http://dx.doi.org/10.7236/JIWIT.2012.12.1.157
JIWIT 2012-1-20

안드로이드 플랫폼에서 온라인 SVC 스트림을 재생하는 비디오 재생기의 설계 및 구현

Video Player for Online SVC Stream in Android Platform

황기태*

Kitae Hwang

요 약 본 논문은 안드로이드 플랫폼에서 온라인 SVC 스트림을 실시간으로 재생할 수 있는 SVC 재생기를 구현한 사례를 소개한다. SVC(Scalable Video Coding)는 한 번의 비디오 인코딩으로 프레임율, 비디오 크기, 화질 등을 선택적으로 재생할 수 있는 확장성을 가진 비디오 인코딩 기법이다. 본 논문에서는 JSVM 오픈 소스를 이용하여 SVC 디코더를 native 형태로 구현하고 자바로 안드로이드 UI를 개발한 뒤 이를 연결하여 SVC 재생기를 구현하였다. SVC 인코딩 시스템을 오프라인으로 구축하고, 온라인 SVC 스트리밍을 실험하기 위해 SVC 스트리밍 서버를 구축하였으며 구현된 SVC 재생기를 실제 모토로이 폰에 탑재하여 온라인 스트리밍에 따른 성능을 평가 분석 하였다. 성능 평가 결과 SVC 재생기는 온라인상에서 QCIF 크기로 10fps의 SVC 비트 스트림을 재생하는데 지터 등의 문제가 없는 것으로 평가되었다.

Abstract This paper introduces an implementation of SVC player which runs on Android platform and can play SVC video stream on line from SVC video server. SVC(Scalable Video Coding) is a scalable video encoding technique which supports three scalability such as temporal scalability, spatial scalability, and quality scalability. To implement the SVC player on Android, we implemented a SVC decoder using JSVM open source written in C/C++ as a native part on Android and developed Android UI in Java. Also we built an SVC encoding system off line and an SVC streaming server to conduct on-line SVC streaming experiments. Finally, after we installed the SVC player developed in this paper on Motoroi mobile phone, we evaluated and analyzed on-line streaming performance of the SVC player. The result showed that the player worked well and it had no jitter in streaming with the size of QCIF and 10fps from a fully encoded SVC video source.

Key Words : SVC, Video Streaming, Android, Video Player

Ⅰ. 서 론

비디오 코딩과 비디오의 전송 분야에 있어 확장성(Scalability)은 최근 10 년 동안 주요한 연구 주제가 되어 왔다[1,2,3]. 현재 TV, PC, 모바일 단말기 등 다양한 비디오 단말기 들은 다양한 비디오 처리 능력과 스크린의 크기를 보이고 있으며 이들 연구의 주요 관심사는 한 번의 비디오 코딩으로 성능과 스크린 크기를 서로 달리하는 모든 단말기 상에서도 인코딩된 비디오 스트림이 잘 출력되도록 하는 확장성을 개발하는 것이었다.

*종신회원, 한성대학교 컴퓨터공학과
접수일자 2011.12.13, 수정완료 2012.1.18
게재확정일자 2012.2.10

Received: 13 December 2011 / Revised: 18 January 2012 / Accepted: 10 February 2012
*Corresponding Author: calafk@hansung.ac.kr
Dept. of Computer Engineering, Hansung University, Seoul, Korea

2003년 ITU-T VCEG 팀과 MPEG video group으로 구성된 Joint Video Team(JVT)이 H.264/AVC의 Annex G 형식으로 새로운 확장성을 가진 비디오 코딩 표준인 Scalable Video Coding(SVC)[4]를 개발하였다. SVC는 H.264/AVC의 확장으로서 시간 확장성(Temporal Scalability), 공간 확장성(Spatial Scalability), 화질 확장성(Quality Scalability)를 지원하는 효율적인 비디오 코딩 표준이다. 즉, SVC로 인코딩된 비디오 신호는 하나의 기본 계층(base layer)과 여러 개의 향상 계층(enhancement layer)로 표현된다. 향상 계층은 기본 계층만을 디코딩하였을 때에 비해 초당 프레임 수나, 프레임의 크기, 비디오의 화질 등을 개선할 수 있다. 다른 관점에서 보면 계산 성능이 열악하거나 스크린이 작은 비디오 단말기의 경우 기본 계층만을 디코딩하든지 아니면 향상 계층 중 처리 가능한 범위의 하위 향상 계층만을 추가적으로 디코딩하면 된다.

