논문 2007-44SP-3-3

H.264에서 주변 움직임 벡터를 이용한 고속 부 화소 탐색 제어 기법

(Fast Sub-pixel Search Control by using Neighbor Motion Vector in H.264)

라 병 두*, 엄 민 영*, 최 윤 식*

(Byeongdu La, Minyoung Eom, and Yoonsik Choe)

요 약

H.264에서 움직임 예측은 전체 부호화 시간의 큰 비중을 차지함으로써 실시간 적용을 위해서 반드시 개선되어야할 부분이다. 이런 문제점 개선을 위해 제안된 알고리즘들의 대부분은 부호화에서 실시간 적용에 문제되는 시간을 줄이고자 전체 검색의 화질을 목표로 빠른 검색 알고리즘을 제안해 왔으며 부 화소관련 알고리즘도 이와 같다. 본 논문은 이런 알고리즘들의 접근과는 다르게 움직임 영상별 부호화된 움직임 예측을 분석하여 이를 바탕으로 현재 매크로 블록의 정수 화소 움직임 벡터와 이미 부호화된 주변 3개 움직임 벡터를 이용하여 부 화소 탐색 진행 여부를 결정하는 것을 제안한다. 또한 분석된 움직임 벡터의 수평/수직 방향성을 이용, 부 화소에서의 수평/수직 방향만의 탐색을 제안하고 위의 두 제안 사항에 대한 적용 결과를 보여 준다.

Abstract

Motion Estimation time in the H.264 has a large portion of encoding time and must be improved for real time application. Most of proposed motion estimation algorithm including Sub-pixel search use the fast search algorithm to speed up motion estimation by targeting the performance of full search in the reference code. This paper proposes a novel fast sub-pixel search control algorithm for H.264 encoder by using neighbor motion vector after analyzing the encoded Motion vector of video sequence. In addition the horizontal/vertical searching method is proposed with the horizontal/vertical directionality of motion vector. And the evaluation is performed with the proposed algorithms and other reference algorithms.

Keywords : H.264, Motion Estimation, Sub-pixel Search

I. 서 론

H.264/AVC는 JVT(Joint Video Team), ITU-T의 VCEG(Video Coding Expert Group)과 ISO/IEC[1]의 MPEG(Motion Picture Expert Group)에 의해 개발된 새로운 영상 부호화 표준이다. H.264/AVC는 연속적인 영상들 간의 시간적 중복을 줄이기 위한 움직임 예측이 중요한 부분을 차지하며 실제 부호화 시간의 약 60%를 차지할 정도로 많은 시간을 소모하는 부분이다. 특히 정확한 움직임 벡터 예측과 압축 효율을 높이기 위해서 많은 영상의 참조와 16x16~4x4까지의 더 많은 서브 블록이 적용됨으로써 복잡성과 많은 계산이 급격히 증가되었다. 움직임 예측의 기본 동작은 정수 화소 탐색과 부 화소 탐색의 두 단계로 진행되며 부 화소 탐색은 다시 확정된 정수 화소 움직임 벡터를 중심으로 진행하는 1/2 화소 탐색과 확정된 1/2 화소 움직임 벡터를 중심으로 추가적인 탐색을 진행하는 1/4 화소 탐색으로 나누어 실행된다. 움직임 예측의 구성 요소로는 탐색 화면 크기, 정합 정도 측정을 위한 도구, 블록 정합 탐색 알고리즘이 있다. 탐색 화면 크기는 보통 기본 매크

* 정회원, 연세대학교 전기전자공학과
(Dept., Electrical & Electronics Eng. Yonsei University)
접수일자: 2006년9월12일, 수정완료일: 2007년4월9일

로 블록 크기의 2배를 탐색 영역으로 설정하며 H.264에서는 중심 점 기준 ± 16(즉 32)을 기준으로 함으로써 1089개의 위치를 탐색한다. 정합 정도 측정을 위한 도구로 H.264에서는 정수 화소 탐색에서는 SAD(Sum of Absolute Difference)가 사용되고 부 화소 탐색에서는 사용 가능한 SAD와 SATD(Sum of Absolute Transformed Difference) 중에서 정확도가 높은 SATD가 주로 사용된다. 블록 정합 탐색 알고리즘은 최대 수준을 낼 수 있는 전체 탐색을 기준으로 최소의 시간으로 전체 탐색의 수준에 근접하는 빠른 탐색 알고리즘들이 제안되고 있으며 그 예로 ARPS-3[2], ARPS-4[3], 2SS[4], FTS[5][6][7]등이 있다. ARPS-3(Adaptive Rood Pattern Search)는 정수 화소 움직임 벡터의 수평/수직 방향성을 분석, 십자가형의 적응적 탐색을 수행하고, ARPS-4(개선된 ARPS-3)는 ARPS-3에 정수 화소 움직임 예측에 대해서는 적응적 문턱 값과 이른 종료를 추가하고 부 화소 움직임 예측에 대해서는 주변 블록 움직임 벡터의 부 화소 일치 위치를 현재 블록의 탐색 위치로 이용하고 문턱 값과 이른 종료를 적용한다. 2SS(Two Step Search)는 정수 화소 검색 이후 수평 방향 2개의 위치 중 최소 SAD를 갖는 위치를 수직 탐색 중심으로 설정한 후 수직 방향 2개의 위치에 대해서 최소의 SAD를 갖는 위치를 최종 부 화소 움직임 벡터로 확정하며, FTS(Flexible Triangle Search)는 정수 좌표에 존재하는 3개의 꼭지점을 이용하여 삼각형의 크기를 키우고 줄여 가면서 최적 위치를 찾는 방법을 사용한다.

