스크린 이미지의 스트리밍을 위한 압축 및 저장 방법

○
이재문, 황기태
한성대학교 멀티미디어 정보처리 전공
한성대학교 컴퓨터 공학 전공,

A Compressing and Storing Method for Stream of Screen Images

○
Jaemoon Lee, Kitae Hwang
Dept. of Multimedia Information Processing, Hansung University
Dept. of Computer Engineering, Hansung University
e-mail : jmlee@hansung.ac.kr ,calafk@hansung.ac.kr

요약

본 논문은 시간에 따라 변하는 스크린 이미지의 스트리밍에 적합한 이미지 압축 및 저장 방법을 제안하는 것이다. 이를 위하여 시간에 따라 변화하는 스크린 이미지의 특성을 일반 동영상과 비교하여 분석 하였으며, 분석 결과에 의거하여 스크린 이미지의 스트리밍에 가장 적합한 압축 및 저장 방법을 제안하였다. 제안된 방법들은 스트리밍 데이터를 효과적으로 지원하는 DirectShow 의 구조에 따라 구현되었으며, 구현된 내용을 보인다. 제안된 방법의 유용성을 보이기 위하여, 구현된 시스템 상에서 압축 성능이 측정되었으며 그 결과를 설명한다.

1. 서론

스크린 이미지의 스트리밍이란 특정 시구간 사이에서 스크린 상에 보여진 모든 이미지를 말한다. 최근 이러한 스크린의 이미지를 효과적으로 저장하는 연구가 활발히 진행되고 있다[1, 2, 8]. 이러한 스크린 이미지의 스트리밍은 특정 프로그램의 사용 설명, 또는 프로그램 실행 결과의 저장 등에 활용된다. 즉, 소프트웨어 패키지를 개발하여 배포할 때, 사용 방법 등 많은 부수적인 문서를 포함하여 배포한다. 그러나 이러한 문서를 보고 학습하기에는 너무나 많은 시간과 노력이 필요하다. 이러한 문서에 대한 대용으로 소프트웨어 패키지의 동작 상태를 전부 동영상으로 저장하여 배포하는 방법이 최근 사용되고 있다[1, 2]. 이 방법은 소프트웨어 패키지가 운영되면서 보여지는 모든 스크린 이미지를 시간 순서에 따라 저장한다. 물론 데이터의 최소화를 위하여 압축 방법이 적용된다. 본 논문은 이러한 스크린 이미지를 압축하는 경우에 적합한 압축 방법 및 이들이 저장에 관한 모델을 제시하는 것이다.

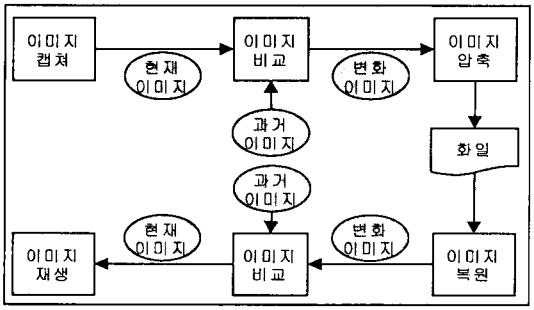
스크린 이미지의 특성은 크게 세 가지로 요약할 수 있다. 첫 번째는 단위 화면의 크기가 매우 크다는 것이다. 최근 모니터 기술의 향상과 가격의 저렴화로 많은 소프트웨어가 1024*768 해상도 이상을 대상으로 제작되고 있다. 또한 대부분의 사용자도 고해상도의 비디오 카드를 채용하여 다양한 컬러 색상을 사용한다. 따라서 하나의 단위 화면 크기는 1024*768 크기에 24bit 컬러를 선택하는 경우 2.25메가 바이트 (2,359,296 바이트)가 된다. MPEG1[3]의 화면 최대 크기가 360*240인 것을 감안하면 약 9배에 해당한다. 두 번째 특징은 스크린 이미지의 변화가 매우 순간적이라는 것이다. 영화 등 일반적인 동영상에서 화면의 변화는 사물의 움직임이 급변하지 못하기 때문에 일반적으로 서서히 변화한다. 반면, 스크린 이미지는 컴퓨터에 의해서 일어나는 현상이므로 비록 부분적이기는 해도 대부분 순간적으로 그 내용이 변화하는 특징을 가지고 있다. 마지막 특징은 화면의 변화가 부분적이라는 것이다. 예를 들어 워드 프로세서와 같은 프로그램을 사용하고 있다고 가정하여 보자. 파일 열기와 같은 경우에는 화면 전체가 변하겠지만 대부분의 경우에 커서가 있는 위치에서 국소적으로 화면이 변한다는 것이다.