현재 SVC에 관한 많은 연구들은 완전 인코딩된 SVC 스트림(fully encoded SVC stream)으로부터 타켓 단말기의 상황이나 네트워크 대역폭의 변화에 적절히 대응하여 전송 비트스트림을 추출하고 전송하는데 집중하고 있다[5]. 또한 최근 들어 SVC 표준을 상업적으로 개발하는 응용 사례들이 늘어나고 있으며 WIFI, 3G 등 무선통신망(cellular network)의 응용 시스템에 적용하려는 노력이 진행되고 있다[6,7].

본 논문은 최근 시장에서 많은 관심을 받고 있는 안드로이드 플랫폼[8]에서 SVC 재생기를 설계 구현한 사례를 소개한다. 기존의 많은 연구들은 오픈 소스로 제공되는 PC용의 SVC 레퍼런스 코드인 JSVM[9]을 기반으로 하며 주로 시뮬레이션을 이용하여 왔다. 본 논문에서는 JSVM에 주어진 SVC 디코더 부분을 수정하고 RTSP와 RTP를 이용하여 스트리밍 서버로부터 SVC 스트림을 받아 재생하는 안드로이드용 SVC 재생기를 구현하였다.

본 논문은 다음과 같이 구성된다. 2장에서는 SVC 스트리밍 시스템을 소개하고 3장에서는 SVC 재생기를 시험할 수 있는 테스트베드 스트리밍 시스템을 구축한 내용을 소개하며, 4장에서는 안드로이드 플랫폼 상에 SVC 재생기를 구현한 내용을 상세히 설명한다. 5장에서는 구현된 SVC 재생기의 성능을 평가하고 6장에서 결론을 맺는다.

II. JSVM을 이용한 SVC 스트리밍 시스템

1. SVC 인코딩

SVC에서 하나의 프레임은 하나의 기본 계층과 여러 개의 향상 계층으로 인코딩된다. 기본 계층은 최소한의 공간 계층과 화질 계층의 데이터로 인코딩되며 모든 향상 계층은 디코딩시에 반드시 기본 계층과 낮은 확장성을 가진 향상 계층에 의존적이다.

SVC에서 확장 구조는 공간 확장성(D), 화질 확장성(Q), 시간확장성(T) 등 3 개의 요소로 표현 된다. 그림 1,2[10]을 참고하면 공간 확장성이란 인코딩된 비디오 속에 QCIF, CID, 4CIF, TV 급 등 모든 비디오 스크린의 크기를 선택할 수 있도록 인코딩되어 있음을 뜻하며, 화질 확장성이란 SNR(Signal to Noise Raio)의 확장성을 가지고 다양한 화질을 선택할 수 있도록 코딩된 것을 뜻한다. 시간 확장성이란 프레임율인 fps(frames per second)를 선택할 수 있도록 다양한 코딩되어 있음을 뜻한다.

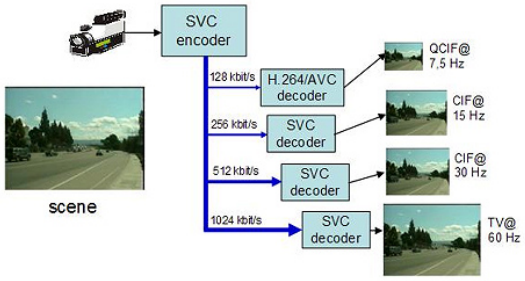

그림 1. SVC 스트리밍 개념
Fig. 1. Concept of SVC streaming

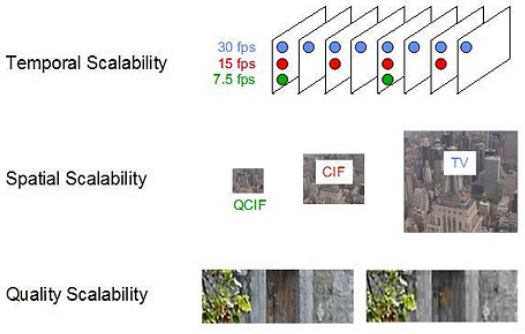

그림 2. SVC의 3 가지 확장성
Fig. 2 . Scalability of SVC

그림 3은 T=(0~3)인 확장성을 가지고 인코딩된 비디오의 계층을 보여 준다. 그림에서 화살표는 의존성을 표시한다. 예를 들면 (D=0, Q=1) 계층은 (D=0, Q=0) 계층에 의존함을 의미한다. (D=0,Q=0)인 계층은 기본 계층이다. 만일 SVC 재생기가 T=3인 확장성을 지원하고자 한다면 (D=0,Q=0)의 기본 계층에서 T=0,1,2,3의 모든 코딩된 비디오가 필요하다. 그러나 만일 T=2인 확장성만 지원하고자 한다면 T=0, T=1 P층의 코딩된 비디오가 모두 필요하다.

