ChatGPT

1. `main.py` – FastAPI Server Code

```
import os
import uuid
import random
import asyncio
from pathlib import Path
from fastapi import FastAPI, UploadFile, File, HTTPException
from fastapi.responses import JSONResponse, HTMLResponse, FileResponse
from fastapi.middleware.cors import CORSMiddleware

# FastAPI 앱 생성
app = FastAPI(title="Gaussian Splatting 3D Reconstruction API", description="Upload images to reconstruct a 3D model using Gaussian Splatting and view the result in a web viewer.")

# CORS 등 필요한 미들웨어 설정 (필요에 따라)
app.add_middleware(CORSMiddleware, allow_origins=["*"], allow_methods=["*"], allow_headers=["*"])

# 작업 관리용 전역 변수
jobs = {}           # job_id -> 정보(dict: status, pub_key, etc.)
pub_to_job = {}   # pub_key -> job_id 매핑
MAX_CONCURRENT = 2
sem = asyncio.Semaphore(MAX_CONCURRENT)   # 동시 실행 세마포어 (최대 2개 작업)

# 경로 기본 설정
BASE_DIR = Path("/data/jobs")   # 모든 job 데이터 저장 베이스 디렉토리
BASE_DIR.mkdir(parents=True, exist_ok=True)

#   COLMAP 및 Gaussian Splatting 실행을 비동기 처리하기 위한 유틸리티 함수
async def run_command(cmd: list, log_file, cwd: Path = None):
  """
  주어진 명령(cmd 리스트)을 서브프로세스로 실행하고, 출력 스트림을 실시간 로그 파일에 기록합니다.
  오류 발생 시 예외를 발생시켜 상위에서 처리합니다.
  """
    # 서브프로세스 실행 (표준 출력과 에러를 합쳐 한 스트림으로 처리)
    process = await asyncio.create_subprocess_exec(
        *cmd, cwd=(str(cwd) if cwd else None),
        stdout=asyncio.subprocess.PIPE, stderr=asyncio.subprocess.STDOUT
    )
    # 출력 스트림을 읽어가며 로그 파일에 기록
    while True:
        line = await process.stdout.readline()
        if not line:
            break
        text = line.decode(errors="ignore")
```

```
            log_file.write(text)
            log_file.flush()
        # 프로세스 종료 대기
        exit_code = await process.wait()
        if exit_code != 0:
            # 에러가 발생한 경우, 로그 파일에 오류 메시지 기록 후 예외 발생
            log_file.write(f"[ERROR] Command {' '.join(cmd)} exited with code {exit_code}\n")
            log_file.flush()
            raise RuntimeError(f"Command failed: {' '.join(cmd)} (exit code: {exit_code})")

#  COLMAP points3D.txt -> PLY 변환 함수
def convert_points3d_to_ply(points3d_txt: Path, ply_path: Path):
    """
    COLMAP에서 생성된 points3D.txt 파일을 읽어 PLY (ASCII 포인트클라우드) 파일로 저장합니다.
    """
    with open(points3d_txt, 'r') as f_txt, open(ply_path, 'w') as f_ply:
        lines = f_txt.readlines()
        # points3D.txt의 유효 데이터 라인만 파싱 (주석 줄 제외)
        points = []
        for line in lines:
            if line.strip().startswith('#') or line.strip() == '':
                continue
            parts = line.strip().split()
            # points3D.txt 포맷: POINT3D_ID, X, Y, Z, R, G, B, ERROR, TRACK[]...
            if len(parts) >= 7:
                X, Y, Z = map(float, parts[1:4])
                R, G, B = map(int, parts[4:7])
                points.append((X, Y, Z, R, G, B))
        # PLY 헤더 작성
        f_ply.write("ply\nformat ascii 1.0\n")
        f_ply.write(f"element vertex {len(points)}\n")
        f_ply.write("property float x\nproperty float y\nproperty float z\n")
        f_ply.write("property uchar red\nproperty uchar green\nproperty uchar blue\n")
        f_ply.write("end_header\n")
        # 각 포인트 좌표와 색상 기록
        for (x, y, z, r, g, b) in points:
            f_ply.write(f"{x:.6f} {y:.6f} {z:.6f} {r} {g} {b}\n")

#  백그라운드에서 실행될 3D 재구성 파이프라인 함수
async def process_job(job_id: str):
    """
    주어진 job_id에 대해 COLMAP 및 Gaussian Splatting 파이프라인을 실행합니다.
