논문 2004-41SP-6-23

H.264 표준의 가변 움직임 블록을 위한 고속 움직임 탐색 기법

(Fast Motion Estimation for Variable Motion Block Size in H.264 Standard)

최 웅 일*, 전 병 우**

(Woong Il Choi and Byeungwoo Jeon)

요 약

기존 비디오 표준과 비교해 볼 때, H.264 비디오 표준이 갖는 중요한 두 가지 특징으로는 높은 부호화 효율과 네트워크 친화성을 들 수 있다. 그러나 이러한 중요한 특성에도 불구하고 H.264 표준은 구현시 요구되는 메모리 대역폭과 연산량의 복잡도가 높기 때문에 실시간 응용에 적용하는데 어려움이 있다. H.264 부호화 기술 가운데 특히 복수 참조 영상을 이용한 다양한 블록 단위 움직임 탐색은 높은 부호화 효율을 갖도록 하는 핵심 요소지만 최적의 움직임 벡터를 찾기 위해 다양한 블록 단위 조합의 모든 경우에 대하여 SAD (Sum of Absolute Difference)를 구해야 하므로 상당한 계산량을 요구한다. 그러므로 본 논문에서는 움직임 탐색의 연산량을 줄이기 위해 정수화소 움직임 탐색 및 부화소 움직임 탐색을 위한 고속 알고리즘을 제안한다. 정수화소 단위 움직임 탐색의 경우, 기존의 고속 움직임 탐색 기법은 H.264의 다양한 블록 단위 움직임 탐색 구조에 그대로 적용할 경우 효과적이지 못하기 때문에 본 논문에서는 종래 다이아몬드 탐색 기반 방법을 계층적 블록 구조에 맞게 개선한 적응적 움직임 탐색 기법을 제안하도록 한다. 또한 부화소 단위 움직임 탐색을 위해서는 움직임 벡터의 통계적 특성을 이용하여 예측벡터를 중심으로 한 다이아몬드 탐색 기반 고속 알고리즘을 제안한다.

Abstract

The main feature of H.264 standard against conventional video standards is the high coding efficiency and the network friendliness. In spite of these outstanding features, it is not easy to implement H.264 codec as a real-time system due to its high requirement of memory bandwidth and intensive computation. Although the variable block size motion compensation using multiple reference frames is one of the key coding tools to bring about its main performance gain, it demands substantial computational complexity due to SAD (Sum of Absolute Difference) calculation among all possible combinations of coding modes to find the best motion vector. For speedup of motion estimation process, therefore, this paper proposes fast algorithms for both integer-pel and fractional-pel motion search. Since many conventional fast integer-pel motion estimation algorithms are not suitable for H.264 having variable motion block sizes, we propose the motion field adaptive search using the hierarchical block structure based on the diamond search applicable to variable motion block sizes. Besides, we also propose fast fractional-pel motion search using small diamond search centered by predictive motion vector based on statistical characteristic of motion vector.

Keywords: H.264 Standard, Fast motion estimation, Diamond search algorithm, Real-time video applications

I. 서 론

움직임 보상은 영상 압축 비디오 부호화시 가장 중요한 기술로써 현재까지 모든 비디오 부호화에서 사용되고 있다. 차세대 비디오 부호화 표준인 H.264의 경우에도 역시 이러한 움직임 보상을 통해 높은 압축 효율을 얻고 있으며, 특히 지난 표준 비디오 압축 방식과는 달리 하나 이상의 참조 영상을 움직임 보상에 이용한다.

그러나 이러한 복수 개의 참조 영상 및 다양한 블록 단위 움직임 보상 기법은 부호화기의 움직임 추정시 각 부호화 모드에 대하여 최적의 움직임 벡터를 찾아내기 위한 반복적 탐색을 요구하기 때문에 막대한 양의 연산량이 필요하다. 16x16 하나의 매크로블록은 그림 1(a)에서 보는 바와 같이 16x16, 16x8, 8x16, 8x8 하위 블록으로 나뉘어 움직임 보상이 가능하며 8x8 블록 단위로 나

* 학생회원, ** 정회원, 성균관대학교 정보통신공학부
(School of Information and Communication Eng., Sungkyunkwan University)

※ 이 논문은 2003년도 한국학술진흥재단의 지원에 의하여 연구되었음. (KRF-2003-041-D20405)
접수일자: 2004년8월2일, 수정완료일: 2004년9월6일

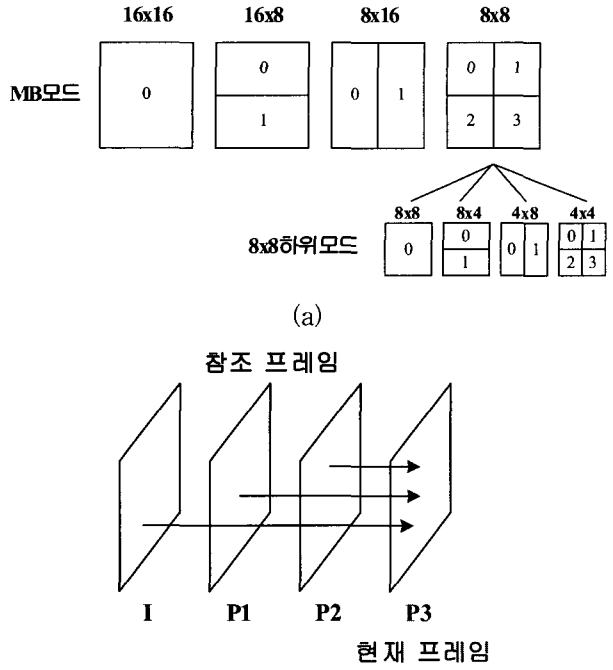
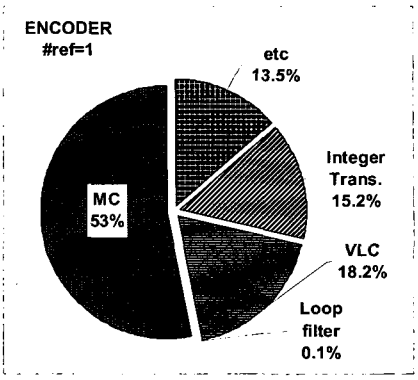


그림 1. (a) 가변 블록 크기의 매크로블록 모드 및 (b) 복수 참조 프레임에 따른 H.264의 움직임 보상 기법

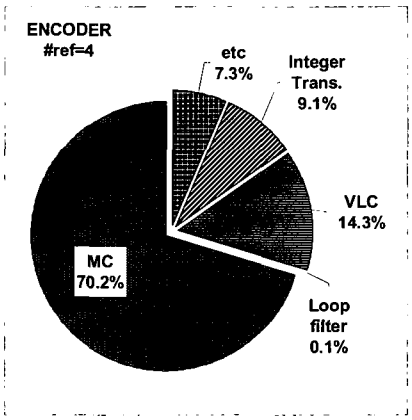
Fig. 1. H.264 Motion compensation using (a) various block sizes and (b) multiple reference frames.

