일반논문 (Regular Paper)
방송공학회논문지 제30권 제4호, 2025년 7월 (JBE Vol.30, No.4, July 2025)
https://doi.org/10.5909/JBE.2025.30.4.660
ISSN 2287-9137 (Online) ISSN 1226-7953 (Print)

HAP 코덱을 활용한 8K 이상 초고해상도 영상 실시간 재생 최적화 연구

전 지 우[a)], 정 현 민[a)‡]

A Study on Real-Time Playback Optimization for Ultra-High-Resolution Video (Beyond 8K) using HAP Codec

Jiwoo Jeon[a)] and Hyunmin Jung[a)‡]

요 약

본 연구는 8K 이상 초고해상도 영상을 위해 최적화된 HAP 코덱 기반 플레이어 구현 방법을 제안한다. 현재 광범위하게 사용되고 있는 H.264, H.265은 하드웨어 디코딩 시 8K까지만 지원하는 한계가 있으며, 소프트웨어 디코딩은 CPU 부하와 CPU-GPU 간 데이터 전송 오버헤드로 인해 실시간 재생이 어렵다. 이러한 문제를 해결하기 위한 대안으로 HAP(Vidvox에서 개발한 실시간 비디오 코덱) 코덱이 주목받고 있으나, 기존 오픈 소스의 HAP 역시 코덱의 특성을 충분히 고려하지 않아, 초고해상도 영상 재생 시 프레임 드랍이 빈번하게 발생한다. 본 연구에서는 HAP 코덱에 최적화된 버퍼링 전략과 멀티스레딩 디코딩을 통해 이러한 한계를 극복하고자 한다. 제안된 방법은 패킷 버퍼와 텍스처 버퍼를 효과적으로 활용하여 디코딩 과정의 시간 변동성에 대응하며, Windows 네이티브 환경에서 Direct3D API를 활용한 최적화된 구현을 통해 하드웨어 성능을 극대화한다.

Abstract

This study proposes an optimized HAP codec-based player implementation that plays ultra-high-resolution videos over 8K without frame drops. The currently widely used H.264 and H.265 codecs have the limitation that hardware decoding only supports up to 8K resolution, and software decoding makes real-time playback difficult due to CPU load and data transfer overhead between CPU and GPU. The HAP codec has been attracting attention as an alternative to solve these problems, but existing open source HAP players also do not sufficiently consider the characteristics of the codec, and frame drops frequently occur when playing ultra-high-resolution videos. This study aims to overcome these limitations with a buffering strategy and multi-threading decoding optimized for the characteristics of the HAP codec. The proposed method effectively utilizes packet buffers and texture buffers to cope with the temporal variability of the decoding process, and maximizes hardware performance through an optimized implementation using Direct3D API in a Windows native environment.

Keyword : HAP codec, Ultra-high-resolution Video Player

Ⅰ. 서 론

최근 디스플레이 기술이 발전함에 따라 8K를 넘어 12K, 16K 이상의 초고해상도를 지원하는 대형 LED 스크린이 전시, 광고, 공연 등 다양한 분야에서 활용되고 있다. 이러한 초대형 디스플레이는 관객에게 높은 몰입감과 압도적인 시각적 경험을 제공한다. 하지만 이를 위해 고품질의 콘텐츠를 안정적이고 실시간성을 보장하도록 재생하기 위한 기술은 여전히 도전적인 과제로 남아있다. 이를 극복하기 위한 방안으로 8K 초과 영상에 대해서 영상을 작은 크기로 분할한 뒤, 다수의 프로젝터나 디스플레이를 동기화하여 재생하는 다중 동기화 방식이 사용되기도 한다[1][2]. 그림 1은 다중 디스플레이 사례로 LG전자가 두바이몰에 설치한 820장의 OLED 디스플레이를 연결한 디지털 사이니지와[3] 제주도의 전시관인 노형 수퍼마켓의 22대 프로젝터를 활용한 프로젝션 맵핑(최대 14772×1920 해상도)을 보여준다[4]. 그러나 이러한 다중 디스플레이 시스템은 구현이 복잡하고 유지보수가 어려운 문제를 가진다. 본질적으로 이를 해결하기 위해 초고해상도 영상을 단일의 디스플레이 시스템에서 효과적으로 재생하기 위한 기술 연구가 필요하다.

대체로 영상 콘텐츠 플레이어에서 주로 사용되는 비디오 코덱은 H.264/AVC[5], H.265/HEVC[6] 계열의 비디오 코덱이다. 이들은 다양한 압축 기법을 바탕으로 압축 효율을 극도로 높인다. 하지만 이는 디코딩 복잡도를 높여 실시간 디코딩 난이도를 높이고, 고사양의 하드웨어가 요구된다는 점에서 단점을 가진다. 이는 특히 초고해상도의 영상 처리에 있어 치명적인 문제를 가진다. 초고해상도 영상 처리에 있어 기존 비디오 코덱이 가지는 기술적 한계 중 하나는 하드웨어 디코딩의 해상도가 제한적이라는 점이다. 최신 GPU와 전용 하드웨어 디코더는 일반적으로 최대 8K의 해상도까지만 지원한다. NVIDIA의 최신 RTX 40 시리즈 GPU조차 H.265 디코딩을 8K/60fps까지만 지원하며, 그 이상의 해상도는 공식적으로 지원하지 않는다[7][8][9]. 따라서 8K를 초과하는 초고해상도 영상의 경우 CPU에서의 디코딩이 필요한데, 이의 실시간 디코딩은 현실적으로 어렵다. 또한 CPU에서 디코딩을 하더라도 이를 디스플레이하기 위해 GPU로 전송하는 과정에서의 병목이 문제가 된다. 8K 고해상도 영상의 한 프레임은 약 100MB이며, 초당 30 프레임으로 전송하기 위해서는 약 3GB/s의 데이터 전송이 요구된다. 이는 PCIe 3.0의 가용 대역폭 중 상당 부분을 차지하는 비중이며, 16K와 같은 더 높은 해상도의 영상 처리를 고려할 경우 문제가 더욱 심화된다. 이러한 문제는 8K 이상의 초고해상도 영상을 기존 비디오 코덱과 재생 기술로 처리하는 데 근본적인 제약으로 작용하며, 특히 실시간 재생 환경에서 프레임 드랍으로 이어져 부드러운 시청 경험을 방해한다. 프레임 드랍은 초고해상도 영상을 분할하여 다중 PC에서 처리할 경우 디스플레이 간 불일치를 유발하며, 초고해상도 영상에서는 작은 화면에서 감지하기 어려운 미세한 프레임 드랍이 관객에게 쉽게 노출되어, 세심한 관리가 요구된다.