본 논문에서는 부 화소 탐색 이전에 현재 매크로 블록의 결정된 정수 움직임 벡터와 현재 매크로 블록 주변의 코딩된 움직임 벡터를 분석하고 이를 이용하여 부 화소 탐색 생략 여부를 결정하는 아이디어를 제안한다. 그리고 결정된 부 화소 움직임 벡터의 많은 움직임 벡터가 수평/수직 방향에 존재함을 분석하여 이를 이용한 아이디어를 제안하고 2가지 제안된 알고리즘 적용시의 실험 결과를 제시한다. 이 논문의 구성은 다음과 같다. SectionⅡ에서는 움직임 예측 방법을 고찰하고 SectionⅢ에서는 기존 제안된 2SS알고리즘을 분석하고 SectionⅣ에서는 움직임 벡터 분석 및 제안하여 SectionⅤ와 Ⅵ에서는 실험 및 결과 그리고 결론을 맺는다.

Ⅱ. H.264에서 움직임 예측 방법 고찰

움직임 예측은 이전에 코딩된 주변 매크로 블록의 움직임 벡터를 참조하여 현재의 움직임 벡터를 예측하는 것으로 시작된다. 그림 1[8]과 같이 현재 매크로 블록의 크기가 이전에 코딩된 주변 분할 블록 크기와 같은지 다른지 혹은 참조할 수 있는 주변 매크로 블록이 있는지 확인한다. 주변 매크로 블록 A, B, C의 움직임 벡터 a(매크로 블록 A의 움직임 벡터), b(매크로 블록 B의 움직임 벡터), c(매크로 블록 C의 움직임 벡터)의 활용 여부와 위치에 따라 예측 움직임 벡터가 결정되어 탐색을 위한 중심 위치로 정해진다. 정해진 중심을 기준으로 각 블록 형태의 정해진 탐색 영역에 따라 최소의 왜곡을 갖는 위치를 찾아 그 블록의 움직임 벡터로 확정한다. 이렇게 각 블록 형태에 따라 움직임 예측이 끝나면 각 블록 형태별 움직임 벡터를 이용하여 최종 코딩을 위한 모드 결정이 진행된다.

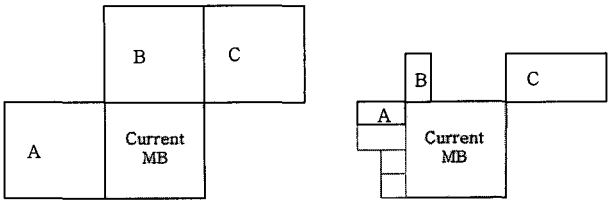

그림 1. 현재의 매크로 블록과 이웃 블록들
Fig. 1. Current MB and neighbor partitions.

Ⅲ. 기존의 2SS 알고리즘 분석

이 알고리즘은 1/2 화소에 대해 수평 화소 탐색 후 수직 화소 탐색하는 빠른 탐색을 제안한 것으로 구성은 다음과 같다. 움직임 보상에서 현재 블록은 참조 화면의 하나의 블록에 의해서 대체되므로 이 예측된 블록은 다음과 같이 나타낼 수 있다.

$$\widehat{B_k}(x,y) = B_{k-1}(x+dx, y+dy) \tag{1}$$

이탈 정도를 나타내는 움직임 벡터로서 dx는 수평을 dy는 수직의 벗어난 정도를 나타내며 부 화소의 경우 다음과 같이 나타낼 수 있다.

$$\begin{aligned} dx &= dx_i + dx_h \\ dy &= dy_i + dy_h \end{aligned} \tag{2}$$

여기에서 dx_i는 수평 방향의 정수 화소 움직임 벡터를 나타내고 dy_i는 수직 방향의 정수 화소 움직임 벡터를 나타낸다. dx_h는 수평 방향의 부 화소 움직임 벡터를 나타내고 dy_h는 수직 방향의 부 화소 움직임 벡터를 나타낸다. 정수 화소를 기준으로 보간된 화면에서 1/2 화소 탐

