# 움직임 추정을 위한 개선된 다단계 연속 제거 알고리즘

## (AMSEA: Advanced Multi-level Successive Elimination Algorithms for Motion Estimation)

정 수 목 † 박 명 순 ††

(Soo-Mok Jung) (Myong-Soon Park)

**요 약** 본 논문에서는 블록 정합 알고리즘(BMA: block matching algorithm)인 다단계 연속 제거 알고리즘(MSEA: multi-level successive elimination algorithm)[1]의 연산량을 줄이기 위하여 네 가지 방안을 제안하였다.

첫 번째 제안 방안은 MSEA에서 서브 블록(sub block)의 합 놈(sum norm)에 대한 절대 오차의 합(SAD: sum of absolute difference)을 계산할 때 부분 왜곡 제거(PDE: partial distortion elimination) 기법을 적용하여 연산량을 감소시킨 알고리즘이다. 두 번째 제안 방안인 적응 SAD 계산 알고리즘은 SAD 계산 시 절대 오차가 큰 값에서부터 작은 값의 순으로 SAD를 계산하면 PDE가 빨리 발생하게 되어 연산량을 줄일 수 있는 성질을 이용한 알고리즘이다. 세 번째 제안 방안인 제거 레벨 추정 알고리즘은 탐색점의 제거 레벨을 추정하고 추정된 레벨에서부터 상위 레벨로 다단계 연속 제거 과정을 수행함으로 추정된 제거레벨보다 낮은 레벨들과 연관된 연산량을 감소시킨 알고리즘이다. 제안된 첫 번째, 두 번째, 세 번째 방안은 움직임 추정의 정확도가 전역 탐색 알고리즘(FSA: full search algorithm) 및 MSEA와 동일하면서 MSEA의 연산량을 효과적으로 감소시킨 알고리즘들이다. 네 번째 제안 방안인 나선형 다이아몬드 그물 탐색 알고리즘은 움직임 추정의 정확도가 거의 100%이면서 움직임 추정에 필요한 연산량을 획기적으로 감소시킨 고속 블록 정합 알고리즘이다.

위의 네 가지 제안 방안에 대한 성능을 평가하기 위하여 실험을 수행하였으며 실험에서 제안 방안들의 효율성을 확인하였다.

**키워드** : 움직임 추정, 블록 정합 알고리즘, 연속제거 알고리즘, 움직임 벡터

***Abstract*** In this paper, we present advanced algorithms to reduce the computations of block matching algorithms for motion estimation in video coding. Advanced multi-level successive elimination algorithms(AMSEA) are based on the Multi-level successive elimination algorithm (MSEA)[1].

The first algorithm is that when we calculate the sum of absolute difference (SAD) between the sum norms of sub-blocks in MSEA, we use the partial distortion elimination technique. By using the first algorithm, we can reduce the computations of MSEA further. In the second algorithm, we calculate SAD adaptively from large value to small value according to the absolute difference values between pixels of blocks. By using the second algorithm, the partial distortion elimination in SAD calculation can occur early. So, the computations of MSEA can be reduced. In the third algorithm, we can estimate the elimination level of MSEA. Accordingly, the computations of the MSEA related to the level lower than the estimated level can be reduced. The fourth algorithm is a very fast block matching algorithm with nearly 100% motion estimation accuracy.

Experimental results show that AMSEA are very efficient algorithms for the estimation of motion vectors.

**Key words** : notiobn estimation, block matcing algorithm, successive elimination algorithm, motion vector

† 정 회 원 : 삼육대학교 컴퓨터학과 교수
jungsm@syu.ac.kr
†† 종신회원 : 고려대학교 컴퓨터학과 교수
myongsp@iLab.korea.ac.kr
논문접수 : 2000년 11월 13일
심사완료 : 2001년 10월 29일

## 1. 서 론

연속적인 비디오 프레임(video frame)에는 매우 큰 시간적 중복성(temporal redundancy)이 존재한다. 시간적 중복성을 제거하기 위하여 움직임 추정(motion estimation)과 움직임 보상(motion compensation) 기법이 이미지 시퀀스(image sequence)를 코딩(coding)하는 방법으로 널리 사용되어져 왔다. 움직임 추정의 정확도(accuracy)와 효율성(efficiency)은 시간적 중복성을 제거하는 효율에 영향을 미친다[2][3].

움직임 추정 방법은 블록 정합 알고리즘(BMA: block matching algorithm)[4][5]과 화소 순환 알고리즘(PRA: pel-recursive algorithm)[6] 두 그룹으로 나누어진다. 구현의 간단함 때문에 블록 정합 알고리즘이 CCITT H.261[7], ITU-T H.263[8] 그리고 MPEG[9] 등 여러 비디오 코딩 표준(video coding standards)으로 널리 채택되어 왔다.

블록 정합 알고리즘에서는 현재 프레임(current frame)이 고정된 크기의 정 사각 블록들로 나누어진다. 나누어진 각 블록의 움직임 벡터(motion vector)는 이전 프레임(previous frame) 상에 설정된 탐색 윈도우 내에서 정합 기준(matching criteria)에 따라 최적 정합 블록(the best matching block)을 찾은 후, 최적 정합 블록과의 상대적인 변위를 계산함으로 구해진다.

블록 정합 알고리즘 중 전역 탐색 알고리즘은 이전 프레임 상에 정의된 탐색 윈도우 내의 모든 정합 블록 후보(candidate matching block)들 중에서 최적 정합 블록을 찾음으로 최적 움직임 벡터(global optimum motion vector)를 찾지만, 전역 탐색을 수행하기 위하여 매우 많은 연산량을 필요로 한다. 따라서 실제적인 응용에서는 전역 탐색 알고리즘이 사용되지 않는다.

전역 탐색 알고리즘의 연산량을 감소시키기 위하여 2차원 로그 탐색(two dimensional logarithmic search)[5], 직교 탐색(orthogonal search)[10], 교차 탐색(cross search)[11], 3단계 탐색(TSS: three-step search)[12] 일차원 전역 탐색(one-dimensional full search)[13], 변형 3단계 탐색(variation of three-step search)[14][15] 등 매우 많은 고속 블록 정합 알고리즘들이 개발되어 왔다. 이러한 고속 블록 정합 알고리즘들은 움직임 보상된 잔여 에러 면(motion compensated residual error surface)이 움직임 벡터의 변위에 대한 볼록 함수(convex function)라는 것을 가정하고 있다[16]. 그러나 이러한 가정은 거의 실제적이지 않다[17]. 그러므로 이러한 고속 블록 정합 알고리즘들을 사용하여 얻어진 움직임 벡터는 근본적으로 국소적 최적치(local optimum)가 된다. 다르게 표현하면 대부분의 고속 블록 정합 알고리즘들은 움직임 추정의 정확도를 희생하여 연산량을 감소시킨다.

Li와 Salari에 의해서 제안된 연속 제거 알고리즘(SEA: successive elimination algorithm)[2]은 이러한 볼록성 가정(convexity assumption) 없이 전역 탐색 알고리즘의 연산량을 감소시켰다. 그 후 Wang 등이 SEA의 연산량을 감소시키기 위하여 세 가지 알고리즘을 제안하였다[3]. 또한 X. Q. Gao 등이 SEA의 연산량을 감소시키기 위하여 MSEA[1]를 제안하였다. 그리고 저자들이 MSEA 및 SEA의 연산량을 감소시키기 위하여 여러 기법들을 제안하였다 [18]-[21]. 본 논문은 MSEA의 연산량을 감소시킨 알고리즘들로서 네 가지 알고리즘으로 구성되어 있다. 제안된 첫 번째, 두 번째 그리고 세 번째 방안은 모두 전역 탐색 알고리즘 및 MSEA와 동일한 움직임 추정 정확도를 가지면서 MSEA의 연산량을 감소시켜 움직임 추정의 속도를 향상시키는 효율적인 알고리즘들이다. 특히 제안된 두 번째 방안은 MSEA뿐만 아니라 모든 종류의 블록 정합 알고리즘에 적용되어 움직임 추정에 필요한 연산량을 감소시키는 매우 유용한 알고리즘이다. 제안된 네 번째 방안은 움직임 추정의 정확도가 거의 100%이며 MSEA의 연산량을 획기적으로 감소시킨 고속 블록 정합 알고리즘이다.

본 논문의 구성은 다음과 같다. 2장에서 MSEA에 대하여 간략히 기술하였고, 3장에서 개선된 다단계 연속 제거 알고리즘(AMSEA)을 제안하였으며, 제안된 방안별로 실험 결과를 기술하였다. 마지막으로 4장에서 결론을 맺었다.

## 2. 다단계 연속 제거 알고리즘(MSEA)

$f_c(i,j)$와 $f_p(i,j)$는 현재 프레임과 이전 프레임 상에서 좌표가 $(i,j)$인 화소(pixel)의 휘도(intensity) 값을 각각 나타내고, 블록(H.263에서 매크로블록의 Y성분)의 크기를 $(N)$x$(N)$, 탐색 윈도우 크기를 $(2M+1)$x$(2M+1)$, 정합 기준 함수로 두 블록 사이의 왜곡을 나타내는 절대오차의 합(SAD: sum of absolute difference)을 사용하는 것으로 한다. $\boldsymbol{B}_c^{(i,j)}$와 $\boldsymbol{B}_p^{(i,j,v_x,v_y)}$를 각각 현재 프레임과 이전 프레임 상에서 좌측 상단 점(top left conner)이 $(i,j)$와 $(i-x,\ j-y)$인 참조 블록과 탐색 윈도우 내에 있는 정합 블록 후보라고 둔다. $\boldsymbol{B}_c^{(i,j)}$는 움직임 벡터를 구하고자 하는 참조 블록이다. 이때 블록내 임의의 점 $(m,n)$에 대하여 식 (1), (2)가 성립한다.

$$\boldsymbol{B}_c^{(i,j)}(m,n) = f_c(i+m,\ j+n) \quad (1)$$

$$\boldsymbol{B}_p^{(i,j,x,y)}(m,n) = f_p(i-x+m,\ j-y+n) \quad (2)$$

여기서 $x$와 $y$는 움직임 벡터 후보(candidate motion vector)의 좌표를 나타내고 $-M \le (x,y) \le M$ 조건을 만족한다. (m,n)은 $0 \le (m,n) \le N-1$ 조건을 만족하는 값이다. 두 블록사이의 SAD는 식 (3)과 같이 정의된다.

