JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 31 No. 11, pp. 223-229, November 2026
https://doi.org/10.9708/jksci.2026.31.11.223

Server-Assisted Integrated 3D Gaussian Splatting Framework for Real-Time Mobile Rendering

Jong-Hyun Kim*

*Associate Professor, College of AI Convergence (Dept. of Design Technology), Inha University, Incheon, Korea

[Abstract]

This study proposes a server-assisted integrated framework for generating 3D Gaussian Splatting (3DGS) scenes from smartphone videos and rendering them in real time on mobile devices. The framework performs background removal, 3DGS training, model compression, and mobile asset conversion on the server, and wirelessly transmits the resulting lightweight data to the mobile client. The received asset can be dynamically loaded and rendered without additional conversion. To support mobile execution, we further implement rendering optimizations and a runtime loader. In the experiments, the model size was reduced from 17.2 MB to 5.8 MB, corresponding to a 66.3% reduction, while achieving 30 FPS on an iPhone 13 with a test PSNR of 29.11 dB. These results demonstrate the feasibility of the proposed end-to-end 3DGS framework for real-time rendering on the tested mobile device under the experimental conditions.

▸ **Key words**: Gaussian Splatting, Real-Time Mobile Rendering, Model Compression, Server-Assisted Framework, Wireless Data Transmission

[요 약]

본 연구에서는 스마트폰으로 촬영한 영상으로부터 3D Gaussian Splatting(3DGS) 장면을 생성하고, 이를 모바일 기기에서 실시간으로 렌더링하기 위한 서버 지원형 통합 프레임워크를 제안한다. 제안 프레임워크는 서버에서 배경 제거, 3DGS 학습, 모델 압축 및 모바일용 에셋 변환을 수행한 후, 생성된 경량 데이터를 모바일 클라이언트로 무선 전송한다. 모바일에서는 수신된 에셋을 별도의 추가 변환 없이 동적으로 로딩하여 실시간 렌더링을 수행할 수 있다. 또한 모바일 실행 환경을 고려한 렌더링 최적화와 런타임 로더를 구현하였다. 실험 결과, 모델 크기를 17.2 MB에서 5.8 MB로 감소시켜 약 66.3%의 압축률을 달성하였으며, iPhone 13에서 30 FPS의 실시간 렌더링과 29.11 dB의 테스트 PSNR을 확인하였다. 이를 통해 제안 프레임워크가 실험에 사용된 모바일 기기(iPhone 13)에서 End-to-End 3DGS 처리와 30 FPS의 실시간 렌더링이 가능함을 확인하였다.

▸ **주제어**: 가우시안 스플래팅, 모바일 실시간 렌더링, 모델 압축, 서버 지원 프레임워크, 무선 데이터 전송

• First Author: Jong-Hyun Kim, Corresponding Author: Jong-Hyun Kim
*Jong-Hyun Kim (jonghyunkim@inha.ac.kr), College of AI Convergence (Dept. of Design Technology), Inha University
• Received: 2026. 09. 18, Revised: 2026. 11. 08, Accepted: 2026. 11. 08.

Copyright © 2026 The Korea Society of Computer and Information
http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945
www.kci.go.kr

I. Introduction

최근 3D Gaussian Splatting(3DGS)은 장면을 다수의 3차원 Gaussian으로 표현하고 이를 효율적으로 투영 및 합성함으로써 높은 품질의 새로운 시점 영상을 실시간으로 생성할 수 있는 방법으로 주목받고 있다. 그러나 일반적인 3DGS 기반 장면 생성 및 렌더링 과정은 고성능 GPU 환경을 전제로 하며, 학습된 Gaussian 데이터의 크기도 커 모바일 기기에서 직접 처리하거나 전송하기에는 부담이 있다. 또한 기존 Unity 기반 3DGS 렌더러는 주로 사전에 변환된 에셋을 사용하는 구조이므로, 새롭게 촬영한 영상으로부터 생성된 데이터를 모바일에서 동적으로 로딩하고 렌더링하기 어렵다.

본 연구에서는 스마트폰으로 촬영한 영상으로부터 3DGS 장면을 생성하고 이를 모바일 기기에서 실시간으로 렌더링할 수 있는 서버 지원형 통합 프레임워크를 제안한다. 제안 방법은 서버에서 프레임 추출, 카메라 자세 추정, 배경 제거, 3DGS 학습, 모델 압축 및 모바일용 에셋 변환을 수행한 후, 생성된 경량 데이터를 모바일 기기로 전송한다. 모바일 클라이언트는 수신된 데이터를 별도의 추가 변환 과정 없이 동적으로 로딩하여 실시간 렌더링을 수행한다.