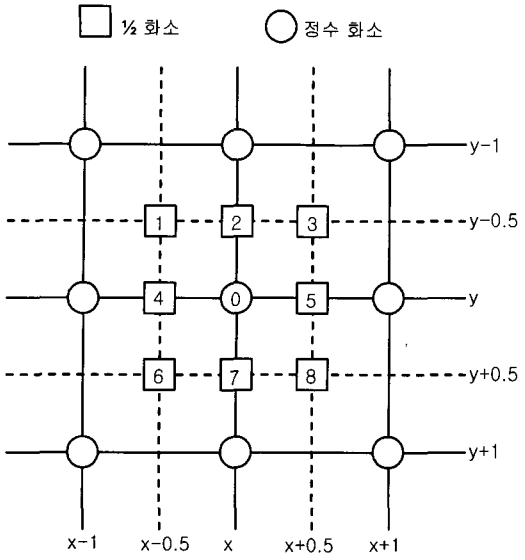

그림 2. 1/2 화소와 정수 화소의 위치
Fig. 2. The positions of half-pixels and integer-pixels.

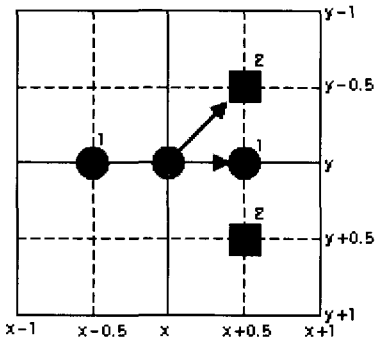

그림 3. 2SS 알고리즘 (움직임 벡터의 1/2 화소 예)
Fig. 3. The 2SS process. The half-pixel component of motion vector is (0.5, -0.5) in this example.

색이 수행된다.

전체 부 화소 검색은 그림 2와 같이 정수 화소 탐색에 의해서 확정된 정수 화소 움직임 벡터가 (0,0) 위치라고 하면 그 (0,0) 위치와 주변 보간된 1/2 화소 8개의 위치가 1/2 화소 탐색에 의해 수행되고 최소 SAD를 갖는 위치가 정해진다. 정수 화소 탐색에 의해 결정된 정수 화소 움직임 벡터 주변 ± 1 화소 이내의 SAD 오류 표면은 96% 이상이 단봉형의 형태를 갖게 되므로 탐색 위치가 정수 화소 탐색 결과의 ± 1 화소 이내에서 1/2 화소 최소로부터 이동함에 따라 SAD 왜곡 함수도 단조롭게 증가할 것이라는 결론을 얻게 된다. 즉 1/2 화소의 SAD 왜곡 함수가 작을수록 이웃에 있는 최소값에 가까워진다. 이에 따라 1/2 화소 움직임 벡터 탐색은 수평/수직 방향에 있는 이웃에서 최소 SAD값을 찾게 되며 알고리즘은 다음과 같다. 첫 번째 단계로 그림 3와 같이 정수 화소 탐색 결과인 움직임 벡터를 초기 탐색 중심으로 설정한 후 수평 탐색 동안 $(x_1-0.5, y_1)$, 그 (x_1, y_1)리고 $(x_1+0.5, y_1)$ 위치에서 SAD를 계산하고 최소 SAD를 갖는 위치를 수직 탐색 중심(x_2, y_2)로 설정한다.

두 번째 단계로 수직 탐색인 $(x_2, y_2-0.5), (x_2, y_2)$ 그리고 $(x_2, y_2+0.5)$위치에 대해 최소 SAD를 갖는 위치를 찾아 1/2 화소 움직임 벡터로 결정한다. 정수 화소 탐색결과인 정수 움직임 벡터가 화면의 상하 혹은 좌우에 존재하면 탐색 위치는 더욱 줄게 된다. 기본적으로 탐색 위치는 전체 탐색 시의 8개 위치에서 4개 위치로 줄게 된다.

IV. 움직임 벡터 분석 및 제안

검토에 사용된 CODEC(enCOder/DECoder)은 JM10.1 참조 코드이고 90개의 화면에서 각각의 매크로 블록에 대해 결정된 정수/부 화소의 움직임 벡터 성향을 분석하고자 9개의 움직임 영상에 대해서 16x16 블록 형태에서 결정된 움직임 벡터를 정리하여 분석을 진행한다. 움직임 벡터는 블록 형태에 따라 너무 많은 자료가 존재하므로 공정성과 편의성을 위해 16x16으로 통일하여 검토 진행하며 움직임 영상에 대한 표기상의 편의를 위해 표 1처럼 정의하여 사용한다. 움직임 영상은 90개 화면을 사용했고 비트 율 조절은 "사용 안함"으로, 양자화 계수는 28로 고정한 상태에서 실험을 수행했다. 실험에서 제시되는 모든 자료 중에서 특별히 언급되지 않은 부분은 혼란을 피하기 위해서 1개의 움직임 영상 당 전체 90개 화면에 대한 99개 매크로 블록의 숫자에 대한 발생 수를 점유율(%)로 나타낸 것이다.