상기와 같이 세 가지 특성을 갖는 스크린 이미지의 압축은 기존에 널리 알려진 동영상 압축 방법인 MPEG[3, 4]은 적합하지 않다. 그 이유는 이 방법이

비교적 작은 화면에 적합할 뿐만 아니라 순간적으로 변화하는 이미지에 대해서는 압축률이 그다지 높지 않기 때문이다. 본 논문에서는 시간적으로 변하는 스크린 이미지를 압축/복원하는 스크린 이미지 관리 엔진의 구성을 제안하며, 스크린 이미지의 특성을 잘 반영하는 스크린 이미지 압축 방법을 제안한다. 또한 제안하는 엔진을 마이크로소프트사의 DirectShow[5]에 적용하여 실제 시스템으로 개발한 결과를 이용하여 성능 평가의 결과도 보인다.

2. 스크린 이미지 스트리밍 관리 엔진

스크린 이미지의 스트리밍 관리 엔진은 주기적으로 주어진 스크린의 영역에 대한 이미지를 캡쳐하여 이를 압축하는 기능을 가져야 하며, 반대로 주기적으로 압축 이미지를 읽어서 이를 복원한 후 화면에 그려야 하는 기능을 가져야 한다. 이러한 기능의 스크린 이미지 관리 엔진은 대상 스크린의 이미지를 효과적으로 압축하는 것이 가장 큰 문제이다. 즉, 압축률을 높여 데이터의 량을 최소화하여야 한다. 압축률을 높이는 방법은 데이터의 캡쳐 주기를 크게 하는 방법이 있으나 이는 전체적으로 화면의 변화를 실시간에 추적할 수 없을 수도 있기 때문에 바람직한 방법이 아니다. 또 다른 하나의 방법은 일반적으로 잘 알려진 전 이미지와 현 이미지에서 변화된 부분만 추출하여 이를 압축하는 방법[4]이다. 이 방법은 이미지의 변화가 부분적인 경우에 매우 높은 압축률을 보장한다. 스크린 이미지의 경우 앞에서 언급하였듯이 화면의 변화가 부분적이라는 특성을 갖고 있다. 이 특성은 전 화면에 비하여 현 화면의 변화가 매우 작은 영역이라는 것을 의미하므로 압출률을 크게 높인다. 스크린 이미지 관린 엔진은 그 목적과 특성에 따라 다양한 구성을 할 수 있으나 본 논문에서는 상기와 같이 스크린 이미지의 압축률을 최대화하는 관점에서 <그림 1>과 스크린 이미지 관리 엔진을 제안한다.



<그림 1> 스크린 이미지 관리 엔진의 구성도

<그림 1>에서 상단은 인코딩 시스템이고 하단은 디코딩 시스템이다. 일반적으로 변화 이미지는 현재 이미지보다 훨씬 작으며, 변화 이미지에는 과거 이미지에 대한 상대적인 위치 정보와 재생 시간정보를 포함하고 있다. 이미지 캡쳐의 출력은 일반적으로 잘 알려진 비트맵(bitmap) 구조를 갖는다. 이미지 비교에서는 두개의 비트맵을 비교하여 변경된 영역을 찾고 이러한 변경된 영역을 포함하는 최소한의 사각 영역을 찾아서 이를 표현하는 작은 비트맵을 생성하여 이를 이미지 압축 모듈에 넘기게 한다. 이미지 압축 모듈에서는 이 이미지를 JPEG[3, 4, 6]으로 압축한다. 각 모듈별 주요 기능은 다음과 같다.

