FHD@240Hz 디스플레이 시스템을 위한 JPEG 기반 정지영상 코덱의 구조

박현상
공주대학교 전기전자제어공학부
vandammm@kongju.ac.kr

JPEG-based Still Image Codec Architecture for Display Systems at FHD@240Hz

Park, Hyun Sang
Division of EECE, Kongju National University

요약

240 Hz 이상의 높은 프레임율을 가지는 LCD 기반 평판 FHD급 디스플레이 시스템은 높은 프레임율로 인하여 디스플레이 패널로 전송해야하는 유효한 데이터율이 1.9GB/s까지 이르게 되며, 수평/수직 동기를 감안하면 2GB/s 이상의 데이터 전송 대역이 필요하다. DRAM을 이용하여 이런 데이터 대역폭을 제공하려면 다수의 메모리 장치를 사용해야하기 때문에, 비용 상승, 전력소모량 증가 등의 문제를 야기한다. 이런 문제를 해결하기 위하여 본 논문에서는 JPEG 기반의 정지 영상 압축 시스템을 제안한다. 제안한 시스템은 8개의 디코더가 동시에 동작하는 구조를 가지고 있으며, 단일 데이터 열로부터 8개의 데이터 열을 용이하게 구분할 수 있도록 128-bit 데이터에 정렬된 64-bit 마커를 사용한다. 제안한 64-bit 마커는 마커 에뮬레이션을 야기하지 않도록 설계되었기 때문에, 인코더와 디코더의 구현 복잡도를 낮출 수 있고 단일 데이터열을 8개의 데이터열로 분리하는 작업을 매우 용이하게 한다.

1. 서론

평판 디스플레이 패널을 채택한 고품격 디스플레이 시스템이 보편적으로 사용되고 있다. LCD는 PDP에 비해서 응답시간이 느려서 동영상에 부적합하다는 평가를 받아왔지만, 발열과 전력사용량 관점에서 우월한 특징을 가지고 있기 때문에, 대부분의 평판 디스플레이 시스템에서는 LCD가 사용되고 있다[1]. LCD에서의 응답시간이란 액정에 인가되는 전압에 따라 액정 셀이 얼마 후에 반응하는지를 나타내며, 밀리초 단위로 표시한다. 응답시간은 작을수록 영상을 빠르게 자주 출력하기 때문에 영상이 더욱 부드럽게 보이게 된다.

빠른 응답 속도를 가지는 디스플레이 장치에서는 높은 프레임율로 화면을 재생(refresh)함으로써 부드러운 영상을 재현할 수 있으며, 대화면 TV나 고부가가치 모니터에서 사용하는 LCD는 240 Hz 이상의 높은 프레임율을 가진다. 그러나 이와 같은 프레임율은 디스플레이 단에서만 해당되는 것으로써, 카메라 단에서의 프레임율은 최대 60 Hz에 불과하다. 그럼에도 불구하고 화면에는 240 Hz로 나타나야하므로 같은 영상이 4번 이상 반복되어 화면에 나타나야 한다. 보편적으로 사용되는 소스 영상의 프레임율이 30 Hz라는 것을 감안하면, 같은 영상이 8번 이상 반복되어 LCD로 나타나는 경우도 발생하게 된다.

FHD(Full High Definition)급 화면의 해상도는 1920x1080이다. 한 화소를 R, G, B, A(alpha blending)와 같은 4 바이트로 표현할 경우, 한 화면에 해당하는 데이터량은 8.3 MB이다. 따라서 초당 240 Hz로 리프레쉬하는 LCD에서 필요로 하는 유효 데이터율은 1.99 GB/s에 해당한다. 이런 데이터율을 충족시키면 영상을 최대 속도가 800 Mbps인 8-bit DDR2[2]를 3개 사용해야만 달성 가능하다. 그럼에도 불구하고 필요한 메모리 용량은 66 Mb나 132 Mb에 불과하기 때문에, 메모리 용량만을 보면 DDR2는 한 개만으로 충분하지만, 데이터율을 만족시키기 위해서 3개의 DDR2를 사용하게 되는 것이다. 이는 부품 및 PCB 등의 BOM(Bill Of Materials)증가, 전력 사용량 증가를 야기하므로 단순 리프레쉬를 위해서라면 영상을 압축해서 저장하는 것이 바람직하다. 이 경우에 필요한 압축률은 4:1 이상이다.

