating the mSAD.

As shown in Eq. (6.6), there are 6 independent indices $(x,y,u,v,\delta_u,\delta_v)$. Therefore, the DG of calculating the mSAD could be six-dimensional. However, there is no data dependence between different x and y. Therefore, the DG of calculating the mSAD is four-dimensional (u,v,δ_u,δ_v). In addition, because there is a high operation reusability in the Eq. (6.6), this task can be further divided into two sub-steps:

$$\begin{aligned}\text{pSAD}[x,y,u,v] &= \min_{\delta_u}\{\text{SAD}[x,y,u-\delta_u,v]\}\\ \text{mSAD}[x,y,u,v] &= \min_{\delta_v}\{\text{pSAD}[x,y,u,v-\delta_v]\}\end{aligned}$$

The DGs of the sub-steps are the same and two-dimensional. Figure 6.13 shows the DG, which is embedded in the 2D (u,δ_u) index space. Note that $-p \le u < p$ and $-1 \le \delta_u \le 1$.

There are two data-dependence edges in this DG. We use $\vec{E}_1$ to denote the read-after-read data dependence of the $\text{SAD}[x,y,u-\delta_u,v]$. The $\text{SAD}[x,y,u-\delta_u,v]$ will be used repeatedly for (1) different u, (2) same $u-\delta_u$. Therefore, one possible choice of the $\vec{E}_1$ is $[1,1]^T$. We use $\vec{E}_2$ to denote the read-after-write data dependence of the partial minimum. One possible choice is $[0,1]^T$. The algebraic representation of the 2D DG is shown below:

SAD ($\vec{E}_1$)	Partial min ($\vec{E}_2$)
$[1,1]^T$	$[0,1]^T$

Transform the 2D DG to a 3D DG.

The size of the 2D DG is 32×3 (assuming $p = 16$). The target architecture is a linear processing array of 16 PUs. Therefore, it is necessary to partition the DG/SFG and execute the SFG in a parallel and pipeline manner. After careful evaluation, we decide to apply the locally sequential globally parallel (LSGP) scheme in this implementation [16].

The first step in applying the LSGP is to transform the 2D DG to a 3D DG whose size is $2 \times 16 \times 3$. Two new indices a and b are introduced. One unit of the u is one unit of the a when the dependence edge does not move across different packing segments. (A packing segment consists of all the computation nodes within two units of sequential u. That is, the packing boundary occurs when 2 divides u.) One unit of the u is 1 unit of the b and -1 unit of the a when the dependence edge crosses the packing boundary of the transformed DG one time. It is obvious that $u = a + 2b$. The 3D DG is shown below:

SAD ($\vec{E}_1$)	Partial min ($\vec{E}_2$)
$[1,0,1]^T$	$[0,0,1]^T$
$[-1,1,1]^T$	

Multiprojecting the 3D DG into a 1D SFG.

We multiproject the 3D DG into a 1D SFG using the following:

$$\vec{d}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \vec{s}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad P_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\vec{d}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \vec{s}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad P_2 = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

To ensure processor availability, $M_3 = 2$. Therefore, we have the allocation matrix and scheduling vector as

$$\begin{aligned} \mathbf{A} &= \mathbf{P_2 P_3} = [0,1,0] \\ \mathbf{S}^T &= \vec{s}_2^T \mathbf{P_3} + M_3 \vec{s}_3^T = [1,0,2] \end{aligned}$$

and the 1D SFG as

SAD ($\vec{E}_1$)		Partial min ($\vec{E}_2$)	
0	($D_1 = 3$)	0	($D_1 = 2$)
1	($D_1 = 1$)		

Using an extremely small buffer (3 Bytes) with the help of local communication, the SAD can be used repeatedly without any extra external memory access. It is obvious that the partial minimum can be used in the same way.

Evaluation of the Implementation.

The allocation matrix and the scheduling vector give us not only the SFG (cf. Figure 6.14), but also some important features of the implementation:

1. **Execution cycles:** The computational time of a block is equal to the difference between the time of the first operation and the time of the last operation, i.e.,

   $$T = \max_{\underline{c}_x, \underline{c}_y} \left\{ \mathbf{S}^T (\underline{c}_x - \underline{c}_y) \right\} + 1$$

   where $\underline{c}_x$ and $\underline{c}_y$ are two computation nodes in the DG.

   In this particular implementation, we have $-16 \leq u \leq 15$, $u = a + 2b$, and $-1 \leq \delta_u \leq 1$. Hence, we have

   $$\begin{aligned} 0 \leq &\quad a &\quad \leq 1 \\ -8 \leq &\quad b &\quad \leq 7 \\ -1 \leq &\quad \delta_u &\quad \leq 1 \end{aligned}$$

   In addition, we do not perform any useful computation if $u + \delta_u < -16$ or $u + \delta_u > 15$. It can be easily shown that this implementation takes 6 cycles by a simple integer linear programming.

Note that there are $32 \times 3 = 96$ computation nodes in the DG. If the parallelism are fully realized on the 16 processors, then the shortest execution time should be $96/16 = 6$ cycles. That is, our implementation achieves the highest efficiency in term of computational time.

The computational time is 6 (cycles/line) $\times$ 32 (lines/block) $=$ 196 cycles/block for the first sub-step. The computational time is also 196 cycles/block for the second sub-step. The total time is 384 (cycles/block) $\times$ 22 (blocks/slice) $\times$ 18 (slices/frame) $= 152 \times 10^3$ cycles/frame. This step adds 4.6 MOPS for each PU.

2. **Memory size:** Because we must store the SAD for 3 cycles and the partial minimum for 2 cycles, the total amount of memory size is 5 bytes.

3. **Internal communication per channel:** Two PUs exchange 3 bytes using the local bus per 6 clock cycles. The internal communication is 76 KB per frame (1/30 seconds). Therefore, this step adds 2.3 MB to the total internal communication per second.

4. **External memory access:** There are 32-byte memory reads and 32-byte memory writes in each sub-step for a fixed u (or a fixed v). There are 64 (operations/line) $\times$ 32 (lines/substep) $\times$ 2 (substeps/block) $=$ 4096 external memory operations/block. Hence, this step adds 4096 (Bytes/block) $\times$ 22 (blocks/slice) $\times$ 18 (slices/frame) $\times$ 30 (frames/sec) $=$ 48 MB/sec external memory access to the requirement of the external communication bandwidth.

   As we mentioned before, without reusing the data, this step will take 206 MB/sec of external memory access. Because our design has special data flow from this operation placement and scheduling, it needs only 24% of that global communication bandwidth.

6.3.3 Implementation of Calculating the Score

The 4D DG of calculating the Score.

Eq. (6.7) can be written as the following:

$$\text{Score}[x,y,u,v] = \text{SAD}[x,y,u,v] + w \sum_{\delta_x=0}^{1} \sum_{\delta_y=0}^{1} \text{mSAD}[x-\delta_x-\delta_y+1, y-\delta_x+\delta_y, u, v]$$

Although there are 6 independent indices $(x,y,\delta_x,\delta_y,u,v)$, there is no data dependence between different u and v. Assuming that $\text{Score}[x,y,u,v] = \text{SAD}[x,y,u,v]$ initially, the DG of calculating the Score is four-dimensional (x,y,δ_x,δ_y). Figure 6.15 shows the core of the DG, which is embedded in the 4D (x,y,δ_x,δ_y) index space. Note that $0 \le x < 22$, $0 \le y < 18$, and $0 \le \delta_x, \delta_y \le 1$ (assuming the picture size is 352×288 and block size is 16×16).

There are two data-dependence edges in this DG. We use $\vec{E}_1$ to denote the read-after-read data dependence of the $\text{mSAD}[x-\delta_x-\delta_y+1, y-\delta_x+\delta_y, u, v]$. The $\text{mSAD}[x-\delta_x-\delta_y+1, y-\delta_x+\delta_y, u, v]$ will be used repeatedly for (1) different x, y, (2) same $x-\delta_x-\delta_y+1$, and (3) same $y-\delta_x+\delta_y$. Therefore, $\vec{E}_1$ is a two-dimensional reformable mesh. One possible choice is $[1,1,1,0]^T$ and $[1,-1,0,1]^T$. We use $\vec{E}_2$ to denote the read-after-write data dependence of the partial score. It is also a two-dimensional mesh. One possible choice is $[0,0,1,0]^T$ and $[0,0,0,1]^T$. The algebraic representation of the 4D DG is shown below:

SAD ($\vec{E}_1$)	Partial sum ($\vec{E}_2$)
$[1,1,1,0]^T$	$[0,0,1,0]^T$
$[1,-1,0,1]^T$	$[0,0,0,1]^T$

Transform the 4D DG to a 5D DG.

The size of the 4D DG is $22 \times 18 \times 2 \times 2$. The target architecture is a linear processing array of 16 PUs. Thus, it is necessary to partition the DG/SFG and execute the SFG in a parallel and pipeline manner. We decide to implement this step on 11 PUs using the LSGP scheme (cf. Section 6.3.2).

The first step in applying the LSGP is to transform the 4D DG to a 5D DG whose size is $2 \times 11 \times 18 \times 2 \times 2$ by introducing two new indices *a* and *b*. One unit of the *x* is one unit of the *a* when the dependence edge does not move across different packing segments. (A packing segment consists of all the computation nodes within two units of sequential *x*. That is, the packing boundary is when 2 divides *x*.) One unit of the *x* is 1 unit of the *b* and -1 unit of the *a* when the dependence edge crosses the packing boundary of the transformed DG one time. Note that $x = a + 2b$. The 5D DG is shown below:

SAD ($\vec{E}_1$)	Partial sum ($\vec{E}_2$)
$[1,0,1,1,0]^T$	$[0,0,0,1,0]^T$
$[-1,1,1,1,0]^T$	$[0,0,0,0,1]^T$
$[1,0,-1,0,1]^T$	
$[-1,1,-1,0,1]^T$	

Multiprojecting the 5D DG into a 1D SFG.

We multiproject the 5D DG into a 1D SFG using the following:

$$\vec{d}_5 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \vec{s}_5 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad P_5 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\vec{d}_4 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \vec{s}_4 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad P_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\vec{d}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \vec{s}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad P_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\vec{d}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \vec{s}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad P_2 = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

To ensure processor availability, $M_5 = 2$, $M_4 = 2$, and $M_3 = 2$. The resulted allocation matrix and scheduling vector will be:

$$\begin{aligned} \mathbf{A} &= \mathbf{P_2 P_3 P_4 P_5} = [0,1,0,0,0] \\ \mathbf{S}^T &= \vec{s}_2^T \mathbf{P_3 P_4 P_5} + M_3 \vec{s}_3^T \mathbf{P_4 P_5} + M_3 M_4 \vec{s}_4^T \mathbf{P_5} + M_3 M_4 M_5 \vec{s}_5^T = [1,0,8,4,2] \end{aligned}$$

Therefore, we have

SAD ($\vec{E}_1$)	Partial sum ($\vec{E}_2$)
0 $(D_1 = 13)$	0 $(D_1 = 4)$
1 $(D_1 = 11)$	0 $(D_1 = 2)$
0 $(D_1 = -5)$	
1 $(D_1 = -7)$	

Because $\vec{E}_2$ is a 2D summation mesh, we apply the summation rule to it (cf. Appendix A.3) so that all the delays of the $\vec{E}_2$ edges are equal to 2.

Because two of the $\vec{E}_1$ edges contain negative delays, we apply the redirection rule to them so as to have positive delays. Moreover, because $\vec{E}_1$ is 2D reformable read-after-read data dependence, we apply the reformation rule to it (cf. Appendix A.3). We let $[-1,1,1,1,0]^T$ become $[-1,1,1,1,0]^T + [1,0,-1,0,1]^T = [0,1,0,1,1]^T$ so that the delay of the $\vec{E}_1$ edge becomes 6. The final SFG becomes the following:

SAD ($\vec{E}_1$)	Partial sum ($\vec{E}_2$)
0 $(D_1 = 13)$	0 $(D_1 = 2)$
1 $(D_1 = 6)$	0 $(D_1 = 2)$
0 $(D_1 = 5)$	
-1 $(D_1 = 7)$	

Evaluation of the Implementation.

The final SFG (cf. Figure 6.16) gives us some important features of the implementation:

1. **Execution cycles:** By a simple integer linear programming, the computational time of fixed u and v is equal to

$$T = \max_{\underline{c}_x, \underline{c}_y} \left\{ \mathbf{S}^T (\underline{c}_x - \underline{c}_y) \right\} + 1 = 144$$

where $\underline{c}_x$ and $\underline{c}_y$ are two computation nodes in the DG.

Note that there are $22 \times 18 \times 4 = 1584$ computation nodes in the DG. If the parallelism is fully realized in the 16 PUs, then the shortest execution time should be $1584/16 = 99$ cycles. That is, our implementation is close to (but not is equal to) the lowest bound of the computational time because only 11 PUs are used in this design.

Since $-16 \leq u, v \leq 15$, the total number of cycles for a frame of picture is $144 \times 32 \times 32 = 147 \times 10^3$ cycles. This step adds 4.4 MOPS for each PU.

2. **Memory size:** Because we only need to store the mSAD for $13 + 5 = 18$ cycles and partial minimum for 2 cycles, the total amount of memory size is 20 bytes.

3. **Internal communication per channel:** Two PUs exchange 1 byte of information using the local bus per 4 cycles. The internal communication is 37 KB per frame (1/30 seconds). Therefore, this step adds 1.1 MB internal communication to the total internal communication.

4. **External memory access:** Because the local data memory and the local bus are exploited for data reusability, each of the mSADs and the SADs will be read only once. There are $32 \times 32 \times 22 \times 18 \times 3 = 1.22$ MB external memory operations per frame (including the write-back of the Score). Hence, this step adds 36 MB to the total external communication.

   Without reusing the data, this step will read each of the mSADs four times, read each SAD once, and write each Score once. That is to say, there will be 73 MB/sec of external memory access. Our design, with the new operation placement and scheduling, needs only 50% of that global communication bandwidth.

6.4 Summary of the Implementation

Because the multimedia applications are getting more and more computation-demanding as well as versatile, new multimedia signal processor must have high performance and high flexibility. Because more and more multimedia applications are running mobile and wireless devices, low cost, low power, and efficiency memory usage become emerging issues in new multimedia system design. Based on the algorithmic features of the multimedia applications and the available VLSI technology, we observe that the future multimedia signal processor will consist of two parts: *array processors* to be used as the hardware accelerator and *RISC cores* to be used as the programmable processor [18]. While some control-intensive functions can be implemented using programmable CPUs,

Step	Implementation on	MOPS per processor	Cache size	Internal Communication	External Communication
1	Processing array	198	8 KB	33 MB/sec	43 MB/sec
2	Processing array	5	80 B	2 MB/sec	48 MB/sec
3	Processing array	4	320 B	1 MB/sec	36 MB/sec
4	RISC core	12	-	-	12 MB/sec

Table 6.3: Implementation of the true motion tracking algorithm on the proposed architectural platform using our operation placement and scheduling scheme (with frame size is 352×288, $p = 16$, $n = 16$, and 16 PUs). The parallelism is almost fully realized. One of the most prominent features is that the design uses a small external memory bandwidth by exploiting small caches.

other computation-intensive functions can rely on hardware accelerators.

