

대용량 3D 모델 뷰어 웹 애플리케이션 아키텍처 설계서

서버 및 로컬 자원 대응형 하이브리드 아키텍처 파이프라인 (초소형 동접자 환경 최적화)

1. 개요 및 설계 목표

본 문서는 서버에 저장된 대용량 3D 자산 조회 및 사용자 로컬 파일의 즉각적인 드래그 앤 드롭(Drag & Drop) 시각화를 동시에 만족하기 위한 웹 기반 3D 뷰어 애플리케이션 아키텍처를 정의한다. 동시 접속자 수 3명 미만의 제약 조건과 대용량 파일 로딩 시의 UX 이탈 방지를 최우선 목표로 삼는다.

핵심 설계 목표

- 초기 체감 속도 극대화:** 수백 MB급 파일 로딩 시 빈 화면 노출을 배제하고 즉각적인 시각 피드백 제공.
- 하이브리드 렌더링 파이프라인 구축:** 서버 자산은 SSR 가상 미리보기 및 백그라운드 CSR 로딩을 결합하고, 로컬 자산은 순수 CSR 처리.
- 인프라 효율성 최적화:** 극소수 동접자 환경에 맞춰 서버 GPU 자원 의무화를 배제하고 정적 프리-렌더링 자산을 활용하여 비용 절감.

2. 시스템 아키텍처 개요

렌더링 메인 엔진은 클라이언트 브라우저(CSR)에 탑재한다. 이를 통해 로컬 드래그 앤 드롭 파일은 네트워크 연산 없이 로컬 메모리에서 즉시 디코딩하며, 서버 파일은 사전 생성된 미디어를 SSR 단계에서 선배포하는 방식으로 이원화 구조를 채택한다.

구분	A. 서버 저장 대용량 파일 조회	B. 로컬 드래그 앤 드롭 파일 조회
초기 렌더링 (SSR)	서버에서 정적 루프 미디어(WebP/WebM) 즉시 서빙	기본 뷰어 레이아웃 및 드롭 존 UI 서빙
데이터 전송	백그라운드에서 원본(Draco 압축 GLB) 비 동기 다운로드	네트워크 전송 없음 (브라우저 File API 직접 로드)
메인 렌더링 (CSR)	다운로드 완료 후 Three.js 웹 컨텍스트로 Hydration	로컬 메모리 주소(Blob URL) 바인딩 후 즉시 파싱
UX 이점	첫 진입 시 무조건적인 시각 움직임 확보 (이탈 방지)	대용량 업로드 대기 시간 제로화 (보안 우수)

3. 시나리오별 상세 시퀀스

3.1 서버 자산 파이프라인 (하이브리드 SSR + CSR)

서버에 파일이 업로드되는 시점에 백엔드 파이프라인에서 '빙빙 돌아가는 360도 회전 움짤(Animated WebP 또는 투명 배경 WebM)'을 미리 자산화(Pre-rendering)해 두는 것이 핵심이다.

- **Step 1 (SSR):** 사용자가 웹페이지 진입 시, 서버는 가벼운 프리비주얼 미디어(WebP)가 포함된 HTML/CSS를 즉시 반환한다. 유저는 즉시 부드럽게 회전하는 고품질 3D 형태를 확인한다.
- **Step 2 (CSR Background):** 화면 한편에 프로그레스 바를 노출하며, 내부적으로 Three.js GLTFLoader가 원본 바이너리를 스트리밍 다운로드한다.
- **Step 3 (Hydration):** 파싱이 끝나고 WebGL 컨텍스트에 3D 메시가 준비되면, 상위 레이어의 WebP 미디어를 CSS Opacity Transition(0.5초)으로 페이드아웃하며 실제 Three.js 캔버스를 활성화한다.

3.2 로컬 자산 파이프라인 (순수 CSR - Drag & Drop)

보안 및 속도를 위해 파일을 서버로 업로드하지 않고 브라우저 샌드박스 내부 메모리에서만 처리한다.

- **Step 1 (Event Capture):** HTML5 Drag & Drop API의 drop 이벤트를 캡처하여 파일 바이너리 객체(Blob)를 확보한다.
- **Step 2 (Local Mapping):** URL.createObjectURL(file)을 통해 해당 파일에 가상 로컬 URI를 임시 부여한다.
- **Step 3 (Direct Loading):** 외부 네트워크 통신 없이 브라우저 내부에 상주하는 Draco 디코더 모듈을 사용해 가상 URI로부터 기하학 데이터를 즉시 압축 해제하고 씬(Scene)에 투입한다.

💡 설계 핵심 요약

- 동접자가 3명 미만이므로, 런타임에 서버가 무거운 실시간 렌더링을 칠 필요가 전혀 없습니다.
- 서버 자산은 '정적 배치 프리렌더링(WebP) + 백그라운드 다운로드' 조합으로 풀고, 로컬 자산은 '클라이언트 메모리 직공급(Blob)'으로 설계하여 인프라 사양을 대폭 낮추면서 최고의 성능을 구현합니다.