만일 SVC 재생기가 T=0인 확장성만 지원한다고 하면, (D=0,Q=1) 계층은 (D=0,Q=0) 계층의 인코딩된 비디오 데이터가 반드시 필요하며 화질을 개선한다. (D=1,Q=1) 계층은 (D=1,Q=0), (D=0,Q=1), (D=0, Q=0) 계층 코딩 비디오가 모두 필요하다.

access unit이란 디코딩을 통해 하나의 프레임을 만들어 내기 위해 상호 의존적이며 필요한 계층들의 집합을 말한다. 그림 4는 가능한 access unit의 사례이다.

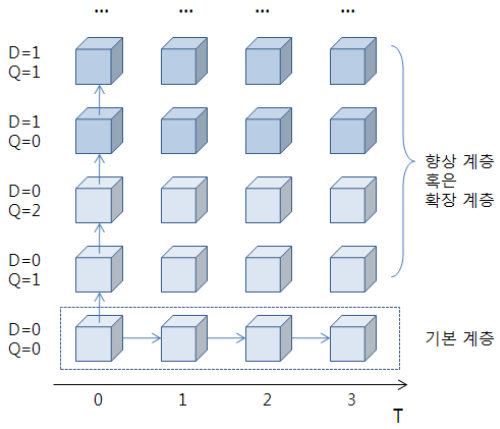

그림 3. SVC로 코딩된 비디오의 계층 사례
Fig. 3. Layers of Sample SVC encoded Video

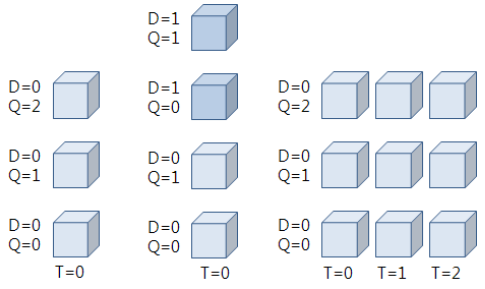

그림 4. 가능한 access unit의 사례
Fig. 4. Possible access units

2. SVC 비트스트리밍 구조

SVC 비트스트림은 하나 이상의 Network Abstraction Layer(NAL) 유닛으로 구성되며 각 NAL 유닛들은 "0x00 00 00 01"값의 4 바이트에 의해 분리된다. NAL 유닛은 4바이트의 헤더와 데이터인 페이로드(payload)로 구성되며 페이로드에 인코딩된 데이터가 저장된다. NAL 유닛은 하나의 계층에 해당하는 인코딩된 데이터를 가지고 있으며, 하나의 원본 소스 프레임을 인코딩하면 기본 계층 NAL과 여러 개의 향상 계층 NAL로 구성된다.

NAL 유닛의 헤더에는 NAL 유닛 타입 정보가 삽입되며 크게 두 가지 종류의 타입이 있다. 첫째, 배치 정보를 담은 Configuration NAL 유닛은 페이로드에 각각 SEI(Sequence Enhancement Information), 혹은 SPS(Sequence parameter Set), PPS(Picture Parameter Set) 정보를 담는다. 나머지 NAL 유닛은 모두 페이로드 압축된 비디오 정보를 담는다. SVC로 인코딩된 NAL 유닛 헤더에는 (D,T,Q) 값이 기록되어 있다.

3. JSVM

IOS/IEC의 MPEG와 ITU-T의 VCEG 그룹의 연합인 JVT(Joint Video Team)에서 SVC 표준화 과정에서 레퍼런스 소프트웨어로 개발한 것이 JSVM(Joint Scalable Video Model)[9]이다. JSVM은 C++로 작성되어 있고 모든 소스가 공개되어 있다. JSVM은 크게 SVC 인코더, SVC 디코더, SVC 비트스트림으로부터 필요한 계층의 비트스트림을 추출하는 비트스트림 추출기, SVC 인코딩 비디오의 NAL 유닛에 관한 정보 보기 등을 위한 소프트웨어 모듈을 제공한다.

III. SVC 스트리밍 시스템 구성

1. 시스템 구성

본 논문에서는 SVC 재생기의 개발 및 테스트를 위해 그림 5와 같은 SVC 스트리밍 시스템을 구축하였다.

본 시스템의 구축을 위해 JSVM을 부분적으로 활용하였다. SVC Encoder는 YUV 파일로부터 SVC 인코딩에 필요한 파라미터를 사용자로부터 받아 기본 계층과 여러 개의 향상 계층을 가진 완전인코딩 SVC 스트림 파일(SVC fully encoded file)을 생성한다. SVC Encoder는 JSVM에서 제공되는 인코딩 실행 모듈을 그대로 이용하였다.

