JNI의 Native영역에서 처리하는 렌더링 기법

신용경°, 박지현, 김미영, 정재일, 이현주, 김상욱
경북대학교 컴퓨터과학과

Rendering Technology Processing on Native Space of JNI

Y. Shin°, J. Pak, M. Kim, J. Jung, H. Lee, S. Kim
Dept. of Computer Science, Kyungpook National Univ.
E-mail : shinyk@woorisol.knu.ac.kr

요 약

효율적인 스트리밍을 지원하기 위하여 JNI(Java Native Interface)를 이용한 플레이어를 구현한다. 미디어 플레이어는 전달 관리기, A/V 디코더, 렌더링 모듈, 자바 그래픽 라이브러리로 구성하고, 렌더링 모듈은 JNI Export Manager, Java Scene Manager, Drawing Manager, JNI API 로 구성한다. 디코딩 된 스트림은 자바가 읽어들일 수 있는 객체 타입으로 변환하는 JNI Export Manager와 장면 구성하는 Java Scene Manager, 자바 그래픽 라이브러리를 이용하여 실제 화면에 드로잉하는 Drawing Manager를 통하여 재생한다. 본 논문은 Java Scene Manager를 Native 영역에서 처리하여 디코딩 된 스트림 객체를 전달받아 RGB변환하고, 장면 구성정보를 이용하여 장면을 구성하여 최종적으로 JNI Export Manager에게 전달한다. 따라서 장면을 구성하기 위한 정보를 JNI API를 통하여 전달할 필요가 없으므로 그 만큼의 성능 향상을 보이고, Native 영역에서 처리하므로 자바 언어에서 처리하는 것보다 효율적이다. 이는 성능 비교표를 통하여 재생시간 향상을 보인다.

1. 서론

멀티미디어 스트리밍은 PC가 인터넷에 연결되기 시작하면서 본격적으로 개발되고 서비스되기 시작한 기술이다[1]. 스트리밍은 미디어 서버와 미디어 플레이어 사이의 멀티미디어 컨텐츠를 연속적으로 전달 해야 한다. 따라서 스트리밍 QoS는 서버와 플레이어 사이의 연결속도나 플레이어의 렌더링 기술에 상당한 영향을 받는다[2]

본 논문은 효율적인 스트리밍을 지원하기 위하여 JNI(Java Native Interface)를 이용한 플레이어를 구현하고, 스트리밍의 렌더링 모듈별 성능분석을 보인다. 플레이어는 전달 관리기, 디코더, 렌더링 모듈로 구성한다. 전달 관리기와 디코더는

Native 코드로 구현하고, 렌더링 모듈은 자바 그래픽 라이브러리를 이용하여 재생한다. Native 코드와 자바 메소드 사이의 스트림 인터페이스를 JNI로 처리한다. 이러한 구조는 미디어 플레이어 전체를 하나의 코드로 구현하는 구조보다 효율적이다[3,4]. JVM(Java Virtual Machine)인 자바 가상 머신에서 스트리밍 서비스되므로 플랫폼에 독립적인 자바의 장점을 가지고, Native 코드는 C 코드를 이용하므로 디코딩 속도를 향상시켜 끊김없는 스트림 전송을 지원한다. 그러나, JNI를 이용하여 Native와 자바 사이의 데이터 릴리즈는 전체 재생시간에 상당한 영향을 준다. 미디어 플레이어의 렌더링 모듈을 향상시켜 데이터 릴리즈 시간을 줄인다.

본 논문의 구성은 2장에서 JNI 스트리밍 구조을 보이고, 3장에서 디코더와 렌더링 모듈사이의 장면구성을 Native 영역에서 처리하는 기법을 설명한다. 4장은 렌더링 모듈별 성능분석을 보이고 5장에서 결론을 맺는다.

2. JNI 스트리밍 구조

미디어 스트리밍은 전달 관리기, A/V 디코더, 렌더링 모듈, 자바 그래픽 라이브러리로 구성하고, 렌더링 모듈은 JNI Export Manager, Java Scene Manager, Drawing Manager, JNI API 로 되어있다. 그림 1은 JNI를 이용한 미디어 스트리밍 구조를 보인다.