초고해상도 영상 플레이어에 있어 HAP 코덱은 이러한 기술적 제약을 극복하기 위한 적합한 대안이다[10][11]. HAP 코덱은 실시간 영상 재생과 GPU 렌더링에 초점을 맞춰 설계된 코덱으로, 초고해상도 영상의 빠른 디코딩에 적합하다. HAP은 내부적으로 두 개의 압축 기법을 포함한다. 하나는 DXT(DirectX Texture Compression)로 GPU에서 인/디코딩이 가능한 압축 방식이다. DXT는 4×4 픽셀 블록 단위로 압축하여 고정된 압축률을 제공하며, 모든 현대 GPU에서 하드웨어 수준으로 지원된다. 다른 하나는 Google의 Snappy이다. Snappy는 빠른 압축/해제 속도에 중점을 둔 알고리즘으로 멀티코어 환경에서 병렬 처리가 가능하다. GPU 텍스처가 지원하는 최대 해상도인 16K까지의 초고해상도 영상에 대한 하드웨어 디코딩을 지원한다는 점에서 초고해상도 영상에 특히 적합하다. 하지만 현재 오픈 소스로 공개된 HAP 코덱 기반 플레이어는 HAP 코덱의 특성을 충분히 고려하지 않아 초고해상도 영상 재생 시 프레임 드

a) 서울과학기술대학교 스마트ICT융합공학과(Dept. of Smart ICT Convergence Engineering, Seoul National University of Science and Technology)
‡ Corresponding Author : 정현민(Hyunmin Jung)
E-mail: hmjung@seoultech.ac.kr
Tel: +82-2-970-6457
ORCID: https://orcid.org/0000-0001-8216-5842
※ This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.RS-2023-00229330, 3D Digital Media Streaming Service Technology)
· Manuscript June 10, 2025; Revised June 30, 2025; Accepted July 1, 2025.

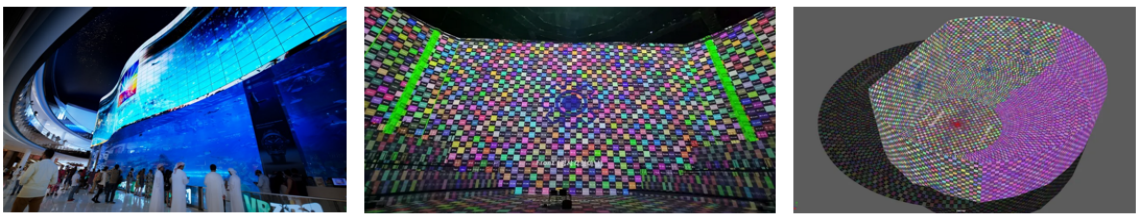

그림 1. 다중 디스플레이를 활용한 초고해상도의 구현의 예 (두바이몰, 노형 수퍼마켓)
Fig. 1. Examples of Ultra-High Resolution Implementation Using Multiple Displays (Dubai Mall, Nohyung Supermarket)

랍이 빈번하게 발생하는 문제를 가진다[10]. 오픈프레임웍스에서 제공하는 HAP 플레이어인, ofxHapPlayer는 프레임 버퍼가 부재하여, 프레임 드랍 빈도가 높고, 복잡한 시킹기능 역시 오버헤드를 높이는 주요한 요인이다. FFmpeg 기반의 HAP 플레이어[12] 역시 프레임 버퍼 크기가 제한적이며, 유사한 문제를 보인다.

본 연구는 이러한 문제점을 해결하기 위해 HAP 코덱의 특성을 분석하고, 이에 적합한 플레이어 구조를 설계하며, 8K 이상의 초고해상도 영상을 프레임 드랍 없이 안정적으로 재생할 수 있는 방법을 제안한다. 특히, 초대형 LED 디스플레이 환경에서 중요한 연속적인 재생 품질과 프레임 동기화를 보장하기 위한 버퍼링 전략을 제안하며, 멀티스레딩 디코딩 시나리오에 중점을 준다. 실험에서는 제안된 HAP 코덱 기반 플레이어가 기존 오픈소스 ofxHapPlayer 대비 우수한 성능을 보였다. 8K 해상도에서 ofxHapPlayer의 6.7% 프레임 드랍률을 0.0%로 완전히 해결하였으며, 고성능 데스크톱 환경에서는 4K부터 16K까지 모든 해상도에서 프레임 드랍 없는 안정적인 재생을 달성하였다. 노트북 환경에서는 12K까지는 안정적인 재생이 가능하였으나, 16K에서는 SSD I/O 성능 제약으로 인한 한계가 확인되었다.