In order to achieve the maximum performance of the parallel and pipelined architecture, systematic methodology, like systolic design methods, which can partition and compile the algorithm is important. Because the gap between the processor speed and memory speed is getting larger and larger, the memory/communication bandwidth is the bottleneck in many systems. A sound operation placement and scheduling scheme should reveal an efficient memory usage. We propose an algebraic multiprojection methodology, capable of manipulating an algorithm with high-dimensional data dependence, to design the special data flow for highly reusable data.

The challenge for years has been to develop efficient systems of conventional block-matching algorithms due to their computational complexities. Because the TMT is even more computation-demanding and control-intensive than conventional block-matching algorithms, it is a greater challenge to have an effective system design of the TMT. In this chapter, the proposed TMT algorithm is partitioned into four parts for a high performance implementation. Table 6.3 gives a brief summary of the implementation of the true motion tracking algorithm. The first three parts of the TMT are computationally intensive and regular for the efficient implementation on the processing array. The last part is complex and less computation-demanding for the easy implementation on the core-processor.

PU_0	PU_1	PU_2	$\cdots$	PU_{n-1}
v=-p	send(SAD[u,v])	send(s[i+u,j+v]	$\cdots$	$\cdots$
u=-p+n	v=-p	send(SAD[u,v])	$\cdots$	$\cdots$
j=0	u=-p+n	v=-p	$\cdots$	$\cdots$
SAD[u,v]=0	j=0	u=-p+n	$\cdots$	$\cdots$
SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	get(SAD[u,v])	j=0	$\cdots$	$\cdots$
j=1	SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	get(SAD[u,v])	$\cdots$	$\cdots$
SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	j=1	SAD[u,v]+=$\cdots$	$\cdots$	$\cdots$
j=2	SAD[u,v]+=abs(s[i+u,j+v]-r[i,j])	j=1	$\cdots$	$\cdots$
$\vdots$	$\vdots$	SAD[u,v]+=$\cdots$	$\vdots$	$\vdots$

Figure 6.12: A "source-level" representation of the code assigned to the processor i ($0 \leq i < n$) after the cache is filled up. Note that after $v > -p$, there is no need for passing `s[i+u,j+v]` (except the very last one) because the `s[i+u,j+v]` is already in the cache. Because the cache is effectively used, there is only one external memory operation per u loop in total and there are only two local bus transactions per u loop per processor.

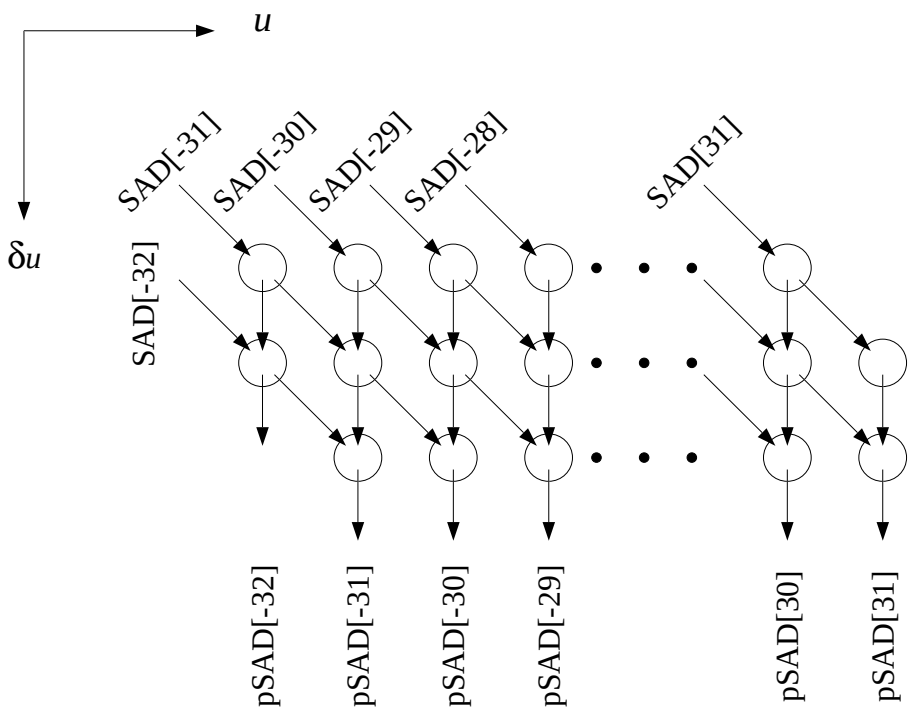


Figure 6.13: The 2D DG of the second step of the true motion tracker. There are two data-dependence edges in this DG. $\vec{E}_1$ denotes the read-after-read data dependence of the SAD$[x, y, u+\delta_u, v]$. The SAD$[x, y, u+\delta_u, v]$ will be used repeatedly for (1) different u, δ_u, and (2) same $u+\delta_u$. Therefore, one possible choice of $\vec{E}_1$ is $[1,1]^T$. $\vec{E}_2$, $[0,1]^T$, denotes the partial minimum data dependence.

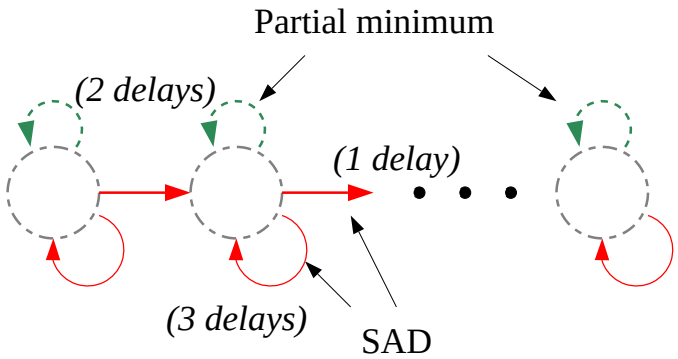


Figure 6.14: The 1D SFG of the second step of the true motion tracker (from multiprojecting 3D DG).

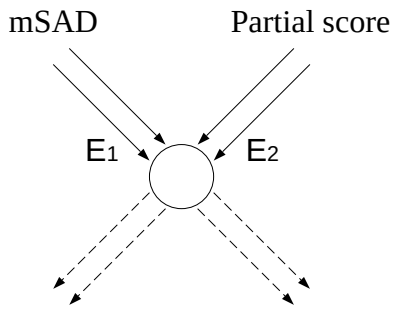


Figure 6.15: The 4D DG of the third step of the true motion tracker. There are two data-dependence edges in this DG. $\vec{E}_1$ denotes the read-after-read data dependence of the mSAD$[x-\delta_x-\delta_y+1, y-\delta_x+\delta_y, u, v]$. The mSAD$[x-\delta_x-\delta_y+1, y-\delta_x+\delta_y, u, v]$ will be used repeatedly for (1) different x, y, (2) same $x-\delta_x-\delta_y+1$, and (3) same $y-\delta_x+\delta_y$. $\vec{E}_1$ is a two-dimensional reformable mesh. One possible choice is $[1,1,1,0]^T$ and $[1,-1,0,1]^T$. $\vec{E}_2$ denotes the read-after-write data dependence of the partial score. It is a two-dimensional mesh as well. One possible choice is $[0,0,1,0]^T$ and $[0,0,0,1]^T$.

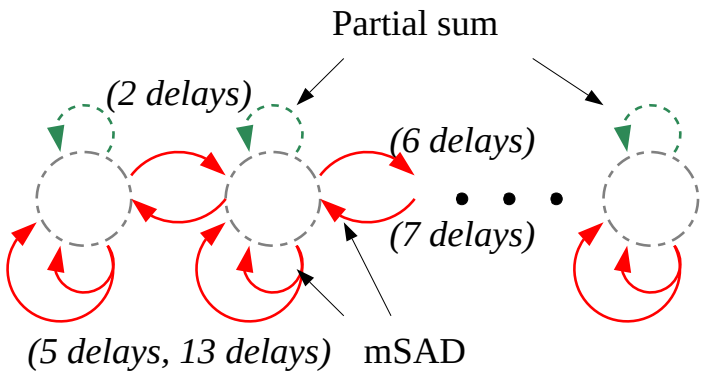


Figure 6.16: The 1D SFG of the third step of the true motion tracker (from multiprojecting 3D DG).

Chapter 7

Conclusions

This work has explored the theory of true motion tracking in digital video with respect to its applications. We have examined basic features of true motion estimation algorithms, and developed a new true motion tracker based on a neighborhood relaxation formulation. This true motion tracker has a number of advantageous properties when applied to motion analysis:

1. dependable tracking—the neighborhood helps to single out erroneous motion vectors
2. motion flexibility—the relaxation helps to accommodate non-translation motion
3. high implementation efficiency—99% of the computations are integer additions.

Consequently, it may be used as a cost-effective motion estimation algorithm for video coding, video interpolation, and video-object analysis. In terms of future research work, we shall continue on these fronts:

7.1 True Motion Tracker Analysis

Gradient-Based Motion Tracker with the Neighborhood- or Temporal-Constraint.

Matching-based motion trackers with the neighborhood-constraint and the temporal-constraint have shown great improvement over conventional matching-based techniques.

In the future, applying these two constraints with gradient-based measurement may also provide great improvement. By combining matching-based or gradient-based measurement with block-constraints, object-constraints, neighborhood-constraints, or temporal-constraints, we define a variety of motion tracking algorithms. Table 1.2 shows examples of how the motion estimation algorithms can be categorized. Since a gradient-based motion tracker with the neighborhood-constraint or the temporal-constraint does not exist, it would be an intriguing challenge to apply these two constraints with gradient-based measurement in the future.

Directional Confidence in Spatial-Neighborhood Weighting Factors.

To achieve higher correctness, in the future, spatial-neighborhood weighting factors should take into account the cross-correlation between vertical/horizontal components of the motion vector and image features.

Although a great deal of effort has gone into researching true motion tracking, some of the most noticeable failures of the TMT still occur in several common situations. For example, we have observed minor tracking errors on points along a long edge (see Figure 5.5).

A possible solution for avoiding this kind of tracking error is to adopt Anandan's scheme [7]: the weighting factor (in $W(\cdot)$ of Eq. (5.2)) takes into account the cross-correlation between vertical/horizontal components of motion vectors and image features. The purpose of such a correlation is that the motion of a block that has a low confidence in vertical movement can be guided by its neighbors that have a higher confidence in vertical movement.

Zero-Mean SAD.

For higher tracking correctness in non-motion brightness changes, we should consider removing the mean intensity of the image.

When there are non-motion brightness changes in images, finding accurate motion vectors becomes difficult. In the fundamental assumption, we assume that $I(x_i(t), y_i(t), t) = I_i$ and I_i is constant across time. However, this assumption could be proved incorrect in real images. For instance, when the lighting decreases over time (e.g., the sun goes behind a cloud), all of the intensities in the image will decrease. As another example, when the surface orientation of an object changes relative to the camera, the amount of light reflected toward the camera alters; and so the intensities will be different (see Figure 3.12). When the intensity of a pixel is no longer conserved over time,

$$I(x(t), y(t), t) = I(x(t+\Delta_t), y(t+\Delta_t), t+\Delta_t) + \Delta_I$$

This means that the fundamental equation (see Eq. (1.4)) for block-based and gradient-based measurement is invalid, i.e.,

$$s(\vec{p}_i, \vec{v}_i) \neq 0$$

Consequently, a motion estimation system that operates under the assumption of image brightness conservation will produce incorrect answers in these situations.

In order to tackle the problem of non-motion brightness changes, we suggest an approach—zero-mean SAD—which takes out the mean of the images before applying the fundamental measurement $s(\vec{p}_i, \vec{v}_i)$.

Probability Motion Vectors.

Instead of describing the motion of a block with a single motion vector, it may be better to use a probability density function to describe the motion of the block.

Nearly all previous techniques for analyzing motion attempt to compute a single motion at each point in space and time. However, it is difficult to describe motion fields in transparency regions using a single motion for each point. For example, a scene viewed through a piece of dirty glass will exhibit two motions at each location: that of the glass, and that of the scene behind it. In these situations, there is more than one motion occurring

within the local region. Such multiple-motion situations are prevalent, and cause problems for motion algorithms.

In [89], Simoncelli establishes a fundamental shift in the representation of visual motion. His new method computes probability distributions over the set of all possible image velocity vectors at a given spatio-temporal location. Viewing the problem probabilistically has many advantages:

1. It produces useful extensions of the standard quadratic gradient techniques for computing motion flow.

2. It provides 2D confidence information, allowing later stages of processing to tailor their use of the velocity estimates according to the shape of the distribution.

3. It provides a framework for "sensor fusion," in which image velocity estimates must be combined with information derived from other uncertain sources.

So, it seems worthwhile to investigate the probability motion estimation algorithm.

Exploiting Inter-Object Motion Information.

Intra-object and inter-object motion modeling should be considered for more accurate object motion estimation. In previous object motion estimation and motion-based video-object segmentation schemes, intra-object information is well exploited. For example, we always assume that the motion vectors within a single object have a good degree of similarity. However, there is also a rich body of inter-object information. For example, a point belonging to an object would not belong to another object and the motion of different objects should also be different.

7.2 Some Promising Application-Domains

True Motion for Error Concealment.

We can also make use of true motion vectors for better error concealment. Error concealment is intended to recover losses due to channel noise (e.g., bit-errors in noisy channels, cell-loss in ATM networks) by utilizing available picture information. Error concealment techniques can be categorized two ways according to the roles that the encoder and decoder play in the underlying approaches. Forward error concealment includes methods that add redundancy at the source end to enhance the error resilience of coded bit streams. For example, in MPEG-2, I-picture motion vectors were introduced to improve error concealment. However, syntax changes are required. Error concealment by post-processing refers to operations at the decoder to recover the damaged areas based on characteristics of image and video signals.

Our scheme is a post-processing error concealment scheme using motion-based temporal interpolation for damaged image regions. We use a true motion estimation algorithm at the encoder. In our work, the syntax is not changed and thus no additional bits are required. Using true motion vectors for video coding can even optimize the bit rate for residual and motion information. As shown in Chapter 3, using true motion vectors for coding offers significant improvement in motion-compensated frame-rate up-conversion over the minimal-residue BMA. The more accurate the motion estimation, the better the performance of frame-rate up-conversion. Because the error concealment problem is similar to the frame-rate up-conversion problem when the error is the whole frame, we believe that the damaged image regions can be interpolated better with help from true motion vectors [21].

Optimizing Frame-Skipping Mode Decision.

We believe that an optimal frame-skipping mode decision should include another criterion: when there is not enough information inside the frame as compared to the previous

and the next frames. According to current video coding standards, the macroblock-skipping mode is used when (1) there is insufficient space in the encoder buffer, and (2) there is too little information inside the macroblock. On the other hand, in current standards, the frame-skipping mode is used only when there is inadequate space in the encoder buffer. In Chapter 3, we demonstrated how to temporally interpolate a frame. When there is no major movement in the scene, it is easy to interpolate a frame from its previous and next frames. That is, there is almost no major information in the scene. Using this criterion, we can skip the frame so that we have more encoder buffer space for future frames.