전달 관리기는 미디어 서버로부터 실시간으로 전송되는 스트림을 각각의 오디오, 비디오 버퍼에 저장하기 위하여 디먹싱 한다. 분리된 오디오, 비디오 스트림은 A/V 디코더에 의해 디코딩 한다. 렌데링 모듈에서 JNI API를 이용하여 자바 그래픽 라이브러리와 인터렉션하여 화면에 보인다[5]. 전달 관리기와 A/V 디코더는 C 언어 메소드로 모듈을 구성하고 렌더링 모듈에 의해 화면에 보이는 그래픽은 자바 언어 메소드로 구성하므로 이 두 다른 타입의 데이터에서 스트림을 인터렉션하기 위해서 JNI API를 호출한다.

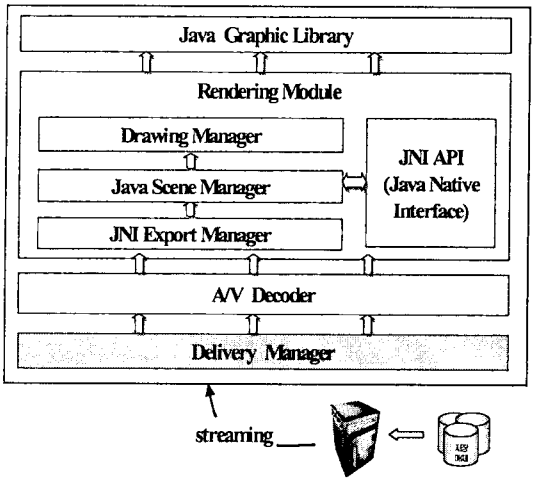

그림 1. JNI를 이용한 미디어 스트리밍 구조

렌더링 모듈은 A/V 디코더에서 디코딩된 오디오, 비디오 스트림중에서 오디오 스트림은 사운드 드라이버를 설정 후 스피커에 재생하고, 비디오 스트림은 자바 그래픽 라이브러리를 통하여 화면에 보인다. A/V 디코더에서 나온 최종 스트림과 렌더링 모듈에서 읽어들일 스트림의 타입이 다르다. C 언어에서 지원하는 포인터 타입을 자바언어에서 지원하는 데이터 타입으로 변화해야 한다. 변화해야 할 데이터 타입이 많으면 플레이어의 성능을 저하시킨다. 따라서 디코딩된 스트림을 Java Scene Manager를 이용하여 장면구성 후 JNI Export Manager에게 데이터를 전달한다. JNI Export Manager에서 자바가 읽을 수 있는 객체

타입으로 변환된 데이터는 Drawing Manager에서 자바 그래픽 라이브러리를 이용하여 화면에 보낸다.

3. Native 영역에서 처리하는 렌더링

그림 2는 일반적인 미디어 플레이어에서 렌더링하는 흐름을 보인다[5]. A/V 디코더에서 최종 생성된 스트림은 자바 메소드가 읽어들일수 있는 데이터 타입으로 변환한다. 이는 그림 2에서 JNI Export Manager에서 한다.

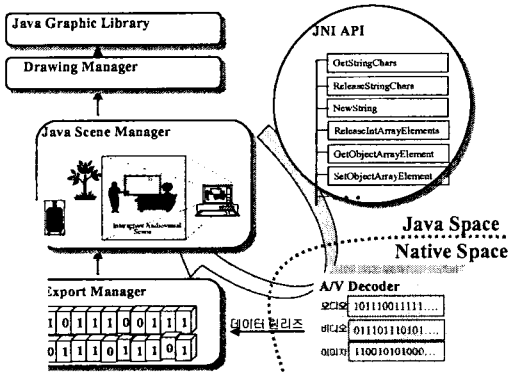

그림 2. 일반적 렌더링에서 장면생성 흐름

자바언어는 객체 지향 언어이고, Native 코드로서의 C 언어는 포인터 타입으로 데이터를 참조한다. 포인터 타입으로 데이터를 넘기는 C 언어값은 자바 언어는 읽을수 없으므로 JNI API를 이용하여 자바 메소드가 읽어들일수 있는 객체타입으로 변환한다. Java Scene Manager는 자바 객체 타입으로 변환된 스트림을 화면에 보이기 위해 객체 기술 정보를 참조하여 장면을 구성한 후 화면에 드로잉한다.

