실시간에서 여러 개의 동영상을 재생하는 미디어 플레이어의 설계 및 구현

조종근[†] · 김상민[††] · 임영환[†††]

요 약

컴퓨터의 성능 향상으로 인해서, 멀티미디어 플레이어에 대한 사용자의 요구는 하나의 동영상 재생기(Media Player)에서 여러 개의 동영상을 동시에 실행시킬 수 있는 복수 동영상 재생기가 일반화 될 것으로 본다. 현재로는 단일 동영상 재생에서 사용한 방법을 그대로 사용한 복수 동영상 재생을 지원하는 제품이 나왔으나, 그로 인한 문제점은 해결하지 못하고 있다. 본 논문에서는 단일 동영상 재생에서 사용되어지는 방법들을 복수 동영상의 재생 시에 적용할 경우에 생기는 문제점들 즉, 복수 동영상 재생시의 깜빡임 현상, 재생 우선 순위 처리 그리고 반투명 처리 문제와 여러 개의 동영상을 재생할 경우에 가장 큰 문제점으로 공간 연출 지원이 이루어지지 않는다는 점을 인식하고, 오버랩핑(overlapping)과 오버레이(overlay)기법을 사용하는 복수 동영상의 공간 재생기를 설계 및 구현하였다.

Design and Implementation of a Media Player That Playing Several Moving Pictures in Real Time

Jong-Keun Cho[†], San-Min Kim[††] and YoungHwan Lim[†††]

ABSTRACT

With the improvement of the computer performance, the users demand for the multimedia player is getting higher, which will be the major force behind the wide distribution of the media player that allows several moving pictures to run simultaneously using multiple windows on the screen. Products that adopted the same method used for a single moving picture and yet support the several moving pictures simultaneously emerged; however, they are not immune from problems. In this paper, we propose method to solve the problems of blinking, play prioritization processor and semi-transparent processor that occur when appling several moving pictures to the method used for a single moving picture and suggested the possible solutions. Taking it one step further, it focused on design and implementation of the media player that allows several moving pictures using overlapping and overlay technique by recognizing the biggest problem, which is that the above media play is not supported when playing several moving pictures simultaneously.

Key words: Multimedia Player, Moving Picture, Blinking

1. 서 론

멀티미디어에 대한 사용자의 요구 중에 동영상에 대한 관심은 다른 것에 비해서 높은 비중을 차지하고 있다. 현재는 단일 동영상 파일 혹은 스트림에 대한 연구가 많이 진행되었으며 많은 해결책이 나와 있고 복수 동영상 스트림에 대한 연구도 많이 진행되어 있다. 그러나 그로 인해서 복수 동영상 재생 시 발생하는 문제점을 파악하고 해결하려는 노력이 부족한 현실이다. 하나의 플레이어에서 멀티비젼과 같이 다

접수일 : 2002년 4월 17일, 완료일 : 2003년 3월 19일
[†] 정회원, 숭실대학교 컴퓨터학과 박사과정
[††] 숭실대학교 컴퓨터학과 석사과정
[†††] 종신회원, 숭실대학교 미디어학부 부교수

수의 영화를 감상하면서 TV도 볼 수 있는 복수 동영상 재생에 있어서 각 동영상의 크기 및 배치가 상당히 중요하다. 왜냐하면 한정된 공간에서 다수의 동영상을 보여줘야 하기 때문에 여러 개의 영화 화면이 겹치는 경우가 발생한다.

본 논문에서는 재생 공간의 새로운 개념으로써 한 개의 동영상 플레이어에서 복수개의 동영상 재생 시 발생하는 문제점을 파악하여 그에 대한 해결책을 제시하고 해결책을 이용하여 소프트웨어를 설계 및 구현하는데 초점을 두고 있다.

2. 관련 연구

복수 동영상을 처리하기 위한 관련된 소프트웨어를 소개하는데 그 중에 2가지가 ESSENCE와 VIP이다.

2.1 ESSENCE

ESSENCE(EasyMedia Streamming Service Engine Consumer Edition)은 EasyMedia사의 스트림 엔진이며 과거 버전 이름은 MuX이다.[1-3] ESSENCE는 멀티 스트림 처리 기술을 보유하고 있으며 네트워크를 통한 스트림 전송이 가능하다. 이렇듯 복수 동영상 재생을 지원하지만 복수 동영상의 공간 배치, 즉, 연출 공간에 대한 처리는 지원해주고 있지 않고 있다. 즉, 사용자가 재생 공간에 대해서 설정할 수 있는 기능이 없으며 복수 동영상을 재생할 수는 있으나 재생 공간에 대해서 처리해 줄 수 있는 기능이 없는 관계로 하드 코딩된 위치에서만 재생이 된다.

본 논문에서는 ESSENCE에서 처리하지 못하고 있는 복수 동영상의 공간적인 배치 문제를 해결하고자 한다.

2.2 VIP

VIP(Visual Interface Player)는 멀티 스트림들 간의 시간적 공간적 편집을 사용자들이 쉽게 해결하고자 개발된 프로그램이다.[4,8] 사용자는 비쥬얼(Visual)한 편집 화면을 이용하여 직관적으로 문제를 해결할 수 있으므로 손쉽게 원하고자하는 시간적 공간적 편집을 행할 수 있다. 매우 쉬운 인터페이스를 통해서 초보자도 쉽게 활용할 수 있는 것이 특징이다. 그러나 VIP에서 제공해주고 있는 공간적 편집, 재생 기능은 매우 단순하여서 동영상의 재생 위치와 크기만 조절할 수 있다. 이로 인해서 재생 영역을 공유하는 곳에서는 서로 간의 화면 갱신 시간차로 인하여 재생시 깜빡임 문제가 발생하고 있다.