4. 클라이언트 구현 명세 (Three.js 핵심 로직)

아래 모듈 사양을 참조하여 프론트엔드 컴포넌트를 설계한다. Draco 압축 해제 코어 라이브러리는 공통 정적 자산 경로(/draco/)에 배치되어야 한다.

```

// 드래그 앤 드롭 및 하이브리드 스트리밍 통합 로더 모듈 구조 예시
import * as THREE from 'three';
import { GLTFLoader } from 'three/examples/jsm/loaders/GLTFLoader.js';
import { DRACOLoader } from 'three/examples/jsm/loaders/DRACOLoader.js';

export class ThreeDViewer {
  constructor(canvasContainer, progressBarElement, previewImageElement) {
    this.container = canvasContainer;
    this.progressBar = progressBarElement;
    this.preview = previewImageElement; // SSR로 그려진 WebP 가상 레이어

    this.initThreeEngine();
  }

  initThreeEngine() {
    this.scene = new THREE.Scene();
    // 렌더러, 카메라, 조명 셋업 (생략)

    // 대용량 처리를 위한 필수 디코더 세팅
    this.dracoLoader = new DRACOLoader();
    this.dracoLoader.setDecoderPath('/draco/'); // Worker 기반 멀티스레드 해제 경로

    this.gltfLoader = new GLTFLoader();
    this.gltfLoader.setDRACOLoader(this.dracoLoader);
  }

  // 시나리오 A: 서버 자산 원격 로드
  loadServerAsset(gltfUrl) {
    this.toggleUI(true);

    this.gltfLoader.load(gltfUrl,
      (gltf) => {
        this.scene.add(gltf.scene);
        this.executeHydration();
      },
      (xhr) => {
        if (xhr.total > 0) {
          const percent = (xhr.loaded / xhr.total) * 100;
          this.progressBar.style.width = percent + '%';
        }
      }
    );
  }

  // 시나리오 B: 로컬 자산 즉시 로드 (Drag & Drop 파일 핸들러 연동)
  loadLocalFile(fileBlob) {
    this.toggleUI(true);

    // 로컬 가상 메모리 매핑 기법 (서버 업로드 차단)
    const localBlobURL = URL.createObjectURL(fileBlob);

    this.gltfLoader.load(localBlobURL,

```

```

    (gltf) => {
      this.scene.add(gltf.scene);
      this.executeHydration();
      URL.revokeObjectURL(localBlobURL); // 메모리 누수 방지
    },
    (xhr) => { /* 로컬 로드는 일반적으로 진척도 표시 없이 즉시 파싱됨 */ }
  );
}

executeHydration() {
  this.toggleUI(false);
  // 부드러운 화면 전환을 위해 CSS 트랜지션 효과 적용
  this.preview.style.transition = 'opacity 0.5s ease';
  this.preview.style.opacity = '0';
  setTimeout(() => { this.preview.style.display = 'none'; }, 500);
}

toggleUI(isLoading) {
  this.progressBar.style.display = isLoading ? 'block' : 'none';
}
}

```

5. 데이터 최적화 파이프라인 (정적 빌드 명세)

대용량 원본 파일(.obj, .fbx, 원시 .gltf)을 클라이언트에 그대로 노출할 경우 네트워크 다운로드 지연 및 브라우저 크래시가 발생하므로, 업로드 혹은 배포 전 아래 가이드라인에 맞춰 자산을 최적화 가공해야 한다.

5.1 자산 압축 가이드라인

- **메쉬 지오메트리 압축:** Google Draco 압축 옵션을 필수로 적용한다. 양자화 비트(Quantization Bits) 설정을 통해 정밀도 손실을 최소화하면서 포지션, 노멀 데이터를 최대 90% 압축한다. (추천 도구: gltf-pipeline CLI)
- **텍스처 가공:** 일반 .png, .jpg 이미지는 GPU 메모리를 직접 점유하므로, VRAM 효율화를 위해 **KTX2(Basis Universal)** 컨테이너 포맷으로 인코딩하여 저장한다.

5.2 프리비주얼 미디어 생성 자동화 가이드

- 서버 백엔드 자산 등록 스크립트 작성 시, Blender CLI 또는 헤드리스 Puppeteer 환경을 이용하여 대상 모델을 중심축(Y-up) 기준 360도 회전시키는 카메라 애니메이션을 수행한다.
- 출력 포맷은 색상 손실이 없으면서 압축률이 뛰어난 **Animated WebP**를 기본 사양으로 정하며, 배경 투명화(Alpha 채널)가 필요한 UI 요소와 결합 시에는 **WebM(VP9 코덱)** 비디오 컴포넌트로 대체 생성한다.