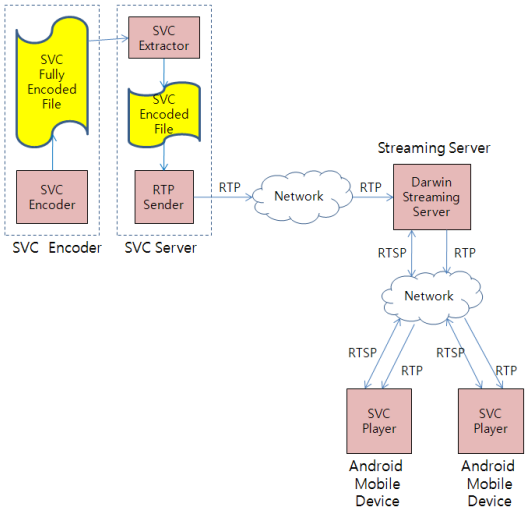

그림 5 SVC 스트리밍 시스템 구축
Fig. 5. Established SVC streaming system

SVC Server는 SVC Player에게 전송하기 위해 필요한 SVC 비트스트림을 전송하는 서브시스템이다. SVC Extractor는 완전인코딩 SVC 스트림 파일로부터 전송하고자 하는 계층을 뽑아내어 SVC encoded file을 생성한다. RTP Sender는 실시간으로 전송율에 맞추어 Streaming Server로 실시간 전송을 시행한다. SVC Extractor는 JSVM에서 제공하는 것를 그대로 활용하였으며 RTP Sender는 새로 작성하였다. JSVM SVC 디코더를 활용하여 SVC 재생기를 구현하였으며, SVC 스트리밍 시스템을 구축하기 위해 JSVM SVC 인코더를 활용하고 SVC 비트스트림 추출기를 부분적으로 수정하였다.

스트리밍 서버(Streaming Server)는 Darwin Streaming Server(DSS)로 구축하였다. DSS는 스트임 입력단에서 소스로 연결되는 장치와 RTSP 연결을 실행하지 않으며 RTP 패킷을 계속받고 있다가 클라이언트의 연결이 들어오면 그때 스트리밍을 시작한다. DSS는 하나의 스트림 입력을 다수의 클라이언트로 동시에 전송한다. DSS는 클라이언트와 연결될 때 RTSP 프로토콜로 연결되며 연결이 이루어지면 스트림 소스로부터 입력되는 RTP 패킷을 그대로 클라이언트로 전송한다.

DSS는 SVC 인코딩된 파일의 구조를 이해할 수 없기 때문에 별도로 SVC 스트리밍 서버를 구현하였다.

SVC Player는 본 논문의 핵심으로서 DSS와 RTSP로 접속하고 RTP로 SVC NAL 유닛들을 실시간으로 전달받는다. 그리고 이들을 적절히 디코딩하여 비디오 프레임을 만들고 안드로이드 모바일 단말기에 재생한다. SVC Player는 구현하기 위해 SVC NAL 유닛들로부터 비디오 프레임을 생성하는 디코딩 작업은 JSVM에서 제공하는 라이브러리를 활용하였다.

Ⅳ. SVC 스트리밍 재생기 구현

1. 구현 범위

본 논문에서 구현된 재생기의 설계 원칙은 다음과 같다.

첫째, 스트리밍 서버나 SVC 릴레이 서버로부터 온라인 스트림 방식으로 전송되는 비트 스트림을 재생하는 SVC 비디오 재생기를 구현한다.

둘째, RTSP와 RTP를 기반으로 어떤 스트리밍 소스와 연결되어도 재생할 수 있게 하지만 중간에 SVC 비트스트림이 프레임 율, 비디오 화면 크기, 해상도 등이 변경되는 것에 대해서는 지원하지 않는다.

셋째, 안드로이드 단말기로 제한한다.

넷째, 처리 속도의 보장을 위해 사용자 입출력은 안드로이드 API를 사용하고 나머지는 native C++로 작성한다. 안드로이드 API는 기본적으로 RTP, RTSP 통신을 위한 라이브러리를 지원하지 않으므로 이들은 모두 C++로 작성한다.

2. 재생기 소프트웨어 구조

본 논문에서 구현된 SVC 재생기는 그림 6과 같은 소프트웨어 모듈로 구성된다.