또한 모바일 GPU 환경을 고려하여 Compute Shader의 스레드 그룹 크기를 조정하고 런타임 로더를 구현하였다. 이를 통해 복잡한 학습과 압축은 서버에서 수행하고 모바일에서는 경량화된 데이터의 렌더링에 집중할 수 있도록 구성하였다. 본 연구는 스마트폰 영상 입력부터 서버 기반 3DGS 처리, 무선 전송 및 모바일 실시간 렌더링까지의 전체 과정을 하나의 파이프라인으로 통합하고, 실제 모바일 환경에서 그 활용 가능성을 검증하는 것을 목적으로 한다.

본 연구의 주요 기여는 다음과 같다. 첫째, 스마트폰 영상 입력부터 서버 기반 장면 복원, 3DGS 학습 및 압축, 모바일용 에셋 변환, 무선 전송 및 모바일 렌더링까지의 과정을 하나의 End-to-End 파이프라인으로 통합하였다. 둘째, 압축된 3DGS 데이터와 Unity 기반 모바일 렌더러 사이의 SH 계수 배열 차이를 해결하기 위한 동적 계수 재배열 및 바이너리 변환 과정을 구현하였다. 셋째, 모바일 GPU의 실행 제약을 고려한 Compute Shader 구성과 런타임 로더를 적용해 실제 모바일 장치에서 처리 시간, 데이터 크기, 메모리 사용량 및 렌더링 성능을 검증하였다.

II. Related Work

3D Gaussian Splatting(3DGS)은 장면을 다수의 3차원 Gaussian primitive로 표현하고 미분 가능한 rasterization을 통해 높은 품질의 새로운 시점 영상을 실시간으로 생성하는 방법이다. Kerbl 등은 명시적인 Gaussian 표현과 효율적인 GPU rasterization을 결합하여 실시간 렌더링이 가능한 3DGS를 제안하였다[1]. 이후 Scaffold-GS는 anchor 기반의 구조화된 표현을 사용하여 불필요한 Gaussian의 중복을 줄이고 시점에 따라 Gaussian 속성을 효과적으로 생성하였다[2]. Mip-Splatting[3]과 Multi-Scale 3D Gaussian Splatting[4]은 각각 filtering 및 multi-scale 표현을 활용하여 카메라 거리나 해상도 변화에서 발생하는 aliasing 문제를 완화하였다.

3DGS는 높은 렌더링 성능을 제공하지만 많은 Gaussian과 속성값을 저장해야 하므로 모델 크기와 메모리 사용량이 증가하는 문제가 있다. Lee 등은 Gaussian 수를 감소시키고 색상 및 기하 속성을 압축하는 Compact 3D Gaussian Representation을 제안하였다[5]. Niedermayr 등은 vector clustering과 quantization-aware training을 활용하여 3DGS를 압축하고 경량 GPU에서의 렌더링 성능을 개선하였다[6]. LightGaussian은 중요도가 낮은 Gaussian을 pruning하고 SH distillation과 vector quantization을 적용하여 모델 크기를 감소시켰으며[7], 본 연구에서도 이러한 압축 방식을 서버 측 모델 경량화 과정에 활용한다. HAC는 hash-grid 기반 context model을 이용하여 Gaussian 속성의 공간적 상관관계를 활용한 압축 방법을 제안하였으며[8], EAGLES는 quantized embedding과 pruning을 통해 저장 공간과 렌더링 비용을 감소시켰다[9].

실시간 3DGS 렌더링에서는 Gaussian 정렬과 alpha blending 역시 주요 연산 비용으로 작용한다. StopThePop은 계층적 정렬 및 rasterization을 통해 기존 3DGS의 popping 및 view inconsistency 문제를 개선하였다[10]. 한편, 본 연구에서는 입력 영상의 불필요한 배경 영역을 제거하기 위해 이미지 및 비디오에 대한 범용 segmentation 모델인 SAM2[11]를 활용한다. SAM2는 첫 프레임에서 지정된 객체 정보를 영상 전체에 전달할 수 있어 3DGS 학습 이전의 전경 분리에 적합하다. 최근 Mobile-GS는 모바일 환경을 대상으로 depth-aware order-independent rendering, SH distillation, vector quantization 및 pruning을 결합하여 모바일 장치에서의

www.kci.go.kr

실시간 Gaussian Splatting 가능성을 제시하였다[12].

