내장형 영상코딩을 위한 재귀적 SPIHT 알고리즘

Recursive SPIHT(Set Partitioning in Hierarchy Trees) Algorithm for Embedded Image Coding

박 영 석*

Young-Seak Park

요 약

EZW(Embedded Zerotree Wavelet) 알고리즘이 소개된 이래 일련의 내장형 웨이브렛 코딩 방법들이 제안되어 져왔다. 이들의 하나의 공통된 특징은 EZW 알고리즘의 기본 아이디어를 근간으로 한다는 점이다. 특히 SPIHT(Set Partitioning in Hierarchy Trees) 알고리즘은 이들 중의 하나로서 산술 코더를 사용하지 않더라도 EZW 와 같거나 혹은 더 나은 성능을 제공할 수 있기 때문에 널리 알려져 왔다. 본 연구에서는 내장형 영상코딩을 위한 재귀적 SPIHT(RSPIHT) 알고리즘을 제안하고 그 유효성을 실험적으로 가한다. 제안한 RSPIHT 알고리즘은 매우 단순하고 정형화된 형태를 지니면서 최악의 경우 시간복잡도 $O(n)$을 가진다. 실험영상들에 대해 T-layer 4 이상에서 SPIHT보다 평균 약 16.4%의 개선된 속도를 얻을 수 있었다. 압축률의 관점에서도 RSPIHT 알고리즘은 실험영상의 T-layer 7 이하에서는 SPIHT와 유사한 결과를 가지나 그보다 큰 T-layer에서는 개선된 결과를 보였다.

ABSTRACT

A number of embedded wavelet image coding methods have been proposed since the introduction of EZW(Embedded Zerotree Wavelet) algorithm. A common characteristic of these methods is that they use fundamental ideas found in the EZW algorithm. Especially, one of these methods is the SPIHT(Set Partitioning in Hierarchy Trees) algorithm, which became very popular since it was able to achieve equal or better performance than EZW without having to use an arithmetic encoder. In this paper We propose a recursive set partitioning in hierarchy trees(RSPIHT) algorithm for embedded image coding and evaluate it's effectiveness experimentally. The proposed RSPIHT algorithm takes the simple and regular form and the worst case time complexity of $O(n)$. From the viewpoint of processing time, the RSPIHT algorithm takes about 16.4% improvement in average than the SPIHT algorithm at T-layer over 4 of experimental images. Also from the viewpoint of coding rate, the RSPIHT algorithm takes similar results at T-layer under 7 but the improved results at other T-layer of experimental images.

Key words : embedded image coding, EZW Coding(embedded zerotree wavelet coding), SPIHT(set partitioning in hierarchy tree) algorithm, layered progressive transmission, RSPIHT(recursive SPIHT), SOT(spatial orientation tree) coding tree structure, significance map, arithmetic coder

Ⅰ. 서론

IT(Inforation Technologies)산업의 발전에 따라 디지털 위성방송, 비디오 컨퍼런싱, 비디오CD 재생, 주문형비디오(VOD), 인터넷 상거래, 텔레뱅킹 등의 다양한 고품질 멀티미디어 서비스의 요구가 급증하고 있다. 그러나 이러한 디지털 기술이 가지는 큰 단점의 하나는 그것이 생성하는 방대한 량의 데이터이다[1,2]. 이러한 방대한 데이터의 문제를 해결하기 위한 필수 기술의 하나가 영상압축이다.

영상압축기법 특히 비가역적(nonreversible) 혹은 손실(lossy) 압축기법들은 하나의 정상 소오스(stationary source)에 대한 무한한 계산량에 따라 최적에 접근한다는

*경남대학교 정보통신공학부
접수 일자 : 2003. 7. 21 수정 완료 : 2003. 9. 26
논문 번호 : 2003-3-9
※본 연구는 2003년도 경남대학교 학술연구장려금에 의해 수행되었음.

정보이론의 소오스 코딩이론의 관점에따라 그들이 더욱 효과적인 만큼 계산적으로 더욱 복잡하게 되는 것으로 알려져왔다[1,2].

그럼에도 불구하고 Shapiro[4]에 의해 소개된 내장형 제로트리 웨이브렛(EZW: embedded zerotree wavelet)이라 불리는 영상코딩기법은 효율성과 복잡도(Complexity)의 병행성 논의을 중단시켰다. 이 기법은 가장 복잡한 기법들과 성능에 있어서 경쟁적일 뿐만 아니라 하나의 내장형 비트 스트림(embedded bit stream)을 생성하는데 극단적으로 빠른 실행시간을 가진다.

하나의 내장형 비트 스트림을 가지고 코드 비트들의 수신은 임의의 시점에서 중단될 수 있고 디코딩되어 복원될 수 있다. 그 의미 깊은 업적에 이어 보다 효과적인 EZW기법 기반 원리를 사용하는 웨이브렛 코딩 알고리즘들이 개발되어 왔다[3,5-7].

대표적인 것으로 SPIHT(Set Partitioning in Hierarchy Trees)알고리즘[3], EBCOT(Embedded Block Coding with Optimized Truncation of the embedded bit-strimes)알고리즘[5], 그리고 SRC(Stack Run Coding) 알고리즘[7]을 들 수 있다. 이 방법들의 공통된 특징은 그들이 EZW 알고리즘에서 발견된 기본 아이디어를 사용한다는 것이다. 그중 EBCOT 알고리즘은 JPEG 2000 표준의 기본으로 채택되었고, SRC알고리즘은 EZW의 계승자라고는 할 수 없으나 모든 서브밴드에 균일양자화기 사용과 인터밴드 예측이 없는 서브밴드 코딩의 면에서 EZW를 닮았기 때문에 초기 와 현대 웨이브렛 코더간의 하이브리드한 형태의 것으로 낮은 복잡도와 양호한 코딩 성능을 가진다. 이들과 다른 웨이브렛 기반 영상코딩 알고리즘들의 성능비교는 WWW(World Wide Web)상[8]에서 찾을 수 있다.