는 경우에는 다시 8x8, 8x4, 4x8, 4x4 등의 블록으로 추가로 나뉠 수 있다. 8x8 하위 블록의 작은 블록 단위 움직임 보상은 압축 효율뿐 아니라 주관적 화질 개선에도 도움을 주지만 빈번한 메모리 접근으로 인하여 복잡도를 추가적으로 증가시키는 원인이 되고 있다. 이러한 8x8 하위 블록의 경우 움직임 탐색시 종래 16x16 매크로블록 단위로 움직임 보상하던 것에 비해 4배의 SAD 연산이 필요하기 때문에 H.264의 경우 각 블록 단위에 따른 움직임 탐색의 총 SAD 연산 수는 16x16 매크로블록 단위의 움직임 보상을 한번 수행하는 기존 H.263의 경우에 비해 7배에 해당한다[1]. 이뿐만 아니라 그림 1(b)와 같이 움직임 보상에 사용되는 참조 프레임이 다양하기 때문에 사용되는 참조 영상의 수만큼 이러한 SAD 연산량의 증가는 배로 커질 수밖에 없다.

그림 2는 이러한 움직임 탐색 및 보상 과정이 전체 부호화기에 차지하는 비중을 알아보기 위해 참조 코덱 JM(Joint Model) 6.1d 부호화기의 복잡도를 분석한 것이다. 단 하나의 참조 영상을 사용한 그림 2(a)의 경우에는 움직임 보상(MC)이 53%를 차지하였고 정수 변환(Integer Transform)이 15%, 가변장 부호화(VLC)가 18%, 그리고 블록화 제거 필터가 0.1%를 차지한 것을



(a) 참조영상 = 1개



(b) 참조영상 = 4개

그림 2. JM 부호화기 복잡도 분석

Fig. 2. Complexity analysis of JM Encoder.

볼 수 있다. 이에 반해 4개의 참조 영상을 사용한 그림 2(b)의 경우에는 움직임 보상이 70%로 증가하였고 상대적으로 정수변환 및 가변장 부호화는 각각 9%, 14%로 줄어든 것을 볼 수 있다. 정수변환과 가변장 부호화가 차지하는 복잡도는 부호화기에서 사용된 R-D 최적화 기법으로 인해 각 부호화 모드별 비트율을 계산하기 위한 반복적 수행에서 기인한 것이다. 만일 복잡도 저하를 위해 R-D 최적화를 사용하지 않는다면, 즉 실제 비트율을 계산하지 않고 왜곡(Distortion)만을 이용하게 되는 경우에는 정수변환과 가변장 부호화가 차지하는 비중이 현격히 줄어들게 되어 움직임 보상에 해당하는 복잡도가 상대적으로 더욱 증가하게 될 것이다. H.264 부호화기의 전체 복잡도를 줄이기 위해서는 가장 높은 비중을 차지하는 움직임 탐색 및 보상에 필요한 연산량을 간소화하는 것이 필수적이다. 그러므로 이러한 움직임 탐색의 연산량을 효과적으로 줄이기 위해 정수 화소 움직임 탐색 및 부화소 움직임 탐색 각각을 위한 고속 움직임 탐색 알고리즘을 제안한다.

먼저 정수 화소 움직임 탐색을 위해서는 다양한 블록

단위 움직임 보상을 위한 고속 움직임 탐색 기법을 제안하고자 한다. 제안기법은 16x16 블록 단위에서 4x4블록 단위까지 다양한 블록 단위에 기반한 움직임 보상 시 각 블록 단위 사이의 상관도를 이용하여 기존 고속 움직임 탐색에 비해 보다 효과적인 탐색이 가능하다. 이를 위해 먼저 기존 제안된 고속 움직임 탐색 기법들을 간략히 소개하고 이러한 고속 움직임 탐색 기법을 가변 블록 단위 움직임 보상에 이용하기 위한 새로운 알고리즘을 제안한다.

또한 H.264 표준에서는 움직임 벡터가 1/4 화소 단위까지 정확도를 갖기 때문에 정수화소 단위 움직임 탐색에 이어서 부화소 움직임 탐색이 수반된다. 여기서 부화소 단위 움직임 탐색은 다양한 블록 단위 움직임 보상으로 인하여 기존 H.263이나 MPEG-4 part 2와 같은 비디오 표준에 비하여 그 수행 횟수가 현격히 증가하였기 때문에 전체 움직임 탐색 연산량에 있어서 상대적으로 많은 비중을 차지한다. 따라서 이러한 부화소 움직임 탐색의 연산량을 줄이기 위해 움직임 벡터의 통계적 특성을 이용하여 예측 벡터를 중심으로 한 다이아몬드 탐색 기법을 제안한다.

II. 기존 고속 움직임 탐색 기법: Diamond 탐색 및 MVFAST 기법

종래 제안된 고속 움직임 탐색 기법으로는 비교적 간단한 다이아몬드 탐색(Diamond Search, DS) 기법을 들 수 있다[2]. 그림 3에서 보는 바와 같이 DS 기법은 탐색의 중심 화소와 그 중심으로부터 시티블록 거리가 2의 화소들로 이루어진 LDSP(Large Diamond Search Pattern)와, 중심 화소로부터 시티블록 거리가 1의 화소들로 이루어진 SDSP(Small Diamond Search Parttern)의 두 가지 형태의 탐색 패턴을 가지고 있다. LDSP는 중심을 포함하여 9개의 탐색 위치점을, SDSP는 중심을 포함하여 5개의 탐색 위치점을 갖고 있으며 둘 다 모두 다이아몬드 모양을 갖는 것을 볼 수 있다.

주어진 블록과 참조 영상 내에서 최소 SAD값을 갖는 동일 크기의 예측블록 간 거리를 그 블록의 최적 움직임 벡터라고 할 때, 이러한 최적 움직임 벡터를 찾기 위해 영상의 x,y 축에 있어서 움직임 벡터가 (0,0) 지점을 시작점으로 하여 최소 SAD를 갖는 지점을 찾아 나간다. 먼저 LDSP의 중심점을 (0,0)으로 하여 각 9개 지점에 대하여 최소 SAD를 갖는 지점을 조사한 다음, 그 지점을 다시 LDSP의 중심점으로 하여 이러한 과정을

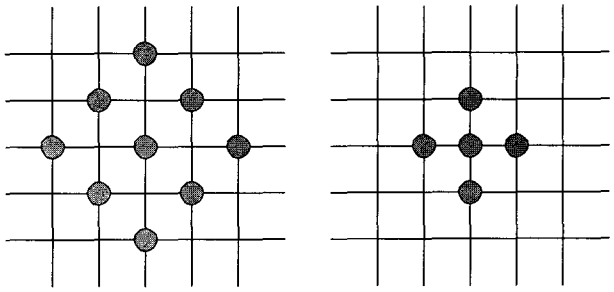


그림 3. 다이아몬드 탐색의 두가지 패턴
(a) LDSP (b) SDSP

Fig. 3. Two types of diamond search pattern.
(a) LDSP (b) SDSP

반복한다. 이러한 반복 과정을 수행하다가 최소 SAD를 갖는 지점이 LDSP의 중심점이 될 경우에는 그 점을 SDSP의 중심점으로 하여 마지막으로 최소 SAD를 갖는 지점을 조사하고 그 지점을 최적 움직임 벡터로 정한다.