2.1 이미지 캡쳐 모듈

이미지 캡쳐 모듈은 스크린의 특정 영역에 대한 이미지를 주기적으로 캡쳐하는 것이 주된 기능이다. 본 논문에서는 상기에서 제안된 이미지 캡쳐 모듈은 특정 응용 프로그램에 대한 윈도우의 캡쳐, 스크린에서 특정 영역을 캡쳐, 스크린의 전 영역을 캡쳐하는 기능을 갖도록 설계하였다. 이미지 데이터에 대한 캡쳐 방법으로는 윈도우 시스템에서 제공하는 DDB(Device Dependent Bitmap)[4, 5, 6]과 DIB(Device Independent Bitmap)[4, 5, 6]을 사용한다. DDB는 윈도우 시스템에서 이미지 데이터를 표현하는 방법이다. 따라서 시스템 독립성을 높게 하기 위하여 캡쳐된 모든 이미지는 DIB로 변환한다.

2.2 이미지 비교 모듈

이미지 비교 모듈의 기능은 전 이미지를 보관하여 현재 캡쳐되어 보내진 이미지와 비교하여 변경된 부분에 대하여 새로운 이미지를 생성한다. 이 모듈은 변경된 이미지를 효과적으로 찾아야 할뿐만 아니라 변경된 이미지에 변경되지 않은 이미지의 포함을 최소화하여야 한다. 전 이미지에서 변경된 부분만 추출하기 위하여 항상 전 이미지를 보관한다. 이 이미지는 메모리 상에서 저장되며, 데이터 포맷은 DIB 형태로 저장/관리한다. 현 이미지와 전 이미지의 비교는 두 이미지의 모든 픽셀을 비교하여 각 픽셀에 대하여 다른 값들을 찾아낸다. 변경된 영역을 표시하기 위하여 변경된 영역을 포함하는 최소의 사각 영역을 찾는 기능이다. 이것은 실제 구현할 때 현 이미지와 전 이미지의 비교 기능에 포함되어 구현되었다. 임의의 사각 영역을 새로운 DIB로 추출한다. 이것은 독립된 헤더 정보를 구축하고 이에 따른 데이터 정보를 사각 영역으로부터 추출해 내는 기능이다.

2.3 이미지 압축/복원 모듈

이미지 압축/복원 모듈은 크기가 다른 다수의 DIB를 순차적으로 각각 JPEG으로 압축/복원하는 것이 주요 기능이다. 순차적으로 입력되는 데이터간에는 전혀 연관성이 없으며, 독립된 데이터로 고려하여

압축/복원을 한다.

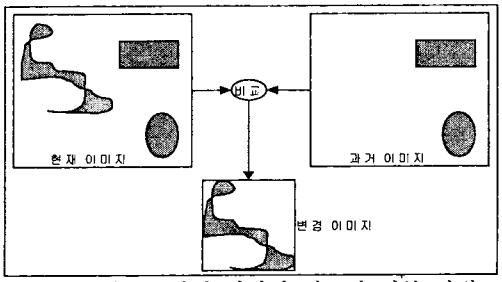
2.4 이미지 재생 모듈

이미지 재생 모듈은 지속적으로 주어지는 DIB 데이터를 특정 윈도우에 지정된 시간에 재생하는 기능이다. 이미지 복원 모듈에 의하여 복원된 DIB 데이터이다. 이 모듈에서는 이를 윈도우 시스템에 적합하도록 DDB 형태로 변환하며, 과거의 스크린 이미지에 대하여 현재 스크린 이미지의 상대적인 위치를 찾아서 재생하게 된다.