그림 2에서 A/V Decoder는 Native 코드로 구현하고, JNI Export Manager, Java Scene Manager, Drawing Manager는 자바 코드로 구현한다. 디코딩은 제한된 시간안에 결과를 반환하여야 하므로 절대적인 시간을 필요로 하고, 비디오가 화면에 보이기 위해서 그래픽 라이브러리가 지원되어야 하므로 이와 같은 구조로 미디어 플레이어는 구현한다[6, 7].

렌더링에서 장면 구성 모듈은 전체 플레이어의 성능저하를 가져오진 않는다. 그러나 장면 구성 모듈을 자바 코드로 구성 하면, 자바의 단점인 실행시간 저하로 인한 렌더링 속도를 상당히 지연시킨다[3]. 따라서 자바 언어로 구현하는 모듈은 Native의 데이터 타입을 자바의 객체타입으로 릴리즈 시키는 JNI Export Manager와 실제로 드로잉하는 Drawing Manager로 한다. Java Scene Manager는 Native 코드로 구현하는 것이 전체 재생시간의 효율을 가져온다.

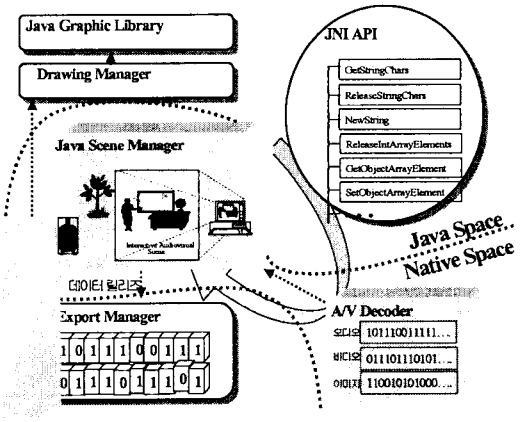

그림 3. Native 영역에 추가한 렌더링 흐름

그림 3은 Na ve 코드에 추가한 렌더링 모듈 흐름을 보인다. A/V Decoder는 디코딩 된 스트림을 Java Scene Manager에게 전달하여 장면 구성 정보를 이용하여 장면을 구성한다. 이는 Native

코드로 구현하여 성능을 향상시킬 수 있다. 구성한 장면 스트림을 자바 객체 타입으로 변환하여 화면에 드로잉한다. 장면 스트림은 자바 그래픽 라이브러리가 읽어 화면에 보이기 전의 스트림 형식이므로 Native 코드에서 전달받는 스트림의 양이 줄어든다. 장면 구성정보, RGB 칼라 변환, 객체 전송 정보등의 데이터는 Native 코드에서 장면을 구성하여 전달하므로 더 이상 필요없는 정보이므로 JNI Export Manager에게 전달하지 않는다. 자바 언어 영역이나 Native 언어 영역에서 JNI API를 호출하는 시간은 플레이어의 전체 성능을 저하시킨다. 따라서 JNI API를 통하여 데이터를 인터렉션하는 양이 작아야 한다.

Java Scene Manager를 Native 코드에 추가하여 렌더링 함으로써, 장면 구성을 위한 데이터는 파싱되어 더 이상 필요 없는 정보가 된다. JNI Export Manager는 장면 구성을 위한 데이터를 제외한 드로잉할 데이터만 자바가 읽어들일 수 있는 객체타입으로 생성한다. 따라서 JNI API를 통한 인터렉션 양이 줄어들어 스트림의 전체 재생 시간을 줄인다.

4. 렌더링 성능 분석

JNI를 이용한 미디어 플레이어는 표 1과 같은 렌더링 모듈 성능 비교를 보인다. 1번 스트림 파일에서 전체 재생시간은 21.99초 소요된다. 렌더링 모듈의 JNI Export Manager는 6.58초, Java Scene Manager는 3.45초 Drawing Manager는 3.07초로 렌더링에 소요되는 시간이 전체의 약 60%이다.

Java Scene Manager가 자바 언어로 구현되어서, 즉 자바 영역에서 처리하여 렌더링 처리 시간의 지연을 초래할 뿐만 아니라, JNI Export Manger가 자바 객체 타입으로 받아와야 하는 데이터가 그만큼 많아지므로 두 모듈이 전체 재생시간을 저하한다.