DS 기법의 경우 (0,0) 지점을 중심으로 탐색을 시작하기 때문에 비교적 움직임이 작은 영상의 경우에는 LDSP의 반복 탐색 수행 횟수가 적지만 움직임이 큰 영상의 경우에는 수행 횟수가 증가하게 되어 속도 향상 측면에서 효율이 떨어지게 된다. 또한 이렇게 탐색 시작점으로부터 매칭 에러가 최소가 되는 지점까지 거리가 멀 경우, 탐색 도중 국부 최소치(local minimum)에 빠져 최적 움직임 벡터를 찾는데 실패할 확률도 커진다. 따라서 탐색의 초기 지점을 최대한 실제 움직임 벡터의 근처에서 시작하도록 하기 위해 주변 매크로블록들의 움직임 벡터를 DS 탐색의 초기 지점으로 이용하는 방법들이 제안되었다[3,4]. 물체의 움직임은 대개 시간적으로 또는 공간적으로 그 상관도가 높기 때문에 기존 제안된 기법들은 주변 매크로블록의 움직임 특성을 이용하여 종래 DS 기법을 개선한 것이다. 이러한 방법 가운데 하나인 MVFAST (Motion Vector Field Adaptive Search Technique) 기법은 주변 블록의 움직임 벡터의 크기를 이용한 움직임 벡터 활성도(MV Activity)에 따라 초기 탐색 위치 지점과 LDSP 또는 SDSP의 탐색 패턴을 정하는 기법이다. MVFAST의 구체적 알고리즘은 다음과 같다[3].

우선 주어진 블록의 좌측, 상측, 상우측 블록의 움직임 벡터들의 크기(Norm), l_i을 각각 구한다. 각 벡터의 크기 가운데 가장 큰 값 l_{MAX}를 각 문턱치 L_1, L_2 (단, $L_1 \leq L_2$)와 비교하여 식 (1)과 같이 움직임 벡터 활성

도(Motion Vector Activity, MVA)를 구한다.

$$MVA = \begin{cases} low, & l_{MAX} \le L_1 \\ medium, & L_1 \le l_{MAX} \le L_2 \\ high, & l_{MAX} > L_2 \end{cases} \quad (1)$$

식 (1)에서 구한 주변 블록을 이용한 움직임 벡터 활성도는 현재 블록의 움직임 벡터 크기와 상관도가 높기 때문에 이 활성도에 따라 움직임 벡터 탐색의 시작점과 패턴을 조절하는 것이 합리적이다. 여기서 움직임 벡터 활성도가 낮은 경우에는 초기 탐색 위치를 (0,0)으로 하고 탐색 패턴도 SDSP만을 사용한다. 즉, 이 경우에는 SDSP의 시작점을 (0,0)으로 하여 최소 SAD가 발생하는 지점을 찾아 다시 그 지점을 중심으로 SDSP를 반복 수행하여 SDSP 중심점에 최소 SAD가 발생할 때 그 지점을 움직임 벡터로 한다. 움직임 벡터 활성도가 높은 경우에는 좌측, 상측, 상우측의 각 주변 움직임 벡터들을 이용하여 그 지점에 대한 SAD를 구한 다음, 최소 SAD를 갖는 지점을 시작점으로 하여 마찬가지로 SDSP의 탐색을 수행한다. 마지막으로 움직임 벡터 활성도가 중간인 경우에는 기존 DS 기법과 동일한 방법으로 (0,0) 지점을 중심으로 LDSP 및 SDSP 탐색을 수행한다.

이러한 MVFAST 기법은 공간상의 움직임 벡터간 상관도를 이용하여 종래 DS에 비해 부호화 효율 및 속도 측면에서 모두 우수한 성능을 보였다. 본 논문에서는 주어진 블록의 움직임 벡터를 예측하는데 있어서 기존 기법에 비해 보다 정확한 모델을 이용함으로써 적응적 탐색기법의 효율을 더욱 증가시키고자 한다. 따라서 매크로블록 내 동일 위치의 각 가변 블록간 움직임 벡터의 분포 특성을 이용한 적응적 탐색 패턴 기법을 제안하도록 한다.

III. 제안된 고속 움직임 탐색 기법

1. 정수화소 움직임 탐색

H.264의 가변 블록 크기 움직임 보상으로 인하여 16x16 매크로블록은 16x16 단위로 하나의 움직임 벡터를 가질 수 있을 뿐 아니라 16x8 및 8x16, 그리고 8x8 하위 블록 단위로 나뉘어 작은 블록 조각에 대해 별도의 움직임 벡터들을 가질 수 있다. 뿐만 아니라 8x8 하위 블록은 8x8 블록 크기의 움직임 벡터와 더불어 8x4, 4x8, 4x4 블록 단위로 나뉜 움직임 벡터도 가질 수 있다. 그러므로 H.264 표준의 가변 블록 크기를 위한 움직임 탐색은 16x16 하나의 매크로블록을 점차 세부적으로 나눠 움직임 벡터를 조사하는 것으로 볼 수 있다. 이 과정을 역으로 생각하면 16x16 하나의 매크로블록 내에 존재하는 최소 4x4 단위의 작은 블록의 움직임 벡터들을 시작으로 점차 큰 영역의 움직임 벡터를 구하는 과정이라고 볼 수도 있다. 이러한 가변 블록 크기의 움직임 보상은 그림 4와 같은 계층적 구조를 갖는다.

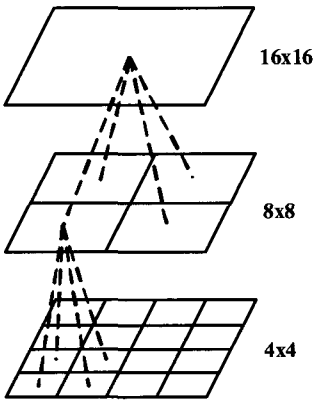


그림 4. H.264 표준의 가변 블록 크기 움직임 보상의 계층 구조

Fig. 4. Hierarchical structure of H.264 motion compen-sation using variable block size.

그림 4와 같이 4x4 블록 크기의 움직임 벡터 4개는 각 해당 위치의 8x8 블록 크기 움직임 벡터와 상관도가 높다. 마찬가지로 4개의 8x8 블록 크기 움직임 벡터는 16x16 매크로블록의 움직임 벡터와 상관도가 높다. 그 이유는 그림 4에서 보는 것처럼 4x4, 8x8, 16x16의 각 계층 간에는 동일한 영상의 움직임을 서로 다른 블록 크기로 나눠 표현한 차이만 있을 수 있기 때문이다. 그러므로 본 논문에서는 이러한 계층적 구조를 갖는 가변 블록 단위 움직임 벡터 구조를 이용한 적응적 탐색 기법을 제안하고자 한다.

만일 매크로블록 내에 동일한 하나의 객체가 존재하여 4x4 블록 단위의 움직임 벡터들이 서로 유사하다면 그 상위 블록, 즉 8x8 블록 단위 및 16x16 블록 단위의 움직임 벡터도 4x4 블록 단위 움직임 벡터와 매우 유사하다. 이러한 경우에는 움직임 벡터 오버헤드의 상대적 비중 감소로 인하여 상위 블록 단위로 갈수록 움직임 보상 효율이 증가하게 된다. 이에 반해 만일 매크로블록 내에 서로 다른 객체가 존재하거나 객체의 움직임이 선형적인 특성에서 벗어나는 등의 이유로 인해 매크로블록 내에 이질적 움직임이 존재하게 되는 경우에는 각 4x4단위 움직임 벡터들이 서로 다르게 분포하게 된다. 이러한 경우에는 4x4 블록 단위로 찾은 움직임 벡터와

8x8 및 16x16 블록의 움직임 벡터간에 유사도가 떨어지게 되며 또한 이 경우 상위 블록에서의 움직임 보상은 왜곡의 증가로 인해 그 효율이 떨어지게 된다. 그러므로 이러한 4x4 블록 단위 움직임 벡터의 분포를 이용하여 상위 블록의 움직임 벡터 탐색 방법을 변화시킨다. 제안 기법은 구체적으로 다음과 같다.