$$\mathrm{SAD}(x,y) = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1}\left|\boldsymbol{B}_c^{(i,j)}(m,n) - \boldsymbol{B}_P^{(i,j,x,y)}(m,n)\right| \quad (3)$$

움직임 추정의 목표는 식 (4)와 같이 최소 SAD를 갖는 최적의 $(x,y)$ 값을 찾는 것이다.

$$d = \min_{x,y} \mathrm{SAD}(x,y) \quad (4)$$

수학적인 부등식 $|\,\|X\|_1 - \|Y\|_1\,| \le \|X-Y\|_1$ [22]를 $X=B_c^{(i,j)}$와 $Y=B_p^{(i,j,x,y)}$에 대하여 적용하면 식 (5)를 얻을 수 있다.

$$|R-M(x,y)| \le \mathrm{SAD}(x,y) \quad (5)$$

여기서

$$R = \left\|\boldsymbol{B}_c^{(i,j)}\right\|_1 = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1}\boldsymbol{B}_c^{(i,j)}(m,n)$$

$$M(x,y) = \left\|\boldsymbol{B}_p^{(i,j,x,y)}\right\|_1 = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1}\boldsymbol{B}_p^{(i,j,x,y)}(m,n)$$

$$\mathrm{SAD}(x,y) = \left\|\boldsymbol{B}_c^{(i,j)} - \boldsymbol{B}_p^{(i,j,x,y)}\right\|_1 = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1}\left|\boldsymbol{B}_c^{(i,j)}(m,n) - \boldsymbol{B}_p^{(i,j,x,y)}(m,n)\right|$$

$R$과 $M(x,y)$는 각각 참조 블록과 정합 블록 후보의 sum norm 들이고 이 값들은 [1]에 기술된 효과적인 방법으로 미리 계산된다.

MSEA에서, 각 블록은 여러 개의 서브 블록(sub-block)들로 나누어진다. 먼저 블록은 크기가 $(N/2)$x$(N/2)$인 4개의 서브 블록으로 나누어진다. 그 후 다시 각 서브 블록은 크기가 $(N/4)$x$(N/4)$인 4개의 서브 블록으로 나누어진다. 이러한 절차가 서브 블록의 크기가 2x2가 될 때까지 반복된다. 블록의 크기가 $(N)$x$(N)$인 경우, 최대 분할 레벨은 $L_{max}=\log_2 N-1$ 이 된다. $L$-level 분할을 가지는 다단계 연속 제거 알고리즘을 $L$-level MSEA라고 하며 이때 $L$은 $0 \le L \le L_{max}$ 조건을 만족하는 값이다. 따라서 SEA는 0-level MSEA에 대응된다. $l$번째 레벨에서 서브 블록들의 수는 $S_l = (2^l)\mathrm{x}(2^l) = 2^{2l}$ 이고 각 서브 블록의 크기는 $(N_l)$x$(N_l)$이다. 이때 $N_l=N/2^l$ 이다.

$l$번째 레벨에서 $R_l^{(u,v)}$, $M_l^{(u,v)}(x,y)$, $\mathrm{SAD}_l^{(u,v)}(x,y)$, $\mathrm{SAD_SB}_l(x,y)$을 아래와 같이 정의한다.

$$R_l^{(u,v)} = \sum_{m=0}^{N_l-1}\sum_{n=0}^{N_l-1}\boldsymbol{B}_c^{(i,j)}(m+uN_l,\ n+vN_l) \quad (6)$$

$$M_l^{(u,v)}(x,y) = \sum_{m=0}^{N_l-1}\sum_{n=0}^{N_l-1}\boldsymbol{B}_p^{(i,j,x,y)}(m+uN_l,\ n+vN_l) \quad (7)$$

$$\mathrm{SAD}_l^{(u,v)}(x,y) = \sum_{m=0}^{N_l-1}\sum_{n=0}^{N_l-1}\left|\boldsymbol{B}_c^{(i,j)}(m+uN_l,\ n+vN_l) - \boldsymbol{B}_p^{(i,j,x,y)}(m+uN_l,\ n+vN_l)\right| \quad (8)$$

$$\mathrm{SAD\_SB}_l(x,y) = \sum_{u=0}^{2^l-1}\sum_{v=0}^{2^l-1}\left|R_l^{(u,v)} - M_l^{(u,v)}(x,y)\right| \quad (9)$$

여기서 $(u,\ v)$는 서브 블록의 취자(index) 이고 $0 \le (u,v) \le 2^l-1$ 이다. $l$은 $0 \le l \le L_{max}$ 조건을 만족하는 값이다. 0 레벨에서, $u$, $v$는 0이고 $R_l^{(u,v)}$, $M_l^{(u,v)}(x,y)$, $\mathrm{SAD}_l^{(u,v)}(x,y)$는 식 (5)에 정의된 $R$, $M(x,y)$, $\mathrm{SAD}(x,y)$와 같다. 그리고 $l$번째 레벨에서 식 (10)이 성립함을 쉽게 알 수 있다.

$$\mathrm{SAD}(x,y) = \sum_{u=0}^{2^l-1}\sum_{v=0}^{2^l-1}\mathrm{SAD}_l^{(u,v)}(x,y) \quad (10)$$

따라서 식 (9)의 $\mathrm{SAD_SB}_l(x,y)$은 다음과 같이 분류될 수 있다.

$$\mathrm{SAD\_SB}_l(x,y) = \begin{cases} |R-M(x,y)| & l=0 \\ \sum_{u=0}^{2^l-1}\sum_{v=0}^{2^l-1}\left|R_l^{(u,v)} - M_l^{(u,v)}(x,y)\right| & 1 \le l \le L \\ \mathrm{SAD}(x,y) & l=L+1 \end{cases} \quad (11)$$

식 (6) ~ 식 (11)을 이용하여 식 (5)가 식 (12)로 표현될 수 있음이 [1]에서 증명되었다.

$$\mathrm{SAD\_SB}_l(x,y) \le \mathrm{SAD\_SB}_{l+1}(x,y) \quad (12)$$

여기서 $l$은 $0 \le l \le L_{max}$ 조건을 만족하는 값이다.

식 (12)는 블록을 서브블록으로 분할하여 만들어지는 각 서브블록간 sum norm의 차의 절대치의 합인 $\mathrm{SAD\_SB}_l$[식 (11)]이 가지는 수학적인 성질을 나타낸 것으로, $l$이 $0 \le l \le L_{max}$ 인 경우 $\mathrm{SAD_SB}_l$은 항상 SAD보다 작거나 같으며 $l$이 작아질수록 $\mathrm{SAD\_SB}_l$ 값이 작아지는 수학적인 성질을 나타낸다. 이 수식을 아래와 같이 정합블록후보를 제거하는데 사용하면 연산량을 매우 효과적으로 줄일 수 있다. 수식 (12)를 기초로 하는 MSEA의 흐름도를 Nassi-Schneiderman chart로 나타내면 그림 1과 같다.

움직임 추정과정에서 현재까지의 최적정합블록보다 더 정합이 잘되는 새로운 최적정합블록이 있는지 찾기 위해서는 정합블록후보에서 SAD를 계산하여 현재까지의 최적정합블록이 가지는 SAD(curr_min_SAD)와 비교하는 정합 평가(matching evaluation: MSEA의 절차 중 step 5)를 수행하여야한다. 이때 정합블록후보에서 구해진 SAD가 curr_min_SAD보다 작은 값을 가질 때 정합블록후보는 새로운 최적정합블록이 되지만, 정합블

록후보에서 구해진 SAD가 curr_min_SAD보다 크거나 같게 되면 정합블록후보는 새로운 최적정합블록이 될 수 없어 제거된다. 이때 정합블록 후보에서 계산되는 SAD연산은 계산 집중적인 연산임으로 매우 많은 연산량을 필요로 한다. 따라서 정합블록후보 제거 시, 매우 많은 연산량을 필요로 하는 SAD를 사용하지 않고 정합블록후보를 제거할 수 있다면 연산량을 감소시킬 수 있다.

MSEA에서는 식 (12)의 성질을 기초로 연산량이 작은 $SAD_SB_l$을 사용하여 정합블록후보가 새로운 최적 정합블록이 될 수 없음을 SAD계산이전에 효율적으로 알 수 있어 연산량을 줄일 수 있다. $SAD_SB_l$의 계산은 [1]에서 제안된 효과적인 방법으로 구해지고 SAD보다 훨씬 적은 연산량이 소요된다. 또한 $l$의 값이 작을수록 $SAD_SB_l$계산에 필요한 연산량은 작아진다.

따라서 수식 (12)에 기초한 MSEA에서는 정합블록후보를 제거하기 위하여, 가장 연산량이 작은 $SAD_SB_0$를 계산 후 curr_min_SAD와 비교한다. 이때 curr_min_SAD가 $SAD\_SB_0$ 보다 작거나 같으면 식 (12)에 따라 curr_min_SAD는 SAD 보다 반드시 작거나 같게 됨으로 새로운 최적 정합 블록이 될 수 없어 정합 블록 후보는 제거된다. 이때 정합 블록 후보가 0-level에서 제거된다고 한다(MSEA 흐름도의 step 4, i=0). 그러나 curr_min_SAD가 $SAD\_SB_0$ 보다 크게 되면 정합 블록 후보를 제거하기 위하여 $l$을 증가시키면서 정합 블록 후보를 제거하는 과정을 반복한다. $L$-level MSEA에서는 정합 블록 후보를 제거하기 위한 과정을 최고 $L$-level까지 수행할 수 있다. 만약 $L$-level MSEA의 경우 0-level에서부터 $L$-level까지 정합 블록 후보를 제거하기 위한 과정을 반복하였어도 제거되지 않았다면 SAD를 계산하여 curr_min_SAD와 비교한다.(step 5) 이때 curr_min_SAD가 SAD보다 작거나 같으면 정합 블록 후보는 제거되나, SAD가 작게되면 정합 블록 후보는 새로운 최적 정합 블록이 되고 curr_min_SAD는 SAD 값으로 대치된다.(step 6)

MSEA를 적용하는 경우 0-level에서 많은 정합블록 후보들이 제거되고, 특히 $L_{max}$-level 까지 수행하면 대부분의 정합 블록 후보들이 제거되기 때문에, MSEA 흐름도 step 5의 SAD계산 전에, 작은 연산량이 소요되는 $SAD_SB_l$만을 이용하여 step 4에서 정합 블록 후보들을 제거할 수 있어 연산량을 획기적으로 줄일 수 있다.