무손실 JPEG[3][4]은 DPCM(Differential Pulse Coded Modulation)기반으로 주어진 화소와, 주변 화소로부터 구해진 예측값의 차이를 엔트로피 코딩해서 전송하는 구조를 가진다. 그러나 45 nm급의 최신 공정에서도 300 MHz이상으로 동작하는 시스템버스를 가지는 SoC 구현이 어렵다는 것을 감안할 때, 1.9 GB/s의 디코딩 성능을 가지는 무손실 JPEG 디코더의 구현은 현실적으로 어렵다. 게다가 영

상에 따라서 압축률이 가변적이어서, 4:1의 압축률을 보장하지도 않기 때문에 디스플레이 장치 내에서는 적용되지 않고 있다. 준무손실(Near-lossless) 압축 기법[3][5]들도 최근에 많이 제안되고 있다. 이런 기법들에서는 압축률은 보장되지만 영상의 화질이 보장되지 않으며, 역시 1.9 GB/s의 디코딩 성능을 가지는 것이 용이하지 않다는 문제를 공유하고 있다.

본 논문에서는 이미 대중화되어 있는 기술인 JPEG[6]에 기반을 둔 정지영상 압축/복원 시스템의 구조를 제안한다. 제안된 시스템은 8개의 범용 JPEG 디코더를 병렬로 사용하여 1.9 GB/s의 데이터 처리율을 충족시킨다. 각 JPEG 디코더의 최대 동작 주파수는 300 MHz 이하로 제한되며, 이런 수준의 동작 속도는 45 nm급의 반도체 공정에서는 충분히 구현가능하다.

2. 본론

2.1. 정지영상 압축 시스템의 구조

FHD영상을 초당 60장 실시간으로 압축하는 것이 목표다. 각 화소는 4개의 색성분으로 구성되어 있기 때문에, 압축해야할 순수한 영상 데이터의 양은 프레임 당 498MB/s(=1920x1080x4x8)가 되므로 300 MHz로 동작할 수 있는 JPEG 인코더를 사용하더라도 2개의 인코더를 병렬 동작시켜야만 하는 규모이다. 범용 JPEG 인코더는 8-bit 데이터를 입력으로 받아들이기 때문에, 2개의 JPEG 인코더를 사용할 경우 그림 1과 같이 16-bit 데이터를 입력으로 제공해야 한다.

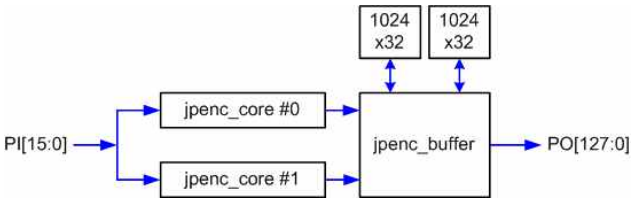

그림 1. 정지영상 압축 시스템의 구조
Fig. 1. Still-image compression system architecture.

그림 1에서 ***jpenc_core***는 범용 JPEG 인코더를 나타낸다. 압축하려는 영상은 4개의 색성분으로 구성되어 있기 때문에, JPEG 인코더 #0은 1, 3번째 색성분을 순차적으로 압축하고, JPEG 인코더 #1은 2, 4번째 색성분을 순차적으로 압축한다. 각 인코더에서 생성된 바이트 단위의 데이터열은 ***jpenc_buffer*** 모듈로 전달된다. 이 모듈의 목적은 동시에 출력되는 데이터열을 병합하여 하나의 데이터열로 만들기 위함이다.

모듈 ***jpenc_buffer***은 인코더들에서 출력되는 데이터열을 2개의 1024x32 크기의 SRAM에 각각 저장한다. SRAM에 저장이 완료되면 JPEG 인코더 #0에 의해서 생성된 데이터를 외부로 모두 출력하고 나서, JPEG 인코더 #1에 의해서 생성된 데이터를 외부로 출력하는 방식으로 2개의 인코더에서 생성된 데이터열을 그림 2와 같이 하나로 통합한다. 그림 2에서 **PI0_EN**, **PI1_EN**은 각 JPEG 인코더에서 출력되는 데이터 스트로브 신호이며, SRAM에 저장하는 신호로 사용된다. *rd_en0*, *rd_en1*은 각 SRAM을 읽기 위한 신호이며, JPEG 인코더 #0에 대응하는 데이터열을 모두 읽은 후에, JPEG 인코더 #1에 대응하는 데이터를 읽어서 출력한다.