Reconstruct a High-Resolution Frame from a Low-Resolution Sequence of Frames.

It is natural to extend our true motion tracker to another application—extraction of a high-resolution frame from video sequences [46, 91]. In Chapter 4, we used one example to demonstrate that true motion is extremely helpful in motion-based spatio-temporal video interpolation. In other words, a good motion tracker with a generalized sampling theorem can successfully combine the information available from multiple time instances. Using the same principal, we will be able to reconstruct a high-resolution image from a number of video frames. This technology can be used not only in video processing but also in image processing. Assume we only have a 300-dpi* (low-resolution) image scanner. If we can scan the image four times with different known offsets, then, according to the generalized sampling theorem, we should be able to reconstruct the 600-dpi (high-resolution) image. Since it is extremely difficult to control the offset less than 1/300 inches, we can first scan the images four times with unknown offsets and then use the TMT to measure the offsets afterward.

*dpi: dots per inch.

Object-Based Video Coding.

Object-based video coding techniques have become more and more important in recent years, e.g., in MPEG-4 [2] and MPEG-7 [4]. Chapter 5 showed that the object motion can be easily obtained once true motion vectors have been obtained. It is natural to utilize a global motion compensation scheme to gain further improvements in coding efficiency. Moreover, object-based coding techniques and block-based coding techniques can be adopted in tandem [19].

7.3 Implementation Considerations

So far, we have focused on the implementation of the TMT on a fixed-scheduling fine-grain parallel architecture. There are many other possible multimedia processor architectures that are yet to be explored. For example, in an out-of-order execution processor, the hardware rearranges the instruction execution to reduce the stalls when dependences are present [83]. Such a scheme can dynamically schedule some instructions whose dependences are unknown at compile time (e.g., they may involve a memory reference), and it simplifies the compiler. Another example: a simultaneous multithreading processor, an extension to current VLIW and superscalar processors, permits issues of multiple instructions from several independent threads [39]. Such effective exploitation of fine-grain instruction-level parallelism and coarse-grain thread-level parallelism leads to an increasing utilization of processor resources. Both architectures provide advantages that the fixed-scheduling fine-grain parallel architecture does not have, and so allow the algorithmic design of the TMT to use some operations that are not allowed or are less efficient in the fixed-scheduling fine-grain parallel architecture. For instance, an algorithm that has an effective system implementation on these kinds of architectures may use more control-intensive but less computation-demanding operations or use more floating-point operations. In this case, for better system implementations, the algorithmic design of TMT must be revised.

Appendix A

Systematic Operation Placement and Scheduling Scheme

In order to achieve the maximum performance of the parallel and pipelined architecture, systematic methodology that can partition and compile the algorithm is important (like systolic design methods). Because the gap between the processor speed and memory speed is growing larger and larger, the memory/communication bandwidth is the bottleneck in many systems. A sound operation placement and scheduling scheme should reveal an efficient memory usage. In this appendix, we describe our algebraic multiprojection methodology, which can manipulate an algorithm with high-dimensional data dependence and can design the special data flow for highly reusable data.

A.1 Systolic Processor Design Methodology

Several useful transformation techniques have been proposed for mapping the algorithm into parallel and/or pipeline VLSI architecture [60]. There are 3 stages in common systolic design methodology: the first is dependence graph (DG) design, the second is mapping the DG to a signal flow graph (SFG), and the third is designing an array processor based on the SFG.

More precisely, a DG is a directed graph, $G =< V, E >$, which shows the dependence

of the computations that occur in an algorithm. Each operation will be represented as one node, $\underline{c} \in V$, in the graph. The dependence relation will be shown as an arc, $\vec{e} \in E$, between the corresponding operations. A DG can be also considered as the graphical representation of a single assignment algorithm. Our approach to the construction of a DG will be based on the space-time indices in the recursive algorithm: Corresponding to the space-time index space in the recursive algorithm, there is a natural lattice space (with the same indices) for the DG, with one node residing on each grid point. Then the data dependencies in the recursive algorithm may be explicitly expressed by the arcs connecting the interacting nodes in the DG, while its functional description will be embedded in the nodes. A high-dimensional looped algorithm will lead to a high-dimensional DG. For example, the BMA for a single current block is a four-dimensional recursive algorithm [111].

A complete SFG description includes both functional and structural description parts. The functional description defines the behavior within a node, whereas the structural description specifies the interconnection (edges and delays) between the nodes. The structural part of an SFG can be represented by a finite directed graph, $G =< V, E, D(E) >$ since the SFG expression consists of processing nodes, communicating edges, and delays. In general, a node, $\underline{c} \in V$, represents an arithmetic or logic function performed with zero delay, such as multiplication or addition. The directed edges $\vec{e} \in E$ model the interconnections between the nodes. Each edge $\vec{e}$ of E connects an output port of a node to an input port of some node and is weighted with a delay count $D(\vec{e})$. The delay count is determined by the timing and is equal to the number of time steps needed for the corresponding arcs. Often, input and output ports are refereed to as sources and sinks, respectively.

Since a complete SFG description should include both functional description (defines the behavior within a node) and structural description (specifies the interconnection—edges and delays—between the nodes), we can easily transform an SFG into a systolic array, wavefront array, SIMD, or MIMD. Therefore, most research has been on how to transfer a DG to an SFG in the systolic design methodology.

There are two basic considerations for mapping from a DG to an SFG:

1. **Placement:** To which processors should operations be assigned? (A criterion might be to minimize communication/exchange of data between processors.)

2. **Scheduling:** In what ordering should the operations be assigned to a processor? (A criterion might be to minimize total computing time.)

Two steps are involved in mapping a DG to an SFG array. The first step is the processor assignment. Once the processor assignment is fixed, the second step is the scheduling. The allowable processor and schedule assignments can be quite general; however, in order to derive a regular systolic array, linear assignments and scheduling attract more attention.

Processor Assignment.

Processor assignment decides which processor is going to execute which node in the DG. A processor could carry out the operations of a number of nodes. For example, a projection method may be applied in which nodes of the DG along a straight line are assigned to a common processing element (PE). Since the DG of a locally recursive algorithm is regular, the projection maps the DG onto a lower dimensional lattice of points, known as the processor space. Mathematically, a linear projection is often represented by a projection vector $\vec{d}$. The mapping assigns the node activities in the DG to processors. The index set of nodes of the SFG are represented by the mapping

$$\mathbf{P} : I^n \rightarrow I^{n-1}$$

where I^n is the index set of the nodes of the DG, and I^{n-1} is the Cartesian product of (n-1) integers. The mapping of a computation $\underline{c}_i$ in the DG onto a node $\underline{n}$ in the SFG is found by:

$$\underline{n}(\underline{c}_i) = \mathbf{P}\underline{c}_i$$

where $\underline{n}(\cdot)$ denotes the mapping function from a node in the DG to a node in the SFG, and the projection basis $\mathbf{P}$, denoted by an $(n-1) \times n$ matrix, is orthogonal to $\vec{d}$. Mathematically,

$$\vec{d}^T \mathbf{P} = 0$$

This projection scheme also maps the arcs of the DG to the edges of the SFG. The set of edges $\vec{m}(\vec{e})$ into each node of the SFG is derived from the set of dependence edges $\vec{e}$ at each point in the DG by

$$\vec{m}(\vec{e}_i) = \mathbf{P}\vec{e}_i$$

where $\vec{m}(\cdot)$ denotes the mapping function from an edge in the DG to an edge in the SFG.

In this paper, bold face letters (e.g., **P**) represent matrices. Overhead arrows represent an n-dimensional vector, written as an $n \times 1$ matrix, e.g., $\vec{e}_i$ (a dependency arc in the DG) and $\vec{m}(\vec{e}_i)$ (an SFG dependency edge that comes for the $\vec{e}_i$). An n-tuple (a point in n-dimensional space), written as an $n \times 1$ matrix, is represented by underlined letters, e.g., $\underline{c}_i$ (a computation node in the DG) and $\underline{n}(\underline{c}_i)$ (an SFG computation node that comes from $\underline{c}_i$).

Scheduling.

A projection should be accompanied by a scheduling scheme, which specifies the sequence of the operations in all the PEs. A schedule function represents a mapping from the n-dimensional index space of the DG onto a 1D scheduling time space. A linear schedule is based on a set of parallel and uniformly spaced hyper-planes in the DG. These hyper-planes are called equi-temporal hyper-planes—all the nodes on the same hyper-plane must be processed at the same time. Mathematically, the schedule can be represented by a schedule vector (column vector) $\vec{s}$, pointing to the normal direction of the hyper-planes. The scheduling of a computation $\underline{c}$ in the DG on a node $\underline{n}$ in the SFG is found by:

$$T(\underline{c}) = \vec{s}^T \underline{c}$$

where $T(\cdot)$ denotes the timing function of a node in the DG to the execution time of the processor in the SFG.

The delay $D(\vec{e})$ on every edge is derived from the set of dependence edges $\vec{e}$ at each point in the DG by

$$D(\vec{e}_i) = \vec{s}^T \vec{e}_i$$

where $D(\cdot)$ denotes the timing function of an edge in the DG to the delay of the edge in the SFG.

Permissible Linear Schedules.

There is a partial ordering among the computations, inherent in the algorithm, as specified by the DG. For example, if there is a directed path from node $\underline{c}_x$ to node $\underline{c}_y$, then the computation represented by node $\underline{c}_y$ must be executed after the computation represented by node $\underline{c}_x$ is completed. The feasibility of a schedule is determined by the partial ordering and the processor assignment scheme.

The necessary and sufficient conditions are stated below:

1. $\vec{s}^T\vec{e} \geq 0$, for any dependence arc $\vec{e}$. $\vec{s}^T\vec{e} \neq 0$, for non-broadcast data.

2. $\vec{s}^T\vec{d} > 0$.

The first condition stands for data availability and states that the precedent computation must be completed before the succeeding computation starts. Namely, if node $\underline{c}_y$ depends on node $\underline{c}_x$, then the time step assigned for $\underline{c}_y$ cannot be less than the time step assigned for $\underline{c}_x$. The first condition means that the causality should be enforced in a permissible schedule. But, if a datum is used by many operations in the DG (read-after-read data dependencies), the causality constraint could be a trifle bit different. As popularly adopted, the same data value is broadcast to all the operation nodes. The data are called *broadcast data*. In this case, there is no delay required. Alternatively, the same data may be propagated step by step via local arcs without being modified to all the nodes. This kind of data, which is propagated without being modified, is called *transmittent data*. There should be at least one delay for transmittent data.

The second condition stands for processor availability, i.e., 2 computation nodes cannot be executed in the same time if they are mapped into the same processor element. The second condition implies that nodes on an equi-temporal hyper-plane should not be projected to the same PE. In short, the schedule is permissible if and only if (1) all the dependency

arcs flow in the same direction across the hyper-planes; and (2) the hyper-planes are not parallel with projection vector $\vec{d}$.

In general, the projection procedure involves the following steps:

1. For any projection direction, a processor space is orthogonal to the projection direction. A processor array may be obtained by projecting the index points to the processor space.

2. Replace the arcs in the DG with zero or nonzero delay edges between their corresponding processors. The delay on each edge is determined by the timing and is equal to the number of time steps needed for the corresponding arcs.

3. Since each node has been projected to a PE and each input (or output) data is connected to some nodes, it is now possible to attach the input and output data to their corresponding processors.

A.1.1 High Dimensional Algorithm

We discuss the concept of high-dimensional algorithms first. An algorithm is said to be n-dimensional if it has n-depth recursive loops in nature. For example, a block-matching algorithm for the whole frame is six-dimensional, as shown in Figure A.1. The indices x, y, u, v, i, j contribute 6D to the algorithm. As another example, a block-matching algorithm for the single block is four-dimensional, as shown in Figure A.3. The indices u, v, i, j contribute 4D to the algorithm.

It is important to respect the *read-after-read* data dependency. If a datum could be read time after time by hundreds of operations and those operations are put closely together, then a small cache can get rid of a large amount of external memory accesses. Since `s[x*n+i+u, y*n+j+v]` will be read time after time for different x, y, u, v, i, j combinations, this algorithm is 6D.

One the other hand, if we ignore the read-after-read data dependency, the DG has only two-dimensional *read-after-write* dependency based on variable `SAD`. Although the DG would become lower dimensional, it would be harder to track the data reusability and reduce the amount of memory accesses.

Transformation to Lower Dimension.

As shown in Figure A.2(a), two loops are folded into one loop to make the algorithm become less-dimensional [111].

The DG becomes three-dimensional because there are only three loop indices. The number of projections in multiprojection become less and it is easier to optimize the scheduling. However, in this modified algorithm, the operation regarding `(u,v+1)` must be executed directly after the operation regarding `(u,v)`. It makes the algorithm become less flexible. Efficient, expandable, and low I/O designs are harder to achieve. Besides, the folding of 6D DG will make it benefit less from some useful graph transformation, as shown in Appendix A.3.

Transformation to Higher Dimension.

We can also construct some artificial indices to make a lower-dimensional DG problem become higher-dimensional DG. For example, the inmost loop of the original algorithm could be modified as shown in Figure A.2(b).

The indices x, y, u, v, i, j_1, j_2 transform this algorithm into a seven-dimensional concept. This approach is not generally recommended because the number of steps for multiprojection increases in order to have the low-dimension design. However, this method provides an option of execution in the order of $j = \{1, N/2+1, 2, N/2+2, \ldots\}$ instead of $j = \{1, 2, \ldots, N/2, N/2+1, \ldots\}$ (simply exchanging the order of the $\mathtt{j}_1$ loop and the $\mathtt{j}_2$ loop). As we will see later in Appendix A.3.7, LSGP and LPGS partitioning can be carried out via multiprojection after a DG is transformed into an artificial higher-dimensional DG.

A.1.2 The Transformation of DG

In addition to the direction of the projection and the schedule, the choice of a particular DG for an algorithm can greatly affect the performance of the resulting array. The following are the two most common transformations of the DG seen in the literature:

- **Reindexing:**

  A useful technique for modifying the DG is to apply a coordinate transformation to the index space (called *reindexing*). Examples for reindexing are plane-by-plane shifting or circular shifting in the index space. For instance, when there is no permissible linear schedule or systolic schedule for the original DG, it is often desirable to modify the DG so that a permissible schedule may be obtained. The effect of this method is equivalent to the re-timing method [82].

- **Localized dependence graph:**

  A locally recursive algorithm is an algorithm whose corresponding DG has only local dependencies—all variables are (directly) dependent upon the variables of neighboring nodes only. The length of each dependency arc is independent of the problem size.

  On the other hand, a non-localized recursive algorithm has global interconnections/dependencies. For example, a same datum will be used by many operations, i.e., the same data value will repeatedly appear in a set of index points in the recursive algorithm or DG. As popularly adopted, the operation nodes receive the datum by broadcasting. The data are called *broadcast data* and this set is termed a broadcast contour. Such a non-localized recursive algorithm, when mapped onto an array processor, is likely to result in an array with global interconnections.