본 논문에서는 VIP의 공간적 편집 기능을 보완하여 완벽한 공간적 편집 및 재생 문제를 해결하고자 한다.

3. 문제점 및 해결 방향

본 논문에서는 하나의 동영상 플레이어에서 여러 개의 동영상을 재생하기 위한 플레이어를 구현하는데 문제가 되는 경우 즉, 여러 개의 동영상이 서로 겹쳐 있을 때 겹쳐 있는 부분까지도 볼 수 있는 부분을 해결하려고 한다. 일반적으로 윈도우 API를 이용한 윈도우 방식의 프로그램으로 한 개의 윈도우에 같은 크기의 다른 윈도우 하나를 올려 놓으면 가려진 부분은 당연히 보이지 않으며, 포함관계가 아닌 일부분만 겹치는 윈도우의 경우 역시 윈도우 포커스가 있는 윈도우가 상위가 되며, 완벽한 상위 윈도우 관계를 표현할 수 없기 때문에 임의로 윈도우의 상하 관계를 설정하기 위해서는 모든 윈도우를 직접 그려주는 방법이 효율적이며, 이 방식으로 본 논문에서는 재생공간의 새로운 개념으로써 하나의 동영상 플레이어 안에서 2개 이상의 동영상을 재생시킬 때 발생하는 동영상간의 깜빡인 문제 즉, 각각의 다른 Frame Rate를 가진 동영상을 뿌려주는 부분의 영역들을 담당하고 있는 프로세스들의 재생 시차 때문에 사람의 눈에는 각각의 동영상이 서로 깜빡거리는 것처럼 보인다. 이처럼 복수 동영상 재생시에 발생할 수 있는 깜빡임 현상, 재생 우선 순위 처리 그리고 반투명 처리와 여러 개의 동영상을 재생할 경우에 가장 큰 문제점으로 공간 연출 지원이 이루어지지 않는다는 점을 인식하고, 오버랩핑(Overlapping)과 오버레이(Overlay)기법을 사용하여 복수 동영상의 공간 재생기를 설계 및 구현하였다.

3.1 복수 동영상 재생 시의 깜빡임 현상

동영상 재생 방법은 재생을 위한 윈도우를 생성하고 설정된 프레임 레이트에 맞추어서 압축된 프레임 정보를 복원하여 윈도우에 프레임을 그리는 것이다.

이러한 방법을 사용하여 복수 동영상 재생시 발생하는 문제점으로 깜빡임 현상을 들 수 있다.

예를 들어 두 개의 동영상이 서로의 일부분이 중복되는 그림 1은 같은 위치에서 동시에 재생이 되며 프레임 레이트(Frame Rate)가 다르다고 가정하고 그림 상의 왼쪽에 위치한 동영상을 A라 정하고 오른쪽에 위치한 동영상을 B라고 정하자. A의 프레임 레이트는 B의 프레임 레이트 보다 두 배 더 높으며 B가 A보다 약간 늦게 재생이 시작되었다고 가정하면 그림 2와 같은 순서로 깜빡임 현상이 발생하게 되며, 이러한 문제점은 각 동영상이 서로 독립적인 화면 갱신 능력으로 인해 비롯된다.

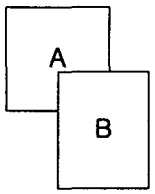


그림 1. 재생공간이 겹치는 동영상

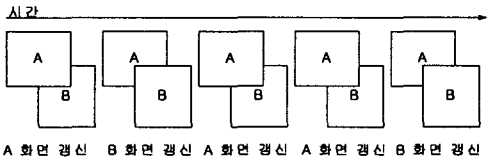


그림 2. 깜빡임 현상

3.2 서로 다른 프레임 레이트를 가진 복수 동영상의 재생

깜빡임 현상의 원인은 크게 두 가지로 나눌 수 있다. 하나는 화면 갱신 시간 차이가 원인이다. 화면 갱신 시간 차이는 또 다시 두 가지로 나눌 수 있다. 하나는 서로 다른 프레임 레이트가 원인이고, 다른 하나는 재생 시작 시간 차이로 인한 원인이다. 깜빡임 현상의 다른 원인은 동영상마다 독자적인 화면 갱신 능력이다.

예를 들어 프레임 레이트가 다른 A, B 두 개의 동영상이 있다고 가정하자. 공간 연출은 그림 3과 같이 A, B는 C 윈도우를 부모로 하고 있다. A가 화면 갱신할 시간이 되면 자신만의 재생 공간을 그리게 되고 이로 인해서 A와 재생 공간이 겹치게 되는 B의 공간에는 A가 그린 것으로 채워지게 된다. 이와 마찬