기존 연구인 LightGaussian[7]은 pruning, SH distillation 및 vector quantization을 통한 3DGS 모델 압축에 초점을 두었으며, Mobile-GS[12]는 모바일 환경에서의 실시간 렌더링을 위해 경량화 및 렌더링 최적화 기법을 제안하였다. 반면 본 연구는 개별 압축 및 렌더링 알고리즘의 개선보다는 스마트폰 영상 입력부터 서버 기반 학습과 압축, 모바일 에셋 변환 및 동적 로딩까지의 전체 과정을 통합하는 데 초점을 둔다. 특히 SH 계수 형식 변환과 모바일 런타임 로딩을 통해 기존 기술을 실제 모바일 환경에서 연계하여 활용할 수 있도록 구성하였다는 점에서 차별성을 갖는다. 또한 Bagdasarian 등[15]은 다양한 3DGS 압축 기법을 종합적으로 분석하고, 모델 크기와 메모리 사용량, 렌더링 품질 사이의 균형이 중요함을 강조하였다. 이러한 분석은 모바일과 같이 자원이 제한된 환경에서 데이터 경량화뿐 아니라 효율적인 저장 및 렌더링 구조를 함께 고려할 필요가 있음을 시사한다.

III. The Proposed Scheme

1. System Architecture

전체 시스템은 모바일 클라이언트, 학습 서버, 두 구성요소 간의 무선 전송 계층으로 구성된다 (Fig. 1 참조).

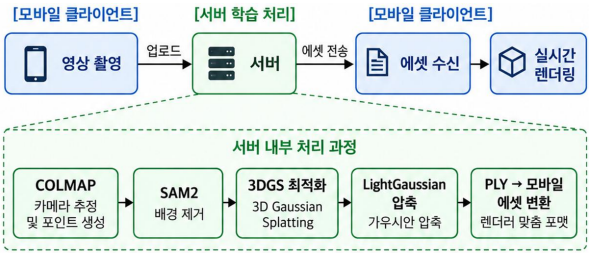


Fig. 1. System architecture of the proposed End-to-End pipeline.

사용자가 모바일 앱에서 촬영한 영상을 서버에 업로드하면, 서버는 배경 제거, 3DGS 학습, LightGaussian 압축, 모바일 에셋 변환까지의 전 과정을 순차적으로 수행한다. 변환된 단일 파일을 무선으로 전송하면, 모바일 클라이언트는 2초 간격으로 학습 진행 상태를 조회하며, 처리가 완료되면 변환된 에셋을 자동으로 다운로드하고 별도의 추가 변환 없이 즉시 렌더링을 시작한다.

본 프레임워크에서는 계산 집약적인 장면 복원, Gaussian 최적화 및 데이터 압축을 서버에 집중시키고, 모바일 클라이언트는 영상 업로드, 처리 상태 확인, 에셋 수신 및 렌더링만을 담당하도록 역할을 분리하였다. 이러한 구조는 모바일 기기의 제한된 연산 성능과 메모리 환경에서 학습 및 변환 과정을 수행하는 부담을 제거하며, 서버에서 생성된 렌더러 호환 데이터를 수신 즉시 사용할 수 있도록 한다.

2. Server Pipeline

서버 파이프라인은 영상 전처리 및 카메라 복원, 전경 분리, 3DGS 학습, 모델 압축, 모바일 에셋 변환 및 데이터 전송 과정으로 구성된다.

Frame Extraction and COLMAP. 업로드된 영상에서 10프레임마다 1장을 추출한다. 이는 30 fps 영상 기준 초당 3장에 해당한다. 이후 Schönberger 등이 제안한 COLMAP[13]을 사용하여 SfM(Structure-from-Motion) 기반의 카메라 포즈 추정과 Sparse Point Cloud 복원을 수행한다. 프레임 샘플링은 연속 프레임 사이의 높은 중복성을 줄이는 동시에 SfM에 필요한 충분한 시점 변화를 확보하기 위한 것이다.

Background Removal via SAM2. COLMAP으로 추출된 프레임에 SAM2[11]를 적용하여 배경을 제거한다. 첫 프레임에서 관심 객체의 바운딩 박스(Bounding Box)를 초기 프롬프트로 지정하고, SAM2의 마스크 예측 기능을 통해 전경 마스크를 생성한다. 이후 생성된 마스크를 SAM2의 비디오 마스크 전파 기능을 이용하여 후속 프레임에 적용한다. 생성된 전경 마스크는 3DGS 학습 과정에서 배경 픽셀을 손실 계산에서 제외하는 데 사용되며, 이를 통해 불필요한 배경 Gaussian 생성을 억제하고 관심 객체 중심의 학습을 수행한다.