II. HAP Codec 소개

1. HAP 코덱 개요

초고해상도 영상 재생에 있어 코덱의 선택은 시스템 성능과 재생 품질에 결정적인 영향을 끼친다. 2012년 Vidvox에서 개발한 HAP 코덱은 기존 범용 코덱과는 근본적으로 다른 접근 방식을 취한다[11]. 일반적인 비디오 코덱이 압축 효율성 극대화에 중점을 두는 반면, HAP 코덱은 실시간 처리 성능과 GPU 친화적인 구조에 초점을 맞추고 있다. HAP 코덱은 이중 압축 구조를 통해 CPU와 GPU 간의 효율적인 작업 분담을 실현한다. 인코딩 과정은 두 단계로 이루어지는데, 그 중 첫 번째 단계는 원본 이미지를 GPU에서 직접 사용 가능한 DXT으로 압축한다. 이 방식은 4:1에서 6:1의 압축률을 제공하며[13][14], 특히 GPU에서 네이티브하게 지원되는 장점을 가진다. 두 번째 단계는 DXT 압축된 데이터를 Google의 Snappy 알고리즘으로 추가 압축한다. Snappy는 압축 효율보다 인/디코딩 속도에 초점을 맞춘 알고리즘으로 파일 크기를 줄이면서 동시에 빠른 디코딩이 가능하다.

디코딩 과정에서는 이 두 단계를 역순으로 적용하여 영상을 디코딩한다. 먼저 CPU에서 Snappy의 디코딩 과정을 거치는데, 이는 멀티코어 시스템에서의 병렬화가 가능하다. 해제된 DXT 상태의 데이터를 GPU로 전송한다. 12K 해상도 기준 한 프레임의 DXT 형식 데이터 크기는 32~50MB로, RGB 데이터 형태인 200MB에 비해 4~6배 가량 데이터 크기가 작으며, CPU와 GPU 간 데이터 전송에 있어 PCIe 대역폭 사용을 크게 줄인다.

기존 H.264/H.265 코덱과 비교하여, HAP은 각 프레임을 독립적으로 압축하는 인트라(Intra) 코딩 방식을 취하기 때문에 무작위 접근성이 우수하고, 병렬 처리에 유리하다. 또한 H.264/H.265가 하드웨어 디코딩에서 8K의 해상도까지만 지원하는 반면, HAP은 GPU 텍스처 제한에 따라 최대 16K 해상도를 지원한다. HAP은 이에 반해 H.264/H.265에 비해 수 배에서 수십 배 수준으로 압축 효율이 낮으며, 이를 극복하기 위한 SSD와 같은 데이터 처리량이 빠른 고성능 저장 장치가 필수적이다. HAP 코덱은 전시, 공연, 대형 LED 스크린과 같이 압축 효율보다 재생 안정성 확보가 중

요한 환경에 특히 적합하다. 그러나 이러한 이점을 최대화하기 위해서는 HAP 코덱의 특성을 고려한 최적화된 플레이어 구현이 필요하다. 특히 CPU에서의 Snappy 압축 해제, GPU로의 데이터 전송, 그리고 GPU에서의 렌더링 과정이 효율적인 파이프라인화가 요구된다.

2. Snappy 압축 소개

Snappy는 Google에서 개발한 데이터 압축 알고리즘으로, 최대 압축 효율보다는 빠른 압축 및 해제 속도에 중점을 둔 설계 철학을 가진다. 이 알고리즘은 초당 수백 메가바이트의 데이터를 처리할 수 있는 뛰어난 처리 속도를 제공하며, 이는 zlib와 같은 기존 압축 알고리즘 대비 5~10배 빠른 수준에 해당한다. Snappy의 압축률은 일반적으로 1.5:1에서 2:1 정도로 다소 제한적이지만, HAP 코덱과 같이 실시간 처리가 중요한 환경에서는 높은 압축률보다 빠른 속도가 더 중요한 요소로 작용한다. 특히 Snappy는 복잡한 사전이나 예측 테이블을 사용하지 않는 단순한 구현 방식을 채택하여 메모리 사용량을 최소화하고, 독립적인 청크 단위로 데이터를 처리할 수 있어 멀티코어 CPU 환경에서 병렬 압축 해제가 가능하다는 장점을 가진다. HAP 코덱에서 Snappy 압축 해제는 CPU에서 수행되는 첫 번째 디코딩 단계로 작용하며, CPU의 여러 코어를 활용한 병렬화를 통해 전체 디코딩 과정의 속도를 크게 향상시키는 핵심 역할을 담당한다.