표 1. 렌더링 모듈 성능 비교 (단위:초)

스트림 파일	전체 재생 시간	JNI Export Manager	Java Scene Manager	Drawing Manager
1	21.99	6.58	3.45	3.07
2	27.59	6.14	2.87	3.22
3	17.57	5.84	2.88	2.18
4	14.37	6.05	3.13	2.29
5	123.98	17.45	23.18	16.46
6	132.58	16.78	27.44	18.53
7	120.51	18.36	24.29	15.98

Java Scene Manager를 Native 영역에서 처리하면 JNI API를 호출할 빈도수가 줄어들고, Native 코드로 파싱 하므로 처리속도를 향상시킨다.

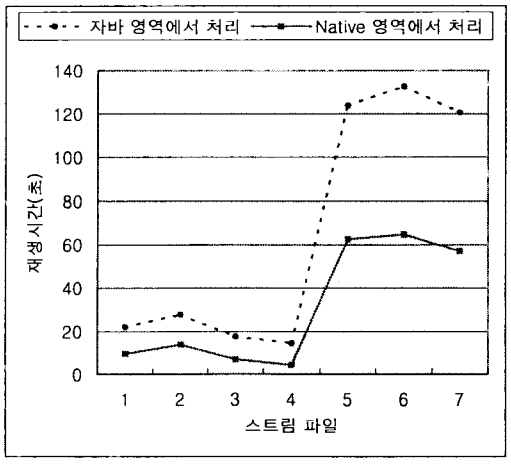

그림 4. 두 영역에서 처리한 재생시간 비교

그림 4는 두가지 영역에서 처리한 렌더링 모듈을 비교한다. 점선은 Java. Scene Manager를 자바 영역에서 처리한 스트림 파일의 전체 재생시간을 보이고, 실선은 Native 영역에서 처리한 스트림 파일의 전체 재생시간을 보인다. Native 영역에서 처리한 전체 재생시간은 자바영역에서 처리한 전체 재생시간의 약 50%가 줄어들었음을 그림 4에서 볼 수 있다.

5. 결론

미디어 스트리밍은 미디어를 실시간으로 재생하려면 대역폭과 스트리밍이 배포되는 네트워크의 품질에 큰 폭으로 의존할 수밖에 없다. 본 논문은 네트워크에서의 제한사항 때문에 발생하는 지연시간을 JNI를 이용한 렌더링 기법으로 향상시켜 고품질의 프리젠테이션을 보여준다. 렌더링 모듈을 Native 언어로 처리하는 영역과 자바 언어로 처리하는 영역으로 분리한다[8]. 렌더링 모듈에서 객체 정보를 이용하여 장면구성 및 RGB 변환을 Native 영역으로 처리하여 JNI API 호출시간을 줄인다.

[참고문헌]

[1] Fitzek, F.H.P. and Reisslein, M, "A prefetching protocol for continuous media streaming in wireless environments", *IEEE J. Select. Area Communication*, vol. 19, pp. 2015 2028, Oct. 2001.

[2] Microsoft Corporation, *Inside Windows Media 인터넷 방송을 위한 스트리밍 기법의 모든 것*, Com & Books, 2000년 6월

[3] T. Lumpp, G. Gruhler, and W. Kuchlin, "Virtual Java Devices Integration of fieldbus based system in the Internet," *In Proc. IEEE Int. Conf. Industrial Electronics, Control and Instrumentation (IECON '98)*, Aachen, Germany, Sept. 1998.

[4] D. Buhler and G. Nusser, "The Java CAN API-a Java gateway to field bus communication," *In Proc. IEEE Int. Workshop on Factory Communication Systems*, 2000

[5] O. Avaro, P. Chou, A. Eleftheriadis, C. Herpel, C. Reader, J. Signes, "The MPEG-4 Systems and description Languages: A Way Ahead in Audio visual information Representation," *Signal Processing: Image Communication (Special issue on MPEG-4)*, 1997.

[6] A. Eleftheriadis, "The MPEG-4 System and Description Languages: From Practice To Theory," *In Pro. IEEE Int. Conf. Circuits and Systems ISCAS '97*, Hong Kong, June 1997.

[7] Y. Shin and S. Kim, "A MPEG-4 Media Presenter on Real-Time OS," In *Pro. Parallel and Distributed Processing Techniques and Applications Int. Conf.* Las Vegas, USA, June. 2000.

[8] Rob Gordon, *Essential JNI: Java Native Interface*, Prentice Hall PTR, 1998