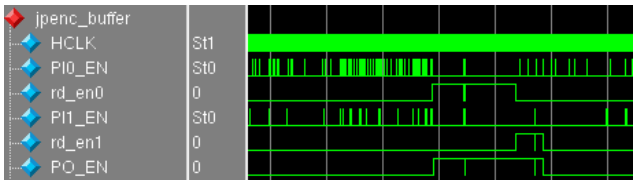

그림 2. 모듈 jpenc_buffer의 타이밍도
Fig. 2. Simulated timing diagram of jpenc_buffer

SRAM의 크기는 각 JPEG 인코더가 생성하는 최대 데이터양을 고려해서 결정해야한다. 본 논문에서 제안한 압축 시스템은 슬라이스 단위로 영상을 압축한다. 슬라이스의 크기는 압축하고자 하는 영상과 같은 수평해상도를 가지나 높이는 8로 고정된 영상 블록을 지칭한다. 각 슬라이스는 독립적인 영상으로 간주되므로, 영상의 시작을 나타내는 SOI(Start of Image) 마커와, EOI(End of Image) 마커가 데이터열의 앞과 뒤에 각각 삽입된다.

2.2. 고속 디코딩을 위한 64-bit 마커

제안한 압축 시스템은 범용 JPEG 인코더의 압축 엔진을 수정 없이 사용하지만 헤더를 생성하는 부분은 새롭게 정의한다. 입력된 영상은 슬라이스와 색성분 단위로 독립적으로 압축되는 구조를 가지기 때문에, 다음과 같이 4종류의 SOI를 정의한다.

0xffd0: SOI0, Start of slice for the 1-st component
0xffd1: SOI1, Start of slice for the 2-nd component
0xffd2: SOI2, Start of slice for the 3-rd component
0xffd3: SOI3, Start of slice for the 4-th component

슬라이스의 끝단에는 EOI(=0xffd9) 마커가 삽입되지만, 슬라이스의 정중앙에도 마커가 삽입된다. 슬라이스의 정중앙에 삽입되는 마커는 HOI로 정의하며 0xffda로 정의한다. 따라서 매 슬라이스마다 3개의 마커를 가지고 있으며, SOI로 시작해서 EOI로 끝나되, 중간에는 HOI가 전체 슬라이스를 2개로 서브슬라이스로 나누는 기능을 한다. 이와 같이 HOI 마커를 중간에 삽입하는 이유는 한 슬라이스를 독립적인 2개의 서브슬라이스로 분리함으로써, 디코더 단에서 각 서브슬라이스를 병렬 처리할 수 있는 여지를 주기 위함이다. 이런 병렬처리는 디코딩 능력을 2배로 향상시킨다. 따라서 제안한 영상압축 시스템에 출력되는 데이터는 다음과 같은 형식을 가진다. 여기서 *MaxSlice*는 슬라이스의 수를 나타낸다.

```
for (i=0; i<MaxSlice; i++){
  SOI0(=0xffd0)
    First sub-slice data for 1-st component
  HOI(=0xffda)
    Second sub-slice data for 1-st component
```

```
  EOI(=0xffd9)
  SOI2(=0xffd2)
    First sub-slice data for 3-rd component
  HOI(=0xffda)
    Second sub-slice data for 3-rd component
  EOI(=0xffd9)
  SOI1(=0xffd1)
    First sub-slice data for 2-nd component
  HOI(=0xffda)
    Second sub-slice data for 2-nd component
  EOI(=0xffd9)
  SOI3(=0xffd3)
    First sub-slice data for 4-th component
  HOI(=0xffda)
    Second sub-slice data for 4-th component
  EOI(=0xffd9)
}
```

일반적인 JPEG 규격에서 마커는 바이트 단위로 정렬되어 나타난다. 따라서 마커가 나타나기 전에 데이터열은 반드시 바이트 단위로 정렬되어있어야 하기 때문에, JPEG 인코더는 Huffman 인코더에 남아있는 비트 데이터를 출력한 후에, 바이트 단위로 정렬하기 위해서 필요한 만큼 1을 삽입(stuffing)한다.