3. 효과적인 이미지 압축 방법

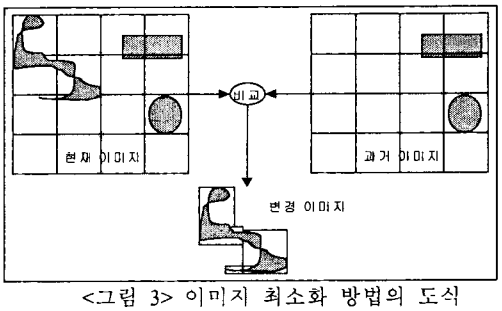
3.1 이미지 압축 방법

스크린에 나타나는 이미지의 변화를 추적하기 위해서는 일정 주기 또는 스크린의 이미지 변화가 있을 때마다 그 이미지를 캡쳐하여 저장하여야 한다. 스크린 이미지를 저장하는 가장 단순한 방법은 두 이미지를 그대로 저장하는 것이다. 그러나 이 경우는 너무 많은 데이터를 저장해야 한다. 이 방법을 다소 보완한 방법이 과거 이미지와 현재 이미지를 그대로 저장하는 것이 아니라 과거 이미지는 그대로 저장하되 현재 이미지는 과거 이미지에 대하여 달라진 영역만 저장하는 것이다. 따라서 저장되어야 하는 데이터는 과거 이미지와 변경 이미지만 저장하면 된다. 이 경우 변경 이미지는 현재 이미지에 비하여 일반적으로 상당히 작게 된다. 이러한 방법은 과거 동영상 저장 방법에 많이 적용된 방법이다. 본 논문에서 특히 이 방법을 채택한 이유는 스크린 이미지의 경우 대부분의 사용에서 화면의 변화가 전체적으로 일어나기보다는 전체 이미지에 비하여 변경되는 부분은 지극히 작기 때문에 변경 이미지의 크기가 현재 이미지의 크기에 비하여 매우 작게 되기 때문이다. <그림 2>는 이에 대한 도식적인 설명이다.



<그림 2> 변경 이미지 최소화 기본 방법

앞에서 설명한 기본 방법에서 변경 이미지는 항상 사각 영역으로 표시된다. 이것은 현재 이미지를 다루는 데이터 포맷이 모두 사각 영역으로 표현되기 때문이다. 이러한 한계 때문에 경우에 따라서는 변경 이미지에서 변경된 영역보다 변경되지 않은 영역이 더 많을 수도 있다. 변경 이미지에 포함된 변경되지 않은 이미지의 량을 최소화함으로써 전체적으로 변경 이미지의 량을 최소화한다는 것이다.



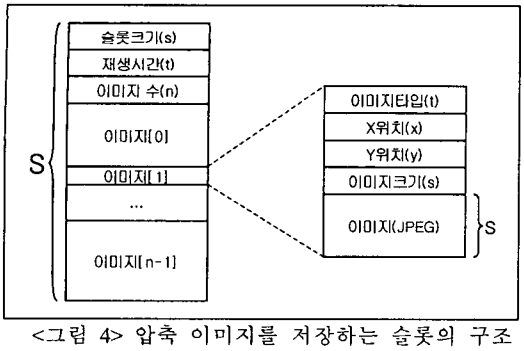
<그림 3> 이미지 최소화 방법의 도식

과거 이미지와 현재 이미지를 비교할 때 전체 이미지를 비교하는 것이 아니라 <그림 3>과 같이 임의의 조각으로 두 이미지를 나눈 후 각각 대응하는 조각 이미지끼리 비교하여 변경된 부분을 찾고 변경된 부분을 포함하는 최소의 사각 영역을 찾아 저장하는 것이다. <그림 3>에서는 과거, 현재 이미지를 각각 16 등분하여 비교하였으며, 5개의 크기가 다른 변경된 이미지를 찾게 되었다. 이렇게 확장된 방법의 결과는 <그림 2>에서 하나의 변경 이미지를 저장하는 대신 <그림 3>에서 크기가 다른 5개의 변경 이미지를 저장하게 된다. <그림 2>의 변경 이미지는 <그림 3>의 5개의 변경 이미지를 포함하고 있기 때문에 <그림 3>의 방법이 저장하는 데이터의 량은 적어진다. 특히 원래의 이미지를 세분화하면 할수록 변경 데이터의 량이 실제로 변경된 데이터와 비슷하게 될 것이다. 하지만 이렇게 확장된 방법은 크기가 서로 다르며 표현되어야 할 위치가 다른 여러 개의 이미지를 관리해야 하는 것과 세분화된 이미지 때문에 변경된 영역을 찾는 비용을 감수하여야 한다.