우선 종래 고속 움직임 탐색 기법을 이용하여 한 매크로블록 내 16개의 4x4 단위 움직임 벡터를 구한다. (본 실험에서는 MVFAST 기법을 이용하였다.) 그리고 그 외 상위 블록들의 경우에는 다음과 같이 구한다. 주어진 블록 내에 존재하는 하위 블록의 움직임 벡터를 기본 벡터, MV_b라고 한다. 예를 들어 8x8 블록의 경우 기본 벡터는 해당 블록 내에 존재하는 4개의 4x4 블록 단위 움직임 벡터들이 되고 마찬가지로 16x8의 경우에는 해당 블록 내에 존재하는 두 개의 8x8 블록 단위 움직임 벡터들이 기본 벡터가 된다. 이러한 기본 벡터를 이용하여 움직임 벡터 분산(MV Deviation) D를 다음 식 (2)와 같은 방법으로 계산한다.

$$D = \frac{1}{N}\sum_{i=1}^{N} MV_b(i) - \overline{MV_b}$$
$$\overline{MV_b} = \frac{1}{N}\sum_{i=1}^{N} MV_b(i) \quad (2)$$

여기서 $MV_b(i)$는 주어진 블록내의 i번째 기본 벡터를 의미하고 $\overline{MV_b}$는 주어진 블록을 구성하는 총 N개의 기본 벡터에 대한 평균 벡터를 나타낸다. 각 기본 벡터들이 서로 다른 방향을 가리키고 있다고 가정하면 D 값은 커지게 될 것이고, 반면 각 기본 벡터들이 서로 유사한 방향을 가리키게 되면 D 값은 작아지게 된다. 즉, D 값은 기본 벡터의 분포를 반영하고 있기 때문에 이 값을 이용하여 기본 벡터와의 유사도를 어느 정도 판단할 수 있다. 따라서 기본 벡터 간의 분산치 D를 이용하여 표 1과 같이 움직임 벡터 탐색의 시작점과 패턴을 정한다.

표 1. 움직임 벡터 분산에 따른 각 탐색 방법
Table 1. Proposed search strategy with regard to motion vector deviation.

	$D=0$	$0<D\le T_1$	$T_1<D\le T_2$	$D>T_2$
탐색 시작점	없음	$\overline{MV_b}$	$\overline{MV_b}$	$\overline{MV_b}$
탐색 패턴	수행안함	SDSP 1회 수행	SDSP	LDSP+SDSP

표 1에서 보는 것처럼, 만일 모든 기본 벡터의 값이 동일한 경우에는 D 값이 0이 되며 이 경우에는 움직임 탐색을 수행하지 않고 기본 벡터 자체를 주어진 블록의 움직임 벡터로 한다. 참고로 표 1의 문턱치 T_1 및 T_2의 값은 몇 가지 영상을 통해 실험적으로 구한 값을 본 실험에 사용하였다. 만일 D 값이 어느 문턱치 T_1 보다 작은 경우에는 주어진 블록 내 존재하는 움직임 벡터들이 거의 고르게 분포한다고 볼 수 있기 때문에 기본 벡터의 평균 벡터 $\overline{MV_b}$를 SDSP의 시작점으로 하여 왜곡이 최소인 지점을 조사한다. 단, 이 경우에는 SDSP의 탐색을 반복적으로 수행하지 않고 단 한번만 수행한다. 그 외에는 평균 벡터 $\overline{MV_b}$를 시작점으로 하여 D 값이 T_1과 T_2 사이에 있는 경우에는 SDSP 탐색을, T_2보다 큰 경우에는 LDSP 및 SDSP 탐색을 수행한다. 즉, D 값이 T_2보다 큰 경우에는 시작점을 제외하면 DS 기법과 동일한 탐색 방법으로 움직임 벡터를 찾게 된다. 기본 벡터의 분산값이 커질수록 기본 벡터의 평균 벡터인 $\overline{MV_b}$와 움직임 벡터의 유사도가 떨어지기 때문에 보다 넓은 범위의 탐색이 필요하다. 그러므로 이를 위해 D 값이 커짐에 따라 다이아몬드 탐색 범위를 키우도록 한 것이다.

또한 각 가변 블록 단위 탐색시 반복적 SAD 연산을 피하기 위해 SAD 값을 저장하여 재사용하는 방법을 사용하였다. 이 SAD 재사용 방법은 이미 H.264 참조 코덱에 구현되어 있는 고속 전체 탐색 기법에서 사용하고 있는 것으로 본 논문에서는 이러한 재사용 기법이 DS에 적용되도록 변형한 것이다. 예를 들어 어느 한 위치에 있어서 8x8 블록 단위로 SAD를 계산하는 경우에는 그 블록 내부에 대해 앞서 8x4, 4x8 또는 4x4 블록 단위로 움직임 탐색시 계산한 SAD 값들이 이미 존재하는 경우에는 중복된 연산을 피함으로 SAD 연산을 크게 줄일 수 있다. 따라서 기본적으로 4x4 블록 단위에서부터 16x16 블록 단위까지 움직임 탐색시 각 위치에 따른 SAD 연산을 4x4 기본 블록 단위로 저장하여 재사용하도록 한다. 주어진 블록의 SAD를 계산하기에 앞서 그 블록 내부의 4x4 블록 단위로 저장된 SAD 값들이 있는지 확인 후 존재하는 경우에는 그 값을 그대로 사용하고 그렇지 않은 내부의 4x4 블록들에 대해서만 SAD 연산을 계산하고 저장한다. 이러한 SAD 연산으로 인하여 추가적으로 필요한 메모리 양을 어느 정도

최소화하기 위해 SAD를 저장할 탐색 범위를 일정 범위 내로 제한하도록 하였다.

이러한 제안된 정수화소 움직임 탐색 기법에 대하여 단계별로 기술하면 다음과 같다.

제안된 정수화소 움직임 탐색

2. 부화소 움직임 탐색

***STEP 1**: 4x4 블록 움직임 탐색*
- 기존 고속 움직임 탐색 기법을 그대로 적용

***STEP 2**: 4x8, 8x4, 8x8 블록 움직임 탐색*
- 해당 4x4 블록 움직임 벡터들의 분산 D를 식 (2)를 통해 구함
- 만일 D값이 0인 경우에는 식 (2)에서 구한 $\overline{MV}_b$를 해당 블록의 움직임 벡터로 결정
- 그 외의 경우에는 $\overline{MV}_b$을 시작점으로 하여 표 1에 나타난 탐색 패턴을 이용하여 다이아몬드 탐색 수행

***STEP 3**: 16x16, 16x8, 8x16 블록 단위 움직임 탐색*
- STEP 2에서 구한 8x8 블록 움직임 벡터들을 이용하여 STEP 2와 동일한 과정을 수행

***STEP 4**: 각 참조 영상에 대하여 STEP 1~3 과정 반복*

H.264 표준의 움직임 벡터는 1/4 화소 단위의 정확도를 갖기 때문에 부화소(Sub-pel) 움직임 탐색이 추가로 필요하다. 부화소 단위 움직임 탐색을 위해서는 먼저 영상 보간(Interpolation)을 통해 1/2위치 및 1/4위치의 각 부화소들을 모두 구해야 한다. 그 다음, 앞서 구한 정수 화소 위치의 움직임 벡터를 중심으로 주변에 위치한 각 부화소 가운데 최소 SAD 값을 갖는 위치 지점을 찾아 최종 움직임 벡터를 확정한다.