In general, global interconnections are more expensive than localized interconnections. In certain instances, such global arcs can be avoided by using a proper projection direction in the mapping schemes. To guarantee a locally interconnected array, a localized recursive algorithm would be derived (and, equivalently, a localized DG). In many cases, such broadcasting can be avoided and replaced by local communication. For example, in Figure A.5, the variable `s[u+i, u+j]` and `r[i, j]` in the inner three loops of the BMA (cf. Figure A.4) are replaced by local variables `S[u,v,i,j]` and `R[u,v,i,j]` respectively.

The key point is that instead of broadcasting the (public) data along a global arc, the same data may be propagated step by step via local arcs without being modified to all the nodes. This kind of data, which is propagated without being modified, is called *transmittent data*.

A.1.3 General Formulation of Optimization Problems

It takes more efforts to find an optimal and permissible linear scheduling than it does to find a permissible linear scheduling. In this section, we show how to derive an optimal design.

Optimization Criteria.

Optimization plays an important role in implementing systems. In terms of parallel processing, there are many ways to evaluate a design: one is to measure by the completion time (T), another is to measure by the product of the VLSI chip area and the completion time ($A \times T$) [67]. In general, the optimization problems can be categorized into:

1. To find a scheduling that minimizes the execution time, for given constraints on the number of processing units [115].

2. To minimize the cost (area, power, etc.) under certain given timing constraints [105].

In either case, such tasks are proved to be NP-hard. *In this paper, we focus on how to find an optimal schedule given an array structure—the timing is an optimization goal, not a constraint.*

Basic Formula.

First, we know that the computation time of a systolic array can be written as

$$T = \max_{\underline{c}_x, \underline{c}_y} \{\vec{s}^T (\underline{c}_x - \underline{c}_y)\} + 1$$

where $\underline{c}_x$ and $\underline{c}_y$ are two computation nodes in the DG.

The optimization problem becomes the following min-max formulation:

$$\vec{s}_{op} = \arg \left[\min_{\vec{s}} \left[\max_{\underline{c}_x, \underline{c}_y} \{\vec{s}^T (\underline{c}_x - \underline{c}_y)\} + 1 \right] \right]$$

under the following two constraints: $\vec{s}^T \vec{d} > 0$ and $\vec{s}^T \vec{e} > 0$, for any dependence arc $\vec{e}$.

The minimal computation time schedule $\vec{s}$ can be found by solving the proper integer liner programming [67, 108, 115] or quadratic programming [116].

A.1.4 Partitioning Methods

As DSP systems grow too complex to be contained in a single chip, partitioning is used to design a system into multi-chip architectures. In general, the mapping scheme (including both the node assignment and scheduling) will be much more complicated than the regular projection methods discussed in the previous sections because it must optimize chip area while meeting constraints on throughput, input/output timing and latency. The design takes into consideration I/O pins, inter-chip communication, control overheads, and tradeoff between external communication and local memory.

For a systematic mapping from the DG onto a systolic array, the DG is regularly partitioned into many blocks, each consisting of a cluster of nodes in the DG. As shown in Figure A.6, there are two methods for mapping the partitioned DG to an array: the locally sequential globally parallel (LSGP) method and the locally parallel globally sequential (LPGS) method [60].

For convenience of presentation, we adopt the following mathematical notations. Suppose that an n-dimensional DG is linear projected to an $(n-1)$-dimensional SFG array of size $L_1 \times L_2 \times \cdots \times L_{n-1}$. The SFG is partitioned into $M_1 \times M_2 \times \cdots \times M_{n-1}$ blocks, where each block is of size $Z_1 \times Z_2 \times \cdots \times Z_{n-1}$. $Z_i = L_i/M_i$ for $i \in \{1, 2, \cdots, n-1\}$,

Allocation.

1. In the LSGP scheme, one block is mapped to one PE. Each PE sequentially executes the nodes of the corresponding block. The number of blocks is equal to the number of PEs in the array, i.e., the array size equals the product $M_1 \times M_2 \times \cdots \times M_{n-1}$.

2. In the LPGS scheme, the block size is chosen to match the array size, i.e., one block can be mapped to one array. All nodes within one block are processed concurrently, i.e., locally parallel. One block after another block of node data is loaded into the array and processed in a sequential manner, i.e., globally sequential.

Scheduling.

In LSGP, after processor allocation, from the processor sharing perspective, there are $Z_1 \times Z_2 \times \cdots \times Z_{n-1}$ nodes in each block in the SFG, which share one PE. An acceptable (i.e., sufficiently slow) schedule is chosen so that at any instant there is at most one active PE in each block.

As to the scheduling scheme for the LPGS method, a general rule is to select a (global) scheduling that does not violate the data dependencies. Note that the LPGS design has the advantage that blocks can be executed one after another in a natural order. However, this simple ordering is valid only when there is no reverse data dependence for the chosen blocks.

Generalized Partitioning Method.

A unified partitioning and scheduling scheme is presented for LPGS and LSGP in [49]. The main contribution includes a unified partitioning model and a systematic two-level

scheduling scheme. The unified partitioning model can support LPGS and LSGP design in the same manner. The systematic two-level scheduling scheme can specify the intra-processor schedule and inter-processor schedule independently. Hence, a greater inter-processor parallelism can be effectively explored.

A general framework for processing mapping is also proposed in [94, 95].

Optimization for Partitioning.

The problem of finding an optimal (or reasonably small) schedule is an NP-hard problem. A systematic methodology for optimal partitioning is described in [113].

A.2 Multiprojection—Operation Placement and Scheduling for Cache and Communication Localities

Similar to systolic design approaches (as shown in Appendix A.1), we present a systematic multimedia signal processing mapping method that can facilitate the design of processor arrays for computationally intensive and regular components.

The key to success in a fixed-scheduling media processor (such as VLIW, SIMD) hinges on the success of the compiler. Similarly, the key components of the proposed implementation are the platform itself, and a compiler to map applications efficiently on the platform (especially, on the array processors). In this section, we present an operation placement and scheduling scheme for the array processors [16, 17]. The key advantages are twofold: (1) This multiprojection method, which deals with multidimensional parallelism systematically, can alleviate the burden of the programmer in coding and data partitioning. (2) It puts a lot of emphasis on cache localities and local communication in order to avoid the memory/communication bandwidth bottleneck, and can lead to faster program execution.

Conventional single projection can only map an n-dimensional DG directly onto an $(n-1)$-dimensional SFG. However, due to current VLSI technology constraint, it is hard to implement a 3D or 4D systolic array. Because the BMA for a single current block is a

four-dimensional algorithm (as shown in Appendix A.1.1), it is impossible to get a 2D or 1D system implementation by one projection. One possible approach is to decompose the BMA into subparts, which (1) are individually defined over index spaces with dimensions less than or equal to three and (2) are suitable to perform the canonical projection. For example, one such decomposition is to take u out first and consider it later as follows:

$$SAD[v] = \sum_{i=1}^{n} \sum_{j=1}^{n} \mid s[i, j+v] - r[i, j] \mid \qquad -p \leq v \leq p$$

Another possible approach is to map an n-dimensional DG directly onto an $(n-k)$-dimensional SFG without DG decomposition and hence a multi-dimensional projection method is introduced [60, 94, 95, 114]. The projection method, which maps an n-dimensional DG to an $(n-1)$-dimensional SFG, can be applied k times and thus reduces the dimension of the array to $n-k$. More elaborately, a similar projection method can be used to map an $(n-1)$-dimensional SFG onto an $(n-2)$-dimensional SFG, and so on. This scheme is called *multiprojection*.

Many design methods, such as, the *functional decomposition*, *index fixing*, and *slice and tile* [12, 31, 32, 59, 92], are the special cases of the multiprojection. Multiprojection can not only obtain the DGs and SFGs from functional decomposition but can also obtain other 3D DGs, 2D SFGs, and other designs that are difficult to obtain from other methods.

A.2.1 Algebraic Formulation of Multiprojection

The process of multiprojection could be written as a number of single projections using the same algebraic formulation as introduced in Appendix A.1. In this section, we explain how to project the $(n-1)$-dimensional SFG to an $(n-2)$-dimensional SFG. The potential difficulties of this mapping are (1) the presence of delay edges in the $(n-1)$-dimensional SFG, and (2) the delay management of the edges in the $(n-2)$-dimensional SFG.

Double-Projection.

For simplicity, we first introduce how to obtain a 2D SFG from a 4D DG by the multi-projection.

Step 1 We project the 4D DG into a 3D SFG by projection vector $\vec{d}_4$ (4×1 column vector), projection matrix $\mathbf{P}_4$ (3×4 matrix), and scheduling vector $\vec{s}_4$ (4×1 column vector) with three constraints: (1) $\vec{s}_4^T\vec{d}_4 > 0$, (2) $\mathbf{P}_4\vec{d}_4 = 0$, and (3) $\vec{s}_4^T\vec{e}_i \geq 0 \quad \forall i$.

The computation node $\underline{c}$ (4×1) in 4D DG will be mapped into the 3D SFG by

$$\begin{bmatrix} T_3(\underline{c}) \\ \underline{n}_3(\underline{c}) \end{bmatrix} = \begin{bmatrix} \vec{s}_4^T \\ \mathbf{P}_4 \end{bmatrix} \underline{c}$$

The data dependence edges will be mapped into the 3D SFG by

$$\begin{bmatrix} D_3(\vec{e}_i) \\ \vec{m}_3(\vec{e}_i) \end{bmatrix} = \begin{bmatrix} \vec{s}_4^T \\ \mathbf{P}_4 \end{bmatrix} \vec{e}_i$$

First, we claim that

$$D_3(\vec{e}_i) \neq 0 \quad \forall\, \vec{m}_3(\vec{e}_i) = 0 \tag{A.1}$$

For $\vec{m}_3(\vec{e}_i) = 0$, $\vec{e}_i$ is proportional to $\vec{d}_4$. For example, $\vec{e}_i = \alpha\vec{d}_4$ ($\alpha \neq 0$). The basic constraint $\vec{s}_4^T\vec{d}_4 > 0$ implies $\alpha\vec{s}_4^T\vec{d}_4 \neq 0$; therefore, $D_3(\vec{e}_i) = \vec{s}_4^T\vec{e}_i \neq 0$.

Step 2 We project the 3D SFG into a 2D SFG by projection vector $\vec{d}_3$ (3×1 column vector), projection matrix $\mathbf{P}_3$ (2×3 matrix), and scheduling vector $\vec{s}_3$ (3×1 column vector) with three constraints: (1) $\vec{s}_3^T\vec{d}_3 > 0$, (2) $\mathbf{P}_3\vec{d}_3 = 0$, and (3) $\vec{s}_3^T\vec{m}_3(\vec{e}_i) \geq 0\ \forall\vec{e}_i$ for broadcasting data. Or, $\vec{s}_3^T\vec{m}_3(\vec{e}_i) > 0\ \forall\vec{e}_i$ for non-broadcasting data.

The computation node $\underline{n}_3(\underline{c})$ (3×1) in the 3D SFG, which is mapped from $\underline{c}$ (4×1) in the 4D DG, will be mapped into the 2D SFG by

$$\begin{bmatrix} T_2'(\underline{c}) \\ \underline{n}_2(\underline{c}) \end{bmatrix} = \begin{bmatrix} \vec{s}_3^T \\ \mathbf{P}_3 \end{bmatrix} \underline{n}_3(\underline{c})$$

The data dependence edges in the 3D SFG will further be mapped into the 2D SFG by

$$\begin{bmatrix} D_2'(\vec{e}_i) \\ \vec{m}_2(\vec{e}_i) \end{bmatrix} = \begin{bmatrix} \vec{s}_3^T \\ \mathbf{P}_3 \end{bmatrix} \vec{m}_3(\vec{e}_i)$$

Step 3 We can combine the results from the previous 2 steps. Let allocation matrix $\mathbf{A} = \mathbf{P}_3\mathbf{P}_4$ and scheduling vector $\mathbf{S}^T = \vec{s}_3^T\mathbf{P}_4 + M_4\vec{s}_4^T$. ($M_4 \geq 1 + (N_4 - 1)\vec{s}_3^T\vec{d}_3$ where N_4 is the maximum number of nodes along the $\vec{d}_3$ direction in the 3D SFG.)

- Node mapping:

$$\begin{bmatrix} T_2(\underline{c}) \\ \underline{n}_2(\underline{c}) \end{bmatrix} = \begin{bmatrix} \mathbf{S}^T \\ \mathbf{A} \end{bmatrix} \underline{c}$$

  where $\underline{n}_2(\underline{c}) = \mathbf{A}\underline{c}$ means where the original computational node $\underline{c}$ is mapped. $T_2(\underline{c}) = \mathbf{S}\underline{c}$ means when the computation node is to be executed.

- Edge mapping:

$$\begin{bmatrix} D_2(\vec{e}_i) \\ \vec{m}_2(\vec{e}_i) \end{bmatrix} = \begin{bmatrix} \mathbf{S}^T \\ \mathbf{A} \end{bmatrix} \vec{e}_i$$

  where $\vec{m}_2(\vec{e}_i) = \mathbf{A}\vec{e}_i$ means where the original data dependency relationship is mapped. $D_2(\vec{e}_i) = \mathbf{S}^T\vec{e}_i$ means how much time delay should be in the edge $\vec{m}_2(\vec{e}_i)$.

Constraints for Data and Processor Availability.

Data Availability Theorem: *Every dependent datum comes from previous computation. To ensure data availability, every edge must have at least one unit of delay if the edge is not broadcasting some data.* $D_2(\vec{e}_i) = \mathbf{S}^T\vec{e}_i \geq 0$ *if* $\vec{e}_i$ *is for broadcasting data.* $D_2(\vec{e}_i) = \mathbf{S}^T\vec{e}_i > 0$ *if* $\vec{e}_i$ *is not for broadcasting data.*

Proof:

$$D_2(\vec{e}_i) \quad = \quad \mathbf{S}^T\vec{e}_i$$

$$
\begin{aligned}
&= (\vec{s}_3^T \mathbf{P}_4 + M_4 \vec{s}_4^T)\vec{e}_i \\
&= \vec{s}_3^T \mathbf{P}_4 \vec{e}_i + M_4 \vec{s}_4^T \vec{e}_i \\
&\geq \vec{s}_3^T \mathbf{P}_4 \vec{e}_i \\
&\quad (\text{from the constraint (3) in step 1}) \\
&> 0 \quad (\text{or, } \geq 0) \\
&\quad (\text{from the constraint (3) in step 2})
\end{aligned}
$$

□

Processor Availability Theorem. *Two computational nodes that are mapped into a single processor could not be executed at the same time. To ensure processor availability,* $T_2(\underline{c}_i) \neq T_2(\underline{c}_j)$ *must be satisfied for any* $\underline{c}_i \neq \underline{c}_j$ *and* $\underline{n}_2(\underline{c}_i) = \underline{n}_2(\underline{c}_j)$.