특히 SPIHT 알고리즘은 그것이 산술엔코더(arithmetic encoder)의 사용 없이도 EZW보다 같거나 더 나은 성능을 가졌기 때문에 매우 유명하게 되었고 JPEG 2000 표준 결정에 있어서 EBCOT에 대항하는 가장 강력한 경쟁자였다[1]. 2D(2-dimentional)에 이어 그 후 EZW 계열 알고리즘은 3D(3-dimensional)로 확장되고[2,9], 그중에서 3D SPIHT[9]는 어떤 모션보상(motion compansation)없이 계산적으로 단순하고 보다 효과적인 비디오 코딩 시스템의 가능성과 뛰어난 수치적, 시각적 결과를 보임으로써 그 유용성과 발전 가능성을 확대하고 있다.

본연구에서는 SPIHT 알고리즘의 이러한 발전가능성에 주목하고, 알고리즘 분석을 통하여 보다 정형화되고 개선된 처리 시간 복잡도(time complexity)와 압축률을 가지는 재귀적 SPIHT(RSPIHT: recursive SPIHT)알고리즘을 개발하고 실험적으로 처리시간 및 압축효율성 관점에서 그 유효성을 평가하고자 한다.

II. SPIHT 알고리즘의 개요

EZW기법은 3가지 기본개념 즉, 1) 크기에 따른 변환 이미지 요소의 부분순서화(partial ordering), 2) 리파인먼트(refinement) 비트들의 순서화된 비트평면 전송, 그리고 3) 다른 스케일에 걸친 이미지 웨이브렛 변환의 자기유사성(self-similarity)의 이용에 의존한다. 그 부분 순서화는 옥타브적으로 감소하는 문턱치(threshold value)들의 집합에 변환요소(계수)크기를 비교한 결과이다. 우리는 하나의 요소가 주어진 문터치 보다 크면 중요하다(significant)고 하고 그렇지 않으면 중요하지않다(insignificant)고 한다.

Shapiro[3]의 EZW알고리즘의 경우는 산술코딩이 필수적이다. 그러나 SPIHT에서는 산술코딩되지않은 이진정보조차도 선행 연구된 EZW 알고리즘들과 같거나 더 좋은 압축결과를 성취할 만큼 압축율이 높고 더구나 산술코딩의 사용은 PSNR(Peak Signal-to-Noise Ratio)을 0.3-0.6dB정도 개선할 수 있고 실행시간 또한 매우 빠른 것으로 보고되었다[2]. 전송코드 혹은 압축영상 파일은 어떤 주어진 코드율로 절단될 수 있고 디코딩될 수 있도록 완전하게 내장(embedded)된다. 물론 엔코딩은 거의 손실없는(nearly lossless) 영상을 표현할 때까지 진행될 수 있다. 거의 손실이 없다는 것은 일반적인 웨이브렛 변환 필터가 비정수 탭 가중치(noninteger tap weight)를 가지며 유한 정도로 절단되는 비정수 변환계수를 생성함으로해서 압축이 가역적(reversible)이지 못할 수 있기 때문이다. 완전한 가역적 압축을 위해서는 S+P 변환[10]과 같은 그런 정수 다해상 변환(integer multiresolution transform)이나 보다 일반적인 리프팅(lifting) 알고리즘[1]을 사용해야만 한다.

2.1 프로그레시브 전송(Progressive Transmission) 알고리즘[3]

영상의 웨이브릿 변환계수, $c(i,j)$들은 크기(magnitude)의 2진 표현에 요구되는 비트의 최소 수에 따라 감소순(descending order)으로 순서화된다고 가정하자. 즉, 식 (1)과 같이 일대일(one-to-one) 매핑, $\pi: I \mapsto I^2$에 따라 순서화된다.

$$|\log_2|c_{\pi(k)}|| \geq |\log_2|c_{\pi(k+1)}|| \;, k=1, \ldots, N \quad (1)$$

그림 1은 크기에 따라 순서화된 계수 리스트의 도식적 2진 표현을 보인다. 각 열 k는 $c_{\pi(k)}$의 비트들을 포함한다. 위에서 첫째 행은 부분 순서화된 계수값을 나타내고 둘째 행은 계수의 부호를 나타낸다. 나머지 행들은 비트평면(bit plan)에 대응하며 바닥에서 위로 번호 붙여져 있고 가장 하위 행은 LSB(least significant bit) 즉, 문턱치 레이어(T-layer: threshold-layer) 6를 가장 상위 행은 MSB(most significant bit) 즉, T-layer 1을 나타낸다.

T-layer는 계층적 전송(layered transmission) 과 문턱

치의 관점에서 각 비트평면을 규정하며 하나의 비트평면은 코딩되어 하나의 전송 층(transmission layer)을 이루는 것으로 규정한다. 여기서 순서화 정보 이외에 디코더는 $2^n \le |c_{i,j}| < 2^{n+1}$인 계수들의 수에 대응하는 수 μ_n 또한 받는다고 가정하자. 그림 1의 예에서 $\mu_5 = 4$, $\mu_4 = 2$, $\mu_3 = 13$ 이다. 그래서 프로그래시브 전송의 가장 효과적인 순서는 화살표로 표시된 것처럼 각 행의 비트를 순차적으로 보내는 것이며 . 그것은 계수들이 크기에 대해 감소 순이고 어떤 열의 선행 0 비트와 첫 번째 1 비트는 순서화와 μ_n으로부터 추론될 수 있으므로 전송될 필요가 없기 때문이다.

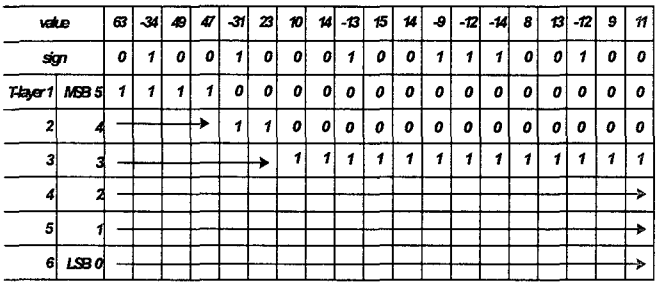


그림 1. 계층적 프로그래시브 전송과정

Fig. 1 Layered Progressive Transmission Process

프로그레시브 전송법은 엔코더에서 사용될 다음의 알고리즘(Algorithm I)으로 구현될 수 있다. 그러나 step 2)의 소팅 패스는 보다 효과적인 방법으로 구현될 필요가 있다. 그래서 SPIHT 알고리즘의 핵심은 μ_n의 전송없이 해당 비트평면(bit plan)의 후술될 중요도-맵(significance map)을 코딩하여 전송하는 것으로 Algorithm I의 step 2)를 대체하는 것이다.