3DGS Training. COLMAP으로부터 얻은 카메라 파라미터와 sparse point cloud를 초기 장면 정보로 사용하고, SAM2 마스크가 적용된 영상으로 Gaussian의 위치, 크기, 회전, 불투명도 및 SH 계수를 최적화한다. 배경이 제거된 프레임을 사용하여 3DGS 학습을 수행한다[1]. 30,000회 반복으로 초기 모델을 학습한 뒤, 5,000회 추가 학습(pruning fine-tuning) 으로 Gaussian 수를 약 1/3로 줄인다. 추가 학습 단계에서는 렌더링 기여도가 낮은 Gaussian을 제거한 후 남은 Gaussian 파라미터를 다시 최적화함으로써 pruning에 의해 발생할 수 있는 품질 저하를 완화한다.

LightGaussian Compression[7]. 학습이 완료된 모델에 대해 세 단계의 압축을 순차적으로 적용한다. Pruning 단계에서는 식 4에 따라 중요도 점수가 낮은 Gaussian의 약 66%를 제거한다. Pruning 이후 5,000회 SH

www.kci.go.kr

Distillation 학습을 통해 SH Degree를 3에서 2로 낮추고, VecTree Quantization(비율 0.6, Codebook 크기 8192)을 적용하여 최종 압축 모델을 생성한다. 이 과정에서 pruning은 Gaussian 개수를 감소시키고, SH distillation은 각 Gaussian이 보유하는 색상 계수 수를 감소시키며, vector quantization은 연속적인 속성값을 codebook 기반 표현으로 대체한다. 따라서 세 단계는 각각 공간적 복잡도, 색상 표현 복잡도 및 속성 저장 비용을 순차적으로 감소시키는 역할을 한다.

PLY-to-Mobile Binary Asset Conversion. 서버는 LightGaussian 압축이 완료된 PLY를 모바일 렌더링 엔진이 직접 사용할 수 있는 단일 이진 파일(.unitygs)로 변환하여 전송한다. Splat 데이터는 Morton 3D 공간 순서로 재정렬한 후 모바일 렌더러에서 사용할 수 있는 이진 데이터 구조로 직렬화한다. 이를 통해 서버에서 변환된 에셋을 모바일 런타임에서 별도의 추가 변환 없이 로딩할 수 있도록 구성하였다. 변환 과정에서 PLY와 Unity 렌더러 간 SH 형식 불일치 문제를 해결하였다. PLY는 SH rest 계수를 채널별 순차 배열(R 8개 → G 8개 → B 8개)로 저장하는 반면, Unity는 계수별 인터리브 배열([$R_0G_0B_0$][$R_1G_1B_1$]...)을 요구한다. 또한 LightGaussian의 degree 3→2 축소로 rest 계수가 24개로 줄어 나머지 슬롯은 zero-padding하였다. 이를 해결하는 동적 채널 재배열 알고리즘을 구현하였다 (Fig. 2 참조).

Algorithm 1 Runtime-Compatible SH Coefficient Rearrangement

Require: PLY attributes P, SH attribute names R, number of coefficients per channel K, number of Gaussians N
Ensure: Interleaved SH coefficient array H

$H \leftarrow \mathbf{0}^{(N \times 48)}$
$K' \leftarrow \min(K, 15)$
for $i = 0$ **to** $K' - 1$ **do**
  $H[:, 3i] \leftarrow P[R[i]]$
  $H[:, 3i+1] \leftarrow P[R[i+K]]$
  $H[:, 3i+2] \leftarrow P[R[i+2K]]$
end for
Reorder Gaussian records using Morton 3D order
Serialize position, scale, rotation, opacity, and H
Write serialized data to a single .unitygs file

Return: .unitygs

Algorithm 1은 입력 PLY의 SH 계수를 채널 단위 배열에서 RGB interleaved 배열로 변환하는 과정을 나타낸다. 먼저 출력 버퍼를 최대 SH 계수 수에 맞추어 초기화하고, 실제 입력 degree에 해당하는 계수만 채운다. Degree 감소로 존재하지 않는 계수는 초기값 0을 유지하므로 별도의 조건 분기 없이 zero-padding이 수행된다. 이후 Gaussian을 Morton 3D 순서로 재배열하고 모든 속성을 단일 바이너리 구조로 직렬화하여 모바일 런타임에서 직접 사용할 수 있도록 한다.