이러한 부화소 움직임 탐색을 위해 현재 H.264 참조 코덱인 JM에서는 그림 5(a)와 같이 2단계 탐색을 수행한다. 먼저 정수 화소 단위 움직임 벡터를 중심으로 현 지점을 포함하여 주변 8개의 1/2 화소 위치에 대해 최소 SAD를 갖는 지점을 조사하고(1단계), 최소 SAD를 갖는 지점을 중심으로 다시 주변 8개 1/4 화소 위치에 대해 최소 SAD를 갖는 지점을 조사하여 최종 움직임 벡터로 정한다(2단계).

현재 JM에서는 부화소 움직임 탐색시 보다 효율성을 높이기 위해 블록간 에러 측정 기준을 SAD 대신 SA(T)D (Sum of Absolute Transformed Difference)를 사용할 수 있다. SA(T)D는 예측 블록과 주어진 블록

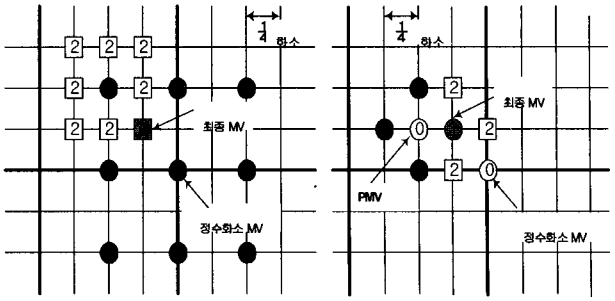


그림 5. 부화소 움직임 탐색의 예 (a) 기존 2단계 탐색 방법 (b) SDSP에 의한 제안 기법

Fig. 5. Examples of Sub-pel motion search: (a) conventional 2-step search method (b) proposed one with SD.

간의 차이값, 즉 잉여값(Residual data)들을 4x4단위 Hadamard 변환한 다음, 각 계수값들의 합으로 나타낸 것이다. 이러한 Hadamard 변환된 계수값들의 에너지를 구함으로써 움직임 보상된 잉여 데이터를 정수 변환(Integer transform)및 양자화했을 때 얻게 될 왜곡을 근사적으로 구할 수 있다. 부화소 단위 움직임 탐색에 필요한 탐색 지점은 그림 5(a)에 나타난 것처럼 총 17개 지점이므로 각 매크로블록 모드별로 4x4 단위 SA(T)D 연산이 17번씩 수행된다. 16x16 모드인 경우 각 탐색 지점마다 16번의 SA(T)D가 수행되며 16x8 및 8x16 모드의 경우에는 한 매크로블록 내에서 각 8번의 SA(T)D가 2차례 수행된다. 그리고 8x8하위 모드의 경우에는 각 8x8 블록 단위별로 8x8, 8x4, 4x8, 4x4의 네 가지 모드에 대하여 4번의 SA(T)D가 수행되므로 각 탐색지점마다 4(8x8)x4(모드)x4(SA(T)D)번 수행된다. 따라서 한 매크로블록 내에서 모든 가변 블록 모드에 대해 수행되는 총 SA(T)D 연산은 다음과 같다.

$$\text{총SA(T)D연산량} = \begin{matrix} 17 \times \{16\,(MB_{16\times 16}) + 8 \times 2\,(MB_{16\times 8}) + \\ 8 \times 2\,(MB_{8\times 16}) + 4 \times 4 \times 4\,(MB_{8\times 8sub})\} \end{matrix} \quad (3)$$

$$= 1094$$

그러므로 부화소 움직임 탐색으로 인하여 하나의 매크로블록 당 총 1094번의 SA(T)D 연산이 필요하며 SAD 연산을 수행하더라도 거의 비슷한 횟수의 연산량이 요구된다. 만일 정수 화소 단위를 위한 고속 움직임 탐색을 적용하는 경우에는 오히려 부화소 움직임 탐색에 수행되는 연산량이 정수 화소 단위보다 더 증가하는 경우가 발생한다. 그러므로 부화소 움직임 탐색을 위한 고속 움직임 탐색 기법이 필요하다.

현재 H.264의 움직임 벡터는 왼쪽, 위쪽, 위오른쪽 주변 블록들의 움직임 벡터의 중간값(Median)인 예측 벡

터와의 차분치를 부호화한다[6]. 그 이유는 움직임 벡터가 공간 상에서 서로 유사도가 높기 때문에 차분 부호화를 통해 엔트로피 부호화 효율을 높이기 위함이다. 실제로 이러한 움직임 벡터 차분치는 그 확률 분포가 기하분포(Geometric distribution)를 갖는 H.264의 Exponential Golomb 부호와 거의 비슷하기 때문에 움직임 벡터를 위한 부호화 효율이 매우 높다[7]. 그러므로 현재 움직임 벡터의 부호 길이를 통해 각 위치별로 움직임 벡터가 존재할 확률 분포를 어느 정도 예측할 수 있다.

그림 6은 PMV (Predictive Motion Vector)를 중심으로 각 움직임 벡터의 부호 길이를 나타낸 것이다. 예를 들어 움직임 벡터가 PMV와 동일한 경우에는 부호 길이는 1비트가 되고 이 지점을 중심으로 시티블록 거리가 1인 주변, 동,서,남,북의 지점에 위치할 경우에는 4비트가 된다. 따라서 이러한 부호 길이가 나타내는 통계적 특성에 따르면 움직임 벡터가 PMV와 동일할 확률은 50% 정도이며 PMV를 포함하여 그 중심으로 SDSP 위치에 움직임 벡터가 존재할 확률이 거의 75%에 이른다는 것을 볼 수 있다.

이러한 확률 모델에 근거하여 본 논문에서는 부화소 움직임 탐색을 위해 PMV를 중심으로 한 SDSP 탐색을 제안한다. 먼저 PMV가 최소 SAD를 갖는 정수 화소 단위 지점, 즉 앞서 구한 정수 화소 단위 움직임 벡터 지점을 중심으로 $\pm 3/4$ 영역에 존재하는 경우에는 예측 벡터와 정수 화소 단위 움직임 벡터 간의 SA(T)D (또는 SAD)를 비교하여 최소값을 갖는 지점을 탐색의 시작점으로 하고 그렇지 않은 경우에는 정수 화소 움직임 벡터를 시작점으로 한다. 이렇게 찾은 시작점을 중심으로 그림 3의 SDSP 탐색을 반복 수행함으로써 최종 움직임 벡터를 결정하게 된다. 이러한 제안 기법에 대한 예가 그림 5(b)에 나타나 있다.

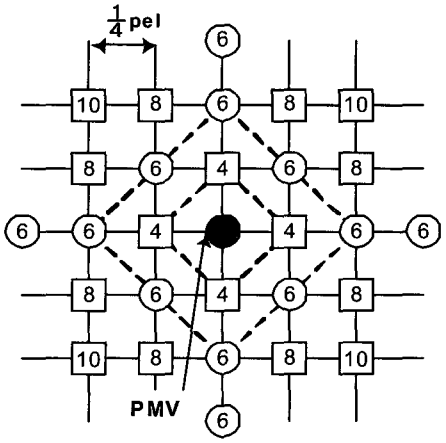


그림 6. 예측 벡터, PMV를 중심으로 주변 각 움직임 벡터의 부호 길이

Fig. 6. Codeword length of motion vectors at surrounding predictive motion vector (PMV).