Algorithm I

step 1) Initialization: output $n = \lfloor \log_2(\max_{(i,j)} |c_{i,j}|) \rfloor$ *to the decoder;*

step 2) Sorting Pass: output μ_n *followed by the pixel coordinates* $\eta(k)$ *and sign of each of the* μ_n *coefficients such that* $2^n \le |c_{\eta(k)}| < 2^{n+1}$;

step 3) Refinement pass: output the nth most significant bit of all the coefficients with $|c_{i,j}| \ge 2^{n+1}$ *(i.e., those that had their coordinates trnasmitted in previous sorting pass), in the same order used to send the coordinates;*

step 4) Quantization-Step Update: decerment n by one, and go to step 2);

2.2 SPIHT 알고리즘 분석

Algorithm I의 소팅패스에 해당하는 SPIHT 알고리즘은 그림 2와 같은 웨이브렛 변환이미지에 대한 SOT(spatial orientation tree)[2] 즉, 중요도-맵의 계층구조의 개념을 사용한다. 그림 3은 웨이브렛 변환 이미지의 예이고 그림4는 그림3의 변환 이미지에 문턱치 32를 적용한 중요도-맵을 보인다. 본 연구에서의 중요도-맵이란 shapiro[3]의 EZW에서와 달리 단순한 비트평면을 의미한다. 설명을 위해 그림 2의 계층적 SOT 트리 구조에 대한 다음의 정의를 사용한다. 이것은 Said and Pearlman[3]의 정의를 확장한 것이다.

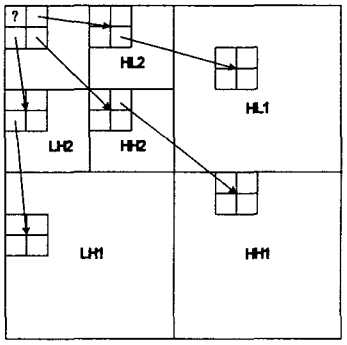


그림 2. 중요도-맵[3]의 계층적 SOT 구조

Fig. 2 The hierarchical SOT structure of significance map[3]

$O(\{(i,j)\})$: 노드 (i,j) 집합의 모든 일세대 직계손(offspring) 좌표의 집합이며, $\{(i,j)\}$가 단일 좌표집합이면 $O(i,j)$로 표기한다.

$D(\{(i,j)\})$: 노드 (i,j) 집합의 모든 자손(descendants) 좌표의 집합이며, $\{(i,j)\}$가 단일 좌표집합이면 $D(i,j)$로 표기한다.

$L(\{(i,j)\})$: $D(\{(i,j)\}) - O(\{(i,j)\})$이며, $\{(i,j)\}$가 단일 좌표집합이면, $L(i,j)$로 표기한다.

$P(\{i,j\})$; 노드 (i,j) 집합의 부모(parent) 좌표의 집합이며, $\{(i,j)\}$가 단일 좌표집합이면 $P(i,j)$로 표기한다.

그리고 임의의 좌표 집합 즉, 파티션(partition) T의 중요도를 나타내기 위해 식 (2)의 함수 $S_n(T)$를 사용한다. 즉, $S_n(T)$는 파티션 T의 계수 중 그 절대값이 문턱치 2^n보다 클 때 1을 그렇지 않으면 0의 값을 가진다. 단일 화소 집합의 표현을 단순화하기 위해 $S_n(\{(i,j)\})$을 $S_n(i,j)$로 표기한다.

$$S_n(T) = \begin{cases} 1, & \max_{(i,j)\in T}\{|C_{i,j}|\} \ge 2^n \\ 0, & Otherwise \end{cases} \qquad (2)$$

SPIHT 알고리즘은 엔코더(encoder)의 코딩 실행경로(execution path)를 디코더에서 재현함으로써 전송되는 중요도-맵의 코딩정보를 통해 중요도-맵을 재구성한다.[3] 이것은 엔코더에서 생성되어 전송되는 데이터의 구조분석을 통해 보다 분명하고 쉽게 이해할 수 있다. 결과

적으로 SPIHT 알고리즘은 중요도-맵에 대한 코딩으로 분류과정에서 SOT상 개개의 루트(root)에 대해 그림 6과 같은 하나의 데이터 구조 정보 즉, SOT 코딩 트리(혹은 중요도 코딩 맵)을 생성하고 부호비트와 함께 전송한다. 여기서 루트 노드의 정보가 3비트인 것은(원 알고리즘에서는 1비트) 원 알고리즘의 완전성을 위해 수정 표현되었다. 즉, 원 알고리즘에서는 루트 노드 이후의 자손 노드들이 존재함을 가정하고 있으나 일반적으로 자손 노드들이 존재하지 않는 경우도 있을 수 있으며 1비트로는 이러한 경우를 표현할 수 없기 때문이다. 이 SOT 코딩트리는 SOT 코딩트리 상의 노드에 대해 다음과 같은 특징을 가진다.

1) 노드 (i,j)는 $S(i,j)=1$ 혹은 $S(D(i,j))=1$ 혹은 $O(P(i,j))=1$이다.

2) 노드 (i,j)가 $L(P(i,j))\neq\emptyset$ 그리고 $S(L(P(i,j)))=0$, 혹은 $L(P(i,j))\in\emptyset$ 그리고 $S(O(P(i,j))=1$이면 자신의 중요도, $S(i,j)$를 나타내는 1비트 정보를 가진다.

3) 노드 (i,j)의 $L(P(i,j))\in\emptyset$ 그리고 $S(O(i,j))=1$이면, 노드 (i,j)는 2비트 b_1b_2 정보를 가지며 b_1은 $S(i,j)$를 그리고 b_2는 $S(O(i,j))$를 나타낸다.

