글. 조인준 KBS 기술연구소 차장

C군의 네버엔딩 스토리,

디지털 영상처리의 이해 - 5

지난 [디지털 영상처리의 이해] 4편에서는 카메라 등의 영상입력 기기에 의해 빛이 어떻게 디지털 영상데이터로 바뀐 후 파일로 저장되는지에 관한 개요를 설명했습니다. 설명은 주로 8비트(bit) 영상데이터의 예를 들었습니다. 영상데이터가 8비트라는 말은 영상 화소의 색상을 나타내는 R(빨강), G(초록), B(파랑)값이 8비트로 표현된다는 것입니다. 이진수의 표현에 8비트를 사용하면 2^8가지의 숫자를 표현할 수 있고, 십진수로 환산했을 때 0~255 사이의 값을 가질 수 있습니다. 더욱이 컴퓨터 기반의 기기들에서 사용하는 자료형들의 사이즈가 바이트(byte, 1바이트 = 8비트)의 배수이므로 8비트 영상데이터는 R, G, B 각각을 1바이트 크기의 자료형에 매핑(mapping)하여 처리하기가 편리합니다. 그렇지만 세상에 존재하는 영상데이터가 모두 8비트는 아닙니다. 방송제작에 사용되는 영상데이터는 10비트가 권장됩니다. 또한, 최근 각광받는 기술인 HDR(High Dynamic Range, 고명암비) 영상기술은 최소 10비트 이상의 영상데이터 사용을 요구하고 있습니다. 추후에 자세히 설명할 기회가 있을 테지만 앞서와 같은 여러 가지 이유로 8비트 이상의 영상 데이터를 사용해야 하는 경우가 있습니다.

그렇다면 바이트의 배수로 된 자료형을 다루는 컴퓨터 기반의 기기들을 이용하여 10비트나 12비트 영상데이터를 어떻게 처리하고 메모리나 파일 등에 쓰거나 읽을 수 있을까요? 이를 설명하기 위해 후반작업 또는 영상의 시각효과 등의 용도로 정의된 영상파일 포맷인 DPX(Digital Picture Exchange) 파일의 예를 들어보겠습니다. DPX 파일은 각 화소의 색을 나타내는 RGB 또는 YC_bC_{rs} 등의 영상데이터를 8비트 이상의 여러 종류의 비트수(10, 12, 16비트 등)를 할당하여 표현할 수 있도록 하고 있어 다양한 비트수의 영상데이터를 어떻게 처리/저장 하는지에 관한 실제적 설명에 유용하기 때문입니다.

앞선 연재에서 파일의 일반적 구조는 [그림 1]과 같다고 설명해 드렸습니다. 헤더에는 영상의 가로/세로 화소수를 포함하여 파일에 포함된 영상 데이터를 활용하는데 필요한 정보가 포함되어 있고, 각 정보의 바이트 수가 정의되어 있어서 그 정의에 따라서 헤더의 정보를 읽으면 필요한 정보를 얻을 수 있습니다.

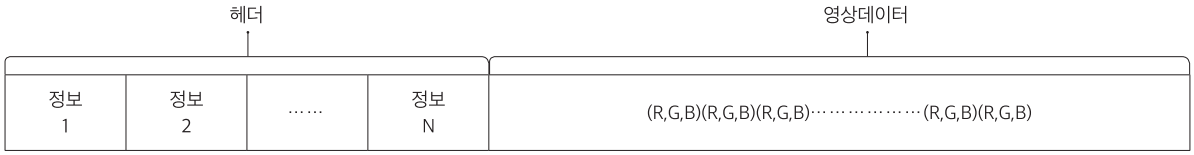


그림 1. 파일의 일반적 구조

DPX 파일의 경우에도 헤더부분에 영상의 화소수 정보 등 파일에 담긴 영상데이터를 읽기 위한 기본적 데이터는 물론, 영상데이터의

형태(RGB, YC_bC_r 등)와 비트수 등의 정보도 포함하고 있습니다. DPX 파일에서 사용할 수 있는 영상데이터의 비트수는 8, 10, 12, 16, 32, 64이며, 8, 10, 12, 16비트는 정수로 표현되는 영상데이터에, 32, 64비트는 소수로 표현되는 영상데이터에 사용됩니다. 만약 DPX 파일 헤더에 영상데이터의 형태가 RGB이고 영상데이터의 표현에 사용된 비트수가 8비트라고 표시되어 있다면 [그림 2]와 같이 각 화소의 색은 R, G, B로 나타내어지고 R, G, B 각각이 8비트(1바이트)로 표현되는 정수(0~255)라는 의미입니다.