본 제안 알고리즘을 단계별로 기술하면 다음과 같다.

제안된 부화소 움직임 탐색

STEP 1: *탐색 초기 지점 선택*

- 정수화소 움직임 탐색을 통해 구한 움직임 벡터를 중심으로 PMV가 $\pm 3/4$ 영역 안에 있는 경우, PMV와 정수화소 움직임 벡터간의 SAD 또는 SA(T)D를 비교하여 최소값을 갖는 곳을 탐색 초기 지점으로 함.
- 그렇지 않은 경우에는 정수화소 움직임 벡터를 탐색의 초기지점으로 함

STEP 2: *SDSP 탐색*

- STEP 1의 탐색 초기 지점을 중심으로 그림 3(b)와 같은 SDSP 형태의 움직임 탐색 수행.
- SDSP 탐색을 반복 수행하면서 SAD 또는 SA(T)D의 최소값이 SDSP의 중심점에 나타나게 되는 경우 그 지점을 최종 움직임 벡터로 결정

STEP 3: *위 과정을 각 참조 영상 및 블록 모드에 대해 반복*

Ⅳ. 실험 결과

표 2에 나타난 실험 조건은 H.264 표준화 과정의 객관적 실험 결과 비교를 위해 제안된 공통 조건이다[8]. 본 실험에는 JM 6.1d 코덱과 최적의 부호화 모드 결정을 위한 R-D 최적화 기법 및 부화소 움직임 탐색을 위한 Hadamard 변환이 사용되었다. 또한 본 실험은 H.264 베이스라인 프로파일에 한하여 고려하였기 때문

표 2. 실험 조건

Table 2. Test condition.

영상 (포멧, 프레임율)	News(QCIF,10), Container(QCIF,10), Foreman(QCIF,10), Silent(QCIF,15), Paris(CIF,15), Mobile(CIF,30), Tempete(CIF,30)
QP	28, 32, 36, 40
움직임 탐색 범위	16(QCIF) / 32(CIF)
참조 프레임수	3
부호화 기술	*H.264 베이스라인 프로파일 *에러강인 기술(RS,FMO,ASO) 사용안함. *R-D 최적화 및 Hadamard 변환

표 3. 각 기법별 속도 향상 비교
Table 3. Speedup comparison among each method.

	QP	DS		MVFAST		제안기법	
		S_{SAD}	$S_{SA(T)D}$	S_{SAD}	$S_{SA(T)D}$	S_{SAD}	$S_{SA(T)D}$
Container (QCIF)	28	80.8	1.0	168.4	1.0	1135.0	3.1
	32	81.2	1.0	173.3	1.0	1168.8	3.1
	36	82.0	1.0	170.4	1.0	1166.3	3.1
	40	82.4	1.0	175.3	1.0	1197.7	3.1
Foreman (QCIF)	28	47.3	1.0	95.4	1.0	254.7	2.1
	32	48.0	1.0	98.3	1.0	275.5	2.1
	36	49.3	1.0	102.4	1.0	317.1	2.2
	40	51.2	1.0	109.8	1.0	394.9	2.4
News (QCIF)	28	75.4	1.0	151.1	1.0	752.3	2.8
	32	76.0	1.0	154.2	1.0	810.9	2.8
	36	77.1	1.0	157.5	1.0	851.6	2.8
	40	77.8	1.0	165.1	1.0	936.6	2.9
Silent (QCIF)	28	72.2	1.0	136.7	1.0	574.1	2.5
	32	72.5	1.0	139.4	1.0	621.1	2.5
	36	73.2	1.0	143.2	1.0	703.5	2.6
	40	74.3	1.0	157.1	1.0	873.2	2.8
Mobile (CIF)	28	252.6	1.0	576.6	1.0	3421.9	2.2
	32	253.4	1.0	577.7	1.0	3480.2	2.2
	36	255.3	1.0	568.2	1.0	3511.6	2.2
	40	259.3	1.0	555.4	1.0	3546.1	2.2
Paris (CIF)	28	287.3	1.0	585.6	1.0	2909.5	2.8
	32	288.5	1.0	595.3	1.0	3053.8	2.8
	36	291.3	1.0	607.6	1.0	3216.7	2.8
	40	294.1	1.0	623.2	1.0	3438.9	2.8
Tempete (CIF)	28	267.8	1.0	556.5	1.0	2891.2	2.2
	32	270.5	1.0	560.4	1.0	3033.7	2.2
	36	273.6	1.0	559.2	1.0	3183.9	2.2
	40	277.8	1.0	558.8	1.0	3410.1	2.4
평균		**156.9**	**1.0**	**329.4**	**1.0**	**1826.1**	**2.6**

에 B 슬라이스 및 CABAC 기술은 사용하지 않았다. 그리고 움직임 벡터의 탐색 범위는 QCIF 포멧 영상인 경우 16, CIF 포멧 영상의 경우에는 32로 하여 실험하였다. 또한 제안 기법에서 정수 화소 단위 탐색을 위해 필요한 표 1의 문턱치 값, T_1과 T_2는 각각 1과 8이 사용되었고 SAD 재사용을 적용할 움직임 벡터는 예측 벡터(PMV)를 중심으로 ±8 범위로 제한하였다.

제안된 고속 움직임 탐색 기법에 대해 기존 다이아몬드 탐색(DS) 및 이를 개선한 MVFAST 기법과 성능을 비교하였다. MVFAST의 경우에는 식 (1)의 문턱치 (L_1, L_2)을 (1,2)로 하여 실험하였다. 각 기법의 성능을 비교하기 위해 다음과 같은 성능 지표를 사용하였다.

$$S_{SAD} = \frac{\#\ \mathrm{SAD}_{4x4}\ \text{for full search}}{\#\ \mathrm{SAD}_{4x4}\ \text{in given method}} \quad (4)$$

$$S_{SA(T)D} = \frac{\#\ \mathrm{SA(T)D}_{4x4}\ \text{for full search}}{\#\ \mathrm{SA(T)D}_{4x4}\ \text{in given method}} \quad (5)$$

수식 (4)와 (5)는 모두 각 고속 움직임 탐색 기법들이 FS(Full Search)에 비해 4x4 블록 단위로 SAD 또는 SA(T)D의 연산량 측면에서 얼마나 속도 향상을 가져오는가에 대한 수치를 나타낸 것이다. 식 (4)의 S_{SAD}는 정수 화소 움직임 탐색에 대한 속도 향상이며 식 (5)의 $S_{SA(T)D}$는 부화소 움직임 탐색에 대한 속도 향상이라고 볼 수 있다. 객관적 성능 비교를 위해 FS를 비롯한 모든 탐색에서 Early termination을 이용한 SAD 연산 생략을 적용하지 않도록 제한하였다. 기존 고속 움직임 탐색 기법 및 제안 알고리즘의 성능을 표 3에서 나타내었다.

표 3에서 보는 바와 같이 먼저 정수 단위 움직임 탐

표 4. 제안 기법 및 기존 알고리즘들의 PSNR 및 비트율 비교
Table 4. PSNR and bitrate of the conventional methods and the proposed one.