이러한 과정이 탐색 윈도우 내에 있는 모든 탐색점에 대하여 수행된 후의 최종 curr_min_SAD가 최소 SAD가 되며 최소 SAD를 가지는 블록이 최적 정합 블록이 된다. 이 때 최적 정합 블록의 좌표에서 참조 블록의 좌표를 뺀 상대적인 변위가 참조 블록의 움직임 벡터가 된다.

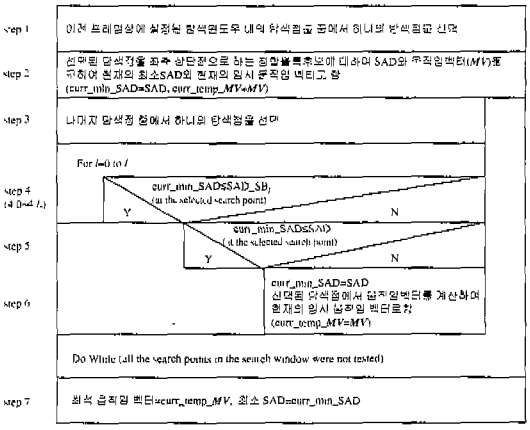


그림 1 $L$-level MSEA 흐름도

## 3. 개선된 다단계 연속 제거 알고리즘(AMSEA : Advanced Multi-level Successive Elimination Algorithms)

### 3.1 $SAD\_SB_l$ 계산에 PDE기법 적용(PDE technique in $SAD\_SB_l$ Calculation)

부분 왜곡 제거(PDE: partial distortion elimination) 기법[23]은 벡터 양자화에서 벡터 코드워드의 셋(a set of vector code-words)으로부터 최상의 재건 벡터(reconstruction vector)를 찾는데 효율적인 속도 향상 기법이다. 전역 탐색 알고리즘 실행 시 SAD 계산에 PDE 기법을 적용하여 연산량을 줄이는 방안이 H.263 표준화 노력(standardization effort) 일환으로 배포된 테스트 모델 1.4a의 텔레노 구현(Telenor implementation)에서 적용되었다[8]. 그리고 SEA의 SAD계산에 PDE기법을 적용하는 방안이 [3]에서 제안되었다.

MSEA에서는 PDE 기법이 SAD 계산에 적용되었으나 $SAD\_SB_l$ 계산에는 적용되지 않았다. 다단계 연속 제거 알고리즘에서 $l$번째 레벨까지 탐색점이 제거되지 않을 경우 $SAD_SB_l$을 계산하기 위하여 참조블록과 정합 블록 후보 각각에서 고려되어야 할 서브 블록의 수는 식 (13)과 같다.

$$\sum_{i=0}^{l} 2^{2i} \quad (\text{단, } 0 \le l \le L_{max}) \tag{13}$$

따라서 SAD_SB$_l$을 계산하여야 하는 $L$-level MSEA에서 연산량을 더욱 감소시키기 위하여 SAD_SB$_l$ 계산에 PDE 기법을 적용할 수 있다.

수식 (9)와 $L$-level MSEA$_L$의 step 4(4.0~4.$L$)에서, 만약 어떤 정합 블록 후보에 대하여 부분적으로 계산된 합(partially evaluated sum, SAD_SB$_{lp}$ where $0 \le l \le L$)이 현재의 최소 SAD(curr_min_SAD)보다 크거나 같으면 그 정합 블록 후보는 최상의 정합 블록이 될 수 없음으로 나머지 계산은 불필요하게 된다. 그러므로 SAD_SB$_l$ 계산에 PDE 기법을 적용하면 정합 블록 후보가 최상의 정합 블록이 될 수 없는지의 여부를 식 (9)를 완전히 계산하기 전에 알 수 있어 연산량을 줄일 수 있다. SAD계산에 사용된 PDE 기법(PDEsad로 표기)과 구분하기 위하여 SAD_SB$_l$에 사용된 PDE 기법을 PDEsb로 표기한다.

한편 식 (9)에서 하나의 항이 더해 질 때마다 부분적으로 계산된 합을 현재의 최소 SAD와 크기를 비교하는 것은 효율적이지 않다. 합리적인 절충은 그림 2에서 보는 바와 같이 각 서브 블록의 1개 행(row)마다 테스트를 하는 것이다. 3-level MSEA에서 하나의 블록은 64(=8x8)개의 서브 블록으로 나눠지기 때문에 8개의 서브 블록 행으로 구성되고 각 서브 블록 행은 8개의 서브 블록으로 구성된다. 따라서 SAD_SB$_3$계산의 경우에는 PDE 테스트가 최대 8번까지 이루어 질 수 있다. PDEsb알고리즘의 흐름도는 그림 3과 같다. 그림 3에서 SAD_SB$_{lp}$는 서브 블록 행에 대하여 차례대로 부분적인 SAD_SB$_l$ 값을 구하여 누적되는 값이다. 따라서 모든 서브 블록 행에 대하여 계산을 완료하면 SAD_SB$_{lp}$는 SAD_SB$_l$이 된다.

본 연구의 실험에서 사용한 영상은 크기가 176x144 pixel인 "salesman.qcif", "suzie.qcif", "foreman.qcif"이고 100 프레임을 테스트하였다. 블록의 크기는 16x16 pixel이고($N$=16) 탐색 윈도우의 크기는 31x31 pixel이고($M$=15) 움직임 벡터는 정수 값만 고려하였다. 본 논문의 모든 실험(3.1, 3.2, 3.3, 3.4)은 이러한 환경 하에서 수행되었다. 영상의 크기, 블록의 크기, 탐색 윈도우의 크기, 정수 움직임 벡터 등은 [1]~[3]에서와 동일하다.

실험 결과가 그림 2와 그림 4에 나타나 있다. 그림 2에서 ($x$ row)는 PDE 테스트가 매 $x$번째 서브 블록 행의 끝에서 수행되는 것을 나타내고 MSEA$_L$은 $L$-level MSEA를 나타낸다. 그림에서 사용되는 Avg. # of m.e./frame은 프레임당 정합 평가(matching evaluation)가 수행되는 평균 횟수를 나타내는 것으로서 정합 평가 횟수는 SAD계산을 하여 현재의 최소 SAD값과 비교되는 step 5를 수행하는 횟수가 된다. Avg. # of rows/m.e. 는 프레임당 SAD 계산에서 부분 왜곡 제거가 일어나기 전에 계산된 행들의 평균수를 나타낸다. (in rows)는 16x16 pixel 크기를 갖는 블록에서 1x16 pixel로 구성되는 1 행에 대한 SAD를 계산하는데 필요한 연산량을 기본 단위로 하여 연산량을 표시하였음을 의미한다. 그림에서 [연산량 (in rows)]은 Avg. # of m.e./frame 값에 Avg. # of rows/m.e.값을 곱하고 SAD계산을 제외한 모든 연산량의 합인 [Overhead(in rows)]를 더함으로 계산되어진다.

MSEA 효율성은 후보 정합 블록이 탐색되는 순서에 의존하게 된다. 최적 정합 블록부터 최악 정합 블록 순으로 테스트를 진행하면 탐색점이 보다 빨리 제거되어 MSEA의 효율이 증가하게 된다. 본 연구의 실험에서는 나선형 탐색(spiral search) 기법을 적용하였다.

PDEsb기법은 MSEA 절차 중 step 4(4.0~4.$L$)에서 SAD_SB$_l$ 값 계산에 필요한 연산량을 줄이는데 사용되는 기법으로 step 5에는 전혀 영향을 미치지 못한다. 따라서 step 5에 의해서 결정되는 Avg. # of m.e./frame 값과 Avg. # of rows/m.e. 값은 PDEsb기법이 적용되어도 일정하게 되며 SAD_SB$_l$의 연산량을 포함하는 Overhead(in rows)가 감소하게 된다. Avg. # of m.e./frame의 값은 8,769.7(MSEA$_1$ salesman), 1,589.1 (MSEA$_2$salesman), 242.3(MSEA$_3$ salesman), 6,182.2 (MSEA$_1$ suzie), 2,066.2(MSEA$_2$ suzie), 606.1(MSEA$_3$ suzie), 4,805.0(MSEA$_1$ foreman), 1,801.6(MSEA$_2$ foreman), 748.6(MSEA$_3$ foreman) 이다. 그리고 Avg. # of rows/m.e. 값은 7.84(MSEA$_1$ salesman), 10.33 (MSEA$_2$ salesman), 14.39(MSEA$_3$ salesman), 11.15 (MSEA$_1$ suzie), 13.40(MSEA$_2$ suzie), 15.25(MSEA$_3$ suzie), 9.70(MSEA$_1$ foreman), 12.65(MSEA$_2$ foreman), 14.62(MSEA$_3$ foreman) 이다.

그림 4의 실험결과에서 보는 바와 같이 MSEA$_L$에 PDEsb 기법을 적용한 MSEA$_L$+PDEsb (단, $0 \le L \le L_{max}$)의 경우 $L$의 값이 증가 할 수록 연산량 감소가 증가하게 되는데 이는 식 (13)에서 보는 바와 같이 $L$의 값이 증가 할 수록 SAD_SB$_l$을 계산하는 서브 블록의 행의 수가 많아져 PDEsb의 효과가 커지기 때문이다. PDEsb기법을 $L$-level MSEA에 적용하였을 경우 연산량이 최대 6.2% 감소되었고, 3-level MSEA에 적용하였을 경우 연산량이 평균 3.6% 감소되었다. 그림 4에 표시된 %는 PDEsb기법을 $L$-level MSEA에 적용한 알고리즘의 경우, $L$-level MSEA의 연산량이 얼마나 감소했는지를 보여주는 비율이다.