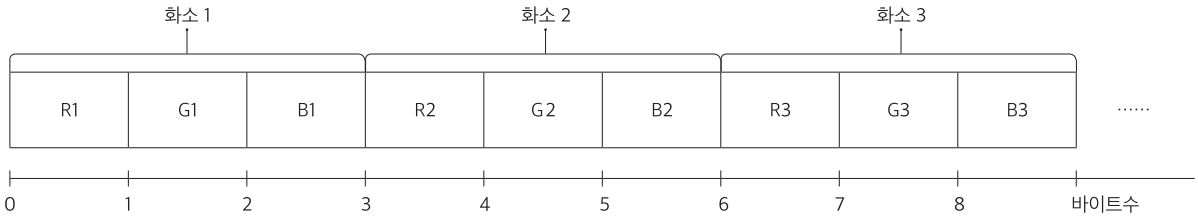


그림 2. 8비트 영상데이터의 바이트 구조

또한, DPX 파일 헤더에 영상데이터의 형태가 RGB이고 영상데이터의 표현에 사용된 비트수가 16비트라고 표시된 경우를 생각해보겠습니다. 이 경우는 [그림 3]과 같이 각 화소의 색은 [그림 2]의 예와 같이 R, G, B로 같지만 R, G, B 각각이 16비트(2바이트)로 표현되는 정수(0~65535)가 됩니다.

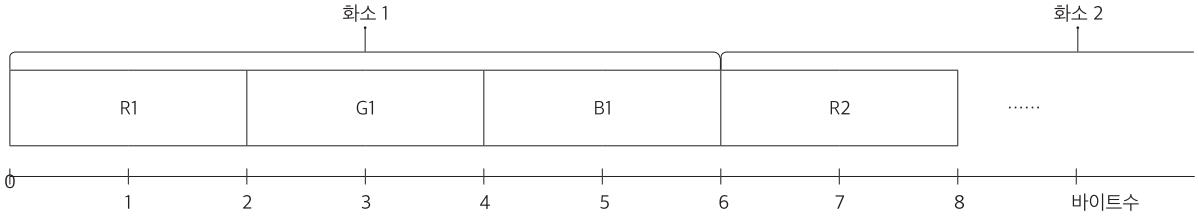


그림 3. 16비트 영상데이터의 바이트 구조

영상데이터에 32비트(4바이트)나 64비트(8바이트)를 사용하는 경우도 정수대신 32비트나 64비트로 표현되는 소수를 사용한다는 점만 빼면 [그림 2]와 [그림 3]의 경우와 동일한 패턴을 따릅니다. 영상데이터가 8, 16, 32, 64비트처럼 바이트의 배수로 이루어진 DPX 파일의 경우, 컴퓨터 기반의 기기로 처리할 때 기기 내부에서 사용하는 자료형들 중 같은 사이즈의 자료형으로 읽어 들이면 간단하게 영상데이터를 읽을 수 있습니다. 무언가 당연한 이야기 같으면서도 정확히 무슨 의미인지 선명하게 와 닿지 않으시죠? 그래서 [그림 4]와 [그림 5]를 준비했습니다. [그림 4]와 [그림 5]의 숫자가 쓰인 카드 9장은 각각의 십진수 12, 271, 9를 빈자리는 0으로 채워 세 자리로 만든 후 순서대로 배열한 것입니다. 바꾸어 말하면 세 자리씩 끊어 읽으면 12, 271, 9와 같이 원래의 숫자를 읽을 수 있는 것입니다. [그림 4]의 경우 숫자 카드의 배열 밑에 세 개씩 묶인 카드꽂이가 있습니다. 0을 이용하여 12, 271, 9를 세 자리로 맞춘 후, 하나로