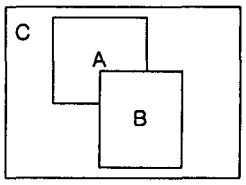


그림 3. 동영상의 부모 윈도우

가지로 B가 화면 갱신할 시간이 되면 자신만의 재생 공간을 그리게 되고 이로 인해서 B와 재생 공간이 겹치게 되는 A의 공간에는 B가 그린 것으로 채워지게 된다. A와 B가 화면 갱신 시간이 되면 부모 윈도우인 C 윈도우에게 화면 재생을 해야함을 알리고 C는 이를 수용하고 A의 화면 정보와 B의 화면 정보를 가져와서 화면을 재구성하게 되면 깜빡임 현상은 발생하지 않게 된다. 대신 C가 화면 갱신 시 A와 B 중 어느 것을 먼저 갱신하느냐에 따라서 A와 B가 겹치는 공간에서 어느 한쪽만 화면에 보이게되는 문제가 발생한다. 그런데 여기서 또 다른 문제가 발생하게 된다. 위에서 언급한 부모 윈도우에게 화면 갱신 할 것을 알리고 부모 윈도우가 화면을 재구성하게 되는 방법을 사용할 경우 화면 갱신을 너무 과도하게 하게되는 문제가 발생한다.

예를 들어 동영상 A, B가 프레임 레이트가 서로 동일하게 15이고 B가 A보다 약간 느린 시점에서 재생이 이루어지게 되면 부모 윈도우 C의 프레임 레이트는 30이 된다. 이보다 더 많은 동영상이 존재하고 모든 동영상의 재생 시간이 동일하지 않을 경우 모든 동영상의 프레임 레이트를 더한 것이 부모 윈도우의 프레임 레이트가 되어 버린다. 이러한 방법을 사용하게되면 동영상의 개수가 많아질수록 부모 윈도우의 프레임 레이트가 높아지는 문제가 발생하게 된다. 그래서 그림 4와 같은 여러 개의 동영상을 동시에 관리하는 모니터 미디움 매니저를 제안하다.

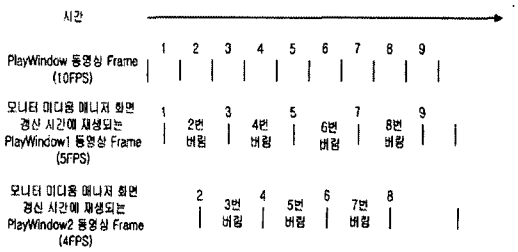


그림 4. 화면 갱신을 제어하는 디스플레이 매니저의 시간에 그려지는 재생 윈도우의 Frame 비교

3.3 재생 우선 순위 처리기

2.2절에서 언급한 동영상 A, B가 각각 부모 윈도우 C에게 화면 갱신할 것을 요구하고 부모 윈도우 C는 A와 B에게 화면 갱신 요청을 받고 화면 갱신을 하게 될 때 발생하는 문제점은 부모 윈도우 C가 화면 갱신 시 A와 B 중 어느 것을 먼저 갱신하느냐에 따라서 A와 B가 겹치는 공간에서 어느 한쪽만 화면에 보이게 되는 문제가 발생한다. 즉, 이러한 문제는 복수 동영상 재생 시에 반드시 해결해야 하는 중요한 요소로 대두된다. 이러한 문제를 해결하기 위해서는 동영상들의 재생 우선 처리기를 둠으로써 해결 할 수 있다.

3.4 반투명 처리기

2.3절에서 언급한 재생 우선 순위 처리 문제를 해결해도 문제는 남아있다. 재생 우선 순위에 밀려 난 동영상은 자신의 일부 재생 영역을 사용하는 우선 순위가 높은 동영상에게 재생 공간을 빼앗기게 되기 때문이다. 즉, 우선 순위에 밀린 동영상은 화면상에서 보여지지 않게 된다. 이러한 문제는 사용자적 요구에 의해서 요구될 수 있는 문제이기도 하기 때문에 본 논문에서는 하나의 문제점으로 인식하였다. 이 문제를 해결하기 위해서 반투명 처리기를 제안한다.

3.5 연구 목표 및 방향

본 논문은 두 가지 목표를 가지고 있다. 첫 번째로 복수 동영상의 공간 연출 시에 발생하는 깜빡임 현상을 해결하는 방법을 제시하는 것이다. 두 번째로 복수의 동영상을 렌더링(Rendering)하여 하나의 동영상을 만드는 방법을 사용하지 않고 실시간으로 복수의 동영상의 공간 연출을 행하는 것이 목표이다.

4. 복수 동영상 공간 연출 방법

4.1 서로 다른 프레임 레이트를 가진 복수 동영상의 재생

복수 동영상 재생시 해결해야 할 문제점인 깜빡임 현상을 제거하기 위해서는 화면 갱신 시간이 같아야 하고 독자적인 화면 갱신을 하지 않고 화면 갱신 관리자가 화면을 갱신하여야한다. 이를 위해서는 화면 갱신을 관리하는 화면 갱신 관리자를 필요로 하며 화면 갱신 관리자는 모든 동영상들의 부모 윈도우여만 한다.

본 논문에서는 화면 갱신 관리자를 모니터 미디움 매니저로 정의하였다. 모니터 미디움 매니저의 역할은 설정된 공간 연출을 기준으로 모든 동영상의 화면 갱신을 관리하는 것이다. 설정된 공간 연출을 기준으로 모니터 미디움 매니저는 공간에 대한 관리를 행하게 된다. 공간에 대한 관리는 스크린의 바탕 색 및 바탕 그림 관리와 재생 윈도우들의 재생 공간의 위치와 크기를 관리하는 것을 말한다.