3. DXT 텍스처 압축 소개

DXT 또는 S3TC(S3 Texture Compression)는 GPU에 최적화된 텍스처 압축 포맷으로, HAP 코덱의 핵심 기술 중 하나이다. 이 압축 기술의 가장 큰 특징은 모든 현대 GPU가 하드웨어 수준에서 DXT 압축 텍스처의 직접 렌더링을 지원한다는 점으로, 텍스처가 압축된 형태로 GPU 메모리에 저장된 상태에서 렌더링 시 실시간으로 압축 해제가 이루어진다. DXT는 고정된 압축률을 제공하는데, DXT1은 약 6:1, DXT5는 약 4:1의 압축률을 가지며, 이는 영상 콘텐츠의 특성에 관계없이 일정한 메모리 요구사항을 보장한다. 또한 4×4 픽셀 블록 단위로 이미지를 압축하는 블록 기반 압축 방식을 사용하여, 각 블록이 독립적으로 처리되므로 텍스처의 일부분만 필요한 경우에도 효율적인 처리가 가능하다. DXT는 손실 압축 방식이므로 원본 이미지와 약간의 화질 차이가 발생할 수 있지만, HAP Q 변형에서는 YCoCg 색상 공간을 사용하여 이러한 화질 손실을 최소화한다. HAP 코덱의 핵심 이점은 DXT 압축된 데이터를 CPU에서 GPU로 전송한 후 GPU 내에서 직접 텍스처로 사용할 수 있다는 점으로, 이는 기존 코덱 방식 대비 CPU-GPU 간 데이터 전송량을 약 1/4에서 1/6로 크게 줄이며, GPU의 하드웨어 기반 DXT 압축 해제 지원으로 디코딩 부하를 현저히 감소시켜 8K 이상의 초고해상도 영상 처리에서 CPU는 Snappy 압축 해제, GPU는 DXT 압축 해제와 렌더링을 담당하는 효율적인 작업 분담을 가능하게 한다.

Ⅲ. HAP 코덱 기반 플레이어 개발

1. 개발 HAP 코덱 기반 플레이어 개요

본 연구는 초고해상도 영상의 안정적인 재생을 위한 HAP 코덱 기반 플레이어를 개발한다. 그림 2는 본 연구에서 개발한 HAP 코덱 기반 플레이어의 개요도를 보여준다. HAP 코덱을 기반으로 인코딩된 비트스트림을 입력으로 받아 패킷 버퍼에 저장한다. 디코딩 과정은 두 과정으로 이루어진다. 먼저 Snappy 압축의 해제를 위해 Packet을 Chunk 단위로 나누고 여러 개의 Snappy 디코더를 거친다. Snappy는 Chunk 단위의 dependency가 없으며, 독립적으로 멀티코어 병렬 처리가 가능하다. 디코딩을 거쳐 만들어진 DXT 포맷의 데이터는 텍스처 버퍼에 저장된 후 렌더링 타임에 맞춰 PCIe를 통해 CPU에서 GPU로 데이터가 전달된다. GPU 상에서 DXT 디코딩이 이루어지고, 재구성된 프레임을 디스플레이함으로써 HAP 코덱 기반 플레이어가 동작하게 된다.

2. 패킷 버퍼와 텍스처 버퍼을 이용한 안정성 확보 방안

기존 오픈소스 플레이어는 패킷 버퍼만을 사용하여

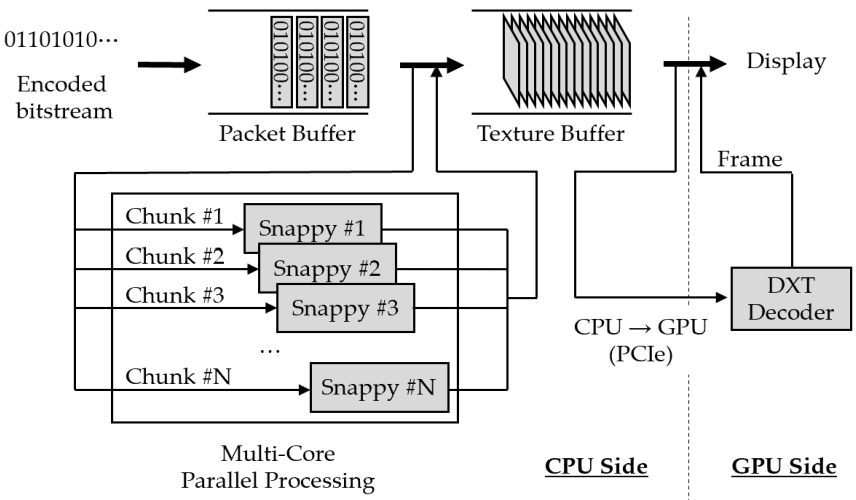

그림 2. 본 논문에서 제안하는 Hap Player 개요도
Fig. 2. Overview of Hap Player proposed in this paper

Render Loop에서 패킷을 읽고 디코딩한 후 화면에 렌더링하는 단순한 구조를 채택한다. 반면 제안된 플레이어는 이중 버퍼링 시스템을 도입하여 I/O, 디코딩, 렌더링 과정 간의 시간적 변동성을 흡수한다. 패킷 버퍼는 파일에서 읽어들인 압축된 HAP 패킷을 저장하는 첫 번째 버퍼로, 사용자가 메모리 용량에 따라 크기를 조절할 수 있으며 기본값은 300MB로 설정되어 있다. Read Thread가 이 버퍼를 지속적으로 채워 디스크 I/O의 불규칙성을 완화하고 안정적인 데이터 공급을 보장한다. 텍스처 버퍼는 Snappy 압축이 해제된 DXT 형식의 텍스처 데이터를 저장하는 두 번째 버퍼로, 압축 해제된 RGB 프레임을 직접 저장하는 방식 대비 메모리 사용량을 절약할 수 있다. 이 버퍼 역시 사용자 설정이 가능하며 기본값은 10프레임으로 구성된다. Decode Thread는 텍스처 버퍼를 지속적으로 채워 CPU 디코딩 과정의 시간적 편차를 흡수하고, 렌더링 루프에 안정적인 텍스처 데이터를 공급한다. 두 버퍼의 크기는 사용자의 하드웨어 사양과 대상 영상의 해상도에 따라 조절할 수 있으며, 고해상도 환경에서는 더 큰 버퍼 용량이 재생 안정성 확보에 도움이 된다.