4) 2)와 3)의 경우가 아니면, 노드 (i,j)는 3비트 $b_1b_2b_3$ 정보를 가지며 b_1은 $S(i,j)$를, b_2는 $S(O(i,j))$를, 그리고 b_3는 $S(L(i,j))$를 나타낸다.

5) 노드 (i,j)는 $S(D(i,j))=1$일 때에 한해서 루트노드 경우는 3개 그리고 그렇지 않은 경우 4개의 자식(children) 노드를 가진다.

63	-34	49	10	7	13	-12	7
-31	23	14	-13	3	4	6	-1
15	14	3	-12	5	-7	3	9
-9	-7	-14	8	4	-2	3	2
-5	9	-1	47	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4

그림 3. 3-레벨 웨이브릿 변환예

Fig. 3 An example of 3-level wavelet transform

1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

그림 4. 문턱치, 32에 대한 중요도-맵

Fig. 4 The significance map for threshold value, 32

SPIHT 알고리즘의 SOT 코딩트리는 웨이브렛 변환에서 에너지는 저주파 영역에 집중된다는 원리에 입각해 알고리즘의 시간 복잡도를 희생하면서도 코딩 볼륨을 줄여보려는 시도로 해석된다. 그러나 T-layer가 증가함에 따라 대부분의 고주파 영역 정보가 코딩 포함되면서 이러한 의도는 의미 없게되고 오히려 역효과를 나타낼 수 있다. 또한 낮은 T-layer에서도 그 효과는 미미한 것으로 파악된다. 왜냐하면 낮은 T-layer에서는 코딩정보의 볼륨이 매우 적기 때문에 SOT 코딩트리 구조가 그렇게 중요한 요소가 되지 못하기 때문이다. 이러한 사실들은 4장의 실험 결과로부터도 확인할 수 있다.

III. 재귀적 SPIHT알고리즘

본 연구에서는 반복적(iterative) 알고리즘인 SPIHT 알고리즘의 시간 복잡도를 개선하고 정형화된 재귀적(recursive) 알고리즘, 즉 RSPIHT 알고리즘을 개발하기 위해 3.1절에서는 보다 단순하고 새로운 중요도-맵의 SOT 코딩 트리구조를 제시하고 3.2절에서는 RSPIHT 알고리즘을 제안한다.

3.1 중요도-맵의 SOT 코딩트리 구조

재귀적 알고리즘을 위한 중요도-맵의 SOT 코딩트리 구조는 중요도-맵 상 하나의 루트에 대한 SOT 코딩트리 상의 노드에 대해 다음처럼 정의된다.

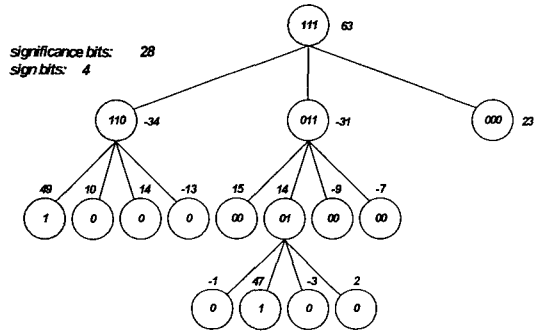


그림 5. 그림 3에 대한 SOT 코딩트리 구조

Fig. 5 The SOT coding tree structure for Fig. 3

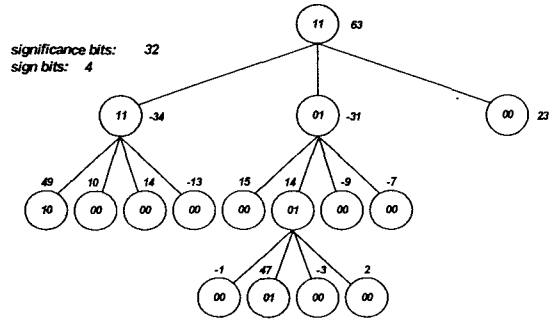


그림 6. 재귀적 알고리즘의 SOT 코딩트리 구조

Fig. 6 The SOT coding tree structure for recursive algorithm

1) 노드 (i,j)는 $S(i,j)=1$ 혹은 $S(D(i,j))=1$ 혹은 $O(P(i,j))=1$이다.

2) 노드 (i,j)는 2비트 정보 b_1b_2를 가지며 b_1은 $S(i,j)$를 그리고 b2는 $S(D(i,j))$를 나타낸다.

3) 노드 (i,j)는 $S(D(i,j))=1$일 때에 한해서 루트노드 경우는 3개 그리고 그렇지 않은 경우 4개의 자식(children) 노드를 가진다.

그림4에서처럼 본 연구에서 제시한 중요도-맵의 코딩트리 구조는 SPIHT 알고리즘의 데이터 구조와 동일하나 각 노드의 비트정보가 다르고 2비트로 일정하다. 이 코딩트리 구조는 그림3과 같은 변환 이미지로부터 재귀적으로 생성, 즉 코딩할 수 있고 그 반대로 수신된 SOT 코딩트리 정보로부터 중요도-맵을 손쉽게 재구성, 즉 디코딩할 수 있다.

3.2 재귀적 알고리즘과 시간복잡도

중요도-맵의 재귀적 SOT 코딩트리 정보의 생성은 정형화되어 매우 단순한 재귀적 알고리즘 즉, Algorithm II로 구현되며 Algorithm I의 소팅패스를 대신한다. 물론 Algorithm I의 소팅패스에서 중요 화소(significant pixel)의 수 μ_n의 전송은 필요없게 된다. 이 알고리즘에서 표현의 간결성을 위해 3가지 함수를 사용한다.

root(T); 파티션 T의 루트 좌표를 리턴한다.

out_significance(x); x를 코딩트리의 노드정보로 전송한다.

out_self_significance$(x,(i,j))$; x를 코딩트리의 노드정보로 전송하고 변환 이미지 상 화소 (i,j)의 값을 0으로(Clean-up) 한다.

out_sign_bit(i,j); 변환 이미지 상 화소 (i,j)의 부호비트(sign bit)를 전송한다.