Wireless Transmission. 서버는 변환 완료 후 에셋 파일을 HTTP 엔드포인트로 제공하며, 모바일 클라이언트는 학습 상태를 주기적으로 폴링하다가 완료 상태가 확인되면 자동으로 다운로드를 시작한다. 전송 대상은 서버에서 사전 변환된 단일 바이너리 에셋 파일이며, PLY 원본 파일은 전송하지 않는다. 이 구조에서는 원본 PLY와 중간 압축 파일을 모바일로 전송하지 않고 최종 .unitygs 파일만 전달하므로, 모바일 측의 저장 공간과 추가 변환 비용을 최소화할 수 있다.

3. Mobile Rendering

모바일 렌더링을 위해 GPU 스레드 그룹 크기 축소, Unity 런타임 로더 구현, Gaussian 정렬 생략의 세 가지 요소를 적용하였다. 이 중 스레드 그룹 크기 축소와 정렬 생략은 모바일 렌더링 성능 확보를 위한 조치이며, 런타임 로더는 서버에서 전송된 에셋의 동적 로딩을 지원한다.

GPU Thread Group Size Reduction. 기존 3DGS[1] 렌더링을 구현한 UnityGaussianSplatting[14] 패키지는 Compute Shader 스레드 그룹 크기를 1024로 고정하고 있으나, iPhone 13(A15 Bionic) 등 모바일 GPU에서는 이 설정으로 렌더링 실패 또는 심각한 프레임 드롭이 발생한다. SHADER_API_MOBILE 매크로를 활용하여 모바일 빌드 시 스레드 그룹 크기가 512로 자동 설정되도록 수정하였다. SplatUtilities.compute는 GROUP_SIZE 매크로를 전체 커널에 일괄 적용하였고, DeviceRadixSort.hlsl은 정렬 알고리즘 자체가 아닌 버퍼 초기화 커널에 한해 동일하게 처리하였다.

Unity Runtime Loader. 기존 UnityGaussianSplatting[14] 패키지는 에디터 빌드타임에 생성된 에셋만 지원하므로, 런타임 동적 로딩을 위해 별도 로더를 구현하는 것이 요구되었다. 이를 위해 서버로부터 수신한 .unitygs 파일을 Unity 런타임에서 동적으로 로딩하기 위해, 파일 I/O는 백그라운드 스레드에서 처리하고 Unity 오브젝트 생성은 메인 스레드에서 수행하는 UnityGSBinaryLoader를 구현하였다.

Deferred Sorting. 3DGS 렌더링에서 매 프레임 Gaussian 정렬은 모바일 환경에서 높은 연산 비용을 유발하므로, 본 구현에서는 실시간 성능 확보를 위해 이를 생략하였다. 향후 Mobile-GS[12]와 같은 모바일 최적화 기법을 접목하여 정렬 정확도와 렌더링 품질을 개선할 예정

www.kci.go.kr

이다. 즉, 본 연구에서는 정확한 depth sorting보다 모바일 환경에서의 안정적인 프레임률 확보를 우선하였다. 이 선택은 카메라 이동이나 Gaussian이 복잡하게 중첩되는 영역에서 blending 순서에 따른 시각적 오차를 발생시킬 수 있으며, 이는 본 프레임워크의 한계로 논의한다.

IV. Experiment and Results

본 연구에서는 약 1분 20초 분량의 스마트폰 영상(1080p, 30 fps)을 사용하여 실험을 수행하였으며, 서버는 NVIDIA RTX 3090 GPU 환경에서 운용하였다. 모바일 렌더링 실험은 iPhone 13(A15 Bionic)에서 수행하였으며, Unity 2022.3 LTS 및 iOS 17.6 환경에서 렌더링 해상도를 1170 × 2532 픽셀로 설정하였다. 파이프라인 전체 소요 시간은 약 15분이며, 최종 모바일 전송 파일(.unitygs)은 5.8 MB로, 3DGS 학습 완료 PLY 파일 17.2 MB 대비 약 66.3%의 크기 감소에 해당한다.