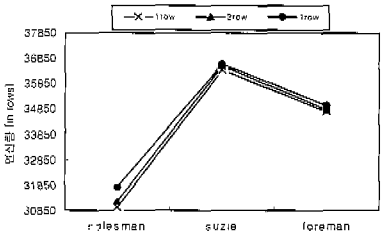


그림 2 PDEsb 테스트 포인터에 따른 $MSEA_3$+PDEsb의 연산량

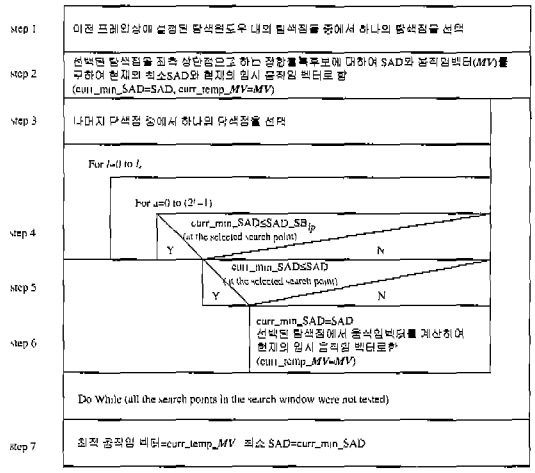


그림 3 PDEsb 알고리즘의 흐름도

### 3.2 적응 SAD 계산 알고리즘(ASCA: Adaptive SAD Calculation Algorithm)

SAD 연산은 매우 많은 연산량을 필요로 하는 계산 집중적인 연산이다. MSEA에서는 프레임 당 많은 탐색점에서 정합 평가가 이루어져 SAD계산에 많은 연산량이 소요된다. SAD 계산에 필요한 연산량을 줄이기 위하여 $0 \le (m,n) \le N-1$ 조건을 만족하는 $(m,n)$에 대하여 $|B_c^{(i,j)}(m,n) - B_p^{(i_x,i_y,x,y)}(m,n)|$의 값들을 조사하였다. $N$=16인 경우, 각 블록마다 256개의 절대 오차(absolute difference) 값들이 존재하고 이 값들은 0에서부터 화소의 최대 휘도 값(maximum pixel intensity value)에 이르는 값들을 가지게 된다. 주목할 만한 특징은 절대 오차 값의 크기가 큰 것들이 서로 모여 있는 것이다. 이는 영상에서 서로 인접한 부분들은 상관이 크기 때문이다. 이러한 특성은 그림 7에 나타나 있다.

만약 SAD 계산을 절대 오차가 큰 값에서부터 작은 값 순으로 행하게 된다면 SAD 계산에서 부분 왜곡 제거(PDEsad)가 매우 빨리 발생하게 되어 SAD 계산에 필요한 연산량이 감소하게 된다. 각 블록에 대하여 SAD를 적응적으로 구하기 위하여 그림 5와 같이 각 블록을 8x8 pixel 크기를 갖는 4개의 서브 블록으로 분할하고, 각 서브 블록에서 8개의 탐색점을 표본 추출(sampling)하였다. 표본 추출된 탐색점에 대하여 각 서브 블록별로 부분 SAD값을 계산한다. SAD 계산 시 사용될 서브 블록의 순서를 정하기 위하여, 구해진 4개의 부분 SAD 값을 정렬(sorting)하여 서브 블록들의

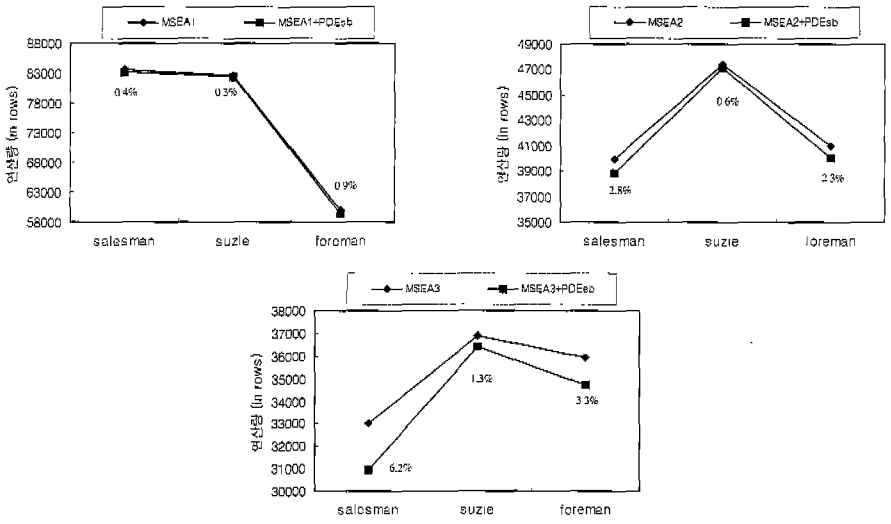


그림 4 PDEsb기법에 대한 성능평가

실제 SAD 크기에 따른 순서를 예측한다. 표본 추출된 탐색점에 대한 부분 SAD 값이 가장 큰 서브 블록에 대하여 가장 먼저 SAD를 구하고 그 후 예측된 서브 블록의 크기 순으로 SAD를 계산한다. 적응 SAD 계산 알고리즘에서는 각 서브 블록의 SAD를 잘 나타낼 수 있는 탐색점들이 각 서브 블록에서 표본 추출되는 것이 바람직하다. 그림 5와 같이 표본 추출하여 서브 블록의 SAD 크기순서를 예측한 정확도는 그림 8과 같다. 그림 8에서 Hit $n$ ($1 \le n \le 4$)은 SAD값이 $n$번 째 클 것으로 추정된 서브 블록이 실제 $n$번 째 크기의 SAD를 갖는 서브 블록과 일치하는 율이다. Hit 1/2는 최고 큰 SAD와 두 번째로 큰 SAD 값을 가질 것으로 예측된 서브 블록이 실제 서브 블록과 일치하는 경우이다. Hit1/2/3/4는 예측된 서브 블록의 SAD 크기 순이 실제 크기 순과 완전히 일치하는 경우이다. 최대 SAD와 최소 SAD사이의 비가 증가할수록 적응 SAD 계산 알고리즘에서 부분 왜곡 제거(PDEsad)가 빨리 발생하게 된다. 그림 7에서 전체 블록의 수는 100 프레임에 대하여 테스트 한 것으로 7,734,000개이다. 적응 SAD 계산 알고리즘의 흐름도는 그림 6과 같다. 그림 6에서 $SAD_b$는 그림 5의 각 서브 블록에서 SAD를 계산하여 누적되는 것을 타낸다. 이때 계산되는 순서는 각 서브블록에서 계산된 부분 SAD를 크기 순으로 정렬하였을 때 정해지는 순으로 수행한다. 따라서 4개의 서브블록 모두에 대하여 수행을 완료하면 $SAD_b$는 SAD가 된다.

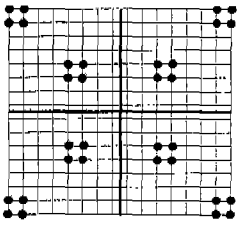

그림 5 블록의 분할과 표본 추출

적응 SAD 계산 알고리즘은 MSEA의 step 5에서 SAD를 효과적으로 계산하기 위한 알고리즘이다. 따라서 이 알고리즘을 적용하면 Avg. # of rows/frame은 감소하나 Avg. # of m.e./frame은 $MSEA_L$의 경우와 동일하다. 그림 9는 적응 SAD 계산 알고리즘을 $MSEA_L$에 적용하였을 때 Avg. # of rows/frame값이 줄어드는 것을 보여주고 있다. 적응 SAD 계산 알고리즘을 $MSEA_L$에 적용하였을 경우 연산량 감소가 그림 10에 나타나 있다.

적응 SAD 계산 알고리즘을 $L$-level MSEA, FSA+

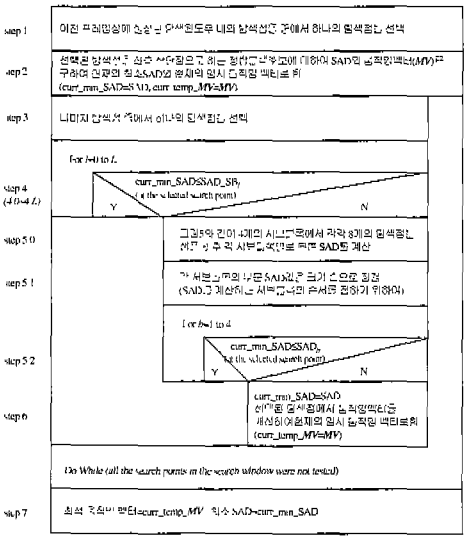


그림 6 적응 SAD 계산 알고리즘의 흐름도

PDEsad, TSS에 적용하였을 때 각각의 최대 연산량 감소는 13.2%, 18.0%, 10.4%이고 $MSEA_0$, FSA+PDEsad, TSS에 적용 시 각각의 연산량이 평균 9.9%, 14.4%, 10.0% 감소되었다. 적응 SAD 계산 알고리즘은 SAD 계산 시 연산량을 감소시키기 기 위하여 MSEA 뿐만 아니라 모든 종류의 블록 정합 알고리즘에 적용될 수 있는 매우 효율적인 알고리즘이다.