3.2 이미지 저장 방법

비디오 데이터는 시간에 따라 변경되어지는 스크린 이미지를 표현하는 데이터이다. 이러한 데이터의 가장 간단한 관리 방법은 주기적으로 이미지를 캡쳐하여 저장하면 된다. 그러나 이 경우 앞에서도 언급하였지만 너무 많은 데이터를 저장해야 하는 단점이 있다. 이러한 단점을 보완하기 위하여 본 논문에서는 변경된 이미지만 저장/관리하며 변경된 이미지를 위한 데이터의 저장을 최소화하기 위하여 이미지를 분할하여 관리한다. 이 결과 임의의 순간에 표현되어야 하는 이미지 데이터가 위치와 크기가 서로 다르면서 다수개가 발생하게 되었다. 이렇게 발생되는 다수의 데이터는 특정 시간에 동시에 그려져야 하므로 이들 데이터는 항상 동시에 관리되어야 한다. 이렇게 동시에 관리 되어야 하는 데이터의 집합을 슬롯이라 명명한다. 슬롯의 구조는 <그림 4>와 같은

구조를 갖는다.



<그림 4> 압축 이미지를 저장하는 슬롯의 구조

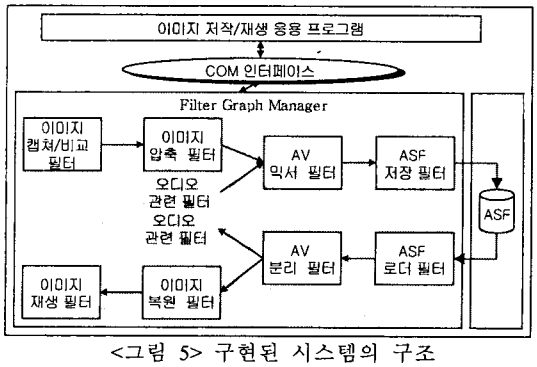
하나의 슬롯 데이터는 재생 시간 t에서 재생되어야 하는 모든 이미지 데이터를 포함하여야 한다. <그림 4>에서 슬롯크기(s)는 하나의 슬롯의 크기를 바이트 단위로 표시한다. 크기를 표시하는 데이터가 필요한 이유는 슬롯 내에 포함된 이미지의 수가 다를 뿐만 아니라 각각의 이미지의 크기도 가변이기 때문이다. 재생시간(t)은 이 슬롯이 재생되어야 하는 시간을 *msec*단위로 저장하고 있다. 이미지 수(n)는 이 재생시간에 과거 이미지와 현재 이미지의 비교 결과에 의하여 생성되는 변경 이미지의 수이다. 슬롯 내에 포함된 이미지는 기본적으로 JPEG로 압축된 데이터로서 <그림 4>의 우측과 같이 구성한다. 이미지 타입(t)은 이 이미지가 스크린 이미지인지 아니면 다른 이미지(마우스 이미지 등)인지를 표시한다. X위치, Y위치는 이 이미지가 재생되어야 하는 위치 정보를 포함하고 있으며, 이미지 크기(s)는 이후 포함된 이미지(JPEG)의 바이트 단위의 크기를 저장한다.

압축 데이터 저장 모듈에서는 슬롯 데이터를 필요에 따라 적절히 포장을 한 후 저장하게 되고 필요시 이러한 포장을 제거한 후 슬롯 데이터를 이미지 복원 모듈로 전송하게 된다. 이미지 복원 모듈은 슬롯 데이터의 정보를 이용하여 n개의 이미지를 복원하게 되며 복원된 이미지를 재생 위치 정보와 함께 이미지 재생 모듈로 전송하게 되고 이미지 재생 모듈은 위치 정보를 이용하여 이미지를 적절히 재생하게 한다.