단계별 처리 시간 및 파일 크기 변화는 Table 1과 같다. Table 1에서 전체 처리 시간 중 가장 많은 비중을 차지하는 단계는 3DGS 학습으로 9분 4초가 소요되었으며, LightGaussian 압축에는 4분 13초가 소요되었다. 반면 모바일 에셋 변환은 약 0.2초로 전체 처리 시간에 미치는 영향이 매우 작았다. 이는 제안하는 바이너리 변환 과정이 전체 파이프라인의 주요 병목이 아니며, 대부분의 계산 비용이 장면 학습과 모델 압축에서 발생함을 보여준다. 3DGS 학습 완료 후 생성된 PLY 파일은 17.2 MB였으며, LightGaussian 적용 후 3.9 MB까지 감소하였다. 이후 모바일 런타임에서 요구되는 SH 데이터 및 바이너리 구조를 포함한 .unitygs 파일은 5.8 MB로 증가하지만, 원본 학습 결과와 비교하면 최종적으로 약 66.3%의 데이터 감소를 유지한다. 따라서 에셋 변환에 필요한 부가 데이터를 포함하더라도 무선 전송에 적합한 수준의 데이터 크기를 확보할 수 있다.

Table 1. Pipeline stage results.

Step	File Size	Processing Time
Background Removal and Frame Extraction	25.8 MB	2m 4s
PLY after 3DGS Learning	17.2 MB	9m 4s
PLY after LightGaussian Compression	3.9 MB	4m 13s
.unitygs (including SH)	5.8 MB	0.2s
Final Mobile Transmission File(.unitygs)	**5.8 MB**	**Total: 15m**

Fig. 2는 모바일 클라이언트의 주요 실행 단계를 보여준다. Fig. 2(a)는 영상 선택 및 서버 업로드 화면, Fig. 2(b)는 서버의 3DGS 학습 진행 상태를 표시하는 화면, Fig. 2(c)는 학습이 완료된 에셋을 다운로드하여 모바일에서 렌더링한 결과이다. 각 화면의 주요 영역을 확대하고 단계별 구분을 명확히 하여 전체 처리 과정을 쉽게 확인할 수 있도록 하였다.

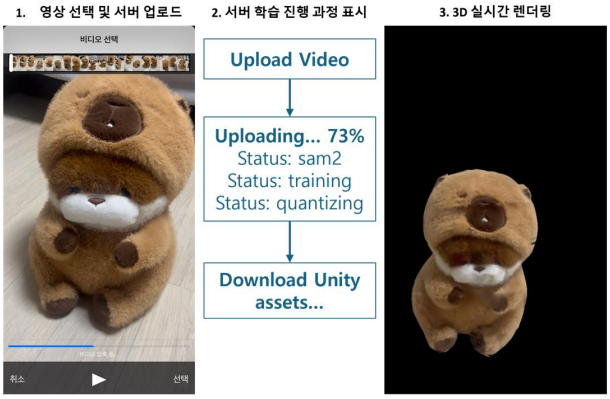


Fig. 2. Mobile client screenshots showing the main processing stages: (a) video selection and upload, (b) server-side 3DGS training progress, and (c) real-time mobile rendering of the downloaded asset.

Fig. 3은 iPhone 13에서 다양한 시점으로 렌더링한 결과를 보여준다. 배경이 제거된 전경 객체가 여러 각도에서 일관된 색상과 형태를 유지하며 다양한 각도로 실시간으로 렌더링됨을 확인할 수 있다.



Fig. 3. Multi-view rendering results on iPhone 13 (30 FPS, 33.2ms, Mem: 155.5MB)

모바일 렌더링 성능은 Table 2와 같다. iPhone 13(A15 Bionic, RAM 6GB) 환경에서 평균 30 FPS의 렌더링 성능과 155.5 MB의 메모리 사용량을 확인하였다. 다만 프레임별 FPS 변동에 대한 정량적인 분석은 수행하지 않았으므로, 장시간 렌더링 안정성에 대한 평가는 제한적이다. 학습 PSNR과 테스트 PSNR의 차이는 1.84 dB로 나타났으며, 이는 학습에 직접 사용되지 않은 시점에서도 장면의 주요 색상과 형상이 유지됨을 보여준다. Fig. 3의 다중 시점 결과에서도 전경 객체의 외형이 시점 변화에 따라 연속

www.kci.go.kr

적으로 유지되는 것을 확인할 수 있다.

Table 2. Mobile rendering performance (iPhone13).