	QP	FS		DS		MVFAST		제안기법	
		PSNR_Y [dB]	bitrates [Kbps]	PSNR_Y [dB]	bitrates [Kbps]	PSNR_Y [dB]	bitrates [Kbps]	PSNR_Y [dB]	bitrates [Kbps]
Container (QCIF)	28	35.96	23.58	35.95	23.73	35.92	23.53	35.88	24.04
	32	33.18	13.21	33.16	13.18	33.15	13.16	33.21	13.25
	36	30.41	7.73	30.39	7.78	30.34	7.71	30.31	7.74
	40	27.71	4.67	27.70	4.67	27.67	4.66	27.69	4.72
Foreman (QCIF)	28	35.95	70.29	35.85	79.09	35.92	72.27	35.88	74.20
	32	33.17	43.08	33.11	48.60	33.12	44.94	33.09	46.56
	36	30.62	27.04	30.51	30.32	30.47	28.04	30.40	29.53
	40	28.12	17.19	27.92	19.12	27.95	18.40	27.88	19.34
News (QCIF)	28	36.68	44.59	36.67	44.77	36.66	45.11	36.65	45.22
	32	33.67	27.57	33.65	27.73	33.64	27.87	33.60	27.99
	36	30.79	16.90	30.77	17.16	30.71	17.13	30.68	17.10
	40	28.10	10.42	27.99	10.44	28.07	10.61	28.02	10.53
Silent (QCIF)	28	35.80	57.96	35.78	58.98	35.8	58.92	35.76	59.01
	32	32.92	34.95	32.90	35.28	32.89	35.33	32.88	35.54
	36	30.36	20.45	30.35	20.72	30.33	20.72	30.29	20.87
	40	27.91	11.81	27.89	12.03	27.92	11.94	27.85	12.26
Mobile (CIF)	28	33.87	1559.31	33.85	1564.17	33.86	1560.00	33.85	1563.75
	32	30.43	761.76	30.41	762.18	30.41	760.50	30.38	762.54
	36	27.37	358.89	27.34	359.94	27.33	359.67	27.30	360.87
	40	24.49	190.62	24.46	192.03	24.45	192.27	24.40	192.87
Paris (CIF)	28	35.45	313.74	35.43	319.05	35.42	317.64	35.39	318.90
	32	32.35	181.76	32.31	185.96	32.3	185.21	32.27	186.39
	36	29.47	101.46	29.41	104.42	29.4	104.04	29.36	105.11
	40	26.79	57.42	26.73	59.01	26.73	59.10	26.68	59.96
Tempete (CIF)	28	34.81	1149.33	34.80	1150.95	34.8	1151.82	34.78	1147.53
	32	31.60	548.67	31.58	549.90	31.57	550.38	31.55	547.77
	36	28.75	265.23	28.72	266.07	28.73	266.97	28.70	264.93
	40	26.11	141.48	26.06	142.38	26.07	142.56	25.99	142.59

색에 필요한 SAD 연산의 경우, DS 기법은 FS에 비해 SAD 연산이 평균적으로 약 1/150로 줄어들었으며 이것은 SAD 연산 측면에서 150배 속도 증가(Speed up)을 가져왔다고 말할 수 있다. MVFAST의 경우에는 FS와 비교해 볼 때 평균적으로 약 330배 가까이 속도 증가가 있었는데 이것은 기존 DS 기법에 비해 두 배 정도 속도가 증가하였다고 볼 수 있다. 제안 기법의 경우에는 정수 단위 움직임 탐색을 위해 기본 벡터의 분포에 따른 적응적 탐색과 각 가변 블록별 SAD 재사용을 통하여 FS에 비해 평균 1800배 이상 속도 향상을 가져왔다. 이것은 기존 DS 기법에 비해 약 12배, MVFAST 기법에 대해서는 약 5.4배 정도 속도가 향상된 것이다. 움직임 탐색 범위가 16인 QCIF 영상에서는 FS에 대한 SAD 속도 증가는 평균 750배 정도이며 움직임 탐색 범위가 32인 CIF 영상의 경우에는 평균 3250배 가까이 되었다. 표 3에서 Foreman 영상의 경우에는 상대적으로 전체 움직임이 크기 때문에 다른 영상에 비해 속도 향상 측면에서 모든 기법들의 성능이 떨어지는 것을 볼 수 있다.

또한 부화소 움직임 탐색시 필요한 SA(T)D에 대하여 기존 기법들과 달리 제안 기법은 2단계 탐색에 비해 평균 2.6배 속도 향상을 가져 왔다. 즉, 제안 기법에서는 부화소 단위 움직임 탐색에 있어서 거의 절반 이상 SA(T)D 연산을 줄일 수 있다. Hadamard 변환을 이용한 SA(T)D의 경우 추가적 변환을 위한 계산이 필요한 것을 생각할 때 상당한 속도 개선이 발생한 것을 예상할 수 있다.

표 4는 각 기법에 따른 PSNR 및 비트율을 비교한 것이다. 율제어(Rate control) 알고리즘을 적용하지 않았기 때문에 각 양자화 계수에 따른 각 PSNR과 비트율을 각 영상별로 나타내었다. DS 및 MVFAST의 경

표 5. 제안 기법 및 기존 알고리즘들의 수행 시간 비교
Table 5. Time saving comparison between the conventional methods and the proposed one.

	QP	정수화소			부화소	
		DS	JM_FME	제안기법	JM_FME	제안기법
Container (QCIF)	28	75.13%	84.52%	96.87%	52.86%	79.71%
	32	74.76%	87.18%	94.29%	35.21%	69.85%
	36	70.30%	87.62%	99.12%	48.23%	65.24%
	40	78.31%	84.46%	94.68%	39.04%	57.07%
Foreman (QCIF)	28	54.38%	67.15%	90.77%	34.97%	51.50%
	32	61.41%	72.57%	87.98%	23.78%	43.58%
	36	59.52%	80.12%	89.95%	25.06%	43.01%
	40	66.69%	78.91%	90.90%	26.08%	61.98%
News (QCIF)	28	76.82%	77.08%	89.07%	33.14%	69.43%
	32	73.21%	83.82%	90.76%	38.71%	59.66%
	36	73.71%	82.61%	99.61%	45.51%	60.77%
	40	71.30%	82.46%	93.84%	15.87%	46.47%
Silent (QCIF)	28	68.33%	89.02%	82.91%	30.04%	55.14%
	32	75.32%	77.10%	87.06%	22.92%	51.35%
	36	69.50%	86.89%	97.61%	25.64%	63.06%
	40	68.54%	81.36%	88.54%	21.06%	51.20%
Mobile (CIF)	28	91.91%	88.80%	98.54%	28.98%	49.21%
	32	91.91%	89.44%	97.93%	34.04%	54.82%
	36	92.57%	88.31%	98.07%	32.16%	49.81%
	40	92.11%	89.12%	97.97%	29.93%	44.12%
Paris (CIF)	28	92.18%	92.40%	97.85%	44.22%	56.81%
	32	92.91%	93.34%	98.80%	39.78%	57.52%
	36	93.69%	92.44%	98.70%	35.71%	60.51%
	40	92.83%	91.98%	98.99%	34.05%	57.72%
평 균		77.39%	84.53%	94.20%	33.21%	56.65%

우 FS에 비해 PSNR 측면에서는 모두 평균 약 0.04dB 정도 손실이 있으며 비트율 측면에서는 각각 3%, 1%의 비트량이 증가하였다. 이에 반해 제안 기법은 FS에 비해 0.08dB 정도 PSNR 손실과 2% 정도의 비트율 증가가 발생하였다. 역시 기존 기법을 비롯한 제안 기법에서도 Foreman 영상의 경우 압축 효율 측면에서 성능 저하가 가장 크게 발생하였다. 이것은 다른 영상에 비해 빠른 움직임을 갖는 영상의 이질적 특성 때문인데 이러한 특성에 좀더 적응하도록 제안 기법의 문턱치를 조절할 필요가 있다.