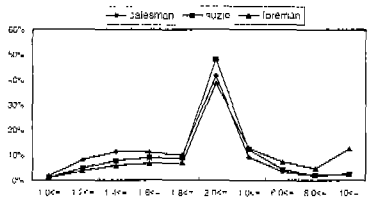


그림 7 (서브 블록의 최대 SAD/서브 블록의 최소 SAD) 비에 따른 블록의 분포

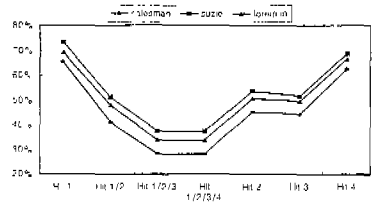


그림 8 표본 추출된 점들을 이용하여 서브 블록의 실제 SAD 크기 순서를 예측한 정확도

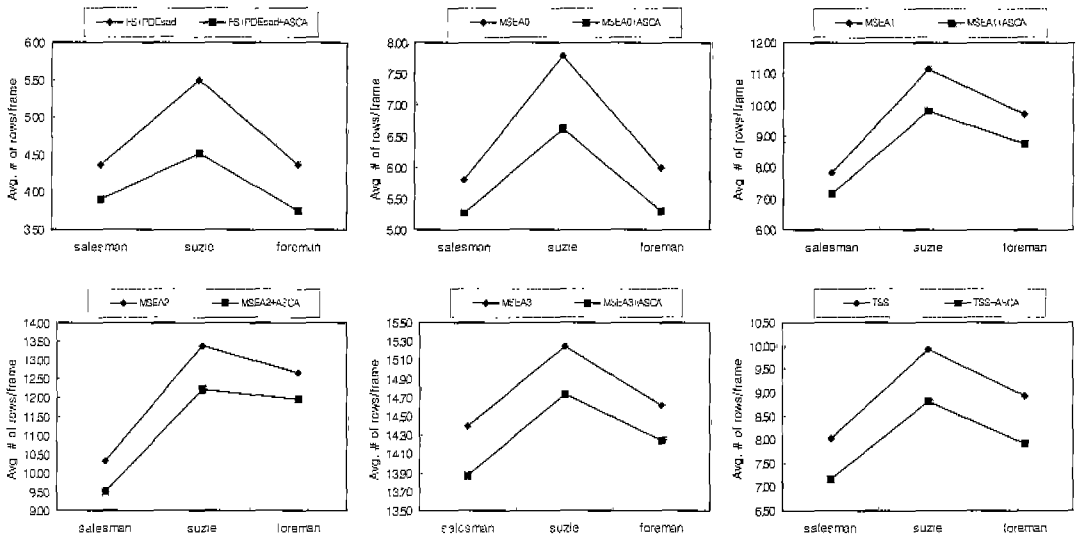


그림 9 각 알고리즘에 적응 SAD 계산 알고리즘을 적용한 경우의 Avg. # of rows/frame 감소

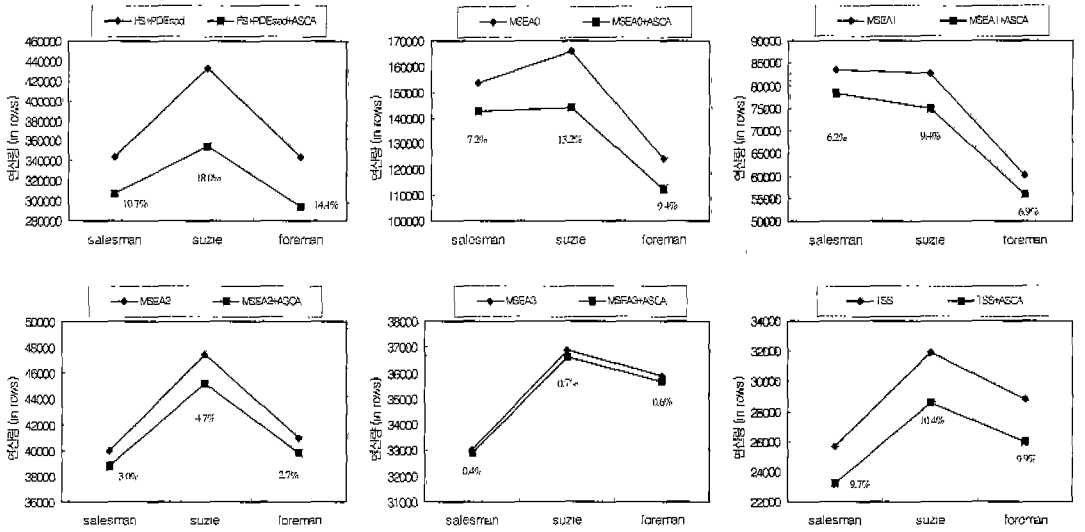


그림 10 각 알고리즘에 적응 SAD 계산 알고리즘을 적용한 경우의 연산량 감소

### 3.3 제거 레벨 추정 알고리즘(ELEA: Elimination Level Estimation Algorithm)

MSEA는 0레벨에서부터 상위레벨로 다단계 연속 제거과정을 수행한다. $L$-level MSEA에서 어떤 탐색점의 제거레벨이 $p$이면, MSEA의 4.0, 4.1, .... , 4.$p$ step들이 순차적으로 수행되고 난 뒤, step 4.$p$에서 탐색점이 제거된다(단, $0 \le p \le L$).

제거 레벨 추정 알고리즘에서는 제거 레벨을 추정하여 추정된 레벨에서부터 상위 레벨로 다단계 연속 제거 과정을 수행한다. 만약 어떤 탐색점의 제거 레벨 $p$를 추정할 수 있다면 4.0, 4.1, ...., 4.($p$-1) step들이 불필요하여 다단계 연속 제거 알고리즘의 연산량을 줄일 수

있다. 제거 레벨 추정 알고리즘의 흐름도는 그림 12와 같다.

제거 레벨의 추정 결과는 다음과 같이 5 가지 경우로 나눌 수 있다.

case 1, *hit1*: 추정된 제거 레벨(EEL: estimated elimination level)이 0이면서 실제적인 제거 레벨(PEL: practical elimination level)과 같은 경우

case 2, *hit2*: EEL이 0보다 크면서 PEL과 같은 경우

case 3, *acceptable miss1*: EEL이 0이면서 PEL보다 더 적은 경우

case 4, *acceptable miss2*: EEL이 0보다 크면서 PEL보다 적은 경우

case 5, *non-acceptable miss*: EEL이 PEL보다 더 큰 경우

제거 레벨 추정 결과가 *hit2*이거나 *acceptable miss2*인 경우에는 연산량 감소에 따른 이득이 있다. 그러나 제거 레벨 추정 결과가 *hit1*이거나 *acceptable miss1*인 경우, step 4.0에서부터 상위레벨로 다단계 연속 제거과정을 수행하기 때문에 연산량을 감소시킬 수 없다. 제거 레벨 추정 결과가 *non-acceptable miss*인 경우는 오히려 손실이 발생하게 된다. 제거 레벨 추정 오버헤드(overhead)를 고려하면 *hit1*이거나 *acceptable miss1*인 경우에도 오버헤드만큼의 손실이 발생한다.

Case 5와 같이 손실이 발생하는 경우, 만약 어떤 탐색점의 PEL과 EEL을 각각 $p$, $e$ 라고 한다면 4.0, 4.1, ..., 4.$p$ step들이 수행된 후 step 4.$p$에서 탐색점이 제거될 수 있음에도 불구하고 제거 레벨 추정이 행하여져 step 4.$e$가 수행되고 step 4.$e$에서 탐색점이 제거된다. 여기서 $p$와 $e$는 $0 \le p \le L$, $p < e \le L_{max}-1$ 조건을 만족하는 값이다. 제거 레벨 추정 알고리즘에서와 3.4의 나선형 다이아몬드 그물 탐색 알고리즘(spiral diamond mesh search algorithm)에서는 MSEA의 절차 중 step 5가 step 4.($L_{max}$+1)과 같다. 만약 더하기와 빼기와 조건 판단이 각각 하나의 연산을 필요로 하고 절대치 계산이 1.5 연산을 필요로 한다고 가정하면, $SAD\_SB_e$ 연산은 $3.5(2^{2e})-1(=3.5S_e-1$, 여기서 $S_e$는 $L$-level MSEA의 $e$번째 레벨에서 서브 블록들의 수)이다. step 4.$e$의 연산량은 $3.5(2^{2e})$이고 4.1, 4.2, ..., 4.($e$-1) step들의 전체 연산량은 식 (14)와 같다

$$3.5\sum_{i=0}^{e-1}2^{2i} \qquad (14)$$

따라서 step 4.$e$의 연산량은 4.1, 4.2, .... 4.($e$-1) step들의 전체 연산량 보다 크게 되어 case 5에서는 손실이 발생하게 된다. 그러므로 제거 레벨 추정 방법은 매우 높은 정확도를 가져야 한다. 이러한 목적을 달성하기 위하여 본 논문에서는 식 (15)와 같이 정확도가 매우 높은 제거 레벨 추정 함수를 사용하였다.

제거 레벨 추정 알고리즘에서 움직임 벡터 탐색 패턴은 그림 11과 같다. 검은색 점은 MSEA가 수행되는 탐색점이고 흰 점은 제거 레벨 추정이 필요한 탐색점을 나타낸다. 움직임 벡터의 탐색 순서는 그림 11에서 번호가 붙여진 순서대로 진행된다. 예를 들어 탐색점 20번에서는, 제거 레벨 추정이 필요하고 인접한 4개의 탐색점(2, 10, 11, 19)의 각 제거 레벨은 이미 구해져 있다. 탐색점 10의 실제적인 제거 레벨과 탐색점 20의 추정된 제거 레벨을 각각 PEL(10), EEL(20)이라고 표현하면, 본 제안 방안에서 사용된 매우 높은 정확도를 가지는 제거 레벨 추정함수는 식 (15)와 같다.

EEL(20)=min(PEL(10), PEL(2), PEL(11), PEL(19)) (15)

여기서 *min*은 4개의 값들 중에서 최소의 값을 의미한다. 식 (15)의 제거 레벨 추정 함수의 정확도는 그림 13에서 보는 바와 같이 매우 우수하나 식 (15)를 사용한 제거 레벨 추정에 필요한 오버헤드가 존재한다. 추정 함수의 오버헤드를 감소시키기 위하여 4개의 탐색점을 {10,2}, {11,19}의 두 그룹으로 분할한다. {10,2}는 탐색점 20의 좌측 하단의 점들이고 또한 탐색점 37의 우측 상단의 점들이다. {11,19}는 탐색점 20의 우측 상단의 점들이고 또한 탐색점 42의 좌측 하단의 점들이다. 그룹 분할에 의하여 식(15)를 식 (16)과 같이 다시 표현 할 수 있다.

EEL(20)=min(lside(20), uside(20)) (16)

여기서lside(20)=min(PEL(10),PEL(2)), uside(20)=min(PEL(11),PEL(19))이며 lside(20)=uside(37), uside(20)=lside(42)이다.

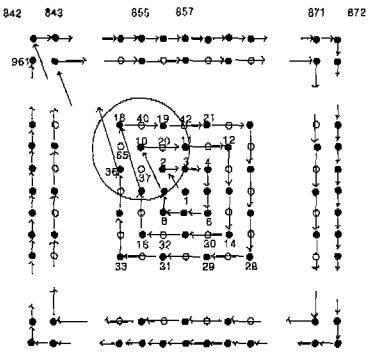


그림 11 제거 레벨 추정 알고리즘을 위한 탐색 윈도우 내에서의 움직임 벡터 탐색 패턴

이 알고리즘에서 만약 PEL(10)이 0이면 lside(20), uside(37), lsidc(40), uside(65)가 0으로 되어 EEL(20), EEL(40), EEL(65), EEL(37)이 0이 된다. 그러므로 인접한 4개 탐색점의 제거 레벨 중 하나만 0이 되어도 제거 레벨의 추정 값이 0이 되고 추정 오버헤드가 최소화된다. 만약 주변 4점의 제거 레벨이 모두 0이 아니면 식 (16)에 의해서 제거 레벨 추정이 간단히 행해질 수 있다.

그림 13에서 보는 바와 같이 평균 (*hit1+hit2*)비율은 89.5% 이고 (*hit1+hit2+acceptable miss1+ acceptable miss2*)비율은 거의 100%이며, *non-acceptable miss* 비율은 거의 0%이다. 그러므로 사용되어진 제거 레벨 추정 함수의 정확도는 신뢰할 만하다.