제안된 플레이어는 재생 안정성을 더욱 향상시키기 위해 프리로드(Pre-load) 메커니즘을 구현하였다. 이 기능은 동영상 재생이 시작되기 전에 패킷 버퍼와 텍스처 버퍼를 모두 설정된 용량까지 미리 채워놓는 방식으로 동작한다. 프리로드 과정에서는 Read Thread와 Decode Thread가 동시에 동작하여 두 버퍼가 각각의 최대 크기까지 채워질 때까지 대기한다. 이러한 사전 준비 과정을 통해 재생 초기에 버퍼가 비어있는 상태에서 발생할 수 있는 버퍼 언더런(Buffer Underrun) 현상을 방지하고, 초고해상도 영상의 첫 번째 프레임부터 안정적인 재생을 보장할 수 있다. 프리로드 메커니즘은 재생 시작 시점에서 충분한 데이터를 미리 확보함으로써 초기 프레임 드랍 발생 가능성을 줄여 전체적인 재생 품질을 안정화한다.

Ⅳ. 실험 결과

1. 실험 환경

본 연구를 통해 개발한 HAP 코덱 플레이어의 성능 평가를 위해 두 가지 실험 환경에서 평가를 진행한다. 표 1은 실험 환경을 보여준다. 하나는 고성능 데스크톱 환경이며, 고성능의 CPU와 GPU를 포함하는 환경에 해당된다. 다른 하나는 상대적으로 성능이 낮은 노트북 환경이다. 이는 상대적으로 낮은 구동 환경에서의 동작 성능을 검증하기

표 1. 실험 환경 사양
Table 1. Experimental Environment Specifications

	CPU	GPU
High performance PC	Intel Core i9-12900K	NVIDIA GeForce RTX 4060
Laptop	Intel Core i7-1165G7	Intel Iris Xe Graphics (integrated GPU)

그림 3. HAP 코덱 플레이어 테스트 화면 (Big Buck Bunny 16K 영상)
Fig. 3. HAP Codec Player Test Screen (Big Buck Bunny 16K Video)

위함이다. 테스트에 사용된 영상은 4K(3840×2160)에서 16K(15360×8640)까지의 다양한 초고해상도 영상을 대상으로 하였다. 그림 3은 실제 테스트 화면을 캡처한 그림으로 테스트에 사용된 영상은 4k 해상도의 Big Buck Bunny의 10초 분량을 다양한 해상도로 HAP 코덱으로 인코딩하여 사용하였다. 4K(3840×2160)부터 16K(15360×8640)까지의 해상도를 테스트하여 초고해상도 영상에서의 성능을 중점적으로 평가하였다. 모든 영상은 60fps로 인코딩되었다.

본 실험에서는 제안된 플레이어(Ours)와 오픈소스 HAP 플레이어(ofxHapPlayer)의 성능을 비교하였다. 본 연구에서는 주요 평가 지표로 프레임 드랍 발생 비율을 고려한다. 하나는 렌더링 루프 동작 시간이다. 이는 각 프레임 별 렌더링 루프에서 소요되는 시간을 측정한다. 60fps 기준 한 프레임 당 16.6ms의 임계치가 있으며, 이를 초과할 경우 프레임 드랍이 발생했음으로 판단하고, 해당 영상은 디스플레이하지 않는다. 다른 하나는 프레임 드랍률이다. 전체 프레임 중, 렌더링 루프 소요 시간을 기준으로 프레임 드랍이 발생한 프레임의 비율을 의미한다.

2. 기존 ofxHapPlayer와의 성능 비교

그림 4와 그림 5는 8K(7680×4320) 해상도에서 본 논문에서 개발한 플레이어와 기존 ofxHapPlayer의 성능을 프레임 드랍 측면에서 비교하며 각각 노트북 환경과 고성능 데스크톱 환경에서의 결과를 의미한다. 각 결과에서 프레임 드랍은 산점도의 형태로 시각화하였으며, 그래프의 가로축은 프레임 번호를 세로 축은 각 프레임의 렌더링 소요

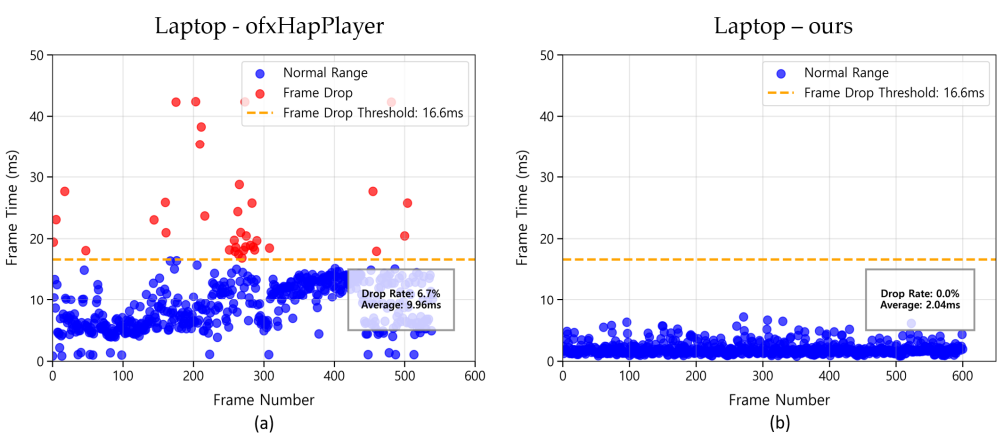

그림 4. 노트북 환경에서의 프레임 시간 산점도 (a) ofxHapPlayer (b) Proposed Player ours
Fig. 4. Frame Time Scatter Plot in Laptop Environment (a) ofxHapPlayer (b) Proposed Player ours