4.2 오버랩핑 처리기

본 논문에서는 모니터 미디움 매니저 안에 재생 우선 순위 처리기를 통하여 재생 우선 순위를 처리하였다. 재생 우선 순위 처리기의 역할은 사용자가 설정한 재생 우선 순위에 부합되도록 화면 갱신 순서를 정하고 모니터 미디움의 화면 갱신 시간에 정해진 순서에 따라 화면 갱신을 행하는 것이다. 이 때 각 동영상의 해상도가 너무 높아서 각 동영상 화면 갱신 시간이 오래 걸리게 되면 재생 공간이 겹치는 곳에서 우선 순위가 제일 높은 동영상의 화면만 화면에 보여져야 하지만 우선 순위가 높은 동영상은 우선 순위가 낮은 동영상에 비해 화면 갱신이 늦게 적용되기 때문에 잠시 동안 이나마 우선 순위가 낮은 동영상이 보이게 된다. 이를 해결하기 위해서 오버랩핑(Overlapping) 처리기는 화면 갱신이 적용되는 비디오 메모리에 직접적으로 화면 정보를 복사하지 않고 화면 갱신에 사용하지 않는 여분의 비디오 메모리 영역 혹은 비디오 메모리 영역이 적어서 사용할 수 없는 경우 일반 메모리에 작업 영역을 생성하여 화면 정보를 적용하게 된다.

4.3 오버레이 처리기

오버레이 처리기의 역할은 선택된 동영상 프레임을 알파 블렌딩(Alpha Blending)처리를 하여 화면에 표시하는 것이다. 알파 블렌딩은 소스(Source)와 데스티네이션(Destination)의 각 픽셀에 대해서 연산을 행하게 되는데, 픽셀 연산은 또 R, G, B에 대해서 각각 연산을 행한다. 알파 블렌딩 수식은 아래의 수식과 같다.[5]

결과물 = source_pixel × alpha 값 +

destination_pixel × *(1 - alpha)*
(단, alpha 값은 0~1사이의 소숫점 값)

오버레이(Overlay) 처리기의 동작 방법을 살펴보자. 모니터 미디움 매니저가 화면 갱신 시간이 되면 재생 윈도우로부터 동영상 프레임을 전달받게 되고 전달받은 동영상 프레임은 오버레이 처리기를 통해서 반투명 처리된다. 이때의 알파 블렌딩 소스는 동영상 프레임이 되며 데스티네이션은 오버레이 처리기의 작업 영역이 된다. 작업 영역은 알파 블렌딩 처리 직전에 초기화된다. 이와 같이 선택된 동영상에 대해서 알파 블렌딩 처리하고 이를 작업 영역에 저장하여 모니터 미디움 매니저에게 전달하게 된다. 모니터 미디움 매니저는 전달받은 화면 정보를 최종적으로 화면에 적용하게 된다.

이와 같이 오버랩핑 처리기와 오버레이 처리기를 통해서 자연스런 공간 연출을 행할 수 있지만, 위에서 언급한 방법만을 사용한다면 공간 연출은 우선 순위 처리 방식 혹은 반투명 처리 방식 중 그림 5와 같이 하나만 사용할 수 있다. 이로 인해 오버레이와 오버랩핑 처리기를 분리하여 관리할 수 있는 새로운 방법이 필요하게 된다.

이러한 문제를 해결하기 위해서 새로운 공간 연출 객체를 추가하였다. 새로이 추가된 객체는 그룹(Group)이며 재생 윈도우들을 하나의 그룹으로 묶고 그룹 단위로 우선 순위 처리 혹은 반투명 처리를 적용하는 것이다. 즉, 그림 6과 같이 동영상 A, B는 Group1으로 묶고 우선 순위 처리를 하여 오버랩핑

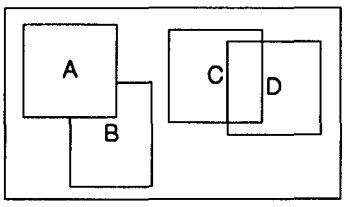


그림 5. 오버랩핑과 오버레이가 각각 적용된 공간 연출

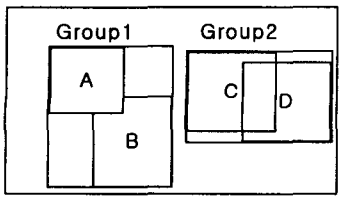


그림 6. Group 개념 추가

처리기에서 처리를 행하고 동영상 C, D는 Group2로 묶고 반투명 처리를 하여 오버레이 처리기에서 처리를 행하는 것이다. 이와 같이 그룹 객체를 추가하게 되면 위에서 언급했던 문제점을 해결할 수 있게 된다.

그러나, 아직 남아 있는 문제가 있다. 바로 그룹간 재생 설정은 처리할 수 없는 점이다. 그룹간에도 재생 설정을 적용하여 우선 순위 처리 혹은 반투명 처리를 행하기 위해서 새로운 공간 연출 객체를 추가하였다. 새로이 추가된 객체는 프레임(Frame)이며 그룹들을 또 하나의 그룹으로 묶는 역할을 한다. 이렇게 묶인 그룹들을 관리하는 프레임들간의 재생 설정도 적용할 수 있도록 프레임은 또 다시 그룹으로 묶일 수 있다. 이렇게 묶인 그룹은 최종적으로 스크린에서 관리를 하게 된다. 이렇게 해서 전체적인 공간 연출 객체를 그림으로 표현하면 그림 7과 같다.