본 논문에서 제안한 시스템은 128-bit 시스템 버스를 사용하는 SoC에서 사용되고, 디코더 단에서 고속 병렬처리가 가능하도록 하는 것이 목표이다. 128-bit 시스템 버스를 사용하는 SoC에서 사용하는 디코더는 128-bit 단위로 데이터를 읽어 들이기 때문에 각 슬라이스, 혹은 서브슬라이스를 신속하게 구분하기 위해서 마커가 128-bit 단위로 정렬되어 있는 것이 효과적이다.

통상적인 JPEG 규격에 의해서 압축할 경우, 데이터열 중에서 의도하지 않은 마커가 생성될 수 있다. 이를 마커 에뮬레이션(emulation)이라 하는데, JPEG 규격에서는 마커 에뮬레이션을 방지하기 위하여 데이터열에서 0xff가 생성될 때마다 0x00을 강제적으로 삽입하도록 규정되어 있다. 따라서 수신단 측에서는 0xff00의 데이터가 발견되면, 이를 0xff로 수정하여 디코딩해야만 한다. JPEG에서 고화질을 유지하기 위해서 양자화를 작게 하면, 마커 에뮬레이션이 자주 발생한다.

마커 에뮬레이션을 위해서 삽입된 0x00은 디코더 단에서 Huffman 디코딩 전에 반드시 제거되어야만 한다. 128-bit 단위로 디코딩할 경우, 128-bit내에 삽입될 수 있는 0x00의 수는 최대 8개가 된다. 임의의 0x00을 제거하는 하드웨어는 대규모의 멀티플렉서로 구현되어 많은 지연시간을 야기하여 하드웨어 구현을 복잡하게 만들고, 충분한 검증을 수반하지 않을 경우 예기하지 못한 오류를 발생할 수 있다는 면에서 디코더의 부담을 가중시킨다. 따라서 본 논문에서는 JPEG 규격에서 정의한 것과 같이 마커 에뮬레이션을 방지하기 위한 기능을 구현하지 않는다. 그 대신 16-bit 마커를 사용하지 않고 64-bit 마커를 다음과 같이 재정의한다.

SOI0: {0xffd0, 8-bit 수평해상도, 8-bit 수직해상도, 0x00000000}
SOI1: {0xffd1, 8-bit 수평해상도, 8-bit 수직해상도, 0x00000000}
SOI2: {0xffd2, 8-bit 수평해상도, 8-bit 수직해상도, 0x00000000}
SOI3: {0xffd3, 8-bit 수평해상도, 8-bit 수직해상도, 0x00000000}
HOI: 0xffff_ffff_ffff_ffda
EOI: 0xffff_ffff_ffff_ffd9

디코더 단에서는 64-bit 마커를 구분하면 되고, 마커는 128-bit 단위로 정렬된 데이터에 삽입되어 나타나기 때문에, 메모리로부터 읽은 128-bit 데이터 중에서 SOI의 경우에는 상위 64-bit, HOI, EOI등은 하위 64-bit만을 참조하면 마커를 검출할 수 있다. 마커 에뮬레이션 가능성을 살펴보겠다.

HOI, EOI 마커의 경우 '1'로만 구성된 매우 긴 데이터열이 발생하면, 마커 에뮬레이션 가능성이 있다. JPEG에서는 양자화된 DCT 계수를 Huffman 코딩할 때, Run/Size에 대한 VLC 코드와, 양자화된 계수의 크기를 전송한다. 양자화된 계수의 길이는 최대 16-bit이며, 임의의 값을 가질 수 있다. JPEG 규격에 정의된 Run/Size에 VLC code들을 살펴보면, 가장 긴 '1'의 열로 끝나는 코드는 다음과 같으며, 끝단에 6개의 '1'을 가진다.

Run/Size = 9/3, Codeword = 1111111110111111 (Table K.5)
Run/Size = 8/A, Codeword = 1111111110111111 (Table K.6)

JPEG 규격에 정의된 Run/Size에 VLC code에서 가장 긴 '1'의 열로 시작하는 코드는 다음과 같으며, 앞단에 15개의 '1'을 가진다.