제거레벨 추정 알고리즘을 $MSEA_L$에 적용하였을 경우 Avg. # of m.e./frame과 Avg. # of rows/m.e.의 변화가 그림 14에 나타나 있다. $MSEA_L$+ELEA알고리즘에서는 탐색패턴이 $MSEA_L$에서 사용한 spiral search pattern과 동일한 순서로 탐색점들이 탐색되기 때문에 Avg. # of m.e./frame과 Avg. # of rows/m.e.의 값이 $MSEA_L$ 경우의 값과 거의 일치하게 된다. 그림 15의 실험결과에서 보는 바와 같이 제거 레벨 추정 알고리즘을 적용하였을 경우 최대 연산량 감소는 4.1%이었고, $MSEA_3$에 제거레벨 추정 알고리즘을 적용하였을 때 평균 2.8%의 연산량이 감소되었다. 비록 제거 레벨 추정 함수의 정확도는 신뢰할 만 하지만 연산량의 감소가 적은데 이는 대부분의 탐색점들의 PEL이 0이어서 제거레벨 추정 알고리즘에서 제거 레벨을 정확하게 추정(Case1 (*hit1*))하여도 step 4.0(0 level)에서 대부분의 탐색점들이 제거되는 경우가 그림 13과 같이 많이 발생하기 때문이다.

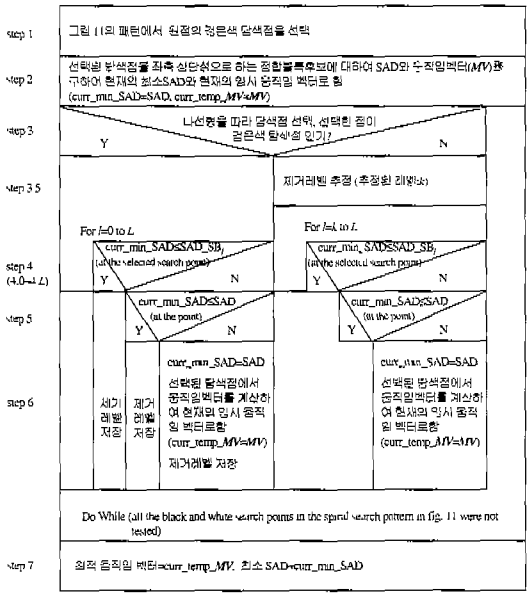


그림 12 제거레벨 추정 알고리즘의 흐름도

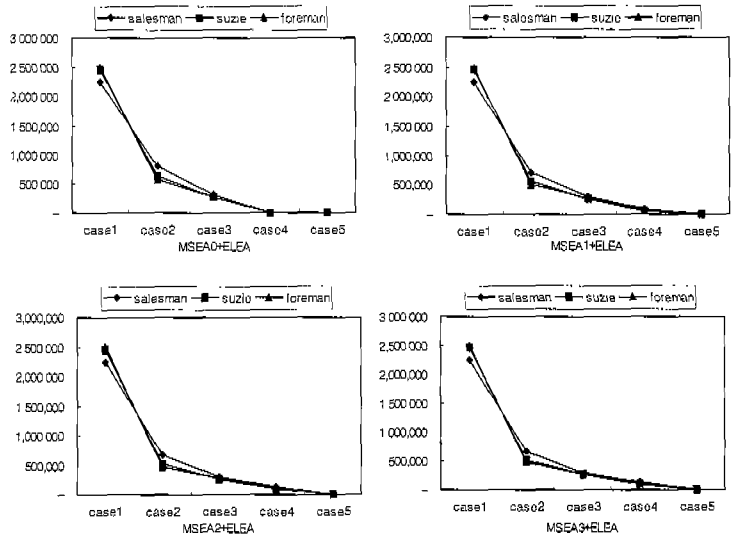


그림 13 제거 레벨 추정 함수의 제거 레벨 추정 결과

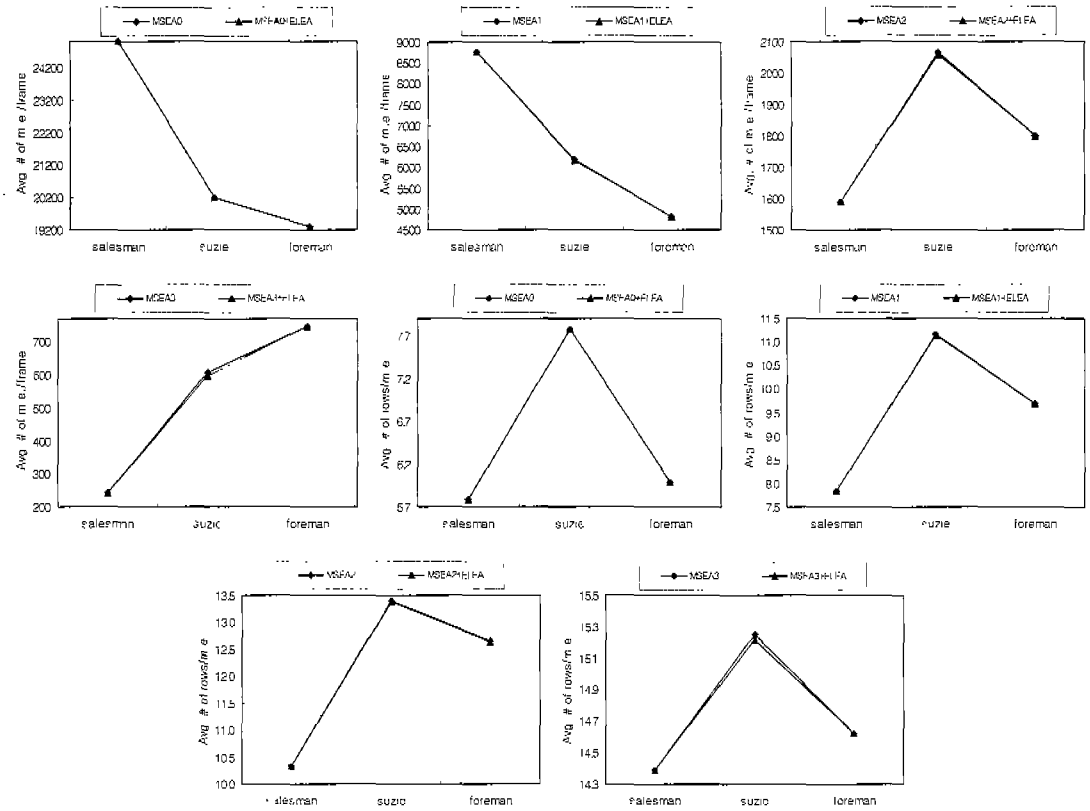


그림 14 제거레벨 추정 알고리즘을 MSEA$_L$에 적용 시 Avg. # of m.e./frame과 Avg. # of rows/m.e.의 변화

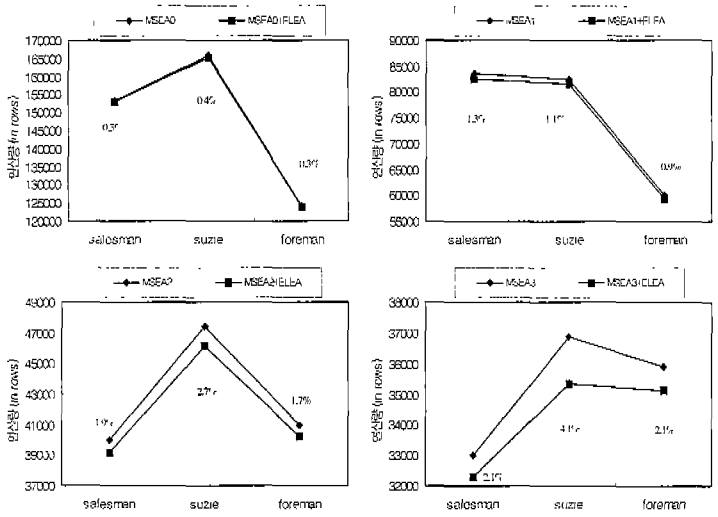


그림 15 MSEA와 결합된 제거 레벨 추정 알고리즘의 연산량

MSEA의 연산량을 보다 효과적으로 감소시키기 위하여 첫 번째(PDEsb), 두 번째(ASCA), 세 번째(ELEA) 방안을 결합(ELEA+PDEsb+ASCA)하여 MSEA에 적용할 수 있다. 이 알고리즘에서는 제거레벨 추정알고리즘의 절차 중 step 4에 PDEsb기법을 적용한 후 step 5에 적응 SAD 계산 알고리즘을 적용하는 기법이다. 이 알고리즘의 움직임추정의 정확도는 FS와 MSEA의 움직임 추정 정확도와 같다. 따라서 MSEA와 동일한 움직임 추정의 정확도를 유지하면서 각 단계별로 연산량을 감소시킨 매우 효과적인 알고리즘이다. 제안된 세가지 방안을 결합하여 MSEA에 적용한 알고리즘의 흐름도는 그림 16과 같다.

(ELEA+PDEsb+ASCA)기법을 $L$-level MSEA에 적용한 경우에 Avg. # of m.e./frame은 $MSEA_L$+ELEA에서의 값과 거의 동일하다. 이는 그림 11과 같은 패턴을 따라 탐색하는 것은 $MSEA_L$에 적용되었던 spiral search pattern과 거의 유사하기 때문이다. 그러나 Avg. # of rows/m.e.의 값은 적응 SAD 계산 알고리즘이 적용되어 $MSEA_L$+ELEA보다 감소하게 되는데 이러한 감소가 그림 17에 잘 나타나 있다.

그림 18은 제안된 첫 번째(PDEsb), 두 번째(ASCA), 세 번째(ELEA) 방안을 결합(ELEA+PDEsb+ASCA)하여 $L$-level MSEA에 적용한 경우의 연산량 감소를 보여 주고 있다. 그림에서 (E+P+A)는 (ELEA+PDEsb+ASCA)를 의미하고 %는 $MSEA_L$+(ELEA+PDEsb+ASCA) 알고리즘에서 연산량 감소가 $MSEA_L$에 비하여 얼마나 일어났는지에 대한 비율을 나타낸다. 이 실험 결과에서 보는 바와 같이 이 경우 연산량이 최대 13.5% 감소하였고 연산량이 평균 8.5% 감소하였다.