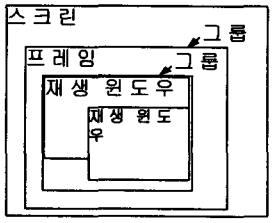


그림 7. 공간 연출 객체

5. 복수 동영상 공간 연출기 구성도 및 설계

5.1 연출기의 구성

복수 동영상 공간 연출기는 모니터 미디움, 모니터 미디움 매니저, 재생 윈도우, 프레임, 그룹, 스크린으로 구성되어 있다. 각각의 기능은 그림 8과 같다.

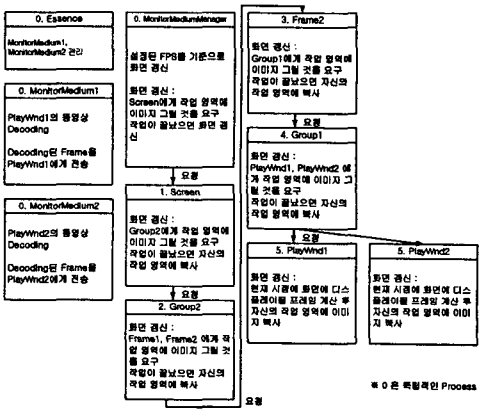


그림 8. 연출기의 구성도

5.2 모니터 미디움

모니터 미디움의 역할은 스트림 엔진인 복수 동영상 플레이어로부터 전송된 압축된 동영상 스트림을 디코딩하여 재생 윈도우에게 전달하는 것이며 가장 중요한 것은 스트림 엔진인 복수 동영상 플레이어와 공간 연출기와의 중간 다리 역할이다. 모니터 미디움의 중간 다리 역할을 살펴보자.복수 동영상 플레이어를 이용한 공간 연출을 위해서 복수 동영상 플레이어 API를 이용하여 재생 윈도우, 프레임, 그룹, 스크린을 구성할 수 있도록 하였다. 이 중에서 재생 윈도우는 모니터 미디움과 매우 밀접한 관련을 가지고 있다. 모니터 미디움은 복수 동영상 플레이어로부터 전달받은 압축된 동영상 스트림을 해독하여 재생 윈도우에게 전달하여야 하기 때문에 압축된 동영상 스트림을 모니터에 디스플레이하기 위해서 압축된 동영상 소스와 모니터 미디움을 하나의 스트림으로 구성하게 된다. 이때 모니터 미디움에 넘겨줘야 하는 정보가 재생 윈도우의 정보이다. 모니터 미디움은 넘겨온 정보를 이용하여 디코딩된 동영상 프레임을 재생 윈도우에게 전달하게 된다.

5.3 비쥬얼 객체

비쥬얼 객체(Visual Object)는 재생 윈도우, 프레임, 그룹, 스크린을 의미한다.

5.3.1 재생 윈도우

재생 윈도우는 모니터 미디움으로부터 전달받은 디코딩된 동영상 프레임들을 관리하고 모니터 미디움 매니저가 현재 프레임 정보를 요구할 시점의 동영상 프레임을 계산하고 계산된 동영상 프레임을 모니터 미디움 매니저에게 전달하는 역할을 한다.

재생 윈도우는 재생 윈도우 별로 프레임 레이트를 다르게 설정할 수 있으며 각자 설정된 프레임 레이트를 이용하여 모니터 미디움 매니저가 동영상 프레임 정보를 요청했을 시점의 동영상 프레임을 계산하는 능력을 가지고 있다. 이를 이용하여 서로 다른 프레임 레이트를 가진 복수 동영상 재생 시 정확한 동기화가 가능하다. 재생 윈도우는 설정된 재생 공간 크기만큼의 작업 영역을 가지고 있다. 작업 영역은 일반 메모리에 할당되며 그룹 객체의 동영상 프레임 정보 요청 시 현재 프레임을 계산 한 후 작업 영역을 현재 프레임으로 구성하고 캡션이 설정되어있으면 작업 영역에 캡션을 그린 후 그룹 객체에게 작업 영역의 내용물을 전달한다.

5.3.2 그룹

그룹 객체는 재생 윈도우 혹은 프레임을 관리하며 오버랩핑 처리기와 오버레이 처리기를 포함하고 있어서 설정된 재생 설정에 따라서 오버랩핑 처리기를 통하여 우선 순위 처리를 하거나 반투명 처리를 적용한다. 그룹은 비쥬얼 객체로써 재생 윈도우와 마찬가지로 자신의 연출 공간 크기만큼의 작업 공간을 가지고 있으며 프레임 혹은 스크린의 요청 시 작업 공간을 구성하고 구성된 작업 공간에 대한 정보를 프레임 혹은 스크린에게 전달한다.

5.3.2.1 오버랩핑 처리기

오버랩핑 처리기는 정해진 재생 우선 순위에 따라서 작업 영역에 영상을 그리는 역할을 한다. 작업 영역에 영상을 그리는 대상은 그룹 객체가 관리하는 비쥬얼 객체이며 비쥬얼 객체는 재생 윈도우 혹은 프레임이다.

처리기의 핵심 동작 알고리즘은 아래와 같다.

비쥬얼 객체 포인터 = 그룹의 헤더
while(그룹의 개수만큼) {
비쥬얼 객체 포인터의 작업 영역의 정보를 얻어온다.
얻어온 비쥬얼 객체 포인터의 작업 영역을 그룹의 작업 영역에 그린다.
비쥬얼 객체 포인터를 그룹의 다음 객체로 변경한다. *}*

작업 영역에 그리는 작업이 끝나면 프레임 혹은 스크린에게 작업 영역의 정보를 전달한다.