4. 이미지 관리 엔진 구현

4.1 이미지 관리 엔진 구현

마이크로소프트사에서 제공하는 DirectShow[5]는 스트림 데이터 처리를 위한 백본을 제공한다. DirectShow는 크게 응용프로그램, 필터 그래프 매니저 및 필터들로 구성된다. 응용 프로그램은 필터 그래프를 통해서 필터들을 제어함으로써 스트림 데이터의 흐름을 적절히 제어할수 있다. DirectShow에서 제공하는 필터 그래프 매니저는 특정 데이터 포맷에 따른 필터를 구성하고, 응용프로그램의 제어에 따라 구성된 필터에 제어를 가한다. 하나의 필터 그래프 매니저에는 다수의 필터가 구성될 수 있는데, 이러한 필터들은 소스 필터, 트랜스폼 필터, 랜더러 필터로 기능상 나누어진다. 본 논문에서는 앞에서 제안한 스크린 이미지 압축 엔진의 유용성을 시험하기 위하여 DiectShow에 이를 적용하여 구현하였다. 엔진 설계에서 보인 모듈들은 DirectShow에서 하나의 필터들로서 구현되었고, 압축 데이터 저장 모듈에서는 슬롯 데이터의 효율적인 관리를 위하여 마이크로소프트사에서 제공하는 ASF(Advanced Streamming Format)[7] 구조로 변형되어 저장/관리 하였다.



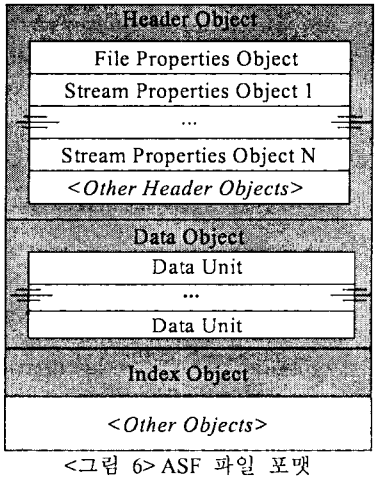
<그림 5> 구현된 시스템의 구조

<그림 5>는 앞에서 설명한 엔진을 DirectShow 구조에 따라 구현한 시스템의 개략도이다. 시스템의 구현은 윈도우 98환경에서 비주얼 C++를 사용하여 구현하였으며, 마이크로소프트사에서 제공하는 DirectShow SDK와 ASF SDK를 최대한 활용하여 구현하였다.

4.2 슬롯 데이터의 저장: ASF 구현

멀티미디어 시스템을 구성하는데 매우 중요한 요소가 멀티미디어 파일 포맷이다. 스트림 데이터들이 서로 동기화가 용이하도록 배치되어야 하며, 순차적인 재생과 임의적인 위치에서의 재생이 용이하도록 설계되어야 한다. ASF[7] 파일 포맷은 널리 사용되는 멀티미디어 파일 포맷으로 많은 멀티미디어 파일들이 ASF 파일 포맷으로 구현되었다. 마이크로소프트사는 최근에 ASF 를 발전시킨 윈도우 미디어 파일 포맷[7]을 개발하여 사용하고 있다. 윈도우 미디어 파일 포맷은 ASF 파일 포맷과 거의 동일하나, 미디어 데이터가 반드시 MPEG4로 인코딩 되어야 한다는 특징을 가진다. 서론에서도 밝혔듯이 시간적으로 변하는 스크린 이미지의 특성상 MPEG 코딩이 적합하지 않기 때문에 새로운 압축 방식으로 멀티미디어 파일을 생성하기 위해서는 윈도우 미디어 파일 포맷 대신 ASF 파일 포맷을 사용한다. ASF 파일 포맷의 입출력을 위해서는 마이크로소프트사의 ASF PDK 라이브

러리를 사용하였다.



<그림 6> ASF 파일 포맷

ASF 파일 포맷은 <그림 6>과 같은 구조를 가진다. 헤더 부분은 스트림 개수, 총 재생 시간 등 파일 전체에 대한 속성과, 각 스트림에 대해 데이터 율, 스트림 종류 등 스트림 속성, 그리고 특정 위치에서 재생이 가능하도록 마킹한 정보 등을 가진다. 헤더 밑에는 여러 스트림의 데이터들이 데이터 유닛이라는 단위로 저장되며, 파일에 읽고 쓰는 단위는 항상 데이터 유닛 단위이다. 파일의 끝에는 임의의 위치에서 재생이 가능하도록 인덱스 정보를 둘 수 있으며 인덱스는 데이터 유닛에 대한 위치 정보를 가진다.