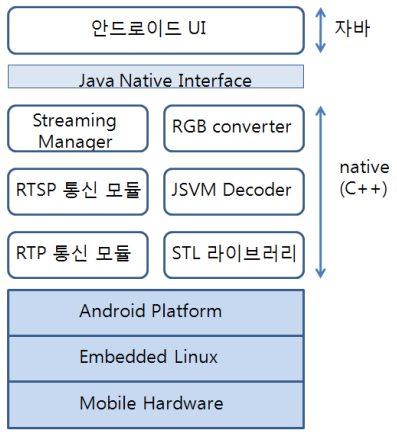

그림 6. 안드로이드 UI와 native 구성
Fig. 6. Android UI and native part

모바일 단말기의 성능 한계를 다소 극복하기 위해 사용자 인터페이스만 안드로이드 API를 이용하여 자바로 작성하고 나머지는 모두 Java Native Interface를 이용하여 C++로 구현하였다. native 모듈은 스트리밍 서버와 초기 연결을 위한 RTSP 통신 모듈과 H.264/SVC NAL 유닛을 실시간으로 전송받는 RTP 모듈, 그리고 NAL 유닛들로부터 디코딩하여 YUV 형식의 프레임을 생성하는 JSVM Decoder 모듈, YUV 프레임을 안드로이드 UI를 이용하여 출력할 수 있도록 RGB로 변환하는 RGB Converter, 그리고 이들 모듈을 초기에 생성하고 통제하는 Streaming Manager 모듈로 구성된다.

3. 재생 과정과 스레드

구체적으로 스트리밍 서버로부터 전송받는 NAL 유닛들이 처리되는 일련의 과정은 그림 7과 같다. 비디오 출력을 위한 안드로이드 UI는 SurfaceView를 사용하였으며 SurfaceView에 프레임 데이터를 공급하는 BitmapBuffer는 RGB로 구성된 프레임 버퍼이다.

들어오는 NAL 유닛들을 디코딩하여 최종적으로 SurfaceView에 출력하기까지의 과정에는 총 4 개의 독립된 스레드가 작동한다.

T1 스레드는 RTP 통신 모듈의 코드를 실행하는 일명 ReceiverThread로서 들어오는 NAL 유닛들을 순서대로 받아 해석하지 않고 그대로 NAL Units Buffer에 저장하는 스레드이다. T1 스레드는 RTP통신 모듈 내에서 작동한다.

T2 스레드는 NAL Units Buffer에서 NAL 유닛이 있으면 항상 가져와서 NAL 유닛을 분석하고 페이로드를 꺼내어 이를 해석하고 변형하여 JSVM SVC Decoder내의 내부 버퍼에 쌓아 놓는다. 보통의 경우는 쌓아 놓고 돌아가지만 디코딩이 가능한 만큼의 NAL 유닛이 도착한 경우 바로 디코딩을 실행한다. 디코딩된 결과는 YUV 형식의 하나의 프레임로서 DecodingBuffer에 저장된다.

그림 7에서 볼 수 있는 바와 같이 SVC 비트스트림에서 하나의 프레임을 구성하고 있는 기본 계층 NAL 유닛과 필요한 향상 계층 NAL 유닛들을 모두 받은 후에야 하나의 프레임으로 디코딩이 가능하다. 그러므로 디코딩 전에 이들 NAL 유닛들을 저장하기 위한 임시 공간으로 JSVM SVC Decoder 내부에는 버퍼를 충분히 가지고 있다.

JT3는 안드로이드 UI에 의해 생성되는 자바 스레드이다. JNI를 통해 C++로 작성된 RGBConverter 모듈을 실행하며 DecoderBuffer에 들어 있는 YUV 형식의 프레임을 RGB 형식으로 변환하여 BitmapBuffer에 QUEUE 저장한다.

JT4 또한 안드로이드 UI에 의해 생성되는 자바 스레드이다. BitmapBuffer에 저장된 RGB 프레임들을 초당 프레임수에 맞추어 하나씩 순서대로 SurfaceView에 출력한다.

T1과 T2는 모두 POSIX 스레드 라이브러리를 이용하여 작성되었으며 JT4는 자바의 java.lang.Thread를 상속받아 작성되었다. JT3 역시 자바의 java.lang.Thread를 상속받아 작성되었으나 native로 작성된 RGBConverter 코드까지 실행되므로 RGBConverter 코드는 DecodedBuffer 버퍼에 대한 동기화를 위해 POSIX 스레드 동기화 라이브러리를 이용하여 작성되었다.

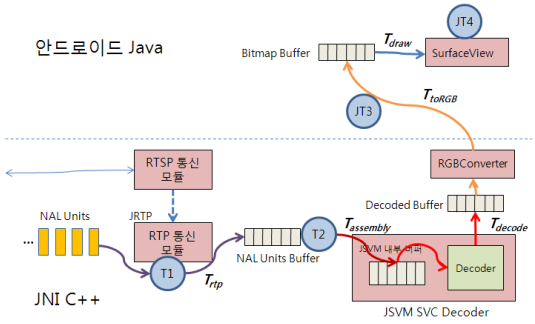

그림 7. 스레드와 스트림의 처리
Fig. 7. Threads and processing of SVC Stream

NAL Units Buffer나 DecodedBuffer, BitmapBuffer들은 모두 여러 개의 버퍼 공간을 가진 가변 길이 큐로 동작하고 버퍼를 중심으로 양단의 스레드는 producer-consumer 방식으로 서로 동기화된다.