Proof: For any $\underline{n}_2(\underline{c}_i) = \underline{n}_2(\underline{c}_j)$, $\mathbf{P}_3\underline{n}_3(\underline{c}_i) - \mathbf{P}_3\underline{n}_3(\underline{c}_j) = 0$

$\Rightarrow \underline{n}_3(\underline{c}_i) - \underline{n}_3(\underline{c}_j)$ is proportional to $\vec{d}_3$.

$\Rightarrow \underline{n}_3(\underline{c}_i) - \underline{n}_3(\underline{c}_j) = \mathbf{P}_4(\underline{c}_i - \underline{c}_j) = \alpha\vec{d}_3$

Since N_4 is the maximum number of nodes along the $\vec{d}_3$ direction in the 3D SFG, $\alpha \in \{0, \pm 1, \pm 2, \ldots, \pm(N_4 - 1)\}$.

$$
\begin{aligned}
&T_2(\underline{c}_i) - T_2(\underline{c}_j) \\
&\quad = \mathbf{S}^T(\underline{c}_i - \underline{c}_j) \\
&\quad = (\vec{s}_3^T \mathbf{P}_4 + M_4 \vec{s}_4^T)(\underline{c}_i - \underline{c}_j) \\
&\quad = \vec{s}_3^T \mathbf{P}_4(\underline{c}_i - \underline{c}_j) + M_4 \vec{s}_4^T(\underline{c}_i - \underline{c}_j) \\
&\quad = \alpha \vec{s}_3^T \vec{d}_3 + M_4 \vec{s}_4^T(\underline{c}_i - \underline{c}_j)
\end{aligned}
$$

1. If $\mathbf{P}_4\underline{c}_i = \mathbf{P}_4\underline{c}_j$, then $\alpha = 0$ and

$$
\begin{aligned}
T_2(\underline{c}_i) - T_2(\underline{c}_j) &= M_4 \vec{s}_4^T(\underline{c}_i - \underline{c}_j) \\
&\neq 0 \quad (\text{by Eq. (A.1)})
\end{aligned}
$$

2. If $\mathbf{P}_4\underline{c}_i \neq \mathbf{P}_4\underline{c}_j$, then $\alpha \in \{\pm 1, \ldots, \pm(N_4-1)\}$

   (a) If $\vec{s}_4^T(\underline{c}_i - \underline{c}_j) = 0$, then

$$\begin{aligned} T_2(\underline{c}_i) - T_2(\underline{c}_j) &= \alpha \vec{s}_3^T \vec{d}_3 \\ &\neq 0 \quad \text{(by the basic constraint of step 2)} \end{aligned}$$

   (b) If $\vec{s}_4^T(\underline{c}_i - \underline{c}_j) \neq 0$, then by assuming $\vec{s}_4^T(\underline{c}_i - \underline{c}_j) > 0$ without losing generality, we have

$$\begin{aligned} & T_2(\underline{c}_i) - T_2(\underline{c}_j) \\ &= \alpha \vec{s}_3^T \vec{d}_3 + M_4 \vec{s}_4^T(\underline{c}_i - \underline{c}_j) \\ &\geq \alpha \vec{s}_3^T \vec{d}_3 + (1 + (N_4-1)\vec{s}_3^T \vec{d}_3)\vec{s}_4^T(\underline{c}_i - \underline{c}_j) \\ &= (\alpha + (N_4-1)\vec{s}_4^T(\underline{c}_i - \underline{c}_j))\vec{s}_3^T \vec{d}_3 + \vec{s}_4^T(\underline{c}_i - \underline{c}_j) \\ &\geq (\alpha + (N_4-1))\vec{s}_3^T \vec{d}_3 + \vec{s}_4^T(\underline{c}_i - \underline{c}_j) \quad (\text{because } \vec{s}_4^T(\underline{c}_i - \underline{c}_j) \geq 1) \\ &\geq 0 + \vec{s}_4^T(\underline{c}_i - \underline{c}_j) \quad (\text{because } \alpha + N_4 - 1 \geq 0) \\ &> 0 \end{aligned}$$

   If $\vec{s}_4^T(\underline{c}_i - \underline{c}_j) < 0$, then let $\underline{c}'_i = \underline{c}_j$ and $\underline{c}'_j = \underline{c}_i$. The condition $T_2(\underline{c}'_i) \neq T_2(\underline{c}'_j)$ for any $\underline{c}'_i \neq \underline{c}'_j$ and $\underline{n}_2(\underline{c}'_i) = \underline{n}_2(\underline{c}'_j)$ holds. So, the proof will.

Q.E.D. from 1, 2(a), and 2(b). □

Multiprojection *n*-Dimensional DG into *k*-Dimensional SFG.

Step 1 Let the *n*-dimensional SFG define as the *n*-dimensional DG. That is, $\underline{n}_n(\underline{c}_x) = \underline{c}_x$ and the $\vec{m}_n(\vec{e}_i) = \vec{e}_i$.

Step 2 We project the l-dimensional SFG into a $(l-1)$-dimensional SFG by projection vector $\vec{d}_l$ $(l \times 1)$, projection matrix $\mathbf{P}_l$ $((l-1) \times l)$, and scheduling vector $\vec{s}_l$ $(l \times 1)$ with basic constraint $\vec{s}_l^T \vec{d}_l > 0$, $\mathbf{P}_l \vec{d}_l = 0$, and $\vec{s}_l^T \vec{m}_l(\vec{e}_i) \geq$ (or $>$) $0 \forall \vec{e}_i$.

The computation node $\underline{c}_i$ $(l \times 1)$ and the data dependence edge $\vec{m}_l(\vec{e}_i)$ $(l \times 1)$ in l-dimensional SFG will be mapped into the $(l-1)$-dimensional SFG by

$$\underline{n}_{l-1}(\underline{c}_i) = \mathbf{P}_l \underline{n}_l(\underline{c}_i) \tag{A.2}$$

$$\vec{m}_{l-1}(\vec{e}_i) = \mathbf{P}_l \vec{m}_l(\vec{e}_i) \tag{A.3}$$

Step 3 After $(n-k)$ projections, the results can be combined. The allocation matrix will be

$$\mathbf{A} = \mathbf{P}_k \mathbf{P}_{k+1} \cdots \mathbf{P}_n \tag{A.4}$$

The scheduling vector will be

$$\begin{aligned} \mathbf{S}^T \;=\; & \vec{s}_{k+1}^T \mathbf{P}_{k+2} \mathbf{P}_{k+3} \cdots \mathbf{P}_n \\ & + M_{k+2}\, \vec{s}_{k+2}^T\, \mathbf{P}_{k+3} \mathbf{P}_{k+4} \cdots \mathbf{P}_n \\ & + M_{k+2} M_{k+3}\, \vec{s}_{k+3}^T\, \mathbf{P}_{k+4} \mathbf{P}_{k+5} \cdots \mathbf{P}_n \\ & \;\vdots \\ & + M_{k+2} M_{k+3} \cdots M_n\, \vec{s}_n \end{aligned} \tag{A.5}$$

where $M_l \geq 1 + (N_l - 1)\vec{s}_{l-1}^T \vec{d}_{l-1}$ and N_l is the maximum number of nodes along the $\vec{d}_{l-1}$ direction in the l-dimensional SFG. Therefore,

- Node mapping will be:

$$\begin{bmatrix} T_k(\underline{c}_i) \\ \underline{n}_k(\underline{c}_i) \end{bmatrix} = \begin{bmatrix} \mathbf{S}^T \\ \mathbf{A} \end{bmatrix} \underline{c}_i \tag{A.6}$$

- Edge mapping will be:

$$\begin{bmatrix} D_k(\vec{e}_i) \\ \vec{m}_k(\vec{e}_i) \end{bmatrix} = \begin{bmatrix} \mathbf{S}^T \\ \mathbf{A} \end{bmatrix} \vec{e}_i \tag{A.7}$$

Constraints for Processor and Data Availability.

If no transmittance property is assumed, every edge must have at least one delay because every dependent data is come from previous computation. It is easy to show that data availability is satisfied, i.e., $D_k(\vec{e}_i) > 0 \forall i$.

One can easily show processor availability is also satisfied., i.e., $T_k(\underline{c}_i) \neq T_k(\underline{c}_j)$ for any $\underline{c}_i \neq \underline{c}_j$ and $\underline{n}_2(\underline{c}_i) = \underline{n}_2(\underline{c}_j)$.

A.2.2 Optimization in Multiprojection

In this operation placement and scheduling scheme, the first step is to find an allocation **A** so that both of the following are satisfied. (1) A node in the SFG corresponds to one unique processor, i.e.,

$$\underline{n}_k(\underline{c}_i) \Rightarrow p_i \quad \forall \underline{n}_k(\underline{c}_i) \in SFG$$

(2) The amount of the global communication is minimized, i.e.,

$$\min_{\mathbf{A}}(\max_{\vec{e}_i}\{\mathbf{A}(\vec{e}_i)\}) \quad \forall \vec{e}_i \in DG$$

After projection directions are fixed, the structure of the array is determined. The remaining part of the design is to find a scheduling that (1) can complete the computation in minimal time and (2) can use a minimal-size cache under processor and data availability constraint, i.e.,

$$\begin{cases} \min_{\mathbf{S}}(\max_{\underline{c}_x,\underline{c}_y}\{\mathbf{S}^T(\underline{c}_x - \underline{c}_y)\}) & \forall \underline{c}_x, \underline{c}_y \in DG \\ \min_{\mathbf{S}}\{\sum_{\vec{e}_i}\mathbf{S}^T\vec{e}_i\} & \forall \vec{e}_i \in DG \end{cases}$$

A method using quadratic programming techniques is proposed to tackle the optimization problem [116]. However, it takes non-polynomial time to find the optimal solution. A polynomial-time heuristic approach, which uses the branch-and-bound technique and tries to solve the problem by linear programming, is also proposed [115].

Here, we propose another heuristic procedure to find a near optimal scheduling in our multiprojection method. In each single projection, from i-dimension to $(i-1)$-dimension, find an $\vec{s}_i$ by

$$\vec{s}_i = \arg\min_{\vec{s}} \left\{ \max_{\underline{n}_i(\underline{c}_x), \underline{n}_i(\underline{c}_y)} \left\{ \vec{s}^T \left[\underline{n}_i(\underline{c}_x) - \underline{n}_i(\underline{c}_y) \right] \right\} \right\} \quad \forall \underline{c}_x, \underline{c}_y \in \mathrm{DG} \tag{A.8}$$

under the following constraints:

1. $\vec{s}_i^T \vec{d}_i > 0$
2. $\vec{s}_i^T \vec{m}_i(\vec{e}_j) \geq 0 \; \forall j$ if $(i-1)$-dimension is not the final goal.
   $\vec{s}_i^T \vec{m}_i(\vec{e}_j) > 0 \; \forall j$ if $(i-1)$-dimension is the final goal.

This procedure will find a linear scheduling vector in polynomial time, when the given processor allocation function is linear. Although we have no proof of optimization yet, several design examples show our method can provide optimal scheduling when the DG is shift-invariant and the projections directions are along the axes. (Nevertheless, it will still be an NP-hard problem for all possible processor allocation and time allocation functions.)

A.3 Equivalent Graph Transformation Rules

In Appendix A.1.1 and Appendix A.1.2, some transformation rules of the DG are introduced. In order to have better designs, we also provide some graph transformation rules that can help us reduce the number of connections between processors, the size of buffer, or the power consumption. Table A.1 shows a brief summary of the rules.

A.3.1 Assimilarity Rule

As shown in Figure A.7, the assimilarity rule can save some links without changing the correctness of the DG. If a datum is transmitted to a set of operation/computation nodes in the DG/SFG by a 2D (or higher-dimensional) mesh, then there are several possible paths

```
for (x = 0; x < Nh; x++)
    for (y = 0; y < Nv; x++)
    {
        for (u = -p; u <= p; u++)
            for (v = -p; v <= p; v++)
            {
                SAD = 0;
                for (i = 1; i <= n; i++)
                    for (j = 1; j <= n; j++)
                        SAD = SAD + |s[x*n+i+u, y*n+j+v] - r[x*n+i, y*n+j]|;

                if (Dmin > SAD)
                {
                    Dmin = SAD;
                    MV[x, y] = [u, v];
                }
            }
    }
```

Figure A.1: The 6D BMA, where N_v is the number of current blocks in the vertical direction, N_h is the number of current blocks in the horizontal direction, n is the block size, and p is the search range. The indices x, y, u, v, i, j contribute 6D to the algorithm. The inner four loops are exactly those as shown in Figure A.3.

Rules	Apply to	Function	Advantages
Assimilarity	2D transmittent data	Keep only one edge and delete the others in the 2nd dimension	Save links
Summation	2D accumulation data	Keep only one edge and delete the others in the 2nd dimension	Save links
Degeneration	2D transmittent data	Reduce a long buffers to a single register	Save buffers
Reformation	2D transmittent data	Reduce a long delay to a shorter one	Save buffers
Redirection	Order independent data (e.g., transmittent or accumulation data)	Opposite the edge	Save problems on negative edges

Table A.1: Graph transformation rules for equivalent DGs. Note that the *transmittent data*, which are used repeatedly by many computation nodes in the DG (see Appendix A.1.2), play a critical role here.

```
for (a = 0; a < Nh * Nv; a++)
{
    x = a div Nv;
    y = a mod Nv;
    for (b = 0; b < (2*p+1) * (2*p+1); b++)
    {
        u = b div (2*p+1) - p;
        v = b mod (2*p+1) - p;
        for (c = 0; c < n * n; c++)
        {
            i = c div n + 1;
            j = c mod n + 1;
            .
            .
        }
    }
}
```

(a)

```
for (j1 = 0; j1 < 2; j1++)
    for (j2 = 1; j2 <= n / 2; j2++)
        SAD = SAD + | s[x*n+i+u, y*n+j1*n/2+j2+v] - r[x*n+i, y*n+j1*n/2+j2] |;
```

(b)

Figure A.2: (a) A 3D BMA that folds two loops in Figure A.1 into one loop. (b) On the other hand, a 7D BMA (x, y, u, v, i, j_1, j_2 7-dimension) can be constructed by modifying the inmost loop index `j` of the original algorithm into two indices $\mathtt{j}_1$ and $\mathtt{j}_2$.

```
for (u = -p; u < p; u++)
    for (v = -p; v < p; v++)
    {
        SAD[u, v] = 0;
        for (i = 0; i < n; i++)
            for (j = 0; j < n; j++)
                SAD[u, v] = SAD[u, v] + |s[i + u, j + v] - r[i, j] |;
    }

for (u = -p; u < p; u++)
    for (v = -p; v < p; v++)
        if (Dmin > SAD[u, v])
        {
            Dmin = SAD[u, v];
            MV = [u, v];
        }
```

Figure A.3: The pseudo code of the BMA for a single current block. In the process of the block-matching motion estimation, the current frame is divided into a number of non-overlapping current blocks, which are (n pixels) $\times$ (n pixels). Each of them will be compared with $2p \times 2p$ different displaced blocks in the search area of the previous frame. SAD is the sum of the absolute differences between the current block and the displaced block in the search area. The motion vector is the displacement that carries the minimal SAD.

```
for (u = -p; u < p; u++)
    for (v = -p; v < p; v++)
    {
      SAD[u, v, 0, n] = 0;
      for (i = 0; i < n; i++)
      {
          SAD[u, v, i, 0] = SAD[u, v, i - 1, n];
          for (j = 0; j < n; j++)
              SAD[u, v, i, j] = SAD[u, v, i, j-1] + |s[i+u, j+v] - r[i, j] |;
      }
    }

for (u = -p; u < p; u++)
    for (v = -p; v < p; v++)
      if (Dmin > SAD[u, v, n, n])
      {
          Dmin = SAD[u, v, n, n];
          MV = [u, v];
      }
```

Figure A.4: A single assignment code of the BMA for a single current block. This pseudo code is exactly the same as shown in Figure A.3. Every element in the `SAD[u, v, i, j]` array will be assigned a value only once—as the name come from.

```
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
    {
        R[u, -p-1, i, j] = r[i, j];
        S[u, -p-1, i, j] = s[u+i, -p-1+j];
    }

for (v = -p; v < p; v++)
{
    SAD[u, v, 0, n] = 0;
    for (i = 0; i < n; i++)
    {
        SAD[u, v, i, 0] = SAD[u, v, i - 1, n];
        for (j = 0; j < n; j++)
        {
            R[u, v, i, j] = R[u, v-1, i, j];
            S[u, v, i, j] = S[u, v-1, i, j+1];
            SAD[u,v,i,j] = SAD[u,v,i,j-1] + | S[u,v,i,j] - R[u,v,i,j] |;
        }
    }
}
```

Figure A.5: An example of the localized recursive BMA. The variables `s[u+i, u+j]` and `r[i, j]` in the inner three loop of the single assignment code shown in Figure A.4 are replaced by locally-interconnected array `S[u,v,i,j]` and `R[u,v,i,j]` respectively.