5.3.2.2 오버레이 처리기

오버레이 처리기는 그룹이 관리하는 비쥬얼 객체의 작업 영역들을 알파 블렌딩 처리를 하여 그룹의 작업 영역을 그리는 역할을 한다.

처리기의 핵심 동작 알고리즘은 아래와 같다.

비쥬얼 객체 포인터 = 그룹의 헤더
while(그룹의 개수만큼) {
비쥬얼 객체 포인터의 작업 영역의 정보를 얻어온다.

얻어온 비쥬얼 객체 포인터의 작업 영역과 그룹의 작업 영역을 알파 블렌딩 처리하여 작업 영역을 갱신한다.
비쥬얼 객체 포인터를 그룹의 다음 객체로 변경한다. }

오버랩핑 처리기와 다른 점은 비쥬얼 객체의 작업 영역을 단순히 복사하는 것이 아니라 비쥬얼 객체의 작업 영역과 그룹의 작업 영역을 알파 블렌딩 처리를 함으로써 작업 영역을 갱신하는 것이다. 이렇게 함으로써 그룹 내의 모든 비쥬얼 객체에 대해서 불투명 처리가 가능하다.

작업 영역에 그리는 작업이 끝나면 오버랩핑 처리기와 같이 프레임 혹은 스크린에게 작업 영역의 정보를 전달한다.

5.3.3 프레임

프레임은 그룹 객체를 관리하며 재생 윈도우의 배경 틀 역할을 한다. 프레임이 관리하는 그룹 객체들도 더블 링크드 리스트로 관리하며 작업 영역 갱신은 오버랩핑 처리기를 이용하기 때문에 리스트에 추가된 순서로 비쥬얼 객체의 작업 영역을 자신의 작업 영역으로 복사한다. 작업 영역에 그리는 작업이 끝나면 그룹 객체에게 작업 영역의 정보를 전달한다.

5.3.4 스크린

스크린은 프레임과 같이 그룹 객체를 관리하며 재생 윈도우의 배경 틀 역할을 한다. 스크린도 프레임과 같이 그룹 객체들을 더블 링크드 리스트로 관리하며 작업 영역 갱신도 오버랩핑 처리기를 이용한다. 프레임과 달리 작업 영역에 그리는 작업이 완료되면 모니터 미디움 매니저에게 작업 영역의 정보를 전달한다.

5.3 모니터 미디움 매니저

모니터 미디움 매니저는 연출 공간을 관리하는 역할을 한다.

모니터 미디움 매니저가 관리하는 비쥬얼 객체는 오직 스크린 뿐이다. 모니터 미디움이 스크린에게 작업 영역 내용을 전달해줄 것을 요청하면 스크린은 자신이 관리하는 그룹에게 요청을 전달 하게되고 그룹은 자신이 관리하는 프레임에게 요청을 전달하며 프레임은 다시 자신이 관리하는 그룹에게 요청을 전달하고 그룹은 자신이 관리하는 재생 윈도우에게 요청을 전달한다. 최종적으로 요청을 전달받은 재생 윈도우는 현재 시간을 참조하여 프레임 버퍼 내에서 현재 동영상 프레임을 찾아낸 후 작업 영역을 구성한다.

5장의 내용들을 요약해 보면 그림 8에서 본 것처럼 모니터 미디엄 매니저에서는 화면이 갱신할 시간이 되면, 먼저 Screen을 그린후에 Frame들을 그리고 Group으로 처리된 재생 윈도우들을 그린다. 이 시간에 재생 윈도우는 그 시간에 재생될 프레임의 정보를

MonitorMediumManager(모니터 미디움 매니저)
while(모니터 미디움 매니저) {
1. 사용자가 설정한 화면 갱신 비율에 맞도록 준비를 한 후에 대기한다.
2. Screen에게 화면 갱신 준비 요청한다.
3. Screen을 화면에 갱신한다. }

Screen(스크린)
if { (화면 갱신 준비 요청을 받았을 경우)
then
1. Frame들에게 화면 갱신 준비 요청한다.
2. Frame들이 준비가 끝났다면 Frame들이 가지고 있는 화면 정보(Bitmap)를 얻어와서 오버랩핑 혹은 오버랩 설정에 맞도록 스크린 화면을 재구성한다.
3. 화면을 갱신한다.
4. Frame들이 제공한 화면 정보들을 이용하여 준비한 스크린의 화면 정보(Bitmap)를 비디오램에 적용한다.

else
1. 화면 갱신 비율에 맞도록 준비를 한 후에 대기한다. }

Group(그룹)
if { (그룹 구성에 대한 요청을 받았을 경우)
then
1. VisualObject 객체를 추가한다.
2. 해당 인덱스에 VisualObject를 추가한다.
3. 해당 Index의 VisualObject를 삭제한다.
4. 리스트의 갯수를 리턴한다.
5. 리스트의 VisualObject 포인터를 리턴한다.
6. 재생을 설정한다.
7. 재생 설정값을 얻어 온다.

else
1. 그룹 구성을 위한 작업을 대기한다. }

Frame(프레임)
if { (화면 갱신 준비 요청을 받았을 경우)
then
1. 재생 윈도우들에게 화면 갱신 준비 요청한다.
2. 재생 윈도우들이 가지고 있는 화면 정보(Bitmap)을 얻어와서 오버랩핑 혹은 오버랩 설정에 맞도록 프레임 화면을 재구성한다.
3. 화면을 갱신한다.
4. 재생윈도우들이 제공한 화면 정보들을 이용하여 준비한 스크린의 메모리 버퍼에 전송한다.

else
1. 프레임 구성을 위한 작업을 대기한다. }

PlayWindow(재생 윈도우)
1. 독립적인 Thread로 자신이 할당받은 비디오 소스를 원본의 디코딩 속도에 맞추어서 디코딩하고 있다.