$MSEA_L$+(ELEA+PDEsb+ASCA) 알고리즘에서는 3가지 알고리즘이 차례대로 적용되어 연산량을 감소시키게 되는데 이 알고리즘의 연산량 감소율은 개별 알고리즘의 연산량 감소율의 합과 거의 같으나 각각의 합과 정확하게 일치하지는 않는다. 이는 PDEsb 알고리즘과 적응 SAD 계산 알고리즘이 $MSEA_L$ 알고리즘에 곧바로 적용된 경우의 연산량 감소율을 고려하였기 때문이다.

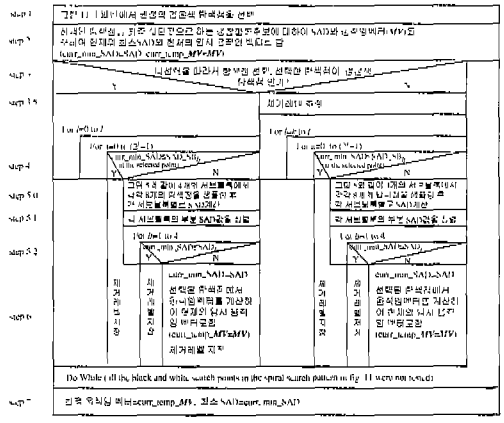

그림 16 (ELE+PDEsb+ASCA)알고리즘이 $MSEA_L$에 적용된 경우의 흐름도

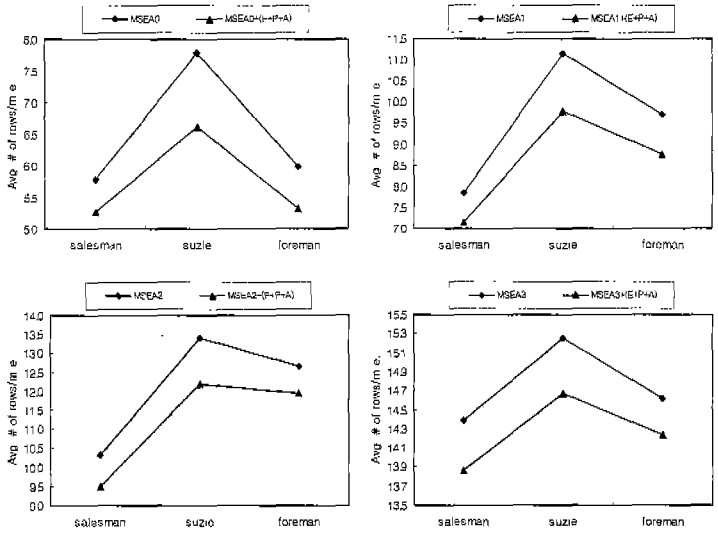


그림 17 $L$-level MSEA와 결합된 (ELEA+PDEsb+ASCA) 알고리즘에서 Avg. # of rows/m.e. 감소

$MSEA_L$에 PDEsb와 적응 SAD계산 알고리즘을 적용하는 경우에는 spiral search pattern을 사용하여 모든 탐색점에서 PDEsb 알고리즘과 적응 SAD계산 알고리즘이 곧바로 적용되나, $MSEA_L$+(ELEA+PDEsb+ASCA) 알고리즘에서는 그림 11과 같이 탐색하게 되기 때문에 검은색 탐색점에서는 PDEsb와 적응 SAD 계산 알고리즘이 동일하게 적용되나 흰색 탐색점에서는 제거레벨 추정후에 적용이 된다. 이러한 절차상의 차이는 그림 3, 그림 6, 그림 12 그리고 그림 16에서 쉽게 확인 할 수 있다. 따라서 PDEsb 알고리즘과 적응 SAD 계산알고리즘에 의해서 연산량 감소가 일어나는 SAD_SB/과 SAD 계산 횟수가 $MSEA_L$ 경우와 $MSEA_L$+(ELEA +PDEsb+ASCA) 경우에 조금씩 서로 차이가 나게 되기 때문에 $MSEA_L$+(ELEA+PDEsb+ASCA) 알고리즘의 연산량 감소율은 ELEA, PDEsb 그리고 ASCA의 연산량 감소율의 합과 거의 같으나 정확하게는 일치하지 않게 된다.

### 3.4 나선형 다이아몬드 그물 탐색 알고리즘(SDMSA: Spiral Diamond Mesh Search Algorithm)

움직임 보상된 잔여 에러 면(motion compensated residual error surface)이 볼록 함수가 아니라는 성질에 기초하여 [3]과 [16]에서 빠른 블록 정합 알고리즘이 제안되었다. 나선형 다이아몬드 그물 탐색 알고리즘은 [3]과 [16]에서 제안된 알고리즘을 변형하여 MSEA에 적용한 것으로서 (ELEA+PDEsb+ASCA)기법이 [3]과[16]의 알고리즘에 추가된 방안이다. 나선형 다이아몬드 그물 탐색 알고리즘의 흐름도는 그림 19와 같다. 나선형 다이아몬드 그물 탐색 알고리즘은 두 번의 탐색 경로를 가진다. 첫 번째 탐색 경로에서는 그림 11의 모든 검은 탐색점들이 원점으로부터 외부로 나선형의 패턴을 따라서 탐색된다. 그러나 첫 번째 탐색 경로(SDMSA의 절차 중 step 1 ~ step 6)에서는 흰 탐색점들은 테스트되지 않고 가장 외곽의 4각형 위의 점들은 검은 탐색점들이 다이아몬드 그물 모양을 유지하게 테스트된다. 그러므로 그림 11의 탐색점 843, ..., 855, 857, ..., 871, ... , 961은 테스트되지 않고 485개의 검은 탐색점들이 첫 번째 탐색 경로에서 테스트되며 이 탐색점들은 전체 탐색 윈도우에 걸쳐 분포하게 된다. 그 후 두 번째 탐색 경로에서는 첫 번째 탐색 경로에서 얻어진 데이타를 기초로 움직임 벡터가 존재할 것으로 예상되는 탐색점들의 주위에 있는 흰 탐색점(첫번째 경로에서 탐색되지 않은 탐색점)

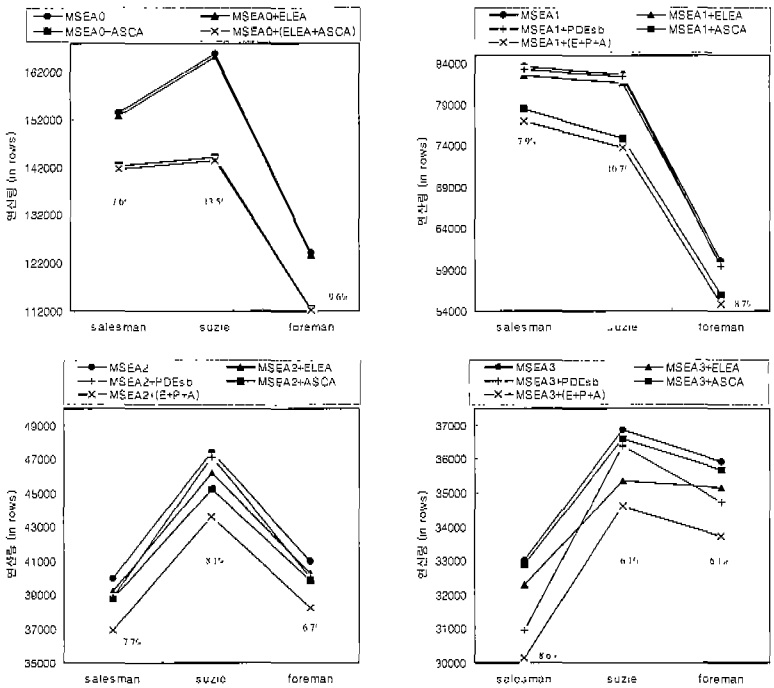


그림 18 $L$-level MSEA와 결합된 (ELEA+PDEsb+ASCA) 알고리즘과 $MSEA_L$+ELEA, $MSEA_L$+PDEsb, $MSEA_L$+ASCA와의 연산량 비교

들에 대하여 (ELEA+PDEsb+ASCA)기법을 사용하여 테스트하게 된다. 두 탐색 경로 수행 완료후의 curr_min_SAD는 최소 SAD가 되고 최소 SAD를 가지는 블록이 정합 블록이 되어 움직임 벡터를 구할 수 있다.

첫 번째 탐색 경로 결과에서 움직임 벡터가 존재할 것으로 예상되는 부분을 알 수 있도록 SDMSA의 절차 중 step 8과 같이 현재 최소 SAD(curr_min_SAD)보다 더 적은 SAD를 가지는 탐색점들을 저장한다. Step 11에서 변수 $\delta$를 제어하여 실시간 연산이 가능하게 부하를 조절 할 수 있고 움직임 추정의 정확도를 조절 할 수 있다. 본 실험에서는 [3]에서와 같이 윈도우의 크기를 ±2로 하였다. 첫 번째 탐색 경로 중에 계산된 SAD의 값이 curr_min_SAD의 값보다 더 적은 경우 그 탐색점의 좌표는 스택에 저장되고 스택에 저장된 탐색점들을 중심으로 윈도우 크기가 $\delta$인 범위내의 흰 탐색점들이 두 번째 탐색 경로에서 테스트된다. 본 실험에서는 프레임 당 평균 스택의 깊이가 4.8 이었다.

탐색 윈도우내의 평균 탐색점의 수를 $S$, 다단계 연속 제거 알고리즘에서 탐색점 당 평균 연산량을 $C$, 프레임 당 스택의 평균 깊이를 $D$, 1 프레임에 대한 움직임 벡터의 갯수를 $V$라고 둔다면 MSEA에서는 하나의 참조 블록에 대한 움직임 벡터를 찾기 위하여 연산량 $SC$가 필요하게 된다. 그러나 $\delta$가 ±2인 나선형 다이아몬드 그물 탐색 알고리즘에서는 연산량이 $(S/2+12D/V)C$로 줄게 된다.

그림 20에서 움직임 추정의 정확도가 표시되었는데 이는 $x/y$로 계산된 값이다. $y$는 100 프레임에 대한 전체 움직임 벡터 개수이고 $x$는 나선형 그물 탐색 알고리즘을 $L$-level MSEA에 적용하여 구해진 움직임 벡터 중 전역 탐색 알고리즘에서 구해진 움직임 벡터와 동일한 움직임 벡터의 수이다. 그림 20의 실험 결과에서 보는 바와 같이 나선형 다이아몬드 그물 탐색 알고리즘의 움직임추정 정확도는 거의 100%이다.