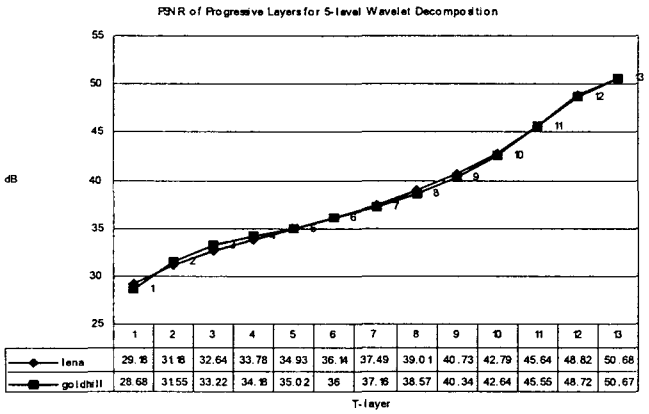
```
Algorithm II
main_recursive_spiht(T,n)
begin
    if T = Ø then return;
    partition T into the child subsets {T1,T2,T3};
    sig_val_1=create_sig_tree(T1, n-1);
    sig_val_2=create_sig_tree(T2, n-1);
    sig_val_3=create_sig_tree(T3, n-1);
    sig_val=sig_val_1+sig_val_2+sig_val_3;
    if sig_val>=1 then sig_val=1
                /* D(0,0)=sig_val */
    out_self_significance(Sn(root(T)));
                /* self significance, Sn(root(T))=Sn(0,0) */
    out_siginificance(sig_val);
                /* descendant significance */
    if S(root(T))=1 then out_sign_bit(root(T));
                /* sign bit */
    if sig_val=1 then
    begin
        out_self_significance(Sn(root(T1)),root(T1));
                /* child_1 self significance */
        out_siginificance(sig_val_1);
                /* child_1 descendant significance */
        if S(root(T1))=1 then out_sign_bit(root(T1));
                /* sign bit */
        out_self_significance(Sn(root(T2)),root(T2));
                /* child_2 self significance */
        out_siginificance(sig_val_2);
                /* child_2 descendant significance */
        if S(root(T2))=1 then out_sign_bit(root(T2));
                /* sign bit */
        out_self_significance(Sn(root(T3)),root(T3));
                /* child_3 self significance */
        out_siginificance(sig_val_3);
                /* child_3 descendant significance */
        if S(root(T3))=1 then out_sign_bit(root(T3));
                /* sign bit */
    end
end
create_sig_tree(T, n)
begin
    if n=1 then return(Sn(root(T)));
                /* leaf node */
    partition T into the child subsets {T1,T2,T3,T4};
    sig_val_1=create_sig_tree(T1, n-1);
    sig_val_2=create_sig_tree(T2, n-1);
    sig_val_3=create_sig_tree(T3, n-1);
    sig_val_4=create_sig_tree(T4, n-1);
    sig_val=sig_val_1+sig_val_2+sig_val_3+sig_val_4;
    if sig_val=0 then return(sig_val);
    if sig_val>=1 then sig_val=1;
    out_self_significance(Sn(root(T1)),root(T1));
                /* child_1 self significance */
    out_significance(sig_val_1);
                /* child_1 descendant significance */
    if S(root(T1))=1 then out_sign_bit(root(T1));
                /* sign bit */
    out_self_significance(Sn(root(T2)),root(T2));
                /* child_2 self significance */
    out_significance(sig_val_2);
                /* child_2 descendant significance */
    if S(root(T2))=1 then out_sign_bit(root(T2));
                /* sign bit */
    out_self_significance(Sn(root(T3)),root(T3));
                /* child_3 self significance */
    out_significance(sig_val_3);
                /* child_3 descendant significance */
    if S(root(T3))=1 then out_sign_bit(root(T3));
                /* sign bit */
    out_self_significance(Sn(root(T4)),root(T4));
                /* child_4 self significance */
    out_significance(sig_val_4);
                /* child_4 descendant significance */
    if S(root(T4))=1 then out_sign_bit(root(T4));
                /* sign bit */
    return(sig_val);
end
```

SPIHT알고리즘은 총화소 수를 n에 대해 최악의 경우 $O(n\log_2 n)$의 비선형다항 시간복잡도를 가진다[14]. 왜냐하면 SOT 코딩트리 정보 생성과정에서 루트로부터 SOT 상 노드 탐색과정에서 각 노드 (i,j)의 정보를 결정하기 위해서는 매번 $S_n(i,j)$의 결정과 더불어 $S_n(D(i,j))$와 $S_n(L(i,j))$의 평가가 부가적으로 필요하기 때문이다. 반면에 본 연구의 RSPIHT 알고리즘은 DFS(Depth First Search)에 의해 모든 화소 즉, 노드를 단 한번 방문하게 됨으로 최악의 경우 $O(n)$의 선형다항 시간복잡도를 가진다.

Ⅳ. 실험 및 고찰

본 연구에서 제안한 재귀적 SPIHT 알고리즘의 유효성

을 실험적으로 확인하기 위해서 처리시간 및 압축효과를 SPIHT 알고리즘과 비교 평가한다. Shapro의 EZW 알고리즘[4]은 이미 SPIHT 알고리즘과의 비교로부터 열등한 것으로 보고되었기 때문에 비교대상에서 제외한다. 그래서 SPIHT 알고리즘과 RSPIHT 알고리즘을 Windows NT Server 환경에서 C언어로 각각 구현하고, 512*512 크기의 Lena와 Goldhill 영상[12]을 실험 영상으로 사용하였다.



