코덱, 동영상 재생 위한 '必技'

글 | PMP인사이드 이덕환(nomark78@pmpinside.com)

코덱이란?

갑자기 자신의 컴퓨터에서 동영상이 재생되지 않을 때 컴퓨터 좀 한다는 사람에게 물어보면 십중팔구 코덱이 문제라고 한다. 흥미로운 것은 그렇게 답변해주는 사람 중 코덱을 확실히 알고 있는 사람은 그리 많지 않다는 것이다.

코덱(CODEC)이란 음성 또는 영상의 신호를 디지털 신호로 변환하는 코더(CODER)와 그 반대로 변환시켜 주는 디코더(DECODER) 기능을 함께 갖춘 기술이다.

필자는 코덱을 이렇게 정의하고 싶다. 코덱은 '약속'이다. 프로그래머와 유저 사이의 무언의 약속이다. 영상을 압축하는 기술을 '코더'라고 하고 그 압축을 푸는 기술을 '디코더'라고 할 때 어떤 방식으로 압축하고 풀 것인가를 정하는 '약속'인 것이다.

더 쉬운 예를 들어보자.

길거리에 나가서 몇 분만 걸어보면 나이 지긋하신 할아버지가 낡은 리어카에 폐재활용품을 잔뜩 싣고 다니는 것을 볼 수 있다. 가끔은 '진기명기' 프로그램에 나갈 수 있을 만큼 산더미 같은 양이 쌓여 있는 리어카도 볼 수 있다. 그렇게 높게, 많이 쌓을 수 있는 비법이 무엇일까? 바로 그 분의 수 십 년간의 노하우다. 어떻게 쌓아야 무너지지 않고 높게, 많이 쌓을 수 있는가를 그분은 나름대로 터득한 것이고 그 방법대로 폐재활용품을 쌓고 다니는 것이다.

이렇게 정해진 방법으로 짐을 쌓는 것처럼 음성이나 영상 신호를 디지털 신호로 압축, 변환하는 것이 '코더(CODER)'인 것이다. 마찬가지로 그 짐을 풀어 내릴 때도 약속대로, 순서대로 내려야 한다. 급하다고 아무거나 먼저 내리다가는 짐은 와르르 무너지고 만다. 이처럼 압축, 변환된 디지털 신호를 디코딩하는 디코더도 같은 약속을 지켜야 한다.

코덱은 또한 폐쇄적이다.

쉽게 말해서 DIVX코덱으로 압축된 파일은 DIVX코덱으로 디코딩해야 한다. 뜬금없이 급하다고 WMV코덱으로 디코딩할 수는 없다.

예를 또 들어, 박씨 할아버지가 그분만의 노하우로 쌓은 짐은 박씨 할아버지만이 풀 수 있다. 옆집 김씨 할아버지가 풀려고 한다면 짐은 무너지고 만다. 그런 노하우와 기술이 코덱의 상용화 가치를 높여 주고 있다. 김씨 할아버지의 짐 쌓기 실력이 세계 최고라면 김씨 할아버지의 기술을 배우기 위해 전국각지에서 폐재활용품 수집가들이 돈을 들고 앞다투어 모일 것이다.

그렇다면 이렇게 어렵고 귀찮은 코덱을 왜 사용할까?

많은 이유가 있겠지만 필자는 영상압축으로 인한 공간의 효율성 증대와 고화질 구현에 초점을 두고 싶다.

700메가짜리 시디 한 장에 압축하지 않은 영상을 넣는다면 수 분은 커녕 수 십초의 영상자료도 넣기 힘들다. 하지만 코덱을 이용해 영상을 압축하면 수 십분은 물론 수 시간까지도 집약이 가능하다. 이런 기술에는 어떤 비밀이 숨어 있을까?

바로 키프레임과 델타 프레임이라는 기술이 숨어 있다.

키프레임이란 쉽게 말해 동영상의 중심점이라고 할 수 있다. 키프레임이 5초 단위라고 하면, 5초 단위로 많은 정보를 가지고 있는 점을 찍으면서 코딩을 하는 것이다.

즉 처음 10초 부분에 100%의 모든 영상정보를 가지고 있고 그 다음 15초 부분에 100%의 모든 데이터를 가지고 있는 것이다. 만약 사용자가 그 중간인 12초부터 영상을 보고자해서 타임슬라이드를 12초로 이동했다면 컴퓨터나 PMP는 12초 지점에서 가장 가까운 키프레임인 10

초로 강제로 이동하고 그 부분부터 플레이가 된다. 왜냐하면 사용자가 원했던 12초 부분은 키프레임이 아닌 '델타 프레임' 이기 때문이다.