표 2를 보면 정수 화소에서 움직임 벡터가 (0,0)인 경우가 전체의 약 17~87%를 차지하고 수평 혹은 수직이 0인 경우를 포함하면 약 67~97%에 이름을 알 수 있다. 1/2 화소에서 움직임 벡터가 (0,0)인 경우는 약 32~82%, 수평 혹은 수직이 0인 경우를 포함하면 약 80~95%에 해

표 1. 9 가지 움직임 영상
Table 1. The index of 9 video sequences.

A	B	C	D	E
Bike	Car phone	Flower garden	Foreman	Missa
F	G	H	I	
Mother & daughter	Silent	Susie	Table tennis	

표 2. 정수/부 화소 움직임 벡터 분석(단위 : %)
Table 2. Analyzing the integer/sub-pixel MV (unit: %).

Seq Typ	Integer-pixel		Half-pixel				Quarter-pixel			
	(0,0)	(0,0) (0,1) (1,0)	(0,0)	8 point	(0,1) (1,0)	(0,0) (0,1) (1,0)	(0,0)	8 point	(0,1) (1,0)	(0,0) (0,1) (1,0)
A	33.25	67.62	45.64	54.36	34.39	80.02	50.61	49.39	34.08	84.69
B	50.80	86.15	47.86	52.14	37.23	85.09	39.50	60.50	40.82	80.32
C	17.67	94.71	32.12	67.88	60.54	92.66	40.40	59.60	46.96	87.37
D	35.98	74.96	43.38	56.62	39.51	82.89	40.19	59.81	40.26	80.44
E	82.29	97.90	73.88	26.12	19.90	93.78	65.16	34.84	25.46	90.61
F	86.80	96.58	72.99	27.01	21.14	94.13	54.07	45.93	33.49	87.56
G	86.15	95.38	82.24	17.76	13.18	95.41	69.64	30.36	21.80	91.44
H	53.08	83.17	48.12	51.88	34.33	82.45	42.79	57.21	34.77	77.56
I	30.75	67.38	47.55	52.45	37.92	85.47	40.48	59.52	41.31	81.80

당함을 알 수 있다. 1/4 화소에서 움직임 벡터가 (0,0)인 경우는 약 39~69%, 수평 혹은 수직이 0인 경우를 포함하면 약 77~91%에 해당함을 알 수 있다. 이를 통해 알 수 있듯이 움직임 벡터에서 (0,0)인 경우와 수평/ 수직 방향인 특성을 찾아냄으로써 움직임 예측의 효율화를 기대해 볼 수 있음을 알 수 있다.

표 3은 그림 1에서 주변 매크로 블록인 A, B, C의 움직임 벡터 a, b, c와 현재 정수 화소 움직임 벡터, 그리고 1/2, 1/4 화소 탐색에 따라 결정된 움직임 벡터를 뽑아서 서로간의 연관성을 분석한 것이다. 움직임 벡터 a, b, c, i(정수 화소 움직임 벡터), h(1/2 화소 움직임 벡터), q(1/4 화소 움직임 벡터)는 움직임 벡터가 각각 (0,0)일 확률이고 "abc"는 주변 움직임 벡터 a, b, c가 전부 (0,0)일 확률로 2~76%이며 영상의 움직임 정도에 따라 달라진다. "ihq"는 정수, 1/2, 1/4 화소 움직임 벡터 전부가 (0,0)일 확률로 1~76%의 분포를 갖는다. 동작이 많은 영상일수록 움직임 벡터가 (0,0)일 가능성은 줄어든다. 여기에서 실제로 중요한 사항은 주변 움직임 벡터 a, b, c가 각각 (0,0)인 조건에서 현재 매크로 블록의 움직임 벡터의 성향이다. 표에서 볼 수 있듯이 현재 매크로 블록의 정수 화소 움직임 벡터가 확정된 후 1/2, 1/4 화소 탐색이 수행되므로 주변 움직임 벡터 a, b, c가 각각 (0,0)이고 현재 매크로 블록의 정수 화소 움직임 벡터가 (0,0)인 조건이 "abci"가 된다. "abci"에서 1/2, 1/4 화소 탐색까지 (0,0)인 조건에서 1/2, 1/4 화소 탐색을 생략할 때 제일 정확한 방법이 되겠지만 1/2, 1/4 화소는 미리 알 수 없으므로 "abci"조건에서 1/2, 1/4 화소 탐색을 생략할 경우 이때의 오류 확률을 예측할 수 있다. 즉 "abci"에서 "abcihq"를 뺀 확률인

표 3. 주변 움직임 벡터와 정수/부 화소 움직임 벡터의 관계
Table 3. Relations between neighbor MV and integer/sub-pixel MV (unit: %).