본 제안 기법이 H.264 부호화기에 구현되었을 경우 실제로 얻을 수 있는 속도 향상을 알아보기 위하여 현재 JM에 구현되어 있는 고속 전역 탐색, FFS(Fast Full Search)를 기준으로 비교대상 방법에 대해 다음과 같이 절감된 시간 TS (Time Saving)를 측정하였다.

$$TS = \frac{T_{ME}(FFS) - T_{ME}(\text{선택방법})}{T_{ME}(FFS)} \quad (6)$$

여기서 $T_{ME}(FFS)$는 FFS 탐색 기법의 총 수행시간을 의미하는 것으로 전체 부호화기 시스템에서 움직임 탐색(ME) 모듈에 해당하는 수행시간을 초단위로 측정한 값이다. 따라서 TS값이 커질수록 상대적으로 FFS 탐색에 비하여 수행 속도의 절감이 그만큼 증가한다는 것을 의미한다. 본 실험에서 FS 대신 특별히 FFS를 기준으로 속도 향상에 관한 성능을 비교하는 이유는 FFS가 가변 블록 단위 움직임 보상을 위해 본 논문에서 적용한 SAD 재사용 기법을 역시 적용하여 FS의 속도를 향상시킨 기법이기 때문이다. 또한 JM을 이용하여 실험하는 대부분의 경우에는 움직임 탐색을 위해 FS 대신 FFS 기법을 이용하기 때문이기도 하다. T_{ME}(선택방법)은 비교할 선택 방법에 대한 움직임 탐색 모듈의 수행시간이다.

표 5는 이러한 FFS에 대한 제안 기법의 속도 향상을 나타낸 것이다. 제안 기법의 성능을 기존 고속 움직임

탐색 기법과 비교하기 위하여 DS 기법을 비롯하여 JVT 표준화 회의에 채택된 고속 움직임 탐색 기법[9] 과도 비교하였다(편의상 JM_FME로 표시). 현재 JM에도 구현되어 있으며 JM_FME 기법은 기존 고속 움직임 탐색 알고리즘들의 국부 최소치 문제를 해결하기 위하여 정수 화소 움직임 탐색 시 광역 탐색(Global search)을 적용함으로써 화질 열화를 최소화한 기법이다. 표 5에서 보는 바와 같이 먼저 정수화소 움직임 탐색을 보면 FFS를 기준으로 DS 기법의 경우 평균 77.39%, JM_FME의 경우에는 평균 84.53% 정도 시간을 단축시켰다. 이에 반해 제안기법은 평균 94.20% 가량 움직임 탐색의 수행 시간을 단축시켜 기존 기법들과 비교해 보아도 가장 우수한 성능을 나타낸 것을 볼 수 있다. 또한 부화소 움직임 탐색의 경우에도 JM의 2단계 탐색에 비해 JM_FME가 평균 33.21% 가량 시간 단축시킨데 반해 제안 기법은 56.65%로 절반이상 수행시간을 단축시킬 수 있었다.

V. 결 론

본 논문은 가변 블록 움직임 보상의 계층적 구조를 이용하여 4x4 블록 단위 움직임 벡터의 분포에 따라 적응적으로 움직임 벡터를 탐색하는 기법을 제안하였다. 또한 각 가변 블록에 있어서 SAD 재사용을 통해 추가적으로 SAD 연산량을 줄일 수 있었으며 뿐만 아니라 부화소 움직임 탐색의 속도 향상을 위한 기법도 제안하였다. 현재 H.264의 움직임 보상은 복수 참조 프레임 및 가변 블록을 이용한 움직임 보상을 통해 높은 부호화 효율을 갖지만 동시에 다양한 경우에 따른 움직임 탐색 수행으로 인하여 부호화기 전체 복잡도를 증가시키는 주요 원인이 되고 있다. 실험 결과, 제안 기법은 기존 고속 움직임 탐색 기법에 비해 큰 화질 열화없이 수행 시간이나 SAD 연산 수 측면에서 모두 높은 속도 향상을 보임으로써 H.264 부호화기의 복잡도 저감에 큰 역할을 할 것으로 기대한다. 제안 기법에 적용된 임계값들을 보다 영상 특성에 맞도록 적응시킴으로써 화질 열화 및 속도 향상 측면에서 개선될 것으로 기대한다.

참 고 문 헌

[1] 이제윤, 최웅일, 전병우, 석민수, "H.264의 가변 블록 움직임 보상을 위한 고속 움직임 벡터 탐색 및 모드 결정법", 대한전자공학회논문지 SP편 제40권 제4호, 49-59쪽, 2003년 7월.

[2] S. Zhu and K. Ma, "A new diamond search algorithm for fast block matching," IEEE Trans. on Image Proc., Vol. 9, No. 2, pp. 287-290, February 2000.

[3] P. Hosur and K. Ma, "Motion vector field adapti -ve fast motion estimation," Second International Conference on Information, Communications and Signal Processing (ICICS '99), Singapore, 7-10, December 1999.

[4] A. Tourapis, O. Au, and M. Liou, "Highly effi -cient predictive zonal algorithms for fast block -matching motion estimation," IEEE Transac -tions on Circuits and Systems for Video Technology, pp. 934-947, October 2000.

[5] Y. Itoh, "Bi-directional motion vector coding using universal vlc," Signal processing: Image commu -nication 14, pp. 541-557, May 1999.

[6] JVT and video study group, "Text of ISO/IEC 14496-10 FDIS Advanced Video Coding," in ISO/IEC JTC1/SC29/WG11 MPEG03/W5555, March 2003.

[7] 최웅일, 전병우, 유국열, 천강욱, "고정 재배정 테이블 기반 동적 UVLC부호화 방법", 대한전자공학회논문지 제39권 SP편 제2호, 56-68쪽, 2002년 3월.

[8] G. Sullivan, G. Bjontegaard, "Recommended Simulation Common Conditions for H.26L Coding Efficiency Experiments on Low-Resolution Progressive-scan Source Material," ITU-T Q.6/16, Doc. #VCEG-N81, September 2001.

[9] Zhibo Chen, Peng Zhou, Yun He, and Yidong Chen, "Fast Integer Pel and Fractional Pel Motion Estimation for JVT," JVT-F017, December 2002.

저 자 소 개

최 웅 일(학생회원)

2000년 성균관대학교 전기·전자·컴퓨터공학부 학사 졸업.
2002년 성균관대학교 전기·전자·컴퓨터공학과 석사 졸업.
2002년~현재 성균관대학교 전기·전자·컴퓨터공학과 박사 과정.

<주관심분야: 영상압축, 영상통신, 영상처리>

전 병 우(정회원)

1985년 서울대학교 전자공학과 학사 졸업.
1987년 서울대학교 전자공학과 석사 졸업.
1992년 Purdue Univ. School of Elec. 박사 졸업.
1993년~1997년 삼성전자 신호처리연구소 수석연구원
1997년~현재 성균관대학교 정보통신공학부 부교수

<주관심분야: 영상인식, 영상압축, 디지털 신호처리>