본 논문에서 구현한 시스템은 ASF 파일에 스크린 이미지와 오디오 데이터를 데이터 유닛에 적절히 저장하게 되는데, 다음과 같은 규칙에 의하여 저장된다.

(1) 시간적으로 변하는 스크린 스트림과 오디오 스트림의 두 스트림만을 다룬다.
(2) 하나의 데이터 유닛은 오직 한 종류의 스트림만을 저장한다.
(3) 각 데이터 유닛은 동기화를 위해 재생 시작 시간에 대한 상대적 시간을 가진다.
(4) 스트림에 관계없이 모든 데이터 유닛은 시간 순서로 정렬되어 저장된다.
(5) 하나의 오디오 데이터 유닛은 여러 번에 걸쳐 캡쳐한 오디오 정보를 합하여 저장한다.

상기에서 (4)의 원칙이 위배되면 재생 시 두 스트림 사이에 동기화가 이루어지지 않는 문제가 발생할 수 있다. 오디오와 스크린이 각각 다른 스레드에 의해 처리된다고 가정하면, 오디오와 스크린이 동시에 캡쳐되었지만 오디오 스트림은 압축 시간이 스크린에 비해 매우 짧기 때문에 시간적으로 후에 캡쳐된 오디오 데이터가 먼저 캡쳐된 스크린 데이터보다 ASF 파일의 앞 부분에 기록되게 되는 현상이 발생한다. 이런식으로 ASF 파일 내에서 모든 데이터 유닛이 시간순으로 정렬되지 않은 현상이 벌어지게 되면, 재생 시 나중에 재생되어도 되는 오디어 데이터 유닛를 먼저 처리하느라고 현재 재생되어야 하는 스크린 데이터 유닛을 늦게 처리하는 현상이 일어나게 되고 스크린의 재생이 부자연스럽고 불연속적으로 될 가능성이 있다. (5)의 원칙은 오디오 샘플의 크기를 사실상 매우 작기 때문에 캡쳐시마다 하나의 데이터 유닛을 두게 되면 빈번한 파일 입출력으로 성능을 나쁘게 만드는 요인이 되기 때문이다.

4.3 성능 평가

구현된 시스템의 성능 평가의 기준은 실 시간적인 요소와 데이터 압축 요소 두 가지가 있다. 전자는 하드웨어의 성능에 크게 좌우된다. 구현된 시스템을 사용하여 실험한 결과 대부분 펜티움 II 300MHZ 이상의 시스템이면 실 시간적으로 스크린 이미지를 압축할 뿐만 아니라, 실 시간적으로 압축된 스크린 이미지를 복원/재생하는 것을 관찰할 수 있었다. 따라서 현재 대부분의 컴퓨터가 펜티움 III, IV로 교체되어 가는 시점이므로 전자의 요소는 성능 평가의 중요한 요소라 할 수 없다. 후자의 요소는 매우 중요하다. 압축된 스크린 데이터는 대부분 교육용으로 활용될 가능성이 많기 때문에 한번 제작된 스크린 데이터는 다수로 복제되어 배포될 것이다. 이 경우 배포 방법은 물리적으로 CD-ROM을 전송하는 방법도 가능하나 네트워크를 통한 온라인 배포도 고려하여야 한다. 이 경우 데이터의 크기는 매우 중요한 요소이다. 따라서 본 논문에서도 스크린 이미지의 압축에 관한 성능 평가만 하기로 한다.

성능 평가를 위해 1024*768 해상도에 16 비트 칼라로 고정하였으며 1초에 2번 캡처하여 100초 동안 실험하였다. 압축하지 않고 비트맵을 그대로 저장하였을 때 314.5MB 정도 되었다. 압축률은 스크린에 출력되는 내용에 따라 달라지게 되므로 3 개의 서로 다른 응용 프로그램이 실행되는 상황에서 압축률을 측정하여 평가하였다. 스크린의 변화가 매우 빈번히 일어나는 상황을 평가하기 위해 AVI 동영상이 출력되는 응용을, 화면이 자주 변하지는 않지만 변하는 영역이 큰 상황을 평가하기 위해 FLASH 응용을, 그리고 화면의 변화가 상대적으로 작고 국소적인 경우를 평가하기 위해 한글 워드를 그 대상 응용으로 설정하였다. 그리고 스크린을 5x5 로 분할하였다.