여기서 새로운 용어가 등장했다. 델타프레임이란 키프레임과 상반되는 것으로, 그 지점에서는 데이터를 100% 가지고 있는 것이 아니라 그 앞에 있는 키프레임으로부터 변화된 정보만을 담고 있다. 대략 50% 정도의 데이터만 가지고 있다고 가정하면 된다. 50% 정도의 미완의 데이터를 가지고 있기 때문에 그 부분 부터의 재생은 불가능하다. 하지만 연속적인 재생은 가능하다.

예를 들어 사용자가 키프레임인 10초 지점으로 이동했다면 다음 키프레임인 15초가 나오기 전까지는 모두 델타프레임이 있는 것이고, 컴퓨터나 PMP는 이런 5초 동안 델타프레임이 가지고 있는 50%라는 데이터와 그 앞 키프레임이 가지고 있는 100%라는 데이터를 비교 · 분석 · 계산하여 델타프레임을 재생할 수 있다.

그렇다면 이런 델타프레임은 왜 사용할까? 앞에서도 밝혔듯이 용량을 줄이고 효율을 높이기 위함이다. 만일 동영상이 모두 키프레임으로만 되어 있다면 어떻게 될까? 30FPS(초당 30프레임)라면 100%의 데이터를 가지고 있는 키프레임이 초당 30개 있다는 것이고 이는 한 시간짜리 동영상일 경우 엄청난 용량의 증가를 가져 온다.

아무래도 키프레임은 델타프레임보다 훨씬 많은 용량을 가지고 있기 때문이다. 쉽게 말해서 완전한 사진이미지(jpg)로만 영상이 구성돼 있다면 용량이 필요이상으로 증가하고 그 효율도 많이 떨어질 수밖에 없다.

따라서 반쪽짜리 사진을 교묘히 이용해 편법을 사용하는 것과 마찬가지로 키프레임과 델타프레임을 사용하는 것이다.

요약하자면, 일정 시간단위로 100%의 데이터를 가지고 있는 완전한 이미지를 만들고 그 사이에는 델타프레임으로 그 빈 자리를 완성하는 것이다.

이런 키프레임과 델타프레임은 각각 I프레임과 B프레임, P프레임에 상응된다. I프레임은 키프레임이라고 할 수 있으며, B와 P프레임은 델타프레임이라고 할 수 있다. 그렇다면 B프레임과 P프레임은 무엇이 다를까?

DIVX계열과 XVID계열의 코덱을 예로 들어 설명하겠다. P프레임은 이전 프레임과 현재 프레임(P프레임)과의 변화된 값들만으로 디코딩이 가능한 델타프레임이다.

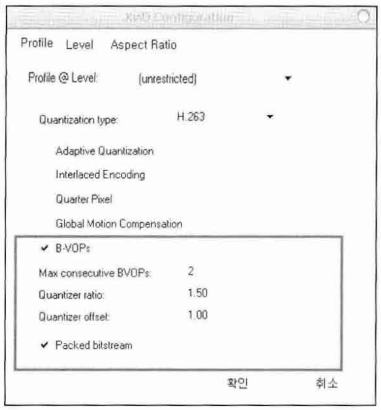
다음 그림을 보자

모든 부분은 이전 프레임과 동일하지만 주인공이 말하는 입주변은 변화하고 있다. 이럴 때 뒷프레임은 어떤 식으로 정보를 담고 있으면 가장 효율이 높을까? 바로 앞선 프레임에서 변화하지 않는 부분은 그대로 두고 변화된 부분만 코드로서 가지고 있는 방법이다.

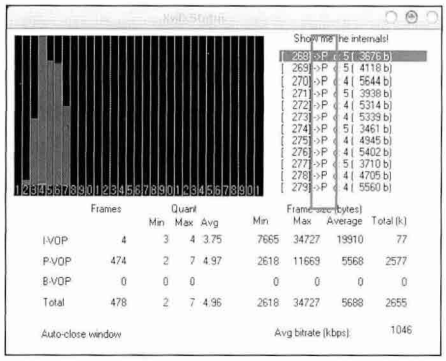
즉 다른 부분은 전혀 변화가 없으니 앞선 프레임의 값을 메모리에서 그대로 가져와서 사용하고 변화된 입주변 값만 완성하면 된다. 현재 프레임에서 나타나는 화면 중 대부분을 계산하지 않고 앞선 프레임 정보만 가져오니 얼마나 높은 효율성인가.