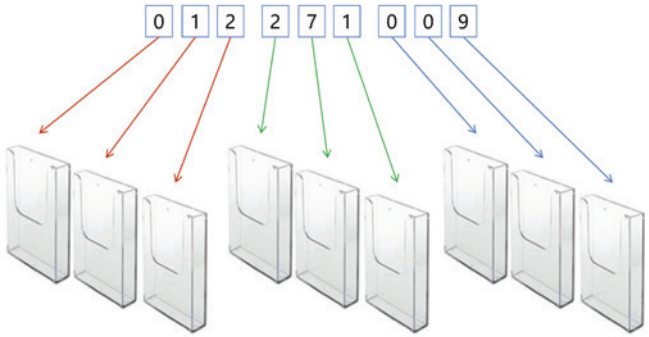


그림 4. 데이터와 이에 일치하는 자료형

조인준
KBS 기술연구소 차장

이어 붙인 9장의 숫자카드들을 세 개씩 묶인 카드 꽂이에 순서대로 꽂으면 012, 271, 009가 되고, 이를 읽으면 일렬로 나열되기 전 원래의 세 숫자들과 같이 12, 271, 9로 읽히겠죠? 여기서 일렬로 나열된 카드들은 파일의 영상데이터, 세 개씩 묶인 카드꽂이는 영상데이터의 사이즈에 맞춘 컴퓨터 기반 기기 내부의 자료형이라고 보시면 됩니다.

하지만 [그림 5]와 같이 카드꽂이가 4개씩 묶여 있는 경우, 일렬로 나열된 세 자리 숫자 카드들의 묶음을 원래의 세 숫자로 분리하기 위해서는 4개씩 묶인 카드 꽂이의 제일 왼쪽 카드 꽂이를 건너뛰며 숫자 카드를 꽂아야 원래의 세 숫자 12, 271, 9로 복원할 수 있습니다. 이렇듯 컴퓨터 기반 기기 내부의 자료형이라 할 수 있는 카드꽂이의 개수가 일렬로 배열한 각 숫자들의 자리수와 맞지 않으면 번거로운 처리를 해야 한 줄로 합쳐지기 전의 원래의 숫자로 분리해 낼 수 있습니다.

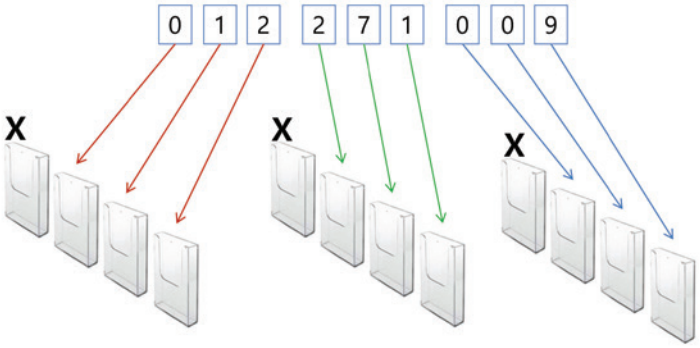


그림 5. 데이터와 이에 불일치하는 자료형

이렇듯 컴퓨터 기반 기기 내부의 자료형의 사이즈가 영상데이터의 사이즈와 맞지 않으면 번거로운 작업을 수행해야 하는데, 10, 12비트 영상데이터가 이런 번거로운 경우에 해당합니다. 구체적인 이해를 위해 DPX 파일의 경우 10, 12비트 영상데이터를 어떻게 저장하고 읽는지 설명해 드리겠습니다. DPX 파일의 경우 영상데이터의 종류와 사이즈 및 저장순서(위→아래, 아래→위, 좌→우, 우→좌 등)에 관한 다양한 옵션을 제공하므로, 예에서는 RGB 영상데이터를 순차적으로 저장하고 읽는 방법을 기준으로 설명하겠습니다.