	1	2	3	4	5	6	7	8	9	10	11	12	13
lena	29.18	31.18	32.64	33.78	34.93	36.14	37.49	39.01	40.73	42.79	45.64	48.82	50.68
goldhill	28.68	31.55	33.22	34.18	35.02	36	37.18	38.57	40.34	42.64	45.56	48.72	50.67

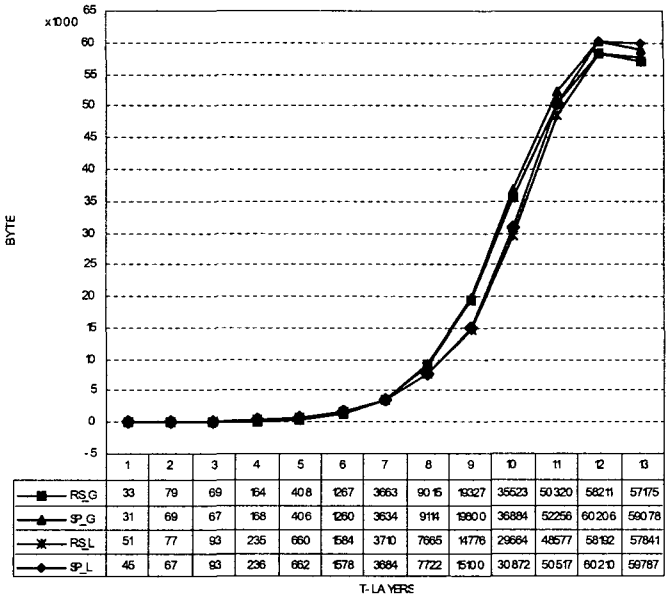
그림 7. T-레이어에 따른 PSNR 비교

Fig. 7 The comparison of PSNR according to T-layer

실험영상에 대한 웨이브렛 변환은 MatLab 환경에서 Harr 웨이브렛[13]을 사용하였다. 본 연구의 목적은 웨이브렛 변환의 효율이 아니라 변환 이미지의 코딩 알고리즘의 개선이기 때문에 특별한 웨이브렛의 사용을 고려하지 않았다. 웨이브렛 변환에 있어서 3, 4, 5-레벨 분해 각각에 대해 실험을 수행하였고 대체로 유사한 결과가 얻어졌으나 그 중에서 5-레벨 분해가 가장 양호한 것으로 나타났다. 본 실험결과 평가에서는 5-레벨 분해변환 계수 데이터를 사용하고 유한한 시그니피컨스-맵의 비트평면을 얻기 위해 변환계수는 정수값으로 절단하여 사용하였다. 알고리즘을 통한 영상코딩 결과와 산술코더를 사용하여 추가적인 압축을 수행한 결과도 비교한다. 산술코딩을 위해서 Mofatt[11]의 산술코더를 사용하였고 컨텍스트 비트(context bits)는 4, 8, 그리고 16비트 중에서 가장 양호한 결과를 주는 8비트를 사용하였다. 처리기는 128MB 메인메모리를 가진 LG-IBM ThankPad T20 노트북을 사용하였다.

$$PSNR = 10\log_{10}\left(\frac{255^2}{MSE}\right) \quad [dB] \qquad (3)$$

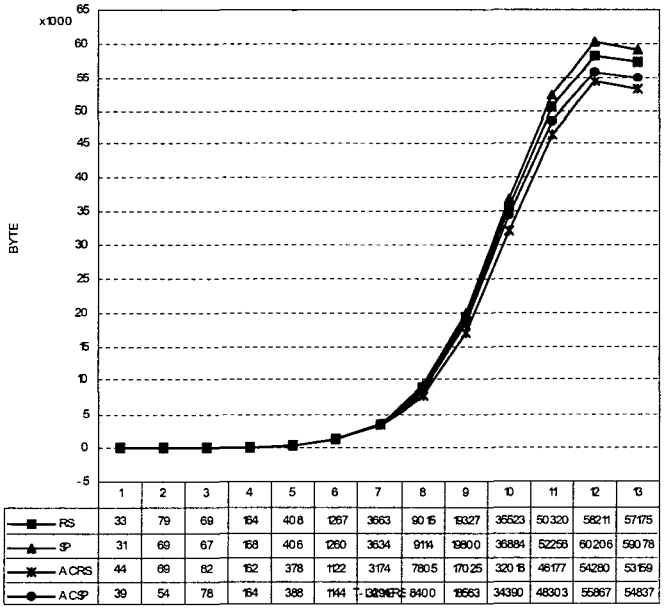
그림 7은 T-layer에 따른 식 (3)의 PSNR(peak signal to noise ratio)을 보이고 있다. 이 식에서 MSE는 원영상과 재구성 영상간의 평균자승오차(mean squared error)를 의미한다. 그림 7로부터 T-layer에 따라 PSNR이 거의 선형적으로 증가함을 알 수 있고 이 특성은 영상의 계층화(Layering) 혹은 영상의 계층적 전송(layered transmission)에 문턱치가 계층화의 기준으로 유효하게 사용될 수 있음을 보이고 있다. 여기서, 총 레이어 수를 N 이라할 때, T-layer i 는 2^{N-i} 의 문턱치를 가진다. 물론 T-layer에 따른 재구성 영상의 PSNR은 양 알고리즘에 대해 동일한 결과를 가진다.



	1	2	3	4	5	6	7	8	9	10	11	12	13
RS_G	33	79	69	164	408	1267	3663	9015	19327	35523	50320	58211	57175
SP_G	31	69	67	168	406	1260	3634	9114	19800	36884	52256	60206	59078
RS_L	51	77	93	235	660	1584	3710	7865	14776	29664	48577	58192	57841
SP_L	45	67	93	236	662	1578	3684	7722	15100	30872	50517	60210	59787

그림 8. 산술코딩없는 코딩율의 비교

Fig. 8 The comparison of coding rate without arithmetic coding



	1	2	3	4	5	6	7	8	9	10	11	12	13
RS	33	79	69	164	408	1267	3663	9015	19327	35523	50320	58211	57175
SP	31	69	67	168	406	1260	3634	9114	19800	36884	52256	60206	59078
ACRS	44	69	82	162	378	1122	3174	7805	17025	32018	46177	54280	53159
ACSP	39	54	78	164	388	1144	[illegible]	8400	18563	34390	48303	55867	54837

그림 9. Goldhill 영상의 코딩율 비교

Fig. 9 The comparison of coding rate for Goldhill image

그림 8은 T-layer에 따른 추가적 산술코딩을 행하지 않은 코딩결과의 볼륨을 바이트 단위로 정리한 것이다.