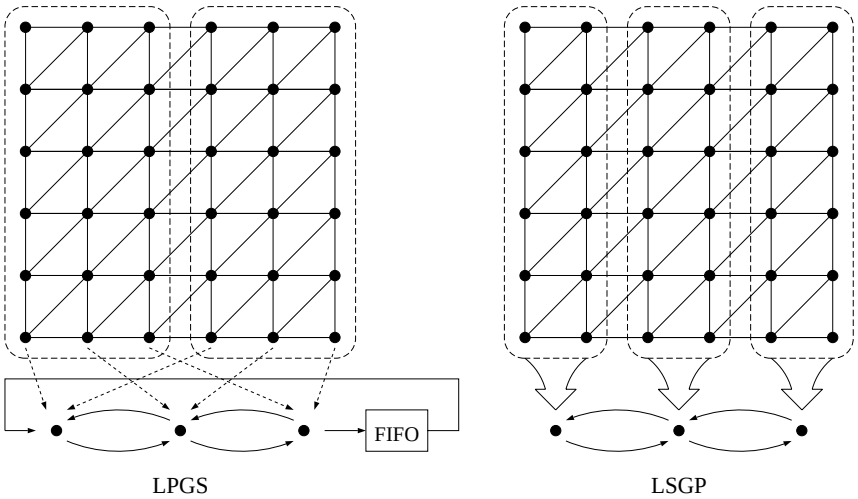


Figure A.6: There are two methods for mapping the partitioned DG to an array: locally parallel globally sequential (LPGS) and locally sequential globally parallel (LSGP).

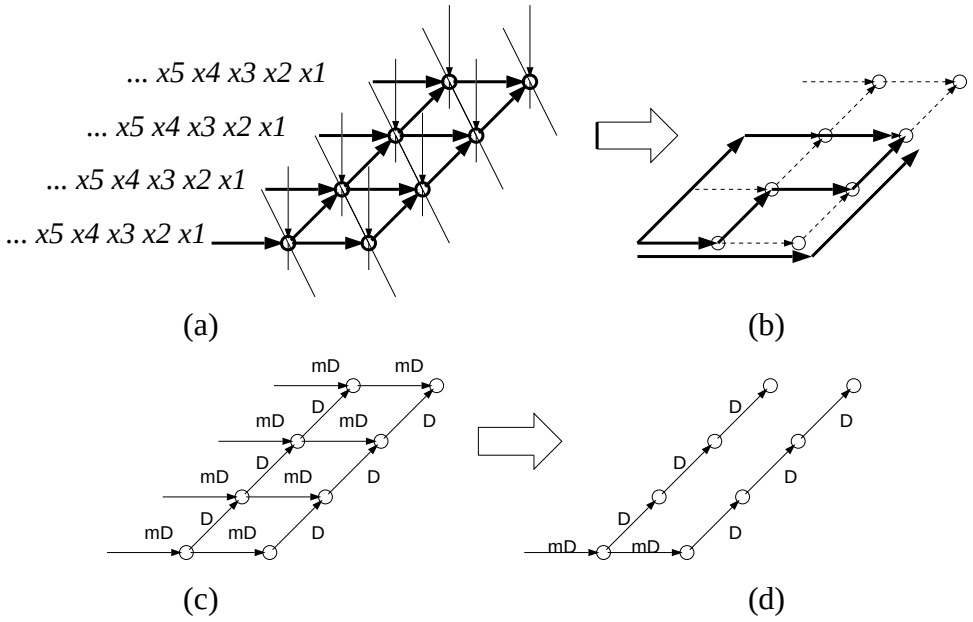


Figure A.7: (a) A high-dimensional DG, where a datum is transmitted to a set of nodes by the solid 2D mesh. (b) There are several paths via which the datum can reach a certain node. (c) During the multiprojection, the dependencies in different directions get different delays. (d) Because the data could reach the nodes by two possible paths, the *assimilarity rule* is applied to this SFG. Only one of the edges in the second dimension is kept. Without changing the correctness of the algorithm, a number of links and buffers are reduced.

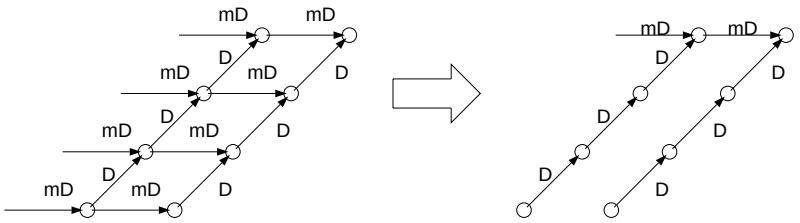


Figure A.8: (a) A datum is the summation of a set of nodes by a 2D mesh in an SFG. During the multiprojection, the dependencies in different directions get different delays. (b) Without changing the correctness of the algorithm, only one of the edges in the second dimension is kept. By the *summation rule*, a number of links and buffers are reduced.

via which the datum can reach a certain node. For example, in the BMA, the `s[i+u,j+v]` can be passed by `s[(i+1)+(u-1),j+v]` via loop *i*, or by `s[i+u,(j+1)+(v-1)]` via loop *j*. Keeping only one edge in the second dimension is sufficient for the data to reach everywhere.

The procedure of keeping only one edge for a set of edges can save a great number of interconnection buffers. Usually, this rule is applied after the final SFG is obtained. In this way, we can get rid of edges with longer delay and more edges.

One of the major drawbacks of this assimilarity rule is that every node must use the same set of data before this rule can be applied. It is not true for any algorithm that uses a 2D mesh to transmit the data. Generally speaking, the data set of a node greatly overlaps with the data set of the other nodes but not identically. In order to reduce the connection edges, we can make all the nodes process the same set of data artificially (i.e., ask the nodes to do some useless computations) and then apply this rule.

A.3.2 Summation Rule

As shown in Figure A.8, the summation rule can save some links without changing the correctness of the DG. Because summation is associative, the order of the summation can be changed. If output is obtained by aggregating a 2D (or higher-dimensional) mesh

of computational nodes, we can accumulate the partial sum in one dimension first, then accumulate the total from the partial sum in the second dimension afterward. For example, in the BMA, the SAD[u,v] is the 2D summation of $| s[i+u, j+v] - r[i, j] |$ over $1 \leq i, j \leq n$. We can accumulate the difference over index i first, or over index j first. We should calculate the data in the direction with fewer buffers first, then rigorously calculate the data in the other direction later.

A.3.3 Degeneration Rule

The degeneration rule reduces the data link when data are transmitted through a 2D (or higher-dimensional) mesh when (1) each node has its own data set and (2) the data sets of two adjacent nodes overlap each other significantly. One way to save the buffer is to let the overlapping data be transmitted from one dimension thoroughly (like that in the assimilarity rule) and let the non-overlapping data be transmitted from the other dimension(s) (unlike that in the assimilarity rule). In the second dimension, it is only necessary to keep non-overlapping data. Figure A.9 shows that only a register is required because the other data could be obtained by the other direction.

A.3.4 Reformation Rule

For 2D or higher-dimensional transmittent data, the structure of the mesh is not rigid. For example, in the BMA, the `s[i+u, j+v]` can be passed by `s[(i+k)+(u-k), j+v]` via loop i and by `s[i+u, (j+k)+(v-k)]` via loop j for $1 \leq k \leq n$. For a different k, the structure of the 2D transmittent mesh is different. The final delay in the designed SFG will be different. As a result, we should choose k, depending on the required buffer size. Generally speaking, the shorter the delay, the fewer the buffers.

For example, Figure A.10(a) shows a design after applying the assimilarity rule. Only a long delayed edge was left. Moreover, the data are transmitted to the whole array. So, we detour the long delayed edge, make use of the delay in the first dimension, and get the

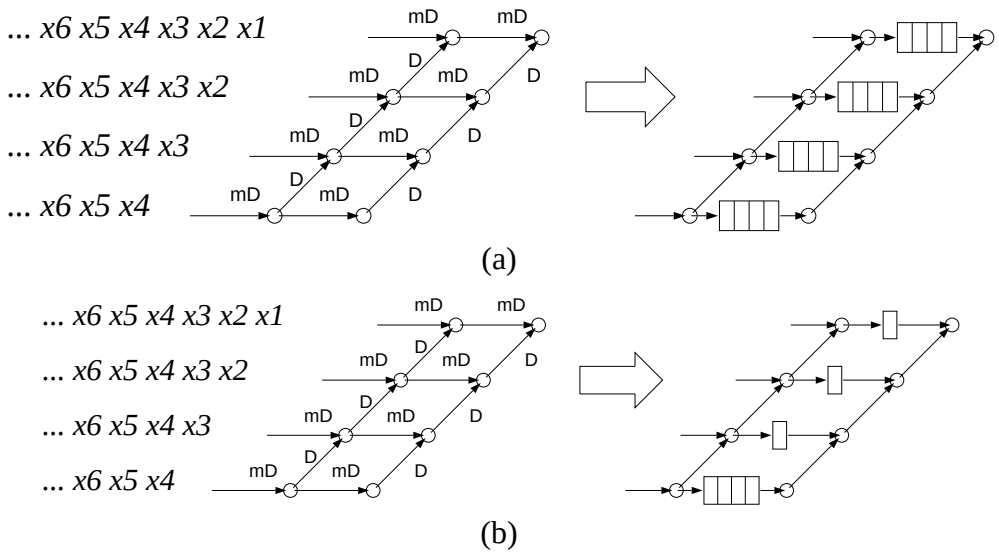


Figure A.9: (a) When transforming an SFG description to a systolic array, the conventional delay management uses $(m-1)$ registers for m units of delay on the links. (b) If the data sets of two adjacent nodes overlap each other, the *degeneration rule* suggests that only a register is required because the other data could be obtained by the other direction.

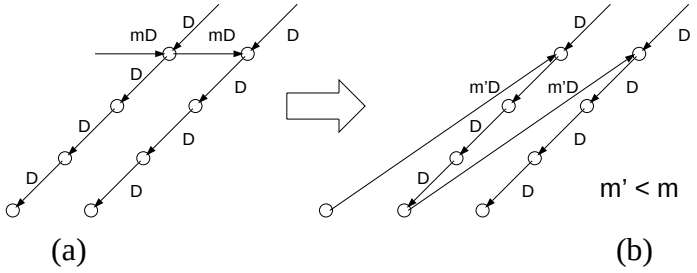


Figure A.10: (a) A high-dimensional DG, where a datum is transmitted to a set of nodes by a 2D mesh, is projected onto an SFG. During the multiprojection, the dependencies in different directions get different delays. Because the data could reach the nodes by more than two possible paths, the assimilarity rule is applied to this SFG. Only one of the edges in the second dimension is kept. (b) The delay (i.e., the number of buffers) could be further decreased when the *reformation rule* transforms the original 2D mesh into a tilted mesh.

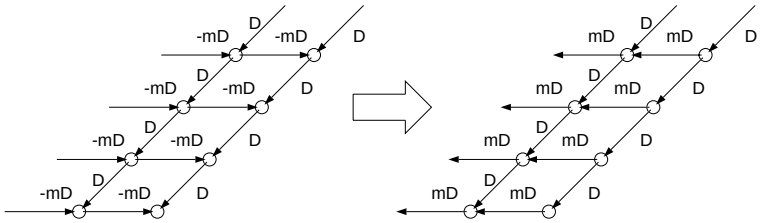


Figure A.11: (a) Generally speaking, an SFG with a negative delay is not permissible. (b) However, if the dependencies have no polarization, then we apply the *redirection rule* to direct the edges with negative delay to the opposite direction. After that, the SFG become permissible.

design shown in Figure A.10(b), where the longest delay is much shorter now.

A.3.5 Redirection Rule

Because some operations are associative (e.g., summation data, transmittent data), the arcs in the DG are reversible. The arcs are reversed to help the design. For example, the datum `s[(i+1)+(u-1), j+v]` is passed to `s[i+u, j+v]` via loop *i* in the BMA. After mapping the DG to a SFG, the delay on the edge is negative. Conventionally, negative delay is not allowed and we must find another scheduling vector $\vec{s}$. This rule tells us to move the data in the opposite direction (passing the `s[i+u, j+v]` to `s[(i+1)+(u-1), j+v]`) instead of re-calculating the scheduling vector (cf. Figure A.11).

A.3.6 Design Optimization vs. Equivalent Transformation Rules

All these rules do not modify the correctness of the implementation, but could accomplish some degree of design optimization.

1. The assimilarity rule and the summation rule have no influence on the overall calculation time. However, these two rules reduce the buffers and links. Generally speaking, these two rules are applied after the SFG is yielded.

2. The degeneration rule does not influence the overall calculation time. It is applied when one would like to transform the SFG into hardware design. It helps the reduction of the buffers and links. However, extra control logic circuits are required.

3. The reformation rule and the redirection rule will have influence on the scheduling problem because these two rules can make some prohibited scheduling vectors become permissible.

These rules help the design optimization but also make the optimization process harder. Sometimes, the optimization process will become a reiterative procedure that consists of (1) scheduling optimization and (2) equivalent transformation.