이렇게 앞선 프레임에서 변화된 값만 가지고 있는 뒷프레임을 P프레임이라고 한다.

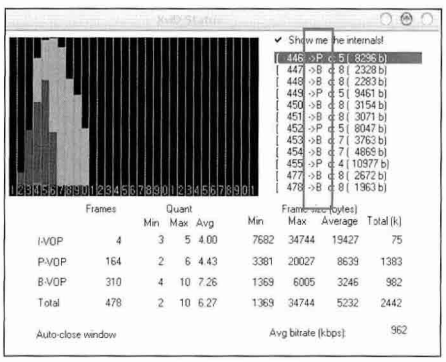
그렇다면 B프레임은 무엇인가? P프레임에서 한 발자국 더 발전한 델타 프레임으로, 이는 앞선 프레임과 뒷 프레임을 비교, 계산해서 그 중간에 위치하는 프레임을 말한다.



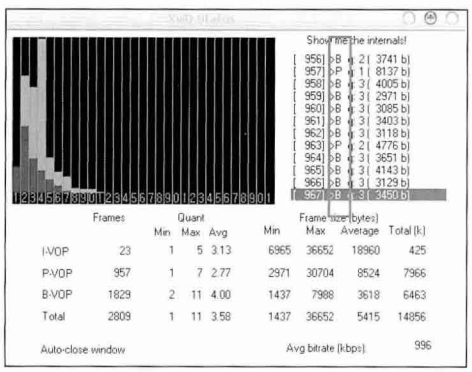
▶XVID코덱으로 인코딩할 경우 B프레임의 채택 유무를 정할 수 있다.



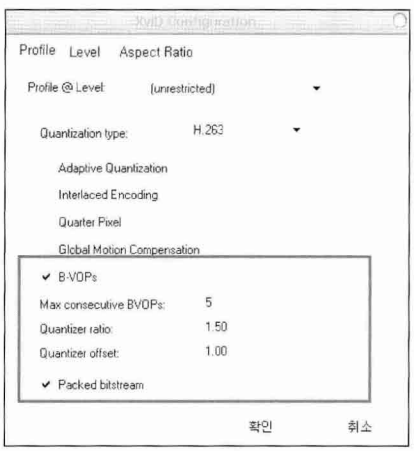
▶B프레임 옵션을 선택하지 않으면 영상 인코딩 때 B프레임은 사용되지 않고 P프레임만 사용된다. 인코딩 정보창의 '우(右)' 상단에서 P프레임만이 사용되는 것을 확인할 수 있다.



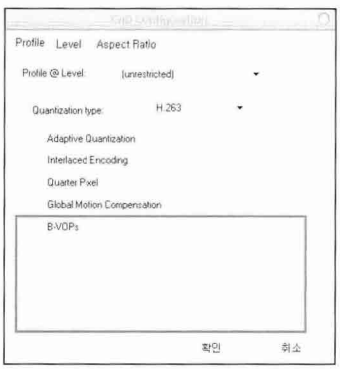
▶인코딩 정보창에서 두 개의 B프레임 사용을 확인할 수 있다.



▶인코딩 정보창에서 5개의 연속된 B프레임을 확인할 수 있다.



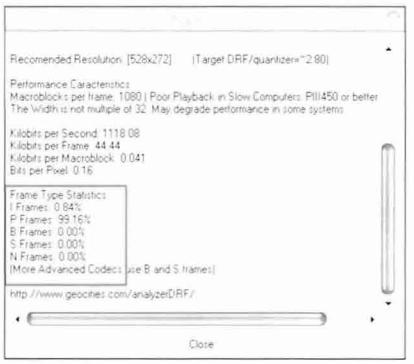
▶B프레임 옵션을 5개로 잡을 경우 영상소스의 상황에 따라 최대 5개까지의 다중 B프레임을 사용한다. 이럴 경우 용량대비 화질은 증가하지만 인코딩과 디코딩 때 많은 부하가 발생된다.



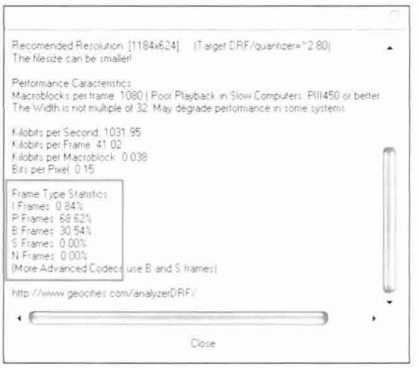
▶XVID코덱의 경우 디폴트값은 두 개의 B프레임이다.