10비트 영상데이터를 저장하고 읽는 방법

DPX 파일의 경우 10비트 영상데이터를 저장할 때 32비트(4바이트) 단위로 [그림 6] 또는 [그림 7]의 구조를 사용하여 저장합니다. [그림 6]과 [그림 7] 최상단의 숫자가 적힌 파란 사각형들은 32비트의 각 비트를 나타내는 인덱스입니다. [그림 6]의 경우 32비트(4바이트)를 첫 번째 화소의 R, G, B값으로 10비트씩 채우고 남은 두 비트를 두 번째 화소의 R값의 10비트 중 처음 2비트를 떼어내어 채운 다음, 다음의 32비트(4바이트)는 두 번째 화소의 R값의 나머지 8비트와 두 번째 화소의 G, B값의 각 10비트, 그리고 세 번째 화소의 R값의 4비트로 채우는 패턴을 반복하여 하나의 비트도 낭비하지 않고 영상데이터에 할당된 모든 비트에 데이터를 가득 채우는 방식입니다.

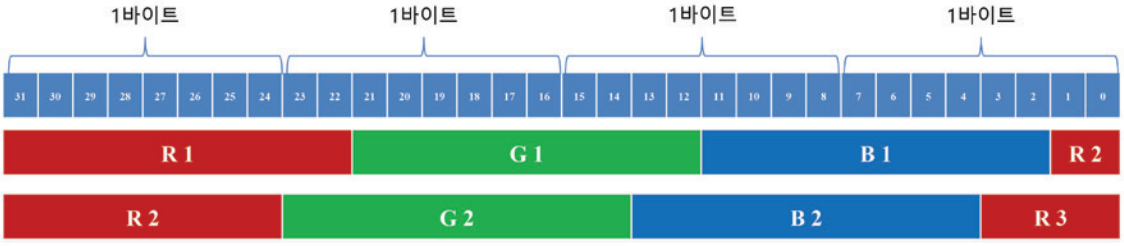
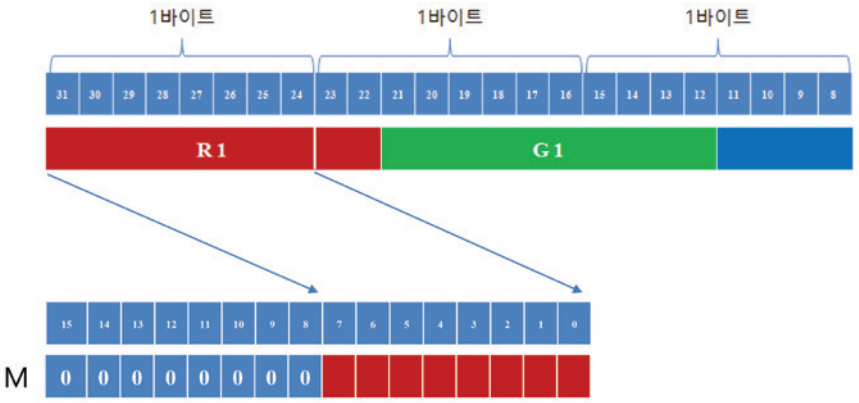


그림 6. 10비트 영상데이터의 저장방법 1

[그림 6]의 방식으로 저장된 R, G, B 영상데이터를 읽는 방법은 10비트를 저장할 수 있는 자료형 중에 가장 작은 2바이트(16비트) 자료형의 메모리를 각 화소의 R, G, B에 하나씩 할당하여 [그림 6]과 같이 빈틈없이 일렬로 이어진 영상데이터에서 순서대로 10비트씩 떼어내어 옮기는 것입니다. 하지만 앞서 말씀드렸듯이 컴퓨터 기반 기기는 바이트의 배수로 데이터를 처리하는 것이 기본이며, 이 때문에 파일을 읽을 때도 바이트의 배수 단위로 읽습니다. 그래서 R, G, B 데이터들이 바이트의 경계를 가로질러 저장되어 있으면 데이터의 앞부분과 뒷부분을 따로 떼어낸 후 하나로 합쳐 읽어내는 번거로운 과정이 필요합니다. [그림 6]의 R1의 예를 들어 10비트의 영상데이터를 2바이트(16비트) 자료형으로 읽는 방법을 자세히 설명드리면 다음과 같습니다.

1) R1값이 존재하는 2바이트 중에 첫 번째 바이트를 읽어서 R1값을 읽기 위해 할당한 2바이트 자료형 메모리 M에 저장한다.