Run/Size = F/A, Codeword = 1111111111111110 (Table K.5)
Run/Size = F/A, Codeword = 1111111111111110 (Table K.6)

따라서 JPEG에 의해서 압축할 때, Huffman 코딩에 의해서 발생할 수 있는 '1' 열의 최대 길이는 37(=6+16+15)다. 그러므로 HOI, EOI를 정의함에 있어서 연속된 '1'의 길이를 38 이상으로 정의하면 JPEG 압축과정에서 HOI, EOI가 발생할 가능성은 전혀 없어진다. 본 논문에서 정의한 HOI, EOI에는 58개의 연속된 '1'이 포함되어있기 때문에 마커 에뮬레이션 가능성은 없다.

본 논문에서 정의한 SOI에는 연속된 16개의 0을 삽입했다. 이 코드는 양자화된 DCT 계수가 -1이고, 연속적으로 5회 이상 발생할 때 16개 이상의 '0' bit가 발생할 수 있다. 양자된 DCT 계수가 -1을 가지고, 연속적으로 5개 이상 발생할 경우라면 AC 성분의 에너지가 매우 작은 경우에 해당하므로 6번째 발생하는 -1의 경우에는 강제적으로 0으로 만듦으로서 마커 에뮬레이션 가능성을 없애는 것이 가능하다. 따라서 제안한 압축 시스템에서 사용하는 양자화기는 양자화후 -1이 연속적으로 6번 발생하면, 6번째 결과를 강제로 0으로 바꾸는 기능을 추가한다.

마커를 128-bit로 정의하는 것도 가능하다. 이 경우 HOI, EOI는 112개의 '1'열 뒤에 0xffda, 0xffd9를 삽입된 형태를 가진다. SOI는 32-bit의 '0'열 대신에 96개의 '1'열을 사용함으로써 마커 에뮬레이션이 절대로 발생하지 않는 압축을 실현할 수 있다. 즉 64-bit 마커를 사용할 경우에는 양자화기의 구조에 약간의 제한을 가해야하지만, 128-bit 마커를 사용하면 그런 제한이 필요 없다. 마커의 비트폭이 커지게 되면 마커 에뮬레이션을 억제함으로써 하드웨어 구현을 용이하게 하지만, 압축효율이 다소 떨어지는 단점도 있다.

2.3. 정지영상 복원 시스템의 구조

4개의 색성분을 가지는 FHD영상을 초당 240장 디코딩하면, 최소 출력 데이터율이 1.99(=1920x1080x4x240) GB/s가 되며, 한 클럭에 1-byte씩 출력하는 디코더라면 2 GHz 이상의 동작주파수를 가져야 하지만, 이는 현실적으로 불가능하다. 따라서 1.99 GB/s의 데이터율을 실현시키려면 300 MB/s의 데이터율을 가지는 디코더라 하더라도 8개 병렬로 구동시켜야만 가능해진다. 그림 3은 1.99 GB/s의 데이터율로 디코딩이 가능한 JPEG 기반 영상 복원 시스템의 구조이다.

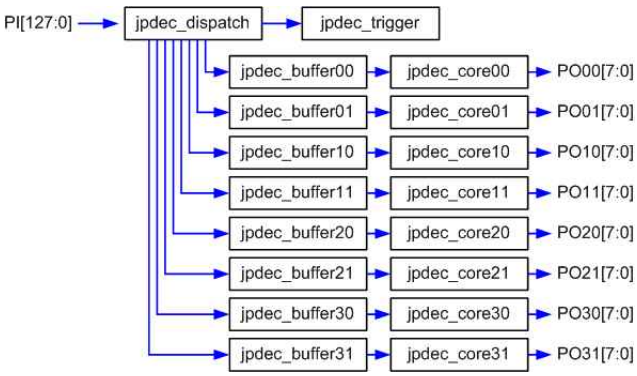

그림 3. 정지영상 복원 시스템의 구조

Fig. 3. Still-image de-compression system architecture.