4. JSVM 디코더 포팅

JSVM 코드는 표준 STL(Standard Template Library)을 사용하고 있지만 안드로이드 플랫폼이 탑재된 리눅스는 STL 라이브러리를 사용하는 C/C++ 소스를 컴파일할 수 없는 큰 문제가 있다. STL 라이브러리를 지원하지 않기 때문이다.

이 문제를 해결하기 위해 STLport라고 불리는 라이브러리를 이용하였다. STLport는 기존의 컴파일러에서 제공되는 STL의 버그 및 속도상의 문제를 해결하기 위하여 오픈 소스로 제공되는 라이브러리이다.

STLport 또한 안드로이드 플랫폼에서 사용하기 위해서 안드로이드 환경에 맞도록 포팅된 STLport 라이브러리를 [11]에서 다운받아 사용하였다.

JSVM 디코더 부분을 STLport 라이브러리와 함께 안드로이드에서 제공하는 NDK 툴을 이용하여 컴파일하였다.

5. YUV 스트림을 RGB 스트림으로 변환

YUV 스트림의 각 픽셀은 Y, U, V의 요소 값으로 이루어져 있다. 안드로이드에서는 화면에 이미지를 출력하기 위해서는 RGB 기반의 픽셀 정보로 구성된 비트맵을 준비하여야 한다. YUV 비트맵을 RGB 비트맵으로 바꾸는 여러 공식이 있다. 이 변환 수식은 대부분 실수 연산을 포함한다. 현재 안드로이드 단말기는 대부분 ARM CPU를 탑재하고 있으며 ARM CPU는 실수처리기를 CPU 내부에 따로 가지고 있지 않기 때문에 실수 연산은 모두 정수 연산을 이용하는 소프트웨어로서 처리된다. 그러므로 실수 연산을 가진 RGB 변환 공식을 사용하면 우리가 실험을 통해 확인하였지만 RGB 변환에 많은 시간이 소요된다. RGB변환 공식을 정수 연산만으로 구성할 수 있지만 이 경우 변환된 RGB로 출력된 이미지는 원본과 색상에 많은 차이를 나타내므로 본 연구에서는 아래와 같은 변환 공식을 사용한다.

R = Y + 1.40200*V

G = Y - 0.34414*U - 0.71414*V

B = Y + 1.77200*U

V. 실험 및 성능 측정

1. 실험 목적

우리는 현존하는 안드로이드 단말기로 다양한 SVC 스트리밍 재생 실험을 진행해보았지만 현재 CPU의 처리 능력 상 초당 30프레임의 SVC 스트림을 처리하기는 어렵다. 현재 안정적인 상황으로 스트리밍이 가능한 대표적인 경우는 7.5fps, QCIF 크기의 영상에 화질 확장성이 선택된 경우이다.

본 논문의 실험은 SVC 재생기에서 처리되는 각 부분의 처리 시간을 평가하여 현재의 대표적인 안드로이드 단말기에서 어느 정도의 시간 성능을 내는지 이해하는데 있으며, 부분별로 소모되는 시간 정도를 이해하는데 있다.

2. 시간 모델

본 논문에서는 들어오는 NAL 유닛으로부터 안드로이드 단말기의 화면에 스트림이 전송되는 시간을 분할하여 각 부분별로 처리되는 시간을 평가한다. 이 시간은 다음과 같으며 그림 7을 참고하면 된다.

(1) T_{rtp}

이 시간은 T1의 의해 소모되는 시간으로서 RTP 통신 모듈에 의해 도착하는 하나의 NAL 유닛을 NAL Units Buffer에 저장하는데 걸리는 시간이다.

(1) $T_{assembly}$

이 시간은 T2의 의해 소모되는 시간으로서 JSVM SVC 디코더가 받은 NAL 유닛을 디코딩하기 위한 전처리 과정을 진행하고 한 프레임의 디코딩에 필요한 만큼의 NAL 유닛을 모으는 과정에 소모되는 시간이다.

(2) T_{decode}

이 시간은 T3의 의해 소모되는 시간으로서 JSVM 내부 버퍼에 임시 저장하고 전처리 과정을 거친 NAL 유닛들을 디코딩하여 YUV 형식의 한 프레임으로 만들어 DecodeBuffer에 저장하는데 소요되는 시간이다.