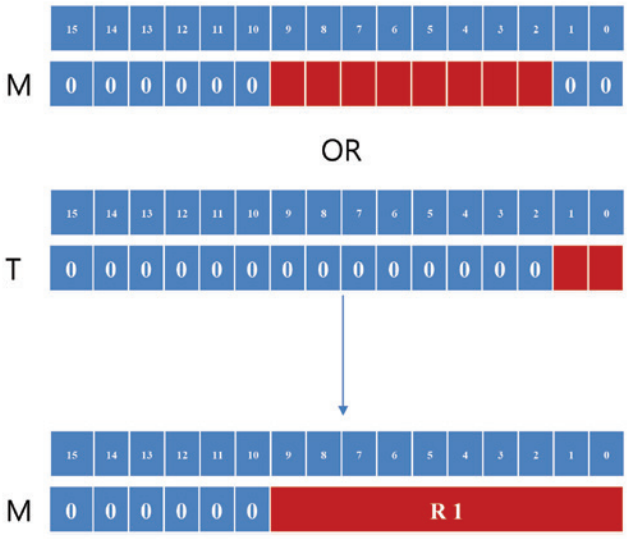


2) R1값을 읽기 위해 할당한 2바이트 자료형 메모리 M에 저장된 R1의 앞쪽 8비트를 2비트 왼쪽으로 쉬프트한다. 쉬프트 시 비워지는 오른쪽 두 비트는 0으로 채워진다.

3) R1이 존재하는 2바이트 중에 두 번째 바이트를 읽어서 임시로 할당한 별도의 2바이트 자료형 메모리 T에 저장한다.

4) 메모리 T의 비트들을 오른쪽으로 6비트 쉬프트한다. 쉬프트 시 비워지는 왼쪽 6비트는 0으로 채워진다.

5) M과 T의 각 비트에서 OR 연산(같은 위치의 두 비트값 중에 하나만 1이면 결과가 1이 되는 연산)을 통해 얻은 결과값을 M에 저장한다.



[그림 6]의 바이트를 가로질러 저장된 다른 영상데이터들도 앞서 설명한 방법과 유사한 방법(영상데이터의 위치에 따라 쉬프트 등의 비트 단위 연산에 차이가 있을 수 있음)으로 2바이트(16비트) 자료형 메모리에 옮겨서 읽을 수 읽습니다.

[그림 6] 이외에도 [그림 7]과 같은 영상데이터 저장방식도 있습니다. [그림 7]의 방식은 32비트(4바이트) 단위로 R, G, B값을 10비트씩 채우고 남은 두 비트는 0으로 채워 사실상 사용하지 않는 방식입니다. 이 방식은 0을 채우는 위치에 따라서 [그림 7]의 A, B 두 가지 방식으로 나뉩니다. 이렇게 저장된 영상데이터를 읽는 방법은 [그림 6]의 예와 같이 R, G, B 각각에 2바이트(16비트) 자료형의 메모리를 할당하여 [그림 7]의 A 또는 B의 규칙에 따라 각각의 데이터를 분리하여 메모리에 옮기는 것입니다.

그림 7. 10비트 영상데이터의 저장방법 2

12비트 영상데이터를 저장하고 읽는 방법

DPX 파일에서 12비트 영상데이터를 저장할 때도 10비트 영상데이터의 저장과 동일하게 32비트(4바이트) 단위로 [그림 8] 또는 [그림 9]의 구조를 사용합니다. [그림 8]의 경우 첫 번째 화소의 R, G값으로 12비트씩 채우고 남은 32비트(4바이트)의 8비트를 첫 번째 화소의 B값의 8비트로 채운 다음, 다음의 32비트(4바이트)는 첫 번째 화소의 B값의 나머지 4비트와 두 번째 화소의 R, G값의 각 12비트, 그리고 두 번째 화소의 B값의 4비트로 채우는 패턴을 반복하여 하나의 비트도 낭비하지 않고 모든 비트에 영상데이터를 가득 채우는 방식입니다. 이 데이터를 읽는 방법은 [그림 6]의 예와 동일하게 2바이트(16비트) 자료형 메모리를 각 화소의 R, G, B에 하나씩 할당하여 빈틈없이 일렬로 이어진 영상데이터에서 순서대로 떼어내어 옮기는 것입니다.