그림 3에서 모듈 ***jpdec_dispatch***는 압축된 데이터를 시스템 메모리로부터 읽으면서, 128-bit 단위로 정렬된 EOI, HOI, EOI만을 보고서 서브-슬라이스 단위로 구분한다. 한 슬라이스는 4개의 색성분으로 구성되어 있고, 특정 색성분의 슬라이스는 2개의 서브 슬라이스로 구성되어 있으므로 ***jpdec_dispatch***에 의해서 모두 8개의 독립적인 단위로 구분할 수 있고, 이들은 독립적으로 JPEG 디코더에 의해서 디코딩될 수 있다. 그림 4는 ***jpdec_dispatch***에서 압축된 데이터가 8방향으로 분리되는 것을 보여주는 타이밍도이다. **PI_EN**은 입력 데이터 스트로브이고, 다른 신호들은 8개의 서브 슬라이스 단위로 구분해주는 출력 데이터 스트로브 신호들이다.

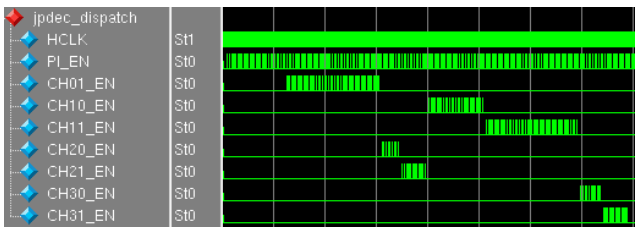

그림 4. 모듈 jpdec_dispatch의 타이밍도

Fig. 4. Simulated timing diagram of jpdec_dispatch

8개의 JPEG 디코더로부터 디코딩된 데이터가 동시에 출력될 수 있도록 8개의 JPEG 디코더에 대한 구동신호는 모듈 ***jpdec_trigger***가 동시에 제공한다. 모듈 ***jpdec_bufferXY***, $0 \le X \le 3$, $0 \le Y \le 1$은 서브 슬라이스에 해당하는 압축된 데이터를 저장하는 버퍼이며, ***jpdec_coreXY***, $0 \le X \le 3$, $0 \le Y \le 1$은 범용 JPEG 디코더이다.

3. 결론

제안한 영상 압축 복원 시스템은 FPGA에서 구현되어 정상 동작이 확인되었으며, 실험에 사용한 어떤 영상에서도 마커 에뮬레이션으로 인한 오동작이 발견되지 않았다. 본 논문에서 정의한 64-bit SOI 마커에는 이론적으로는 마커 에뮬레이션의 가능성을 가지고 있었으나 실험에 사용한 컴퓨터 그래픽까지 포함한 수십 종의 다양한 영상에서는 그런 경우가 발생하지 않았다. 물론 발생하더라도 양자화기에서 방지하는 장치를 구비하였기 때문에 디코딩 에러를 야기하지는 않는다.

현존하는 모든 종류의 압축 알고리즘은 잡음이 많은 영상에 대해서는 압축효율이 극단적으로 떨어지거나, 압축률을 높일 경우에는 화질이 크게 떨어지는 근본적인 한계를 가지고 있다. 그럼에도 불구하고 JPEG을 4:1 압축률에서 적용하면 자연영상 뿐만 아니라, 컴퓨터 그래픽에 의한 인공적인 영상에 대해서도 블록화 현상 없는 높은 화질을 가능하게 한다.

제안한 영상압축 시스템은 LCD를 채택한 평판 디스플레이 시스템에서 높은 프레임율을 제공하고자할 때, 메모리 용량과 필요한 대역폭을 큰 폭으로 감축시킬 수 있기 때문에, 효율적인 디스플레이 시스템의 구현에 적합하다.

4. 참고문헌

[1] 장진, *정보디스플레이개론*, 청문각, 2007년

[2] Micron technology, Inc., "*256Mb: x4, x8, x16 DDR2 SDRAM*", 2003.

[3] S. D. Rane and G. Sapiro, "Evaluation of JPEG-LS, the New Lossless and Controlled-Lossy Still Image Compression Standard, for Compression of High-Resolution Elevation Data," *IEEE Trans. on Geoscience and Remote Sensing*, Vol. 39, No. 10, pp. 2298-2305, Oct. 2001.

[4] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS," *IEEE Trans. Image Processing*, vol. 9, no. 8, pp. 1309-1324, Aug. 2000.

[5] E. Magli, "Optimized onboard lossless and near-lossless compression of hyperspectral data using CALIC," *IEEE Geoscience and Remote Sensing Letters*, vol. 1, no. 1, pp. 21-25, Jan. 2004.

[6] ISO/IEC, ISO/IEC 10918-1:1994, Information technology - Digital compression and coding of continuous-tone still images: Requirements and guidelines, 1994.