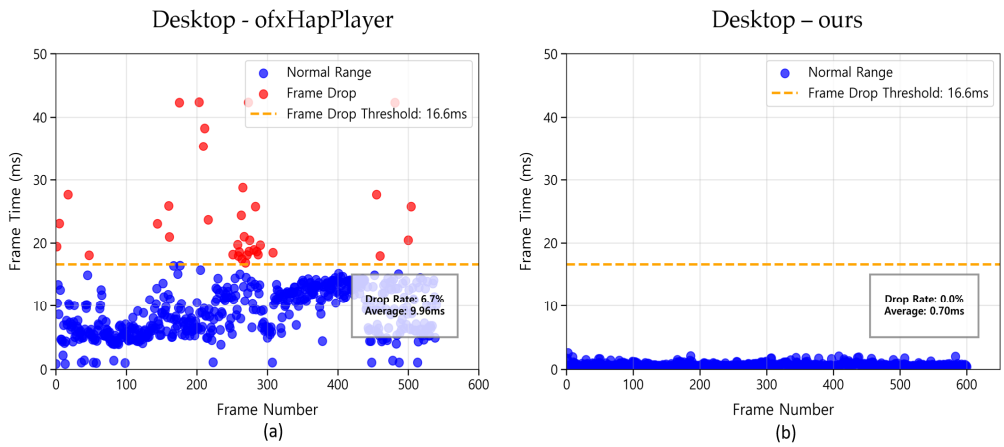

그림 5. 데스크 환경에서의 프레임 시간 산점도 (a) ofxHapPlayer (b) Proposed Player ours
Fig. 5. Frame Time Scatter Plot in Desktop Environment (a) ofxHapPlayer (b) Proposed Player ours

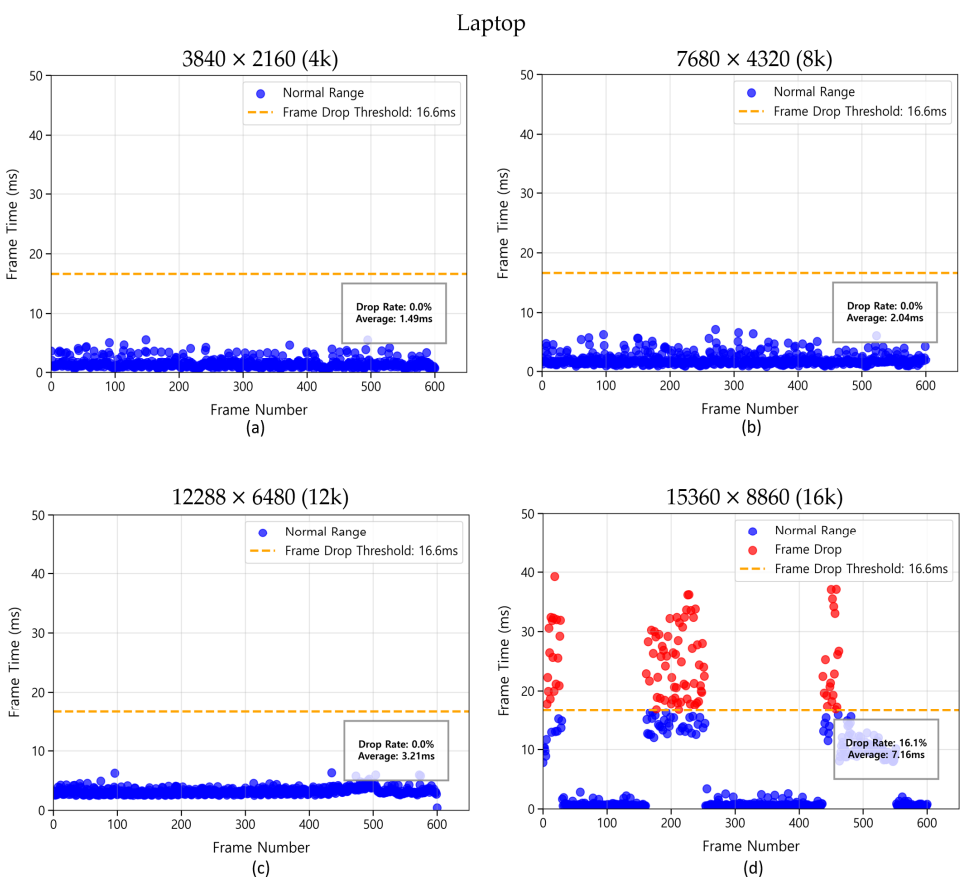

그림 6. 노트북 환경에서 해상도 별 프레임 시간 산점도 (a) 4k 해상도 (b) 8k 해상도 (c) 12k 해상도 (d) 16k 해상도
Fig. 6. Frame Time Scatter Plot by Resolution in Laptop Environment (a) 4K Resolution (b) 8K Resolution (c) 12K Resolution (d) 16K Resolution

시간을 의미한다. 16.6ms 임계선을 초과하여 프레임 드랍이 발생한 프레임은 빨간색으로 표시하였으며, 임계값 이내에 렌더링 된 프레임은 파란색으로 표시하였다. 그림 4의 노트북 환경에서는 기존 ofxHapPlayer의 경우 동일한 8K 샘플에서 프레임 드랍이 발생한 프레임 수는 40장으로 크게 증가하여 6.7%의 프레임 드랍률을 보였다. 개발 플레이어의 경우 고성능 데스크톱 환경에 비해 몇몇 프레임의 소요 시간이 증가하여 10ms 수준의 소요 시간이 관찰되었으나, 임계치를 초과하는 프레임 드랍은 확인되지 않았다. 그림 5의 데스크톱 환경에서의 8K 결과를 보면, 기존 ofxHapPlayer에서 총 600장의 프레임 중 프레임 드랍이 발생한 프레임 수는 5장으로 0.8%의 프레임 드랍률이 확인된 반면, 개발 플레이어는 프레임 드랍 없이 모든 프레임이 정상적으로 렌더링 된 것을 확인할 수 있다.