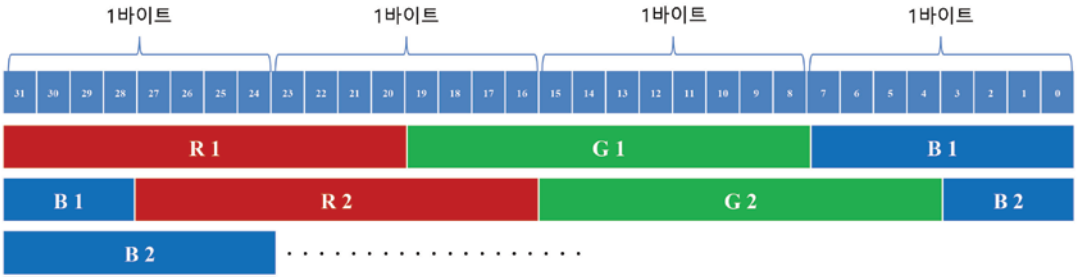


그림 8. 12비트 영상데이터의 저장방법 1

[그림 9]의 방식은 32비트(4바이트) 단위로 R, G 또는 B, R값 등을 12비트씩 채우고 남은 4비트는 0으로 채운 후 사실상 사용하지 않는 방식입니다. 이 방식은 0을 채우는 위치에 따라서 [그림 7]의 예와 유사하게 [그림 9]의 A, B 두 가지 방식으로 나뉩니다. 이렇게 저장된 영상데이터를 읽는 방법은 R, G, B 각각에 2바이트(16비트) 자료형 메모리를 할당하여 [그림 9]의 A 또는 B의 규칙에 따라 순서대로 2바이트(16비트)씩 읽어낸 후, 이진수의 아랫자리에 4비트의 0이 붙어 있는 경우 오른쪽 4비트 시프트 연산을 통해 0을 없애면 됩니다.

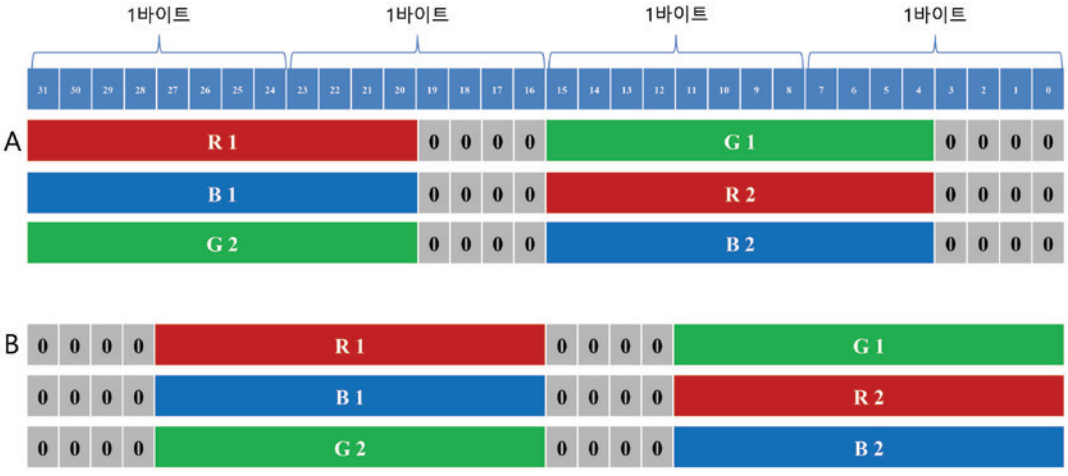


그림 9. 12비트 영상데이터의 저장방법 2

지금까지 바이트(8비트) 기반의 처리를 하는 컴퓨터 기반 기기들에서 어떻게 10, 12비트와 같이 바이트의 배수가 아닌 사이즈를 갖는 영상데이터를 처리하는지 개념적인 설명을 하였습니다. 위 설명을 통해 10비트나 12비트 영상이 왜 컴퓨터 기반 기기에서 처리할 때 까다로울 수 있는지에 관해 대략적인 감각을 갖게 되셨다면 유익할 것이라 생각합니다.

다음 연재에서는 키(Key) 신호를 이용한 영상합성이 실제로 어떤 방식으로 처리되는지에 관해 간단한 계산을 동반한 설명을 드리겠습니다.