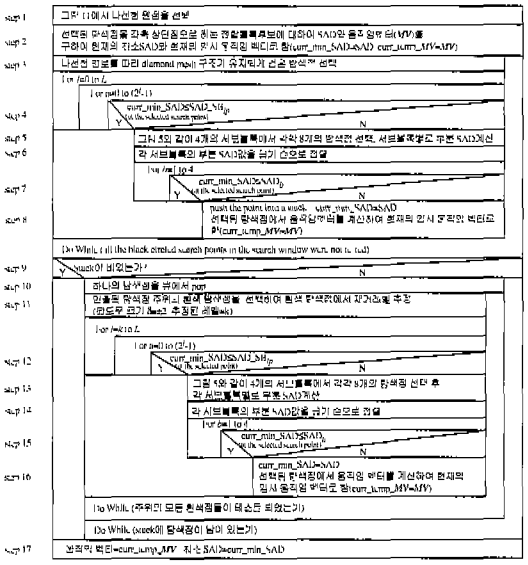

그림 19 나선형 다이아몬드 그물 탐색 알고리즘

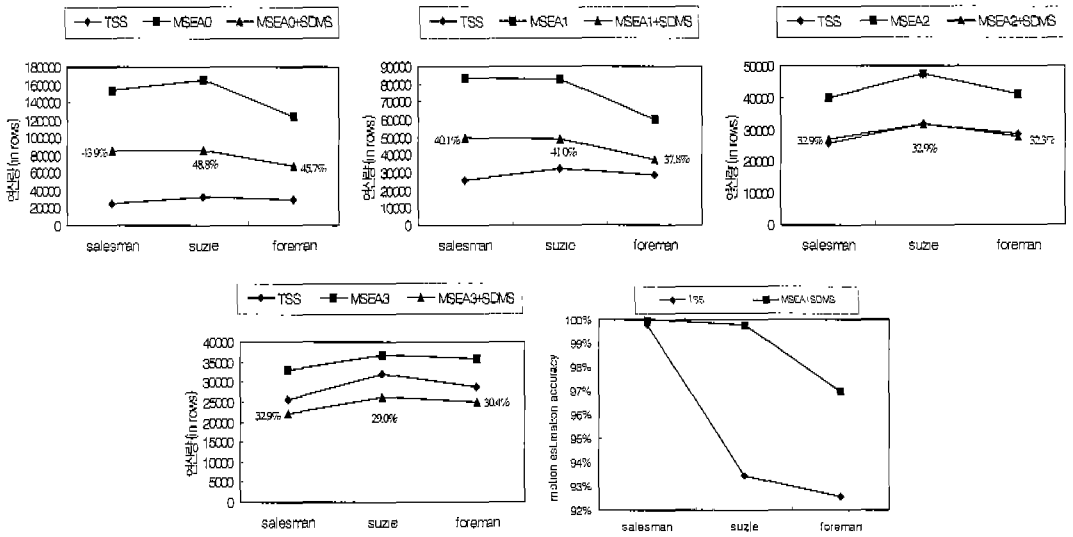


그림 20 나선형 다이아몬드 그물 탐색 알고리즘을 $L$-level MSEA에 적용한 실험 결과와 TSS와의 성능비교

나선형 다이아몬드 그물 탐색 알고리즘을 적용하였을 경우 최대 48.84%, 평균 37.3%의 연산량 감소가 있었으며, 나선형 그물 탐색 알고리즘의 움직임 추정 정확도는 TSS의 정확도 보다 훨씬 높고 $MSEA_3$와 결합된 나선형 그물 탐색 알고리즘의 연산량은 TSS의 연산량보다 평균 15.1% 감소되었다. 그러므로 $L$-level MSEA와 결합한 나선형 다이아몬드 그물 탐색 알고리즘은 매우 높은 움직임 추정 정확도를 가지는 고속 블록 정합 알고리즘으로 받아들여질 수 있다.

## 4. 결 론

본 논문에서 움직임 추정을 위한 블록 정합 알고리즘인 다단계 연속 제거 알고리즘의 연산량을 감소시키기 위하여 개선된 다단계 연속 제거 알고리즘을 제안하였다. $SAD\_SB_l$ 계산에 부분 왜곡 제거 기법을 적용한 방안과 제거 레벨 추정 알고리즘은 다단계 연속 제거 알고리즘의 연산량을 감소시키기 위한 효율적인 방안이다. 적용 SAD 계산 알고리즘은 SAD 계산 시 연산량을 줄일 수 있는 알고리즘으로 MSEA 뿐만 아니라 모든 종류의 블록 정합 알고리즘에 적용될 수 있는 매우 유용한 방안이다. 나선형 다이아몬드 그물 탐색 알고리즘은 움직임 추정의 정확도가 매우 높은 고속 블록 정합 알고리즘이다. 본 논문에서 제안된 개선된 다단계 연속 제거 알고리즘(AMSEA)은 낮은 비트 율(bit-rate)과 질 높은 코딩을 필요로 하는 비디오 코딩 응용에 매우 효율적인 해결책이다.

## 참 고 문 헌


[1] X. Q. Gao, C. J. Duanmu, and C. R. Zou, "A multilevel successive elimination algorithm for block matching motion estimation," IEEE Trans. Image Processing, Vol. 9, No.3, pp. 501-504, Mar. 2000.

[2] W. Li and E. Salari, "Successive elimination algorithm for motion estimation," IEEE Trans. Image Processing, vol. 4, No.1, pp. 105-107, Jan. 1995.

[3] H. S. Wang, R. M. Mersereau, "Fast algorithms for the estimation of motion vectors," IEEE Trans. Image Processing, vol. 8, No.3, pp. 435-438, Mar. 1999.

[4] H. G. Musmann, P. Pirsh, and H. J. Gilbert, "Advances in picture coding," Proc. IEEE, vol. 73, pp. 523-548, Apr. 1985.

[5] J. R. Jain and A. K. Jain, "Displacement measurement and its application in interframe image coding," IEEE Trans. Commun., vol. COM-29, no. 2, pp. 1799-1808, Dec. 1981.

[6] A. N. Netravali and J. D. Robbins, "Motion compensated television coding: Part I," Bell Syst. Tech. J., vol. 58, pp. 631-670, Mar. 1979.

[7] CCITT Standard H.261, "Video codec for audiovisual services at px64 kbit/s," ITU, 1990.

[8] ITU-T DRAFT Standard H.263, "Video coding for narrow telecommunication channel at (below) 64kbit/s," ITU, Apr. 1995.

[9] ISO-IEC JTC1/SC2/WG11, "Preliminary text for MPEG video coding standard," ISO, Aug. 1990.

[10] A. Puri, H.-M. Hang, and D. L. Schilling, "An efficient block matching algorithm for motion compensated coding," Proc. Int. Conf. Acoust., Speech, Signal Processing, 1987, pp. 25.4.1-25.4.4.

[11] M. Ghanbari, "The cross-search algorithm for motion estimation," IEEE Trans. Commun., vol. 38, no. 7, pp. 950-953, July 1990.

[12] T. Koga, K. Iinuma, Y. Iijima, and T. Ishiguro, "Motion compensated interframe coding for video conferencing," Proc. Nat. Telecommunications Conf., Nov. 1981, pp. G5.3.1- G.5.3.5.

[13] M. J. Chen, L. G. Chen, and T. D. Chiueh, "One-dimensional full search motion estimation algorithm for video coding," IEEE Trans. Circuits Syst. Video Technol., vol. 4, pp. 504-509, Oct. 1994.

[14] H. M. Jong, L. G. Chen, and T. D. Chiueh, "Accuracy improvement and cost reduction of 3-step search block matching algorithm for video coding," IEEE Trans. Circuits Syst. Video Technol., vol. 4, pp. 88-91, Feb. 1994.

[15] R. Li, B. Zeng, and M. L. Liou, "A new three-step search algorithm for block motion estimation," IEEE Trans. Circuits Syst. Video Technol., vol. 4, pp. 438-442, Aug. 1994.

[16] B. Liu and A. Zaccarin, "New fast algorithms for the estimation of block motion vectors," IEEE Trans. Circuits Syst. Video Technol., vol. 3, no. 2, pp. 148-157, Apr. 1993.

[17] K. H. K. Chow and M. L. Liou, "Genetic motion search algorithm for video compression," IEEE Trans. Circuits Syst. Video Technol., vol. 3, pp. 440-446, Dec. 1993.

[18] S. M. Jung, S. C. Shin, H. K. Baik, and M. S. Park, "Advanced multi-level successive elimination algorithms for motion estimation in video coding," Lecture Notes in Computer Science 1983, Proc. of International Conference of Intelligent Data Engineering and Automated Learning, 431-442 December, 2000, Hong Kong

[19] S. M. Jung, S. C. Shin, H. K. Baik, and M. S. Park, "Nobel successive elimination algorithms for the estimation of motion vectors," Proc. of the IEEE International Symposium on Multimedia Software Engincering, pp. 332-335 December, 2000, Taiwan
[20] S. M. Jung, S. C. Shin, H. K. Baik, and M. S. Park, "An efficient multi-level successive elimination algorithm for motion estimation in video coding," Proc.(CD version) of International Conference on Signal Processing Applications & Technology, August, 2000, USA
[21] S. M. Jung, S. C. Shin, H. K. Baik, and M. S. Park, "New fast successive elimination algorithm," Proc.(CD version) of the 43rd IEEE Midwest Symposium on Circuits and Systems, August, 2000, USA
[22] T. M. Apostol, "Mathematical Analysis. Reading," MA: Addison-Wesley, 1975.
[23] A. Gersho and R. M. Gray, "Vector Quantization and Signal Compression," Boston, MA: Kluwer, 1991.

정 수 목

1984년 경북대학교 전자공학과(학사). 1986년 경북대학교 대학원 전자공학과(석사). 1986년 ~ 1991년 LG정보통신 연구소 연구원. 2002년 고려대학교 대학원 컴퓨터학과(박사). 1998년 ~ 현재 삼육대학교 컴퓨터학과 조교수. 관심분야는 Multimedia, Video coding

박 명 순

1975년 서울대학교 전자공학과 학사. 1982년 University of Utah 전기공학과 석사. 1985년 University of Iowa 전기전산공학과 박사. 1975년 ~ 1980년 국방과학연구소 연구원. 1985년 ~ 1987년 Marquette 대학교 조교수. 1987년 ~ 1988년 포항공과대학 전자전기공학과 조교수. 1988년 ~ 현재 고려대학교 컴퓨터학과 교수. 관심분야는 병렬처리, IP QoS, IP Multicast, Internet Computing