Seq Type	a, b, c, i, h, q (x=0,y= 0)						conditional possibility				Error
	a	b	c	i	h	q	abc	ihq	abci	abcihq	
A	36.4	37.0	36.0	33.3	45.6	50.6	22.4	19.9	19.6	16.8	2.8
B	32.2	36.4	35.0	50.8	47.9	39.5	15.6	18.7	12.0	8.5	3.5
C	11.6	14.2	14.3	17.7	32.1	40.4	2.5	1.2	1.0	0.6	0.4
D	20.4	23.2	20.8	36.0	43.4	40.2	6.5	10.2	5.9	3.8	2.1
E	70.9	74.1	72.6	82.3	73.9	65.2	56.4	53.8	53.0	42.3	10.7
F	57.6	61.3	57.1	86.8	73.0	54.1	38.7	46.6	37.4	29.4	8.0
G	73.4	76.2	76.2	86.2	82.2	69.6	58.8	64.9	56.8	50.8	6.0
H	38.8	45.6	43.3	53.1	48.1	42.8	22.9	24.6	19.5	14.9	4.5
I	34.6	34.6	34.5	30.7	47.6	40.5	18.3	17.8	15.0	12.2	2.8

"Error" 위치가 그에 해당하는 확률을 나타낸다. 계산된 "Error" 위치를 참조하여 여러 움직임 영상 적용에 따른 차이를 예상할 수 있다. Idea 1은 1/2과 1/4 화소 탐색 이전에 움직임 벡터 예측으로 사용하는 주변 매크로 블록의 움직임 벡터인 a, b, c를 별도의 배열에 저장하고 정수 화소 움직임 벡터가 확정된 후 주변 움직임 벡터 a, b, c가 전부 (0,0)이고 정수 화소 움직임 벡터가 (0,0)인 경우 1/2, 1/4 화소 탐색을 생략한다. 프로그램에서의 적용 방법은 다음과 같다.

Idea 1 : 현재 매크로 블록의 움직임 벡터와 주변 움직임 벡터 참조하여 부 화소 탐색 결정.

```
if(!(a==0 && b==0 && c==0 && MV==0))
  {
    Sub_Pixel_search();
  }
```

사람의 눈은 고정된 상태에서 멈춰있는 물체를 보거나 움직이는 물체를 보며, 본인이 움직이면서 다른 정지해 있는 물체를 보거나 다른 물체가 움직이는 것을 보는 경우가 대부분이다. 평지에서 움직이는 사물을 보면 멈춰있거나 좌우로 움직이는 것이 대부분이다. 카메라로 사물을 촬영할 때도 이와 동일한 성향을 가지며 영상의 방향은 대부분 멈춰있거나 수평/수직 방향의 성향을 가진다. 따라서 화면의 변화를 반영하는 움직임 벡터는 영상의 이런 성향을 갖게 된다. 표 2에서 볼 수 있듯이 부 화소의 경우 (0,0)과 (1,0) 그리고 (0,1)의 확률을 합하면 약 77~95%를 차지함을 알 수 있다. 나머지는 사선 방향으로 약 23%이하를 차지함을 알 수 있다.

표 4. 부 화소 움직임 예측에서 움직임 벡터의 수평/수직 방향성

Table 4. The directionality of sub-pixel MV (unit : %).

	Half-pixel search						Quarter-pixel search					
	8 point	(0,-1)	(0,1)	(-1,0)	(1,0)	sum	8 point	(0,-1)	(0,1)	(-1,0)	(1,0)	sum
A	54.4	7.5	7.8	9.5	9.6	34.4	49.4	9.1	9.8	7.3	7.9	34.1
B	52.1	9.6	10.0	8.9	8.7	37.2	60.5	10.9	11.6	9.6	8.7	40.8
C	67.9	3.7	1.8	29.0	26.1	60.5	59.6	7.3	4.0	17.9	17.8	47.0
D	56.6	9.2	9.6	11.0	9.8	39.5	59.8	9.4	9.7	10.8	10.3	40.3
E	26.1	5.4	3.6	4.0	6.9	19.9	34.8	7.3	5.3	5.1	7.8	25.5
F	27.0	5.8	5.4	3.0	6.8	21.1	45.9	7.0	9.0	6.9	10.6	33.5
G	17.8	2.4	2.1	4.5	4.2	13.2	30.4	4.0	4.8	6.8	6.2	21.8
H	51.9	13.3	8.1	6.2	6.7	34.3	57.2	10.4	10.0	6.8	7.6	34.8
I	52.5	8.1	7.3	11.0	11.6	37.9	59.5	7.5	8.8	12.2	12.8	41.3