A.3.7 Locally Parallel Globally Sequential and Locally Sequential Globally Parallel Systolic Design by Multiprojection

In Appendix A.1.4, LPGS and LSGP were introduced briefly. In this section, we delineate a unified partitioning and scheduling scheme for LPGS and LSGP into our multiprojection method. The advantage of this unified partitioning model is that various partitioning methods can be achieved by choosing projection vectors. The systematic scheduling scheme can explore more inter-processor parallelism.

Equivalent Graph Transformation Rules for Index Folding.

A unified re-indexing method is adopted to fold original DG into a higher-dimensional DG but with a smaller size in a chosen dimension. Then, our multiprojection approach is applied to obtain the LPGS or LSGP designs. The only difference between LPGS and LSGP under our uniform approaches is the order of the projection. Our approach is even better in deciding the scheduling because our scheduling is automatically inherited from multiprojection scheduling instead of hierarchical scheduling.

Index Folding.

In order to map an algorithm into a systolic array by LPGS or LSGP, we propose a re-indexing method for the computational nodes into a higher-dimensional DG problem.

An example is shown in Figure A.12. We want to map a 2×6 DG into a smaller 2D systolic array. Let u, v be the indices ($0 \leq u \leq 1$, $0 \leq v \leq 5$) of the DG.

First, we will re-index all the computational nodes (u, v) into (u, a, b). The 2D DG becomes a 3D DG ($2 \times 2 \times 3$) where an a means 3 units of v, a b means 1 unit of v, and $0 \leq a \leq 1$, $0 \leq b \leq 2$. Then, a node at (u, a, b) in the 3D DG is equivalent to the node at $(u, (3a + b))$ in the original 2D DG.

After this, by multiprojection, we can have the following two partitioning methods:

1. **LPGS**

   If we project the 3D DG along the a direction, then the nodes that are close to each other in the v direction will be mapped into the different nodes. That is, the computation nodes are going to be executed in parallel. This is an LPGS partitioning.

2. **LSGP**

   If we project the 3D DG along b, then the nodes that are close to each other in the v direction will be mapped into the same node. That is, the computation nodes are going to be executed in a sequential order. This is an LSGP partitioning.

Note that we must be careful about the data dependency after transformation. One unit of original v will be 0 unit of a and 1 unit of b when the dependence edge does not move across different packing segments. (In the example, a packing segment consists of all the computation nodes within three units of sequential v. That is, the packing boundary is when 3 divides v.) One unit of the v is 1 unit of the a and -2 unit of the b when the dependence edge crosses the packing boundary of the transformed DG one time.

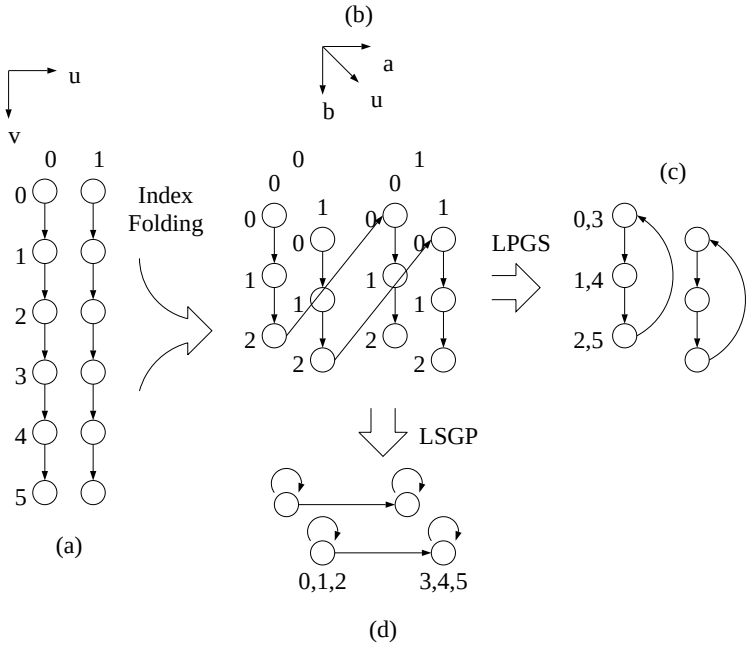


Figure A.12: (a) shows a 2×6 DG. (b) shows an equivalent $2 \times 3 \times 2$ DG after index folding. (c) an LPGS partitioning when we project the 3D DG along the a direction. (d) an LSGP partitioning when we project the 3D DG along the b direction.

Bibliography

[1] "H.263 Test Sequence." `ftp://bonde.nta.no/pub/tmn/qcif_source`.

[2] "MPEG-4 Requirements Document." ISO/IEC JTC1/SC29/WG11 Coding of Moving Pictures and Associated Audio MPEG98/W2194, Mar. 1998.

[3] "MPEG-4 Video Verification Model V7.0." ISO/IEC JTC1/SC29/WG11 Coding of Moving Pictures and Associated Audio MPEG97/N1642, Apr. 1997.

[4] "MPEG-7 Requirements Document." ISO/IEC JTC1/SC29/WG11 Coding of Moving Pictures and Associated Audio MPEG98/N2208, Mar. 1998.

[5] *Proc. of Int'l Solid-State Circuits Conference*, Feb 1997.

[6] "Will HDTV Be Must-See?" Time, p. 41., April 13 1998.

[7] P. Anandan, "A Computational Framework and an Algorithm for the Measurement of Visual Motion," *International Journal of Computer Vision*, vol. 2, no. 3, pp. 283–310, 1989.

[8] J. L. Barron, D. J. Fleet, and S. S. Beacuchemin, "*Systems and Experiment* Performance of Optical Flow Techniques," *International Journal of Computer Vision*, vol. 13, no. 1, pp. 43–77, 1994.

[9] M. Bierling, "Displacement Estimation by Hierarchical Block Matching," in *Proc. of SPIE Visual Communication and Image Processing*, vol. 1001, pp. 942–951, 1988.

[10] R. Castagno, P. Haavisto, and G. Ramponi, "A Method for Motion Adaptive Frame Rate Up-conversion," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 6, no. 5, pp. 436–446, Oct. 1996.

[11] J. Chalidabhongse and C.-C. J. Kuo, "Fast Motion Vector Estimation Using Multiresolution-Spatio-Temporal Correlations," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 7, no. 3, pp. 477–488, June 1997.

[12] S. Chang, J.-H. Hwang, and C.-W. Jen, "Scalable Array Architecture Design for Full Search Block Matching," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 5, no. 4, pp. 332–343, Aug. 1995.

[13] F. Chen, J. D. Villasenor, and D. S. Park, "A Low-Complexity Rate-Distortion Model for Motion Estimation in H.263," in *Proc. of IEEE Int'l Conf. on Image Processing*, vol. II, pp. 517–520, Sept. 1996.

[14] M. C. Chen and A. N. Willson, Jr., "Rate-Distortion Optimal Motion Estimation Algorithm for Video Coding," in *Proc. of IEEE Int'l Conf. on Acoustics, Speech, and Signal Processing*, vol. IV, pp. 2098–2111, May 1996.

[15] Y.-K. Chen and S. Y. Kung, "A Multi-Module Minimization Neural Network for Motion-Based Scene Segmentation," in *Proc. of IEEE Workshop on Neural Networks for Signal Processing*, (Kyoto, Japan), Sept. 1996.

[16] Y.-K. Chen and S. Y. Kung, "A Systolic Design Methodology with Application to Full-Search Block-Matching Architectures," *Journal of VLSI Signal Processing Systems*, vol. 19, no. 1, pp. 51–77, 1998.

[17] Y.-K. Chen and S. Y. Kung, "An Operation Placement and Scheduling Scheme for Cache and Communication Localities in Fine-Grain Parallel Architectures," in *Proc.*

of Int'l Symposium on Parallel Architectures, Algorithms and Networks, (Taipei, Taiwan), pp. 390–396, Dec. 1997.

[18] Y.-K. Chen and S. Y. Kung, "Multimedia Signal Processors: An Architectural Platform with Algorithmic Compilation," *Journal of VLSI Signal Processing Systems*, vol. 20, no. 1/2, Oct. 1998.

[19] Y.-K. Chen and S. Y. Kung, "Rate Optimization by True Motion Estimation," in *Proc. of IEEE Workshop on Multimedia Signal Processing*, (Princeton, NJ), pp. 187–194, June 1997.

[20] Y.-K. Chen, Y.-T. Lin, and S. Y. Kung, "A Feature Tracking Algorithm Using Neighborhood Relaxation with Multi-Candidate Pre-Screening," in *Proc. of IEEE Int'l Conf. on Image Processing*, vol. II, (Lausanne, Switzerland), pp. 513–516, Sept. 1996.

[21] Y.-K. Chen, H. Sun, A. Vetro, and S. Y. Kung, "True Motion Vectors for Robust Video Transmission," in *Proc. of SPIE Visual Communications and Image Processing*, Jan. 1999.

[22] Y.-K. Chen, A. Vetro, H. Sun, and S. Y. Kung, "Frame Rate Up-Conversion and Interlaced-to-Progressive Scan Conversion Using Transmitted True Motion," submitted to *1998 Workshop on Multimedia Signal Processing*, Dec. 1998.

[23] Y.-K. Chen, A. Vetro, H. Sun, and S. Y. Kung, "Optimizing INTRA/INTER Coding Mode Decisions," *ISO/IEC JTC/SC29/WG11 (Coding of Moving Pictures and Associated Audio) M2884*, Oct. 1997.

[24] Y.-K. Chen, A. Vetro, H. Sun, and S. Y. Kung, "Rate Optimization Based on True Motion," *ISO/IEC JTC/SC29/WG11 (Coding of Moving Pictures and Associated Audio) M2235*, July 1997.

[25] L. Chol, H.-B. Lim, and P.-C. Yew, "Techniques for Compiler-Directed Cache Coherence," *IEEE Parallel and Distributed Technology*, vol. 4, no. 4, pp. 23–34, Winter 1996.

[26] Chromatic Research, "Mpact 2 Media Processor Data Sheet." `http://www.mpact.com/tech/mpact2.pdf`, Feb. 1998.

[27] K.-W. Chun and J.-B. Ra, "An Improved Block Matching Algorithm Based on Successive Refinement of Motion Vector Candidates," *Signal Processing: Image Communication*, no. 6, pp. 115–122, 1994.

[28] G. de Haan and E. B. Bellers, "De-interlacing of video data," *IEEE Trans. on Consumer Electronics*, vol. 43, no. 3, pp. 819–824, Aug. 1997.

[29] G. de Haan, P. W. A. C. Biezen, H. Huijgen, and O. A. Ojo, "True-Motion Estimation with 3-D Recursive Search Block Matching," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 3, no. 5, pp. 368–379, Oct. 1993.

[30] G. de Haan and P. W. Biezen, "Sub-Pixel Motion Estimation with 3-D Recursive Search Block-Matching," *Signal Processing: Image Communication*, vol. 6, no. 3, pp. 229–239, Jun 1994.

[31] L. De Vos, "VLSI-Architectures for the Hierarchical Block-Matching Algorithm for HDTV Applications," in *Proc. of SPIE Visual Communications and Image Processing*, vol. 1360, pp. 398–409, 1990.

[32] L. De Vos and M. Stegherr, "Parameterizable VLSI Architectures for Full-Search Block-Matching Algorithm," *IEEE Trans. on Circuits and Systems*, vol. 36, no. 10, pp. 1309–1316, Oct. 1989.

[33] G. Demos, "MPEG-2 Video Adjustments For Advanced Layered Coding." ISO/IEC JTC1/SC29/WG11 Coding of Moving Pictures and Associated Audio MPEG98/M2975, Feb. 1998.

[34] L. Dreschler and H.-H. Nagel, "Volumetric Model and 3D Trajectory of a Moving Car Derived from Monocular TV Frame Sequences of a Street Scene," *Computer Graphics and Image Processing*, no. 20, pp. 199–228, 1982.

[35] F. Dufaux and M. Kunt, "Multigrid Block Matching Motion Estimation With an Adaptive Local Mesh Refinement," in *Proc. of SPIE Visual Communication and Image Processing*, vol. 1818, pp. 97–109, 1992.

[36] F. Dufaux and F. Moscheni, "Motion Estimation Techniques for Digital TV: A Review and a New Contribution," *Proceedings of the IEEE*, vol. 83, no. 6, pp. 858–876, June 1995.

[37] J.-L. Dugelay and H. Sanson, "Differential Methods for the Identification for 2D and 3D Motion Models in Image Sequences," *Signal Processing: Image Communication*, vol. 7, no. 1, pp. 105–127, Mar. 1995.

[38] S. Dutta, A. Wolfe, W. Wolf, and K. J. O'Connor, "Design Issues for Very-Long-Instruction-Word VLSI," in *VLSI Signal Processing* (W. Burleson, K. Konstantinides, and T. Meng, eds.), vol. IX, pp. 95–104, 1996.

[39] S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, R. L. Stamm, and D. M. Tullsen, "Simultaneous Multithreading: A Platform for Next-Generation Processors," *IEEE Micro*, vol. 17, no. 5, pp. 12–19, Sept./Oct. 1997.

[40] C. Fan, N. Namazi, and P. Penafiel, "New Image Motion Estimation Algorithm Based on the EM Technique," *IEEE Trans. on the Pattern Analysis and Machine Intelligence*, vol. 18, no. 3, pp. 348–352, Mar. 1996.

[41] Federal Communications Commission, "Final Report and Recommendation of the Federal Communications Commision Advisory Committee on Advanced Television Service." `http://www.atsc.org/finalrpt.html`, Nov. 1995.

[42] P. J. S. G. Ferreira, "Incomplete Sampling Series and the Recovery of Missing Samples from Oversampled Band-Limited Signals," *IEEE Trans. on Signal Processing*, vol. 40, no. 1, pp. 225–227, Jan. 1992.

[43] E. Francois, J.-F. Vial, and B. Chupeau, "Coding Algorithm with Region-Based Motion Compensation," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 7, no. 1, pp. 97–108, Feb. 1997.

[44] K. Gaedke, H. Jeschke, and P. Pirsch, "A VLSI Based MIMD Architecture of a Multiprocessor System for Real-Time Video Processing Applications," *Journal of VLSI Signal Processing*, vol. 5, no. 2-3, pp. 159–169, Apr 1993.

[45] D. L. Gall, "MPEG: A Video Compression Standard for Multimedia Applications," *Communications of the ACM*, vol. 34, no. 4, Apr. 1991.

[46] W. Gao and X. Chen, "Stochastic Approach for Blurred Image Restoration and Optical Flow Computation on Field Image Sequence," *Journal of Computer Science and Technology*, vol. 12, no. 5, pp. 385–399, Sept. 1997.

[47] S.-C. Han and J. W. Woods, "Frame-rate Up-conversion Using Transmitted Motion and Segmentation Fields for Very Low Bir-rate Video Coding," in *Proc. of IEEE Int'l Conf. on Image Processing*, vol. II, pp. 747–750, Oct. 1997.

[48] A. R. Hurson, K. M. Kavi, B. Shirazi, and B. Lee, "Cache Memories for Data Systems," *IEEE Parallel and Distributed Technology*, vol. 4, no. 4, pp. 50–64, Winter 1996.