Item	Result
Average FPS	30
Memory Usage	155.5 MB
Training PSNR	30.95 dB
Test PSNR	29.11 dB

학습된 모델의 테스트 PSNR은 29.11 dB로, 모바일 실시간 렌더링 제약하에서도 충분한 시각적 품질을 달성하였다. 이상의 결과를 통해, 본 시스템이 스마트폰 영상 입력부터 모바일 실시간 렌더링까지의 전체 파이프라인을 약 15분 이내에 자동으로 처리하며, 5.8 MB의 경량 에셋으로 30 FPS 실시간 렌더링을 달성함을 검증하였다.



Fig. 4. Illustrative multi-view visualizations of two additional object models: (a) potted plant and (b) sneaker, each shown from four viewpoints.

Fig. 4는 기존 인형 모델 외에 화분과 운동화 객체를 대상으로 구성한 다중 시점 시각화 예시를 보여준다. 각 객체를 네 가지 시점에서 제시하여 서로 다른 형상과 표면 특성을 가진 객체의 외관을 확인할 수 있도록 하였다.

V. Discussion

실험 결과를 분석하면, 제안 프레임워크의 주요 처리 병목은 서버 측 3DGS 학습과 모델 압축 과정에 집중되어 있다. Table 1에서 3DGS 학습과 LightGaussian 압축에는 각각 9분 4초와 4분 13초가 소요된 반면, 모바일 에셋 변환은 약 0.2초로 상대적으로 짧았다. 이는 에셋 변환 과정의 계산 부담은 작지만, 전체 콘텐츠 생성 시간을 단축하기 위해서는 서버 측 학습 및 압축 과정의 최적화가 필요함을 보여준다. 또한 압축된 PLY 파일은 3.9 MB였으나, 모바일 렌더러에서 요구하는 SH 데이터와 바이너리 구조를 포함하면서 최종 에셋은 5.8 MB로 증가하였다. 이는 모바일 런타임 호환성을 확보하기 위해 일정 수준의 저장 공간이 추가로 요구됨을 의미한다.

모바일 렌더링 측면에서는 iPhone 13에서 평균 30 FPS와 155.5 MB의 메모리 사용량을 확인하였으며, 테스트 PSNR은 학습 PSNR보다 1.84 dB 낮은 29.11 dB로 측정되었다. 그러나 현재 실험에서는 프레임별 성능 변동을 분석하지 않았으므로 장시간 렌더링 안정성을 일반화하기 어렵다. 또한 렌더링 비용을 줄이기 위해 Gaussian sorting을 생략하였으므로, Gaussian이 복잡하게 중첩되는 장면에서는 blending 순서에 따른 시각적 오차가 발생할 수 있다. 따라서 제안 프레임워크는 제한된 모바일 환경에서의 실시간 렌더링 가능성을 확인하였으나, 다양한 모바일 GPU와 복잡한 장면에서의 성능 및 품질 검증이 추가적으로 필요하다. 향후에는 서버 처리 시간 단축과 모바일 환경에 적합한 Gaussian 정렬 및 렌더링 최적화를 통해 이러한 한계를 개선할 계획이다.

VI. Conclusion

본 연구에서는 스마트폰 영상 업로드부터 서버 기반 3DGS 학습·압축·에셋 변환, 모바일 실시간 렌더링까지의 전 과정을 통합한 End-to-End 시스템을 제안하였다. 서버에서 학습·압축·에셋 변환을 일괄 처리하고 단일 파일을 무선 전송함으로써, 모바일이 수신 즉시 렌더링을 시작할 수 있는 구조를 실현하였다. 또한 PLY와 모바일 렌더러 사이의 SH 계수 형식 차이를 해결하는 동적 계수 재배열과 바이너리 변환 과정을 구현하고, 모바일 GPU 환경에 맞는 Compute Shader 구성 및 런타임 로더를 적용하였다. 실험에서는 학습 완료 PLY 대비 약 66.3% 감소된 5.8 MB의 모바일 에셋을 생성하였으며, iPhone 13에서 평균 30 FPS의 실시간 렌더링 성능을 확인하였다.

향후에는 Gaussian sorting을 포함한 모바일 렌더링 정확도를 개선하고, 서로 다른 성능의 모바일 장치 및 보다 복잡한 실제 장면을 대상으로 확장된 성능 분석을 수행할 예정이다. 또한 Mobile-GS[12]와 같은 모바일 특화 렌더링 기법을 접목하여 렌더링 품질과 처리 효율을 개선하고, 스마트 글래스 및 모바일 XR 환경으로 적용 범위를 확대할 계획이다.

www.kci.go.kr

REFERENCES

[1] Kerbl, B., Kopanas, G., Leimkühler, T., & Drettakis, G. “3D Gaussian Splatting for Real-Time Radiance Field Rendering.” ACM Transactions on Graphics, 42(4), Article 139, 1-14. 2023. DOI: 10.1145/3592433.