표 4는 1/2, 1/4 화소 움직임 예측에서 수평/수직 방향성을 상세히 나누어 분석한 것이다. 표시된 8개의 위치는 (0,0) 움직임 벡터를 제외한 것으로 8개 위치의 전체 확률을 나타내며 "sum"은 수평/수직 방향의 움직임 벡터를 갖는 수를 점유율(%)로 나타낸 것이다. Idea 2는 부 화소 탐색을 수평과 수직 방향만(수평 혹은 수직 움직임 벡터가 0인 경우) 하는 것으로 프로그램에의 적용은 다음과 같다.

Idea 2 : 부 화소 탐색에서 수평/수직 방향만 탐색.

```
if(!(MV_x * MV_y))
{
  Half_Pixel_search();
}
if(!(MV_x * MV_y))
{
  Quarter_Pixel_search();
}
```

Idea 3은 Idea 1과 Idea 2를 동시에 적용한 것으로 다음과 같다.

Idea 3 : Idea 1과 Idea 2를 동시에 적용

```
if(!(a==0 && b==0 && c==0 && MV==0))
  {
    if(!(MV_x * MV_y))
    {
      Half_Pixel_search();
    }
    if(!(MV_x * MV_y))
    {
      Quarter_Pixel_search();
    }
  }
```

V. 실험 및 결과

제안된 알고리즘은 JVT에서 제공된 JM10.1 참조 코드를 사용하였다. 실험 조건은 다음과 같다.

1) 실험 이미지: 9개 QCIF 각각 90 frame
   적용된 이미지는 표 1 참조
2) 율-왜곡 최적화 : On
3) 인트라 주기 : 0(첫 번째만 I 프레임)
4) Rate Control : Off, 양자화 계수 : 28
5) 각 방법에 대해 SATD로 움직임 벡터 결정
6) ΔPSNR : 전체 검색 기준 비교 (+:증가, -:감소)
7) 평균 비트 율 증감 계산 (+:증가, -:감소)

$$\Delta B/R(\%)=\frac{Bit-rate(proposed)-Bit-rate(full\_search)}{Bit-rate(full\_search)}\times 100$$

8) 평균 계산 시간 증감 계산 (+:증가, -:감소)

$$\Delta Time(\%)=\frac{Time(proposed)-Time(full\_search)}{Time(full\_search)}\times 100$$

위에서 제안된 아이디어를 기준으로 움직임 벡터 예측 시 참조하는 주변 움직임 벡터를 별도의 배열에 저장하고 현재 매크로 블록의 정수 화소 움직임 벡터를 결정하였다. 이를 바탕으로 주변 움직임 벡터 a, b, c와 정수 화소 움직임 벡터가 (0,0)이면 1/2, 1/4 화소 탐색을 생략하는 방법을 Idea 1(I_1)이라 하고 부 화소인 1/2, 1/4 화소에 대해 수평/수직 방향만 탐색하는 것을 Idea 2(I_2)로, Idea 1과 Idea 2를 같이 적용한 방법을 Idea 3로 정의하고 적용하였다. 또한 적용된 기법의 상대적 비교를 위해서 참조 코드의 빠른 전체 탐색 방법과 빠른 탐색관련 제안된 논문 중 2SS(Two Step Search)논문의 기법을 프로그램에 반영하여 도출된 자료를 제안된 방법과 비교하였다.

표 5, 6, 7은 영상에 따른 각 기법의 적용 결과를 보여준다. 표의 제일 좌측은 적용된 알고리즘을 나타내고 표의 제일 위는 표 1에서 구분지어 준 각각의 영상을

표 5. 검색 방법에 따른 움직임 예측 시간 비교
Table 5. Comparison of ME $\Delta Time$(unit : %).

	A	B	C	D	E	F	G	H	I
I_1	-5.3	-3.1	-1.2	-1.5	-12.3	-8.7	-12.0	-5.2	-2.7
2SS	-6.3	-6.1	-5.3	-4.4	-5.4	-5.1	-6.1	-5.7	-4.6
I_1+2SS	-9.4	-8.0	-5.6	-7.3	-11.2	-11.9	-12.9	-10.1	-7.5
I_3	-10.1	-10.4	-5.6	-7.6	-14.5	-11.6	-15.8	-11.8	-10.0
I_2	-6.3	-8.8	-5.9	-7.8	-7.9	-10.0	-8.2	-7.5	-8.4

표 6. 검색 방법에 따른 부 화소 검색 시간 비교
Table 6. Comparison of Sub-pixel $\Delta Time$ (unit : %).