RS_G(recursive SPIHT for Goldhill image)와 RS_L(recursive SPIHT for Lena image)는 각각 Goldhill 영상과 Lena 영상에 대한 RSPIHT 알고리즘을 의미하고, SP_G(SPIHT for Goldhill image)와 SP_L(SPIHT for Lena image)는 각각 Goldhill 영상과 Lena 영상에 대한 SPIHT 알고리즘을 의미한다. 이 결과로부터 T-layer 7까지는 양 알고리즘이 거의 유사한 결과를 가지나 그 이후는 RSPIHT알고리즘 SPIHT 알고리즘보다 최대 약 3.9% 그리고 평균 약 3%의 개선된 압축효과를 가진다.

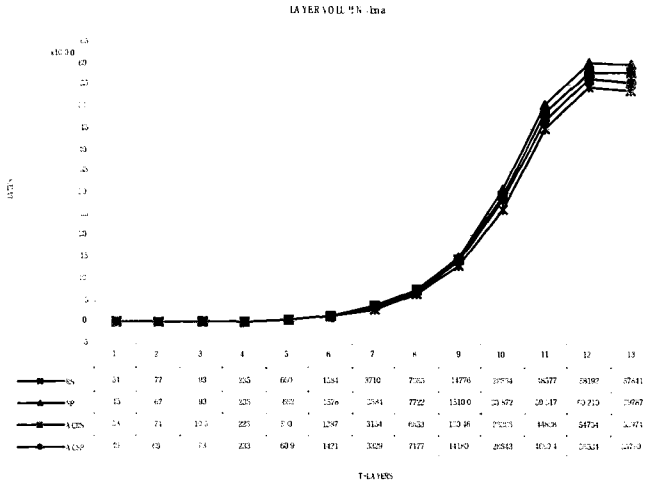


그림 10. Lena 영상의 코딩율 비교

Fig. 10 The comparison of coding rate for Lena image

그림 9와 10은 실험 영상 각각에 대한 산술코딩없는 경우(RS, SP)와 추가적 산술코딩 경우인 ACRS(arithmetic coded RSPIHT)와 ACSP(arithmetic coded SPIHT)의 코딩 볼륨을 보인다. 역시 어느 경우나 T-layer 7 이후는 SPIHT보다 RSPIHT 알고리즘이 효과적임을 알 수 있고 ASRS는 ASSP보다 goldhill 영상 경우 약 2.8%에서 8.3%까지 그리고 lena 영상 경우 약 3.1%에서 8%까지의 개선된 결과를 보이며 전체 평균으로는 약 5.5%의 개선된 결과를 보인다.

그림 11은 T-layer에 따른 bpp(bit-per-pixel) 단위로 누적 압축률(cumulated compression rate)을 정리한 것이다. 그 결과로부터 역시 T-layer 7까지는 모두가 거의 유사한 결과를 보이고 있으나 그 이후는 RSPIHT 알고리즘이 효과적임을 알 수 있다. 구체적으로 T-layer 8 이상에서 산술코딩을 수행하지 않은 경우는 SPIHT보다 평균 2.3% 그리고 산술코딩을 수행한 경우는 평균 약 5.7%의 개선된 결과를 보이고 있다.

그림 12는 양 알고리즘의 T-layer별 코딩 처리시간을 비교하고 있다. 본 연구의 알고리즘 구현에서는 표준 C 라이브러리를 이용하여 가능한 처리시간의 개선을 꾀하였으나 특별한 실시간 처리를 위한 배려는 없었다. 알고리즘의 시간복잡도의 관점에서 뿐만아니라 실제 처리시간의 경우도 T-layer 4 이상에서는 RSPIHT 알고리즘이 SPIHT보다 평균 약 16.4%의 개선을 확인할 수 있다. 특히 RSPIHT 알고리즘은 최악의 경우 시간복잡도가 $O(n)$이기 때문에 T-layer에 관계없이 대체로 100ms 내외의 일정한 처리시간을 가지나 SPIHT는 T-layer의 증가에 따라 처리시간이 최대 약 2배까지 증가함을 보이고 있다.

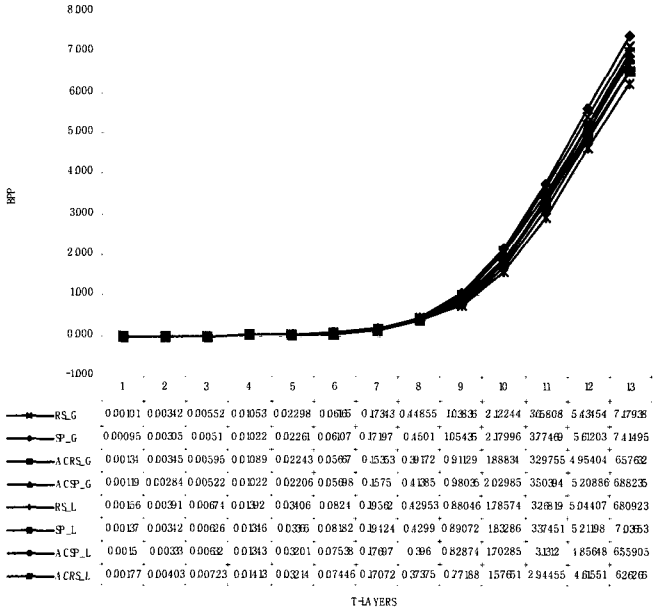


그림 11. T-레이어에 따른 누적코딩율의 비교

Fig. 11 The comparison of cumulated coding rate according to T-layer

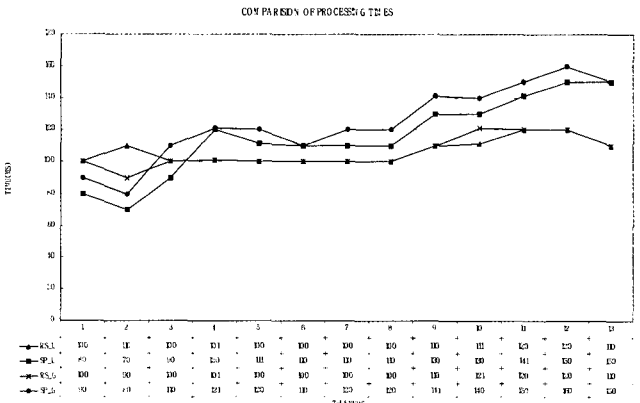


그림 12. 처리시간의 비교

Fig. 12 The comparison of processing time

V. 결론

본 연구에서는 내장형 영상코딩을 위한 SPIHT알고리즘의 우수한 성능과 2D 뿐만아니라 동영상을 위한 3D 압축에서의 발전 가능성에 주목하고 개선된 재귀적

SPIHT(RSPIHT) 알고리즘을 제안하고 그 유효성을 실험적으로 평가하였다.