{
do {
1. 현재 디코딩한 비트맵을 보관한다.
2. 화면을 갱신한다.
3. 보관했던 비트맵을 프레임에게 전송한다.
}
while(화면 갱신 준비 요청을 받았을 경우);
return(0) ;
}

가지고 있고 이를 모니터 미디엄 매니저가 사용하게 되면 된다. 이러한 메카니즘을 사용하게 되면 Frame Rate가 다른 재생 윈도우들에 대한 처리가 가능해진다.

첫 번째로 재생 윈도우 알고리즘은 각각 재생 윈도우가 설정된 Frame Rate를 기준으로 현재 재생될 Frame에 대한 정보를 판별하여 모니터 미디움 메니져에게 전달하게 되며, 재생 윈도우의 Frame Rate가 15 FPS(Frame Per Second) 일 경우 각 Frame들은 두 번씩 사용된다. 기본적으로 모니터 미디엄 매니저는 30 FPS로 화면을 갱신한다.

두 번째로 재생 윈도우가 현재 Frame 계산하는 알고리즘은 Window API중 시스템 시간을 알려주는 GetTickCount()를 이용하여 현재 유효 Frame을 판별한다. GetTickCount()는 1000ms단위의 현재 시간값을 알려주며, 재생이 시작되면 그 순간의 GetTickCount()값을 저장하고, 다음 Frame이 나올 TickCount값까지 기다린다. TickCount값이 다음 Frame으로 갱신할 시간이 되면 유효 Frame을 변경한다.

위의 내용들을 의사코드(pseudo-code)로 나타내면 아래와 같다.

6. 실험 및 결과 분석

6.1 실험 방법

실험에 사용한 연출 공간은 다음과 같이 설정하였고 대략적인 공간 배치는 그림 9와 같다.

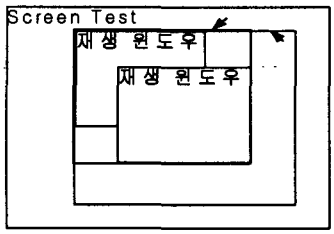


그림 9. 실험에 사용된 공간 연출

스크린 : 위치 : x 좌표 = 0, y 좌표 = 0
크기 : 가로 640 픽셀, 세로 480 픽셀
배경 그림 : 설정
그룹 멤버 : 그룹1
캡션 : Screen Test

그룹1 : 그룹 멤버 : 프레임
재생 설정 : 오버랩핑(우선 순위)

프레임 : 위치 : x 좌표 = 100, y 좌표 = 50
크기 : 가로 500 픽셀, 세로 400
배경 색 : 초록
그룹 멤버 : 그룹2
캡션 : Frame Test

그룹2 :
그룹 멤버 : 재생 윈도우1, 재생 윈도우2
재생 설정 : 오버랩핑(반투명)

재생 윈도우1 :
위치 : x 좌표 = 0, y 좌표 = 0
크기 : 가로 352 픽셀, 세로 288 픽셀

재생 윈도우2 :
위치 : x 좌표 = 50, y 좌표 = 50
크기 : 가로 352 픽셀, 세로 288 픽셀

6.2 실험 결과

공간 연출기를 실험한 동작 화면을 캡쳐한 결과는 그림 10과 같다.

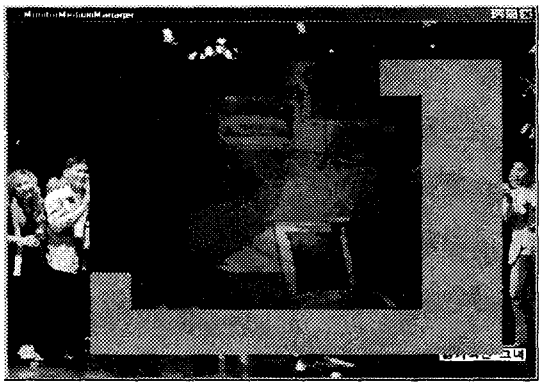


그림 10. 실험 결과 화면

스크린에는 배경 그림이 설정되어 있으며 스크린에는 한 개의 그룹이 있으며 이 그룹은 한 개의 프레임을 가지고 있으며 이 프레임은 또 한 개의 그룹을 소유하고 있고 이 그룹은 두 개의 재생 윈도우를 포함하고 있는 것을 알 수 있다. 이 중에서 투명 처리가 적용된 그룹은 프레임 내에 존재하는 그룹임을 알 수 있으며 두 재생 윈도우가 공유하고 있는 재생 공간에는 서로의 영상이 알파 블렌딩이 적용되어서 나타나고 있는 것을 확인할 수 있다.

또한 재생 시간이 완료될 시점까지 재생윈도우, 프레임, 그룹, 스크린 모두에서 깜빡임 현상이 나타나지 않았음을 미루어 깜빡임 현상이 해결됐음을 알 수 있다.