(3) T_{toRGB}

JT3의 의해 소모되는 시간으로서 Decoded Buffer에서 YUV 형식의 한 프레임을 RGB 프레임으로 변환하고 Bitmap Buffer에 저장하는데 소요되는 시간이다.

(4) T_{draw}

JT4의 의해 소모되는 시간으로서 Bitmap Buffer에서 하나의 RGB 프레임을 가지고 와서 실제 SurfaceView에 그리는 시간이다.

2. 실험 및 결과

SVC 재생기의 작동을 실험하기 위해 SVC 인코더를 이용하여 완전인코딩으로 SVC 인코딩된 비디오를 생성하였다. 그리고 SVC extractor를 이용하여 전송할 SVC 비트스트림을 추출하여 파일로 저장하였다. 추출된 SVC 비트스트림은 7.5fps, QCIF의 영상 크기이다. 테스트가 진행되면 RTP Sender는 프레임율에 따라 SVC 비트스트림의 전송을 시작한다. 그림 8은 구현된 SVC 재생기가

실행되는 화면을 보여주며 화면의 동영상은 JSVM에 의해 테스트용으로 주어진 것이다. 그림 8에서 영상이 왼쪽에 치우쳐 있는 것은 QCIF 크기의 작은 영상을 데모 구현의 편리를 위해 둔 것이다.

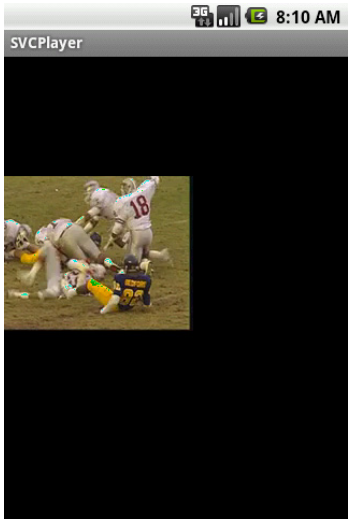

그림 8. 실행 영상 캡쳐
Fig. 8. Captured Running SVC Video streaming

실험을 위해 사용한 안드로이드 단말기는 표 1과 같은 사양을 가진 모토로이를 이용하였다. 무선 네트워크는 54Mbps의 대역폭을 가진 WiFi 무선 네트워크에서 진행되었다. 실제 실험을 통해 측정한 시간 값은 표 2와 같다. 각 시간은 100번 이상의 반복 실험치의 평균값이다. 각 시간을 측정할 때 사용된 스레드 외에 다른 스레드를 블록시킨 상태에서 실험을 진행하였기 때문에 다른 스레드로 인한 대기 시간이나 스레드 동기화로 인한 지연시간을 포함하지 않으며 순수한 작업 시간만을 포함한다.

표 1. 모토로이 단말기의 하드웨어 사양
Table 1. Hardware spec. of Motoroi Phone

구성 요소	사양
CPU	ARM CPU TI OMAP3430,600MHz
Memory	RAM 256MB ROM 384MB
디스플레이	3.7인치 WVGA(480x854) 16M 컬러 TFT LCD
WiFi	802.11 b/g

표 2. 시간 측정 결과
Table 2. Measured time of each time part

시간 구분	소요시간(ms)
T_{rtp}	0.5
$T_{assembly}$	52.5
T_{decode}	2
T_{toRGB}	25
T_{draw}	2.5
합	82.5

본 실험의 결과, JSVM SVC 디코더에서 필요한 NAL 유닛을 모으고 이들 유닛을 디코딩하는 시간은 $T_{assembly}$ + T_{decode}이며 전체 시간의 66%로서 많은 시간이 소모되고 있음을 보여준다. 또한 CPU가 실수 연산 처리 장치를 가지고 있지 않기 때문에 YUV 비트맵을 RGB 비트맵으로 변환하는 T_{toRGB} 역시 전체 시간의 30%로서 큰 비중을 차지한다. 만일 QCIF(176x144) 대신 CIF(352x288) 크기라면 더 많은 시간이 소요될 것이다. 그러나 이 시간은 추후 모바일 단말기에 사용되는 CPU가 실수 연산 처리 장치를 내부적으로 가지게 되면 해결될 수 있을 것이다. RTP 패킷을 받는 수신 시간이나 LCD에 출력하는데 걸리는 시간은 비교적 작아 전체 성능에 큰 요소가 되지 않는다.

현재 실험 결과만을 보면 QCIF 크기의 한 프레임이 처리되는데 걸리는 시간은 약 83ms로서 현재와 동일한 화질 확장성을 유지할 때 QCIF 크기로 10fps의 SVC 비트스트림을 재생하는 데는 지터 등과 같은 문제는 없다.