3. 다양한 해상도에서의 성능 평가

제안된 플레이어의 다양한 해상도에서의 성능을 평가하기 위해 4K, 8K, 12K, 16K 해상도에서 두 환경 모두에서 테스트를 진행하였다. 그림 6에서 보여지듯이, 노트북 환경에서는 4K, 8K, 12K까지는 프레임 드랍 없이 안정적인 재생이 가능하였으나, 16K 해상도에서는 상당한 프레임 드랍이 발생하였다. 이는 HAP 코덱의 큰 파일 크기로 인한 SSD I/O 성능 제약이 주요 원인으로 분석되며, 테스트에

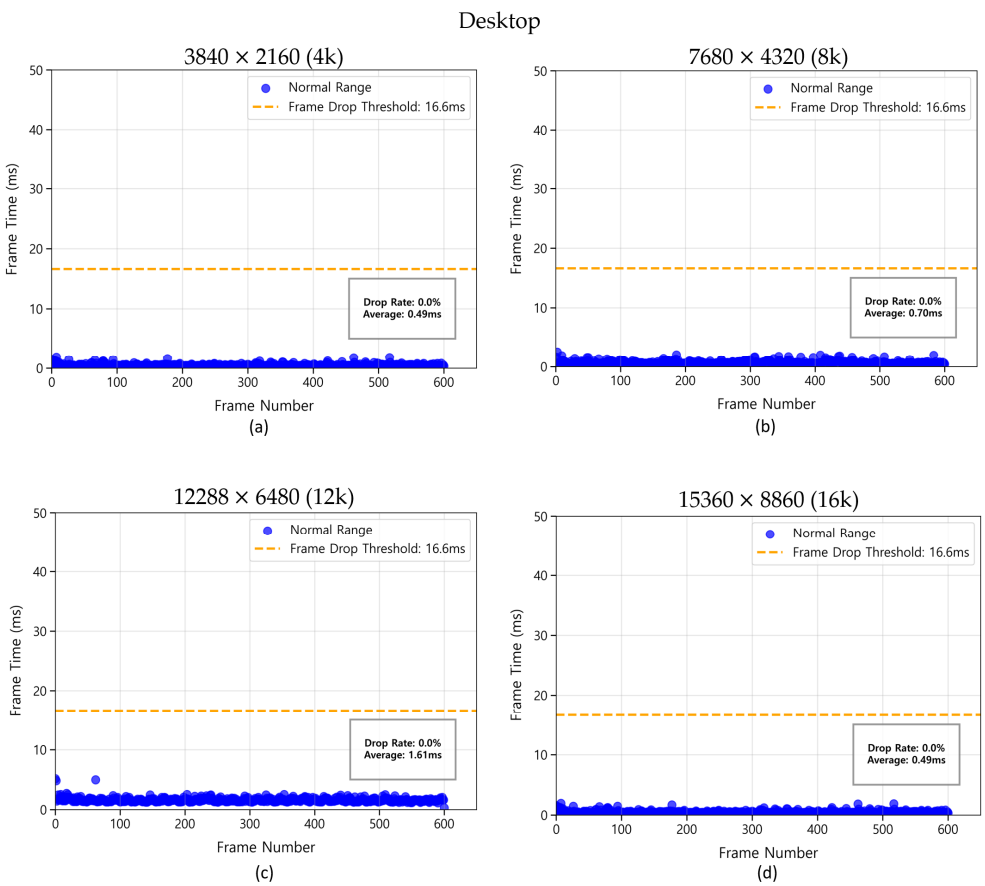

그림 7. 데스크톱 환경에서 해상도 별 프레임 시간 산점도 (a) 4k 해상도 (b) 8k 해상도 (c) 12k 해상도 (d) 16k 해상도
Fig. 7. Frame Time Scatter Plot by Resolution in Desktop Environment (a) 4K Resolution (b) 8K Resolution (c) 12K Resolution (d) 16K Resolution

사용된 노트북의 SSD가 16K 해상도의 높은 데이터 처리량을 안정적으로 지원하지 못하여 패킷 버퍼의 언더런이 발생한 것으로 판단된다. 반면 그림 7에서 확인할 수 있듯이, 데스크톱 환경에서는 4K, 8K, 12K, 16K 해상도 모두에서 0.0%의 프레임 드랍률을 보였다. 산점도에서 확인할 수 있듯이, 해상도가 증가함에 따라 렌더링 시간이 소폭 증가하지만 모든 경우에서 16.6ms 임계치 이하를 유지하여 안정적인 재생이 가능하였다.

Ⅴ. 결론 및 향후 계획

본 연구에서는 8K 이상 초고해상도 HAP 코덱 영상의 안정적인 실시간 재생을 위해 최적화된 플레이어를 제안하였다. 제안된 플레이어는 패킷 버퍼와 텍스처 버퍼를 활용한 이중 버퍼링 시스템, 멀티스레딩 디코딩, 그리고 프리로드 메커니즘을 통해 디코딩 과정의 시간적 변동성을 효과적으로 흡수하였다. 실험 결과, 기존 오픈소스 HAP 플레이어 대비 프레임 드랍률을 크게 개선하였으며, 고성능 데스크톱 환경에서는 16K 해상도까지, 노트북 환경에서는 12K 해상도까지 안정적인 재생이 가능함을 확인하였다.