	A	B	C	D	E	F	G	H	I
I_1	-29.9	-18.6	-7.6	-9.3	-64.5	-48	-70.0	-28.1	-15.0
2SS	-35.5	-35.8	-32	-26.2	-28.7	-28.2	-35.7	-30.8	-25.3
I_1+2SS	-52.5	-47.3	-34.1	-43.5	-59.1	-65.3	-75.1	-54.5	-41.5
I_3	-56.6	-60.7	-33.9	-45.4	-76	-63.5	-91.6	-63.6	-55.2
I_2	-35.5	-51.7	-35.8	-46.4	-41.7	-54.8	-47.7	-40.7	-46.1

표 7. 검색 방법에 따른 $\Delta B/R$ 비교
Table 7. Comparison of $\Delta B/R$(Bit-rate, unit : %).

	A	B	C	D	E	F	G	H	I
I_1	0.86	2.11	-0.06	1.93	1.57	2.52	1.69	0.03	0.17
2SS	1.18	3.51	0.46	3.93	2.04	2.02	0.88	2	0.67
I_1+2SS	1.51	5.15	0.53	6.7	4.06	5.17	1.97	2.06	0.98
I_3	2.88	5.2	0.88	5.95	7.2	5.5	2.84	3.75	1.65
I_2	2.77	3.78	0.86	3.92	1.92	2.82	0.88	3.18	1.54

나타낸다.

표 5를 통해서 정수와 부 화소 전체를 포함한 움직임 예측에서 I_3 방법이 E 영상에서 약 14%의 시간 감소로 효과가 큰 것을 알 수 있고 표 6과 그림 5를 통해서 부 화소만을 기준으로 I_3 방법이 G 영상에서 약 91%의 시간 감소로 효과가 큰 것을 알 수 있다. 표 7은 각 방법에 따른 Bit-rate을 비교한 것으로 I_3 방법이 E 영상에서 약 7%의 증가가 발생함을 알 수 있다. I_1과 2SS를 동시에 적용하면 I_3보다 안정적이고 빠른 검색이 이루어 짐을 볼 수 있다. 그림 4는 각 방법에 따른 PSNR을 비교한 것으로 -0.06dB가 가장 낮은 수준이다. Idea 1(I_1), Idea 2(I_2)에서 특정 움직임 영상의 PSNR이 전체 탐색보다 높은 경우가 있는데 이는 Bit-rate가 더 많이 할당되어 좋아진 것임을 알 수 있다.

Idea 1의 경우 그림 4에서 볼 수 있듯이 움직임 영상 G에서 PSNR 감소가 있지만 B나 D에서는 오히려 상승함을 알 수 있고 Idea 1의 경우 그림 5처럼 움직임이 많은 움직임 영상 C나 D에서는 움직임 예측 시간의 감소가 두드러지지 않으나 정지 특성이 강한 움직임 영상 E나 G의 경우 움직임 예측 시간의 감소가 큼을 알

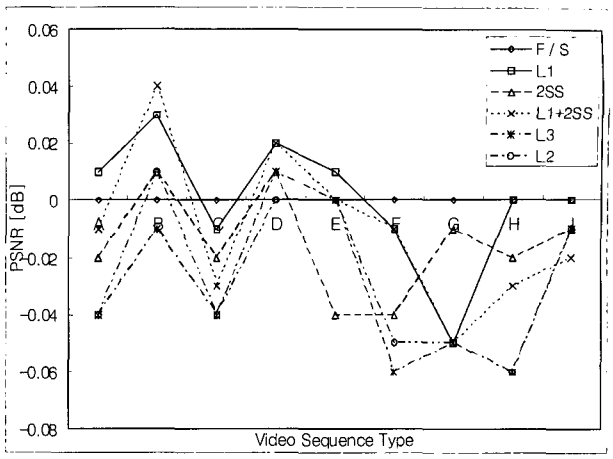

그림 4. 빠른 전체 탐색 기준으로 PSNR 비교
Fig. 4. Comparison of the PSNRs.

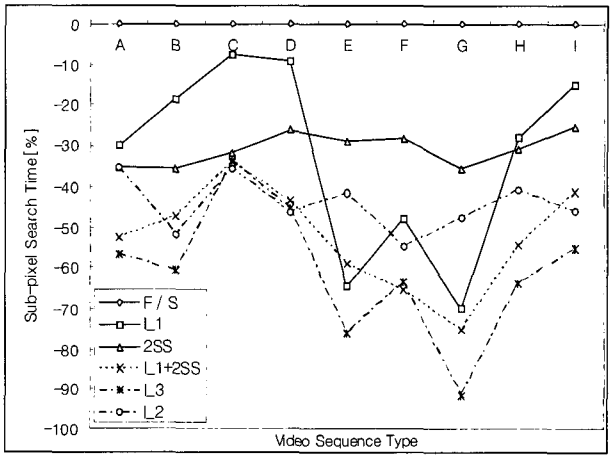

그림 5. 부 화소 검색 시간 비교
Fig. 5. Comparison of the Sub-pixel Search time saving.