본 논문에서는 구현된 SVC 재생기를 검증하고 SVC 스트리밍을 재생하는데 있어 시간 요소를 구분하고 각 시간 요소들이 어느 정도의 오버헤드를 가지는지 대략적인 직관을 얻고자 함이다. 본 논문의 실험 결과는 각 시간 요소와 SVC 스트림의 특성의 상관성을 고려한 충분한 결과는 아니지만 단말기가 수용할 수 있는 비트스트림에 대해서 상대적인 시간 오버헤더를 분석하는 결과를 얻었다. 다양한 스트림의 특성과 재생기의 각 시간 요소와의 상관 관계와 WiFi가 아닌 3G 네트워크, NAL 유닛의 오류가 발생하는 경우, 다양한 단말기 장치들에 따라 재생기의 각 시간 요소의 관계 등은 추후 연구를 통해 실험할 예정이다.

VI. 결론

본 논문은 SVC 비디오 스트림을 온라인상에서 실시간으로 재생 가능한 안드로이드용 SVC 재생기를 개발한 사례를 소개하였다. SVC 스트리밍 서버를 비롯한 테스트베드를 구축하고 모토롤라 사의 모토로이 단말기에 개발된 SVC 재생기를 실행시켜 작동 검증 및 성능을 평가하였다. 실험 결과 스트림으로부터 NAL 유닛을 모으고 디코딩하는데 소요되는 시간이 전체 시간의 66%를 차지하였으며 단말기는 실수 연산 처리 장치를 가지고 있지 않기 때문에 YUV 비트맵을 RGB로 변환하는데 약 30%의 시간 소모를 보였다. 이 문제는 단말기의 CPU에 실수 연산 처리 장치를 포함시킴으로써 개선 가능하다.

SVC 기술을 상용화하려는 국내외적인 연구가 진행중인 가운데, 본 논문의 의의는 안드로이드용 SVC 재생기의 구체적인 설계 구현 내용을 제시하고 실제 단말기에서 작동하고 성능을 테스트하여 시간을 분석한 점에 있다. 추후 스트림의 특성과 재생기의 각 시간 요소의 상관 관계, 다양한 단말기와 네트워크 상황에서의 성능을 평가하고 실험할 계획이다.

참 고 문 헌

[1] H. Kirchhoffer, H. Schwarz, and T.Wiegand. “CE1: Simplified FGS,” Joint Video Team, Doc. JVT-W090, Apr. 2007.

[2] Schwarz, H.; Marpe, D.; Wiegand, T.; "Overview of the Scalable Video Coding Extension of the H.264/AVC Standard," Circuits and Systems for Video Technology, IEEE Transactions on, vol.17, no.9, pp.1103-1120, Sept. 2007

[3] Jubran, M.K.; Bansal, M.; Kondi, L.P.; "Transmission of Scalable H.264 Coded Video Over Wireless MIMO Systems with Optimal Bandwidth Allocation," ICASSP 2007. IEEE International Conference on, vol.1, pp.I-849-I-852, April 2007

[4] Advanced Video Coding for Generic Audiovisual Services. ITUT Rec. H.264 and ISO/IEC 14496-10 MPEG-4 part 10 Advanced Video Coding (AVC), 2003.

[5] Peng Chen, Jeongyeon Lim, Bumshik Lee, Munchurl Kim, Sangjin Hahm, Byungsun Kim, Keunsik Lee, Keunsoo Park, “A network-adaptive SVC Streaming Architecture," Advanced Communication Technology, The 9th International Conference on, vol.2, pp.955-960, Feb. 2007

[6] Shoaib, M.; Waheed, M.; "Streaming Video in Cellular Networks Using Scalable Video Coding Extension of H.264-AVC," Wireless Communications, Networking and Mobile Computing, 2008. 4th International Conference on, pp.1-4, Oct. 2008

[7] Wiegand, T.; Noblet, L.; Rovati, F.; , "Scalable Video Coding for IPTV Services," Broadcasting, IEEE Transactions on, vol.55, no.2, pp.527-538, June 2009

[8] http://www.android.com

[9] J. Reichel, H. Schwarz, and M. Wien, “Joint scalable video model JSVM-5,” JVT-R202 (Output Document from Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG), Jan. 2006.

[10] http://ip.hhi.de/imagecom_G1/savce/index.htm

[11] STLport, http://www.stlport.org/grammer's Reference, Wiley Press

저자 소개

황 기 태(정회원)

- 서울대학교 컴퓨터공학과 학사
  서울대학교 컴퓨터공학과 석사
  서울대학교 컴퓨터공학과 박사
- 경력
  University of Callifornia, Irvine 방문교수
  University of Florida 방문 교수

<주관심분야 : 모바일 시스템>

※ 본 과제는 한성대학교 연구년 연구비를 지원받았음