본 연구는 복잡한 다중 디스플레이 동기화 방식 대신 단일 시스템에서 초고해상도 콘텐츠를 재생할 수 있는 방법을 제시하였다. 다만 HAP 코덱의 큰 파일 크기로 인한 I/O 성능 제약이 확인되었으며, 향후 더 높은 압축률을 제공하면서도 GPU 가속 처리가 가능한 새로운 코덱 개발을 진행할 예정이다.

참 고 문 헌 (References)

[1] S. J. Kim, J. K. Heo, and J. W. Kim, “A Study on the GPU-based multi-vision system for 12K super ultra highdefinition video,” Proceedings of the 2017 KICS Winter Conference, Seoul, Korea, pp. 936-937, Jan. 2017.

[2] H. Y. Park, S. G. Yoo, Y. T. Moon, M. O. Kim, and Y. T. Shin, “The Design and Implementation of Multiple Digital Signage Video Sync Technology for Ultra-high Resolution,” Journal of Broadcast Engineering, Vol. 21, No. 5, pp. 651-661, Sept. 2016.
doi: https://doi.org/10.5909/JBE.2016.21.5.651

[3] LG Electronics, “LG Unveils World’s Largest OLED Screen in Dubai,” Press Release, Aug. 13 2017. Online: https://www.lgcorp.com/media/srelease/7873 (accessed May 4 2025).

[4] Mediasync Co., “Nohyung Supermarket projection-mapping installation,” MediasyncPlayer Portfolio, n.d. Online: http://www.mediasyncplayer.com/#portfolioportfolioModal-49 (accessed May 4 2025).

[5] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, “Overview of the H.264/AVC Video Coding Standard,” IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, No. 7, pp. 560-576, July 2003.
doi: https://doi.org/10.1109/TCSVT.2003.815165

[6] G. J. Sullivan, J. Ohm, W. J. Han, and T. Wiegand, “Overview of the High Efficiency Video Coding (HEVC) Standard,” IEEE Transactions on Circuits and Systems for Video Technology, Vol. 22, No. 12, pp. 1649-1668, Dec. 2012.
doi: https://doi.org/10.1109/TCSVT.2012.2221191

[7] NVIDIA Corp, “Maximum Resolution Support,” NVIDIA Video Codec SDK Documentation v12.2, 2023. Online: https://docs.nvidia.com/video-technologies/video-codec-sdk/12.2/nvdec-video-decoder-api-prog-guide/index.html (accessed May 4 2025).

[8] AMD Corp, “Powerful Media Engine With Support for H.265/HEVC, H.264 and AV1－designed for up to 8K,” Radeon™ Graphics for Creators web page, n.d. Online:https://www.amd.com/en/products/graphics/radeon-for-creators/video-editing.html (accessed May 4 2025).

[9] Intel Corp, “Hardware Accelerated Video Decode,” 12th Gen Intel® Core™ Processors Datasheet, Vol. 1, Doc ID 655258, Rev. 010, Aug. 8 2022, Table ‘Maximum Resolution’. Online: https://edc.intel.com/.../hardware-accelerated-video-decode/ (accessed May 4 2025).

[10] T. Butterworth, “ofxHapPlayer,” GitHub repository, ver. 2, commit 6304b23, Dec. 8 2014. Online: https://github.com/bangnoise/ofxHapPlayer (accessed May 4 2025).

[11] VIDVOX LLC, “HAP: A New Video Codec for Digital Motion Graphics,” 2013. Online: https://hap.video/ (accessed May 4 2025).

[12] FFmpeg Developers, “libavcodec/hapdec － HAP Decoder Implementation,” FFmpeg Documentation, ver. 7.0, 2024. Online: https://ffmpeg.org/doxygen/7.0/hapdec_8c.html (accessed May 4 2025).

[13] A. C. Beers, M. Agrawala, and N. Chaddha, “Rendering from Compressed Textures,” Proceedings of SIGGRAPH ’96, New Orleans, USA, pp. 373-378, Aug. 1996.

[14] K. Roimela, T. Aarnio, J. Itäranta, and V. Heikkinen, “High Dynamic Range Texture Compression,” ACM Transactions on Graphics, Vol. 25, No. 3, pp. 707-712, July 2006.
doi: https://doi.org/10.1145/1141911.1141944

저 자 소 개

전 지 우

- 2025년 2월 : 서울과학기술대학교 전자IT미디어공학과 학사
- 2025년 3월 ~ 현재 : 서울과학기술대학교 스마트ICT융합공학과 석사과정
- ORCID : https://orcid.org/0009-0007-1615-5380
- 주관심분야 : 렌더링 엔진, 동영상 코덱, 영상처리, 컴퓨터 비전

정 현 민

- 2014년 2월 : 경희대학교 전자전파공학과 학사
- 2016년 2월 : 서울대학교 전기정보공학부 석사
- 2020년 8월 : 서울대학교 전기정보공학부 박사
- 2023년 3월 ~ 현재 : 서울과학기술대학교 스마트ICT융합공학과 조교수
- ORCID : https://orcid.org/0000-0001-8216-5842
- 주관심분야 : 실감미디어(VR, AR, XR, 메타버스), 영상처리, 컴퓨터 비전