[49] Y.-T. Hwang and Y.-H. Hu, "A Unified Partitioning and Scheduling Scheme for Mapping Multi-Stage Regular Iterative Algorithms onto Processor Arrays," *Journal of VLSI Signal Processing Applications*, vol. 11, pp. 133–150, Oct. 1995.

[50] Intel, "Intel MMX Technology–Developer's Guide." `http://developer.intel.com/drg/mmx/manuals/dg/devguide.htm`, 1997.

[51] ITU Telecommunication Standardization Sector, "ITU-T Recommendation H.263 Video Coding for Low Bitrate Communication." `ftp://ftp.std.com/vendors/PictureTel/h324/`, May 1996.

[52] J. C.-H. Ju, Y.-K. Chen, and S. Y. Kung, "A Fast Algorithm for Rate Optimized Motion Estimation," in *Proc. of Int'l Symposium on Multimedia Information Processing*, (Taipei, Taiwan), pp. 472–477, Dec. 1997.

[53] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active Contour Models," *International Journal of Computer Vision*, vol. 1, no. 4, pp. 321–331, 1988.

[54] K. Kawaguchi and S. K. Mitra, "Frame Rate Up-Conversion Considering Multiple Motion," in *Proc. of IEEE Int'l Conf. on Image Processing*, vol. I, pp. 727–730, 1997.

[55] J. Kim and J. W. Woods, "3-D Kalman Filter for Image Motion Estimation," *IEEE Trans. on Image Processing*, vol. 7, no. 1, pp. 42–52, Jan. 1998.

[56] M. Klima, P. Dvořák, P. Zahradnik, J. Kolář, and P. Kott, "Motion Detection and Target Tracking in a TV Image for Security Purposes," in *Proc. of IEEE Int'l Conf. on Security Technology*, pp. 43–44, 1994.

[57] U. V. Koc and K. J. R. Liu, "DCT-Based Subpixal Motion Estimation," in *Proc. of IEEE Int'l Conf. on Acoustic, Speech, and Signal Processing*, vol. IV, pp. 1931–1934, May 1996.

[58] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion Compensated Interframe Coding for Video Conference," in *Proc. of National Telecommunication Conference*, vol. 2, (New Orleans, LA), pp. G5.3.1–5.3.5, Nov./Dec. 1981.

[59] T. Komarek and P. Pirsch, "Array Architectures for Block Matching Algorithms," *IEEE Trans. on Circuits and Systems*, vol. 36, no. 10, pp. 1301–1308, Oct. 1989.

[60] S. Y. Kung, *VLSI Array Processors*. Englewood Cliffs, NJ: Prentice Hall, 1988.

[61] S. Y. Kung, Y.-T. Lin, and Y.-K. Chen, "Motion-Based Segmentation by Principal Singular Vector (PSV) Clustering Method," in *Proc. of the IEEE Int'l Conf. on Acoustics, Speech, and Signal Processing*, (Atlanta, GA), pp. 3410–3413, May 1996.

[62] D.-H. Lee, J.-S. Park, and Y.-G. Kim, "Video Format Conversions Between HDTV Systems," *IEEE Trans. on Consumer Electronics*, vol. 39, no. 3, pp. 219–224, Aug. 1993.

[63] J.-B. Lee and S.-D. Kim, "Moving Target Extraction and Image Coding Based on Motion Information," *IEICE Trans. Fundamentals*, vol. E78-A, no. 1, pp. 127–130, Jan. 1995.

[64] M.-H. Lee, J.-H. Kim, J.-S. Lee, K.-K. Ryu, and D.-I. Song, "A New Algorithm for Interlaced to Progressive Scan Conversion Based on Directional Correlations and Its IC Design," *IEEE Trans. on Consumer Electronics*, vol. 40, no. 2, pp. 119–129, May 1994.

[65] R. B. Lee, "Subword Parallelism with MAX-2," *IEEE Micro*, vol. 16, no. 4, pp. 51–59, Aug. 1996.

[66] X. Lee and Y.-Q. Zhang, "A fast hierarchical motion-compensation scheme for video coding using block feature matching," *IEEE Trans. on Circuits and Systems for Video Tech.*, vol. 6, no. 6, pp. 627–635, Dec 1996.

[67] G.-J. Li and B. W. Wah, "The Design of Optimal Systolic Array," *IEEE Trans. on Computer*, vol. 34, no. 1, pp. 66–77, Jan. 1985.

[68] J. Li, X. Lin, and Y. Wu, "Multiresolution Tree Architecture with its Application in Video Sequence Coding: A New Result," in *Proc. of SPIE Visual Communication and Image Processing*, vol. 2094, pp. 730–741, 1993.

[69] Y.-T. Lin, Y.-K. Chen, and S. Y. Kung, "A Principal Component Clustering Approach to Object-Oriented Motion Segmentation and Estimation," *Journal of VLSI Signal Processing Systems*, vol. 17, no. 2, pp. 163–188, Nov. 1997.

[70] Y.-T. Lin, Y.-K. Chen, and S. Y. Kung, "Object-Based Scene Segmentation Combining Motion and Image Cues," in *Proc. of IEEE Int'l Conf. on Image Processing*, vol. I, (Lausanne, Switzerland), pp. 957–960, Sept. 1996.

[71] P. Lippens, V. Nagasamy, and W. Wolf, "CAD Challenges in Multimedia Computing," in *Proc. of Int'l Conf. on Computer-Aided Design*, pp. 502–508, 1995.

[72] B. Liu and A. Zaccarin, "New Fast Algorithms for the Estimation of Block Motion Vectors," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 3, no. 2, pp. 148–157, Apr. 1993.

[73] D. Matzke, "Will Physical Scalability Sabotage Performance Gains?" *IEEE Computer*, vol. 30, no. 9, pp. 37–39, Sept. 1997.

[74] J. Mendelsohn, E. Simoncelli, and R. Bajcsy, "Discrete-Time Rigidity-Constrained Optical Flow," in *Proc. of Int'l Conf. on Computer Analysis of Image and Patterns*, Sept. 1997.

[75] J. L. Mitchell, W. B. Pennebaker, C. E. Fogg, and D. J. Le Gall, *MPEG Video Compression Standard*. Chapman and Hall, 1996.

[76] P. Moulin, R. Krishnamurthy, and J. W. Woods, "Multiscale Modeling and Estimation of Motion Fields for Video Coding," *IEEE Trans. on Image Processing*, vol. 6, no. 12, pp. 1606–1620, Dec 1997.

[77] H. G. Musmann, M. Hötter, and J. Ostermann, "Object-Oriented Analysis-Sysnthesis Coding of Moving Images," *Singal Processing: Image Communication*, vol. 1, no. 2, pp. 117–138, Oct. 1989.

[78] K. Nadehara, I. Kuroda, M. Daito, and T. Nakayama, "Low-Power Multimedia RISC," *IEEE Micro*, vol. 15, no. 6, pp. 20–29, Dec. 1995.

[79] M. O'Connor, "Extending Instructions for Multimedia," *Electronic Engineering Times*, no. 874, p. 82, Nov. 1995.

[80] M. T. Orchard, "Predictive Motion-Field Segmentation for Image Sequence Coding," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 3, no. 1, pp. 54–70, Feb. 1993.

[81] A. Papoulis, "Generalized Sampling Expansion," *IEEE Trans. on Circuits and Systems*, vol. 24, no. 11, pp. 652–654, Nov. 1977.

[82] N. L. Passos and E. H.-M. Sha, "Achieving Full Parallelism Using Multidimensional Retiming," *IEEE Trans. on Parallel and Distributed Systems*, vol. 7, no. 11, pp. 1150–1163, Nov. 1996.

[83] D. Patterson and J. Hennessy, *Computer Architecture: A Quantitative Approach*. San Francisco, CA: Morgan Kaufmann Publishers, 2nd ed., 1996.

[84] Philips Electronics, "TRIMEDIA TM1000 Programmable Media Processor." `http://www-us2.semiconductors.philips.com/trimedia/products/tm1.stm`, 1997.

[85] A. Puri, H.-M. Hang, and D. L. Schilling, "An Efficient Block Matching Algorithm for Motion-Compensated Coding," in *Proc. of the IEEE Int'l Conf. on Acoustics, Speech, and Signal Processing*, (Dallas, Texas), pp. 25.4.1–25.4.4, 1987.

[86] J. M. Rehg and A. P. Witkin, "Visual Tracking with Deformation Models," in *Proc. of IEEE Int'l Conf. on Robotics and Automation*, vol. 1, pp. 844–850, Apr. 1991.

[87] V. Seferidis and M. Ghanbari, "Generalized Block-Matching Motion Estimation Using Quad-Tree Structured Spatial Decomposition," *IEE Proc.-Vis. Image Signal Process*, vol. 141, no. 6, pp. 446–452, Dec. 1994.

[88] T. Sikora, "MPEG Digital Video-Coding Standard," *IEEE Signal Processing*, vol. 14, no. 5, pp. 82–100, Sept. 1997.

[89] E. P. Simoncelli, *Distributed Analysis and Representation of Visual Motion*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, Cambridge, MA, Jan. 1993.

[90] R. Srinivasan and K. R. Rao, "Predictive Coding Based on Efficient Motion Estimation," *IEEE Trans. on Communications*, vol. 33, no. 8, pp. 888–896, Aug. 1985.

[91] R. L. Stevenson and R. R. Schultz, "Extraction of High-Resolution Frames from Video Sequences," in *Proc. of IEEE Workshop on Nonlinear Image Processing*, (Neos Marmaras, Greece), June 1995.

[92] M.-T. Sun, "Algorithms and VLSI Architectures for Motion Estimation," *VLSI Implementations for Image Communications*, pp. 251–282, 1993.

[93] K. Suzuki, T. Arai, K. Nadehara, and I. Kuroda, "V830R/AV: Embedded Multimedia Superscalar RISC Processor," *IEEE Micro*, vol. 18, no. 2, pp. 35–47, Mar./Apr. 1998.

[94] J. Teich and L. Thiele, "Partitioning of processor arrays: a piecewise regular approach," *INTEGRATION: The VLSI Journal*, vol. 14, no. 3, pp. 297–332, 1993.

[95] J. Teich, L. Thiele, and L. Zhang, "Partitioning Processor Arrays under Resource Constraints," *Journal of VLSI Signal Processing*, vol. 17, no. 1, pp. 5–20, 1997.

[96] Telenor R&D, "H.263 Encoder Version 2.0." `ftp://bonde.nta.no/pub/tmn/software/`, June 1996.

[97] Texas Instruments, "TMS320C6000 Product Information." `http://www.ti.com/sc/docs/dsps/products/c6000/index.htm`, 1998.

[98] Texas Instruments, "TMS320C8x Product Information." `http://www.ti.com/sc/docs/dsps/products/c8x/index.htm`, 1998.

[99] R. Thoma and M. Bierling, "Motion Compensating Interpolation Considering Covered and Uncovered Background," *Signal Processing: Image Communications*, vol. 1, pp. 191–212, 1989.

[100] C. Tomasi and T. Kanade, "Shape and Motion from Image Streams: a Factorization Method–Part 3, Detection and Tracking of Point Features," Tech. Rep. CMU-CS-91-132, Carnegie Mellon University, Apr. 1991.

[101] K. M. Uz, M. Vetterli, and D. LeGall, "Interpolative Multiresolution Coding of Advanced Television with Compatible Subchannels," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 1, no. 1, pp. 86–99, 1991.

[102] J. van Meerbergen, P. Lippens, B. McSweeney, W. Verhaegh, A. van der Werf, and A. van Zanten, "Architectural Strategies for High-Throughout Applications," *Journal of VLSI Signal Processing*, vol. 5, no. 2-3, pp. 201–220, Apr 1993.

[103] J. L. van Meerbergen, P. E. R. Lippens, W. F. J. Verhaegh, and A. van der Werf, "PHIDEO: High-Level Synthesis for High Throughput Applications," *Journal of VLSI Signal Processing*, vol. 9, no. 1-2, pp. 89–104, Jan 1995.

[104] L. Vandendorpe, L. Cuvelier, B. Maison, P. Queluz, and P. Delogne, "Motion Compensated Conversion from Interlaced to Progressive Formats," *Signal Processing: Image Communication*, vol. 6, no. 3, pp. 193–211, June 1994.

[105] W. F. Verhaegh, P. E. Lippens, E. H. Aarts, J. H. Korst, J. L. van Meerbergen, and A. van der Werf, "Improved force-directed scheduling in high-throughput digital signal processing," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 8, pp. 945–960, Aug 1995.

[106] F.-M. Wang, D. Anastassiou, and A. N. Netravali, "Time-Recursive Deinterlacing for IDTV and Pyramid Coding," *Signal Processing: Image Communication*, vol. 2, no. 3, pp. 365–374, Oct. 1990.

[107] J. Y.-A. Wang and E. H. Adelson, "Representing Moving Images with Layers," *IEEE Trans. on Image Processing*, vol. 3, no. 5, pp. 625–638, Sept. 1994.

[108] Y. Wong and J.-M. Delosme, "Optimization of Computation Time for Systolic Array," *IEEE Trans. on Computer*, vol. 41, no. 2, pp. 159–177, Feb. 1992.

[109] K. Xie, L. Van Eycken, and A. Oosterlinck, "A New Block-Based Motion Estimation Algorithm," *Signal Processing: Image Communication*, vol. 4, no. 6, pp. 507–517, Nov. 1992.

[110] H. Yamauchi, Y. Tashiro, T. Minami, and Y. Suzuki, "Architecture and Implementation of a Highly Parallel Single-chip Video DSP," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 2, no. 2, pp. 207–220, June 1992.

[111] H. Yeo and Y.-H. Hu, "A Novel Modular Systolic Array Architecture for Full-Search Block Matching Motion Estimation," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 5, no. 5, pp. 407–416, Oct. 1995.

[112] S. Zafar, Y. Q. Zhang, and B. Jabbari, "Multiscale Video Representation Using Multiresolution Motion Compensation and Wavelet Decomposition," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 1, pp. 24–35, Jan. 1993.

[113] K.-H. Zimmermann, "A Unifying Lattice-Based Approach for the Partitioning of Systolic Arrays via LPGS and LSGP," *Journal of VLSI Signal Processing*, vol. 17, no. 1, pp. 21–47, 1997.

[114] K.-H. Zimmermann, "Linear Mappings of n-Dimensional Uniform Recurrences onto k-Dimensional Systolic Array," *Journal of Signal Processing System for Signal, Image, and Video Technology*, vol. 12, no. 2, pp. 187–202, May 1996.

[115] K.-H. Zimmermann and W. Achtziger, "Finding Space-Time Transformations for Uniform Recurrences via Branching Parametric Linear Programming," *Journal of VLSI Signal Processing*, vol. 15, no. 3, pp. 259–274, 1997.

[116] K.-H. Zimmermann and W. Achtziger, "On Time Optimal Implementation of Uniform Recurrences onto Array Processors via Quadratic Programming," *Journal of Signal Processing Systems for Signal, Image, and Video Technology*, vol. 19, no. 1, pp. 19–38, 1998.