표 1과 표 2의 결과를 살펴보면 투명 처리를 하였을 때와 하지 않았을 때 매우 뚜렷한 성능 차이를 보임을 알 수 있다. 동영상의 개수가 많아질수록 성능이 느려지는 가장 큰 원인은 압축된 동영상의 해독 때문이다. 이러한 문제는 중앙처리장치의 성능에 좌우하기 때문에 좋은 성능의 처리장치를 이용하면 해결될 것으로 판단된다.

표 1. 6.2와 공간연출은 동일하고 투명처리를 사용하지 않고 초당 프레임 레이트를 20으로 설정하고 재 생 윈도우 개수만 다를 경우 실제 재생되는 초당 프레임 레이트

재생 윈도우 개수	초당 프레임 수(FPS)
1	20
2	20
3	18
4	17
5	15

표 2. 6.2와 공간연출은 동일하고 재생 윈도우들은 모두 투명처리를 사용하고 초당 프레임 레이트를 20으로 설정하고 재생 윈도우 개수만 다를 경우 실제 재생되는 초당 프레임 레이트

재생 윈도우 개수	초당 프레임 수(FPS)
1	7
2	4
3	3
4	2
5	1

이와 달리 투명처리 하였을 때와 하지 않았을 경우 매우 큰 성능 차이를 보이는데, 이는 투명 처리로 인한 오버헤드 때문이다. 투명 처리를 하기 위해서는 압축된 YUV 포맷의 영상을 RGB로 변환하고 이를 픽셀별로 비교하여 적용하기 때문에 많은 계산이 필요하다. 이로 인해서 투명 처리로 인한 오버헤드가 발생하는 것이다. 이러한 문제는 비디오 카드에서 하드웨어 투명 처리를 지원해 주거나 고성능의 중앙처리장치를 사용함으로써 해결될 것으로 판단된다.

참 고 문 헌

[1] 임영환, "ComBiStation : 분산 멀티미디어 컴퓨팅 환경을 위한 컴퓨터 플랫폼," 정보과학회 논문지, 제 2권, 제 1호, pp.160-181, 1996.

[2] Baker R. A. Dowring, K.Finn, E.Rennison, D.H.Kim, and Y.H. Lim, "Multimedia Processing Model for a Distributed Multimedia I/O System," Proceedings of 3rd International Workshop on Network and Operating Systems for Digital Audio/Video, pp.233-239, 1993.

[3] Rennison. E, R.Bker, D.H.Kim, and Y.H.Lim, "MuX : An X Co-Existant Time-Based Multimedia I/O Server," The X Resource, Issue pp.213-233, 1, 1992.

[4] 임영환, 이명수, 이선혜, 우시연, "하이퍼 프리젠테이션을 위한 아이콘 프로그래밍 도구," 한국정보 처리학회 논문지 제 5권 제 6호, 1998.

[5] 나우누리 게임 제작 포럼(telnet://nownuri.net : go ngm)

[6] K.R.Rao, J.J. Hwang, Techniques & Standards for Image · Video & Audio Coding, Prentice

Hall, 1996.
[7] R.J.Clarke, Digital Image Compression Algorithms and Standards Academic Press, 1995.
[8] VIP, http://www.easymedia.co.kr
[9] Francis. Li, Anoop. Gupta, Elizabeth. Sanocki, Li-wei. He, Yong. Rui, "Browsing Digital Video," Technical Report, 1999.
[10] J. Yu and Y. Xiang, "Hypermedia Presentation and Authoring System," Proceedings of the 6th International WWW Conference, pp. 153-164, April 1997.
[11] R. Alur, T. A. Henzinger, and O. Kupferman, "Alternating-time Temporal Logic," In Proc, 38th IEEE Symp, Foundations of Comp, SCI, pp. 100-109, 1997.
[12] L. D. Alfaro, T. A. Henzinger, O. Kupferman, "Concurrent Reachability Games," IEEE Symposium on Foundations of Computer Science, 1998.
[13] SYMN : Synchronized Multimedia Working Group, Http://www.w3.org/AudioVideo.
[14] Senna. J, "Streaming Media for The Enterprise," InfoWorld, February(14), pp. 63-66, 2000.

조 종 근

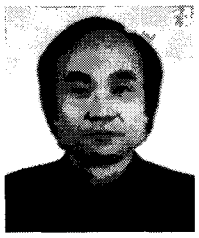
1998년 성결대학교 컴퓨터공학과 졸업(공학사)
2001년 숭실대학교 컴퓨터학과 졸업(공학석사)
2001년~현재 숭실대학교 컴퓨터학과 박사과정

관심분야 : 멀티미디어, 컴퓨터 그래픽스

김 상 민

1999년 숭실대학교 컴퓨터학과 졸업(공학사)
2002년 숭실대학교 컴퓨터학과 석사과정

관심분야 : 멀티미디어



임 영 환

1977년 경북대학교 수학과 졸업(이학사)
1979년 한국과학원 전산학과 졸업(공학석사)
1985년 Northwestern Uni versity 전산학과(공학박사)
1979년~1996년 한국전자통신 연구소 책임연구원
1996년~현재 숭실대학교 미디어학부 부교수
관심분야 : 멀티미디어

교 신 저 자

조 종 근 156-743 서울시 동작구 상도 5동 1-1 숭실대학교 별관 211호 미디어 연구실