수 있다. Idea 2의 경우 PSNR과 움직임 예측 시간 모두 단조로운 변화를 가져감을 알 수 있다.

VI. 결 론

움직임 예측에서 현재 매크로 블록의 정수 화소 움직임 벡터가 확정된 후 주변 움직임 벡터 3개(a, b, c)와 확정된 정수 화소 움직임 벡터가 (0,0)이면 1/2, 1/4 화소 탐색을 생략하는 Idea 1 방법은 움직임의 변화가 적은 영상에 크게 효과가 있어서 움직임 예측의 시간 감소에 많은 효과가 있는 것을 알 수 있다. 또한 1/2, 1/4 화소 탐색에 대해서 수평/수직 방향만 탐색하는 Idea 2의 경우 Idea 1보다 움직임 영상에 따라 효과의 폭은 작지만 일정 수준의 안정된 효과를 나타낸다. Idea 1과 Idea 2를 같이 적용한 Idea 3의 경우 JM10.1 참조 코드의 빠른 전체 탐색에서 부 화소에 걸린 시간을 평균 약

61% 줄이는 효과가 있었다. Idea 1은 1/2, 1/4 화소 탐색 이전에 진행 여부를 결정하므로 향후 다른 빠른 탐색 알고리즘과 연계하여 적용하면 움직임 예측의 효율성 개선에 더욱 효과가 있으리라 본다.

참 고 문 헌

[1] Thomas Wiegand, Gary J. Sullivan, Gisle Bjontegaard, and Ajay Luthra, "Overview of the H.264/AVC Video Coding Standard", IEEE Transaction on Circuits and System for Video Technology, Vol.13, No.7, pp.560-576, July 2003.

[2] Kai-Kuang Ma and Gang Qiu, "Unequal-Arm Adaptive Rood Pattern Search for Fast Block-Matching Motion Estimation in the JVT/H.26L," IEEE, vol.1, pp. 901-904, Sept, 2003.

[3] Wei Zhenyu, Jiang Baochenn, Zang Xudong and Chen Yu, "A new Full-pixel and Sub-pixel Motion Vector Search Algorithm for Fast Block-matching Motion Estimation in H.264," IEEE, pp. 587-590, 20-22 Oct, 2004.

[4] Bo Zhou and Jian Chen, "A Fast Two-step Search Algorithm for Half-pixel Motion Estimation," IEEE, vol. 2, pp. 611-614, May, 2003.

[5] M. Rehan, P. Agathoklis and A. Antoniou, "Flexible Triangle Search Algorithm for Block Based Motion Estimation," IEEE, vol. 1, pp. 233-236, Aug, 2003.

[6] M. Rehan, P. Agathoklis and A. Antoniou, "Block-based Motion Estimation using an Enhanced Flexible Triangle Search Algorithm," Canadian Conference, pp. 269-272, May, 2005.

[7] M. Rehan, P. Agathoklis, "Half-pixel Accurate Motion-Estimation using a Flexible Triangle Search," IEEE, pp. 257-260, Aug, 2005.

[8] Iain E. G. Richardson, "H.264 and MPEG-4 Video Compression," Willey, pp. 176, May, 2004.

저 자 소 개

라 병 두(정회원)

1994년 건국대학교 전자공학과 학사 졸업.
2001년 아주대학교 정보전자 공학과 석사 졸업
2005년~현재 연세대학교 전기전자공학과 박사과정
1994년 삼성전자 수원 Display 사업부 입사
2001년~현재 삼성전자 기흥 반도체 미디어 개발팀(SYS.LSI) 책임 연구원
<주관심분야 : 영상 신호처리, 반도체 SOC>

최 윤 식(정회원)

1979년 연세대학교 전기공학과 학사 졸업
1984년 Case Western Reserve Univ. 시스템공학과 졸업.
1987년 Pennsylvania State Univ. 전기공학과 석사 졸업
1990년 Purdue Univ. 전기공학부 박사 졸업
1990년~1993년 (주)현대전자 산업전자 연구소 책임 연구원
1993년~현재 연세대학교 전기전자공학부 정교수
<주관심분야 : 비디오, 영상 신호처리, HDTV>

엄 민 영(정회원)

2001년 연세대학교 전파공학과 학사 졸업.
2004년 연세대학교 전기전자 공학과 석사 졸업.
2004년~현재 연세대학교 전기전자공학과 박사과정
<주관심분야 : 웨이블릿, 비디오, 영상신호처리>