압축률은 압축없이 저장한 ASF 파일의 경우에 대해 본 논문의 알고리즘을 이용하였을 경우의 ASF 파일의 크기로 계산하였다.

$$\text{압축률} = \frac{SI - SI_{Compressed}}{SI} \times 100\ (\%)$$

여기서 SI 는 압축 없는 스크린 이미지를 말하며, $SI_{Compressed}$ 는 압축된 스크린 이미지를 말한다.

JPEG 압축질	10%	30%	75%
AVI 응용	99.37	99.31	99.17
FLASH 응용	99.62	99.59	99.52
한글워드	99.70	99.69	99.52

<표 1> 스크린 이미지의 상대적 압축률

<표 1>은 세 응용프로그램이 실행되는 동안 본 논문에서 제안하는 압축 알고리즘의 성능인 압축률의 결과를 보이고 있다. 본 실험에서 JPEG 압축질을 변화 시키면서 본 논문의 압축 알고리즘의 성능을 측정하였다. JPEG 는 손실 압축(loss compression) 알고리즘으로서 압축 시 원 이미지의 정보를 손실하면서 압축하기 때문에 압축 후 복원하면 본래의 이미지로 완벽하게 복원할 수 없는 알고리즘이다. 이때 JPEG 압축질이란 JPEG 알고리즘의 실행 시 압축 정도를 말하는 것으로 압축질이 작으면 작을수록 압축 시 원 이미지에 대한 손실(loss) 정도가 커진다. 즉 압축한 이미지를 화면에 출력하면 원 이미지와의 차이가 크게 나타난다. JPEG의 압축질을 낮추면 당연한 결과로서 본 논문의 압축률이 향상된다. 그러나 압축된 스크린을 복원하여 재생하면 화질이 떨어지게 된다. 본 논문에서는 JPEG 압축질을 75%로 설정하는 것이 화질 및 압축율이 대체로 좋은 것으로 평가되었다.

<표 1>에서 볼 수 있듯이 본 논문의 알고리즘은 대단히 우수한 압축률을 보이고 있다. 실험에서 AVI는 계속적으로 화면이 변하며, 또한 셋 중 가장 빈번히 화면이 변하므로 압축률이 셋 중에서 가장 낮다. 한글 워드는 사용자의 입력이 상대적으로 느리며 변화하는 화면의 영역이 매우 국소적이므로 압축률이 제일 좋게 평가되었다. FLASH 의 경우 화면의 변화되는 영역은 상대적으로 크나 화면이 변화하는 시간 간격이 작지 않다.

5. 결론

본 논문은 시간적으로 변하는 스크린 이미지를 캡쳐하여 저장/관리하는 방법을 제안하였다. 스크린 이미지의 특성을 분류하여 그 특성에 따라 데이터의 량을 최대한 압축하는 압축 방법을 제안하였으며, 이렇게 압축된 데이터를 효율적으로 관리하는 스크린 이미지 관리 엔진을 제안하였다. 제안된 스크린 이미지 관리 엔진은 마이크로소프트사에서 제공하는 DirectShow에 적용하여 실제로 구현하였다.

구현된 시스템을 이용하여 성능평가 결과 화면의 내용에 관계없이 99%이상의 높은 데이터 압축률을 보이는 것을 알 수 있었다.

참고문헌

[1] 로터스, "screencam", http://www.lotus.co.kr

[2] 미리온 시스템즈, "wincam 2000", http://www.wincam.net

[3] 정제창, "최신 MPEG", 교보문고, 1995

[4] C. Wayne Brown and Barry J. Shepherd, "Graphics File Formats: Reference and Guide", Manning, 1995.

[5] 마이크로소프트, "DirectShow", MSDN, 1999.

[6] John Miano, "Compressed Image File Formats", Addison-Wesley, 1999

[7] 마이크로소프트, "Advanced Streamming Format", MSDN, 1999.

[8] 이재문, "시간적으로 변하는 스크린 이미지 압축 방법", 한성대학교 정보통신논문집 3권, 71-77, 2000