[2] Lu, T., Yu, M., Xu, L., Xiangli, Y., Wang, L., Lin, D., & Dai, B. “Scaffold-GS: Structured 3D Gaussians for View-Adaptive Rendering.” Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 20654-20664. 2024. DOI: 10.1109/CVPR52733.2024.01952.

[3] Yu, Z., Chen, A., Huang, B., Sattler, T., & Geiger, A. “Mip-Splatting: Alias-Free 3D Gaussian Splatting.” Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 19447-19456. 2024. DOI: 10.1109/CVPR52733.2024.01839.

[4] Yan, Z., Low, W. F., Chen, Y., & Lee, G. H. “Multi-Scale 3D Gaussian Splatting for Anti-Aliased Rendering.” Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 20923-20931. 2024.

[5] Lee, J. C., Rho, D., Sun, X., Ko, J. H., & Park, E. “Compact 3D Gaussian Representation for Radiance Field.” Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 21719-21728. 2024. DOI: 10.1109/CVPR 52733.2024.02052.

[6] Niedermayr, S., Stumpfegger, J., & Westermann, R. “Compressed 3D Gaussian Splatting for Accelerated Novel View Synthesis.” Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 10349-10358. 2024. DOI: 10.1109/CVPR52733.2024.00985.

[7] Fan, Z., Wang, K., Wen, K., Zhu, Z., Xu, D., & Wang, Z. “LightGaussian: Unbounded 3D Gaussian Compression with 15x Reduction and 200+ FPS.” Advances in Neural Information Processing Systems (NeurIPS), 37. 2024. DOI: 10.52202/079017-4447.

[8] Chen, Y., Wu, Q., Lin, W., Harandi, M., & Cai, J. HAC: “Hash-Grid Assisted Context for 3D Gaussian Splatting Compression.” European Conference on Computer Vision (ECCV), 422-438. 2024. DOI: 10.1007/978-3-031-72667-5_24.

[9] Girish, S., Gupta, K., & Shrivastava, A. “EAGLES: Efficient Accelerated 3D Gaussians with Lightweight EncodingS.” European Conference on Computer Vision (ECCV), 54-71. 2024. DOI: 10.1007/978-3-031-73036-8_4.

[10] Radl, L., Steiner, M., Parger, M., Weinrauch, A., Kerbl, B., & Steinberger, M. “StopThePop: Sorted Gaussian Splatting for View-Consistent Real-Time Rendering.” ACM Transactions on Graphics, 43(4), Article 64, 1-17. 2024. DOI: 10.1145/3658187.

[11] Ravi, N., Gabeur, V., Hu, Y. T., Hu, R., Ryali, C., Ma, T., Khedr, H., Rädle, R., Rolland, C., Gustafson, L., Mintun, E., Pan, J., Alwala, K. V., Carion, N., Wu, C. Y., Girshick, R., Dollár, P., & Feichtenhofer, C. “SAM2: Segment Anything in Images and Videos.” International Conference on Learning Representations (ICLR). 2025.

[12] Du, X., Wang, Y., Zhan, K., & Yu, X. “Mobile-GS: Real-Time Gaussian Splatting for Mobile Devices.” International Conference on Learning Representations (ICLR). 2026.

[13] Schönberger, J. L., & Frahm, J.-M. “Structure-from-Motion Revisited.” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 4104–4113. 2016. DOI: 10.1109/CVPR.2016.445.

[14] Pranckevičius, A. UnityGaussianSplatting. GitHub Repository. https://github.com/aras-p/UnityGaussianSplatting.

[15] Bagdasarian, M. T., Knoll, P., Li, Y., Barthel, F., Hilsmann, A., Eisert, P., & Morgenstern, W. “3DGS.zip: A Survey on 3D Gaussian Splatting Compression Methods.” Computer Graphics Forum, 44(2), e70078. 2025. DOI: 10.1111/cgf.70078.

Authors



Jong-Hyun Kim received the B.A. degree in the Department of Digital Contents at Sejong University in 2008. He received M.S. and Ph.D. degrees in the Department of Computer Science and Engineering at Korea University, in 2010 and 2016. Prof. Kim is an Associate Professor in the College of Software and Convergence (Dept. of Design Technology) in Inha University. His current research interests include fluid animation and virtual reality.

www.kci.go.kr