본 연구에서 제안한 RSPIHT 알고리즘은 매우 정형화된 형태를 지니면서 최악의 경우 시간복잡도 $O(n)$을 가지기 때문에 T-layer에 대체로 무관하게 일정한 처리속도를 가지며 실험 영상들에 대해 T-layer 4 이상에서 SPIHT보다 평균 약 16.4%의 개선된 속도를 얻을 수 있었다. 그리고 T-layer에 따른 신호대잡음비(PSNR)는 SPIHT와 동일하며 선형적으로 증가한다는 특성을 가진다. 이러한 특성들은 성능이 다른 네트웍 자원을 충분히 활용하기 위한 계층적 프로그래시브 전송(layered progressive transmission)을 위해서는 매우 의미있는 것으로 평가된다.

압축률의 관점에서도 RSPIHT 알고리즘은 실험영상의 T-layer 7 이하에서는 SPIHT와 유사한 결과를 가지나 그보다 큰 T-layer에서는 개선된 결과를 보였다. 구체적으로 T-layer 8 이상에서 산술코딩을 행하지 않은 경우 T-layer당 압축률은 평균 약 3% 개선되고 산술코딩을 행한 경우 평균 약 5.5%의 개선이 있었다. 그리고 T-layer에 따른 누적 압축률 또한 산술코딩없는 경우 평균 약 2.3% 그리고 산술코딩있는 경우 평균 약 5.7% 개선되었다. 물론 이러한 결과들은 웨이브렛변환 필터나 산술코더의 개선에 따라 더욱 개선될 수 있다.

향후 연구의 고려점으로는 특히 RSPIHT를 비롯한 EZW 계열 알고리즘의 단점으로 지적될 수 있는 비트에러에 대한 민감성 문제해결 방안과 네트웍 응용을 위한 데이터 비트율 제어를 포함하는 스케일어빌리티(scalability) 등을 포함하는 효과적인 블록코딩 개념의 고안, 그리고 동영상의 내장형 압축을 위한 Layered 3D 계층구조(hierarchy) 및 알고리즘의 개발 등을 들 수 있다.

참고문헌

[1] D. S. Taubman and M. W. Marcellin,"JPEG2000: Image Compression Fundamentals, Standards and Practice, Kluwer Academic Publishers, 2002

[2] J. Y. Tham, S. Ranganath, and A. A. Kassim, "Highly scalable wavelet-based video codec for very low bit-rate environment," IEEE Journal on Selected Area in Communications, vol. 16, pp. 12--27, Jan. 1998.

[3] Amir Said, William A. Pearlman, A New Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees, IEEE Transactions on Circuits and Systems for Video Technology, Vol 6, pp. 243-250, 1996.

[4] J. M. shapiro, "Embedded image coding using zerotrees of wavelet coefficients", IEEE Transactions on Signal Processing, Vol. 41, No. 12 , pp. 3445 3462 Dec. 1993.

[5] David Taubman, "High Performance Scalable Image Compression with EBCOT", IEEE Transaction on Image Processing, Vol. 9, No. 7, July 2000.

[6] Z. Xiong, K. Ramchandran, and M. Orchard, "Space-frequency Quantization for Wavelet Image Coding", IEEE Trans. on Image Processing, Vol. 6, pp. 677-693, May 1997.

[7] M. Tsai, J. Villasenor, and F. Chen, "Stack-run Image Coding,"IEEE Trans. on Circuits Syst. Video Technol., Vol. 6, pp. 519-521. Oct. 1996.

[8] Image Communication Lab, Wavelet Image Coding: PSNR Results. UCLA School of Engineering and Applied Sciences. Available http://www.icsl.ucla.edu/~ipl/psnr_results.html.

[9] Y. W. Chen and W. A. Pearlman, "Three-Dimensional Subband Coding of Video Using the Zero-Tree Method," in symp. on Visual commun. and Image Processing, SPIE, Vol. 2727, Mar. 1996.

[10] A. Said and W. A. Pealman, "Reversible Image Compression via Multiresolution Representation and Predictive Coding," in Proc. SPIE Conf. Visual Commun. and Image Processing '93, SPIE, Vol. 2094, pp. 664-674, Nov. 1993.

[11] Moffat, Neal, and Witten, "Arithmetic Coding Revisited", ACM Transactions on Information Systems, Vo;.16, No. 3, pp. 256-294, July 1998. or Available ftp://munnari.mu.oz.au/pub/ arith_coder

[12] Available http://www.fhtw-berlin.de/Projekte/Wavelet/still.html

[13] M. Misiti, Y. Misiti, G. Oppenheim, and J. M. Poggi, Wavelet Toolbox User's Guide, The MathWork Inc., March 1996.

[14] 박영석, "SPIHT 영상코딩 알고리즘의 시간복잡도 해석", 한국신호처리·시스템학회논문지, 제4권 1호, 2002

박 영 석 (Young-Seak Park)
正會員
1979년 영남대학교 전자공학과 (공학사)
1981년 한양대학교 전자공학과 (공학석사)
1985년 한양대학교 전자공학과 (공학박사)
1990년~1991년 일본 우정성 통신총합연구소(관서선단연구센터) 초빙과학자
1990년 ~ 1991년 일본 긴끼이동통신센터 객원연구원
2001년 ~ 2002년 미국 North Carolina 주립대학(NCSU) 교환교수
1985년 ~ 현재 경남대학교 정보통신공학부 교수
관심분야 : Software Engineering, Web-based Software Design & Development, Pattern Recognition, Image Processing, Computer Network & Network Computing