P프레임은 비교할 대상이 하나 밖에 없기 때문에 다소 많은 정보를 필요로 하지만 B프레임은 비교할 대상이 두 개이기 때문에 P프레임보다 더 용량을 줄일 수 있다.

이런 I · B · P프레임들로 영상이 압축돼 파일에 담겨지는 것이다.



▶XVID코덱으로 인코딩된 영상 중 B프레임 없이 P프레임만으로 인코딩 된 영상파일을 DRF Analyzer로 분석해 보았다. 전체 프레임 중 I프레임은 0.84%, P프레임은 99.16%가 사용되었다.



▶두 개의 B프레임을 사용해서 인코딩한 영상일 경우 전체 프레임 중 I프레임은 0.84%, P프레임은 68.62%, B프레임은 30.54%가 사용되어서 효율성이 증가되었다.

DIVX계열의 코덱은 'IBPBPBP~' 의 형태를 취하면서 인코딩을 한다. 이는 전형적인 방법으로 평가되며, 인코딩 혹은 디코딩 때 많은 부하를 유발하지 않는다.

반면 XVID는 IBBPBBPBBP~식의 구조를 가지고 있다. B프레임이 연속으로 들어 있는 구조이다. 게다가 B프레임이 연속으로 배열되는 수는 인코딩 때 자유롭게 조절이 가능하다. DIVX보다 더 진일보 된 알고리즘으로 같은 용량일 때 더 많은 정보를 담을 수 있다. 높은 화질을 구현할 수 있는 건 당연지사.

하지만 양지가 있으면 음지가 있는 법. 바로 이런 B프레임은 인코딩은 물론 디코딩 때 DSP칩이나 GPU · CPU에 많은 부하가 걸리게 한다. 그도 그럴 것이 앞선 프레임뿐 아니라 뒷 프레임과의 상관관계를 계산해야 하는 B프레임이 하나도 아닌 하나 이상이 들어 있으니 어쩔 수 없는 노릇이다.

이제 PMP를 예로 들어 보자. 1초에 24프레임으로 구성되어 있는 XVID코덱 동영상이 있다고 하면 이를 DSP칩이 디코딩할 때 파일의 헤더 부분 인덱스를 검사하여 동영상과 음성을 일치시키며 디코딩한다. 이 과정에서 오류가 있다면 음성과 영상이 일치하지 않는 이른바 '싱크 어긋남 현상' 이 일어난다.

예를 들어 IBBPBBP~ 구조의 XVID의 경우 1초에 24프레임을 정확히 디코딩해야 하는데 DSP칩에서 다중으로 되어 있는 B프레임을 실시간으로 디코딩하지 못해 24프레임을 디코딩하는데 1.1초가 걸렸다고 하자. 0.1초가 별것 아닌 것처럼 보일 수 있지만 대략 10초 후에는 영상이 음성보다 1초나 늦게 나오게 된다. 이런 현상은 계속 누적이 되고 몇 분 후에는 도저히 영상 감상이 불가능할 정도가 될지도 모른다.

이런 싱크의 어긋남을 막기 위해서 많은 경우 DSP과정에서 프레임을 드롭시켜 버린다. 즉 1초에 24프레임을 완전하게 디코딩할 수 없다면 어쩔 수 없이 몇몇 프레임은 디코딩하지 않고 버린다. 화면의 변화가 심한 영상, 즉 액션장면이나 카메라의 패닝장면 등에서 쉽게 이 경우를 목격할 수 있다. 인간의 눈은 잔상효과에 약하기 때문에 약한 수준의 프레임 드롭은 문제시 되지 않는다.

하지만 이런 프레임 드롭도 심할 경우 영상 감상에 많은 불편함을 준다. 특히 영상을 디코딩하는 비트수가 18비트(26만 칼라)이상, 혹은 24비트(1600만 칼라)일 경우, 많은 색의 데이터를 실시간으로 디코딩해야 하고 그와 동시에 다중의 B프레임을 디코딩해야 하기 때문에 레임 드롭은 눈에 띄게 증가된다.

이런 실시간 디코딩 성능은 DSP칩의 성능과 이를 컨트롤하는 코덱 솔루션에 달려 있다.K