    """
    # 세마포어 획득: 동시에 최대 2개 작업만 실행
    async with sem:
        job = jobs[job_id]
        job_dir = Path(BASE_DIR / job_id)
        log_path = job_dir / "run.log"
        # 로그 파일 열기 (추가 모드)
        with open(log_path, 'a', encoding='utf-8', buffering=1) as log_file:
            try:
```

```
                # 상태 갱신: 실행 중
                job["status"] = "RUNNING"
                log_file.write(f"=== Job {job_id} started (pub_key={job['pub_key']}) ===\n")
                log_file.flush()
                # COLMAP feature extraction 단계
                log_file.write(">> [1/6] Feature extraction 시작...\n")
                log_file.flush()
                images_path = job_dir / "upload" / "images"
                database_path = job_dir / "colmap" / "database.db"
                # colmap 실행: feature_extractor
                await run_command([
                    "colmap", "feature_extractor",
                    "--database_path", str(database_path),
                    "--image_path", str(images_path),
                    "--ImageReader.camera_model", "OPENCV",   # 기본 카메라 모델 (OPENCV 핀
홀)
                    "--SiftExtraction.num_threads", "8"
                ], log_file)
                # colmap 실행: exhaustive_matcher (전역 매칭)
                log_file.write(">> [2/6] Feature matching 시작...\n")
                log_file.flush()
                await run_command([
                    "colmap", "exhaustive_matcher",
                    "--database_path", str(database_path),
                    "--SiftMatching.num_threads", "8",
                    "--SiftMatching.guided_matching", "1"
                ], log_file)
                # colmap 실행: mapper (Structure-from-Motion 재구성)
                log_file.write(">> [3/6] Sparse reconstruction (SfM) 시작...\n")
                log_file.flush()
                sparse_path = job_dir / "colmap" / "sparse"
                sparse_path.mkdir(parents=True, exist_ok=True)   # 출력 디렉토리 생성
                await run_command([
                    "colmap", "mapper",
                    "--database_path", str(database_path),
                    "--image_path", str(images_path),
                    "--output_path", str(sparse_path),
                    "--Mapper.num_threads", "8"
                ], log_file)
                # sparse 결과 (0번 모델) 존재 여부 검사
                model0_path = sparse_path / "0"
                if not model0_path.exists():
                    # 재구성된 모델이 없으면 오류로 처리
                    raise RuntimeError("COLMAP reconstruction failed: no sparse model
generated.")
                # colmap 실행: image_undistorter (이미지 왜곡 보정 및 모델 변환)
                log_file.write(">> [4/6] Undistorting images 및 변환된 모델 생성...\n")
                log_file.flush()
                work_path = job_dir / "work"
                work_path.mkdir(parents=True, exist_ok=True)
                undistort_sparse = work_path / "sparse"
```

```
            undistort_images = work_path / "images"
            await run_command([
                "colmap", "image_undistorter",
                "--image_path", str(images_path),
                "--input_path", str(model0_path),
                "--output_path", str(work_path),
                "--output_type", "COLMAP"
            ], log_file)
            # Gaussian Splatting convert.py는 COLMAP 절차를 포함하지만 이미 수동으로 수행했으므
로 생략

            # GS 학습(train.py) 실행
            log_file.write(">> [5/6] Gaussian Splatting 학습 (train.py) 시작...\n")
            log_file.flush()
            gs_repo_dir = Path(__file__).resolve().parent / "gaussian-splatting"
            train_script = gs_repo_dir / "train.py"
            # train.py 실행 (`-s` 파라미터에 work 디렉토리를 지정)
            await run_command([
                "python", str(train_script),
                "-s", str(work_path)
            ], log_file, cwd=gs_repo_dir)
            # 학습 완료 후 metrics.py 실행 (메트릭 계산 및 보고서 생성)
            log_file.write(">> [6/6] Metrics 계산 (metrics.py) 및 결과 처리...\n")
            log_file.flush()
            metrics_script = gs_repo_dir / "metrics.py"
            metrics_dir = job_dir / "metrics"
            metrics_dir.mkdir(parents=True, exist_ok=True)
            await run_command([
                "python", str(metrics_script),
                "-s", str(work_path),
                "-o", str(metrics_dir)
            ], log_file, cwd=gs_repo_dir)
            # point cloud (.ply) 생성 - COLMAP sparse 점들을 PLY 포맷으로 내보내기
            export_dir = job_dir / "export"
            export_dir.mkdir(parents=True, exist_ok=True)
            ply_path = export_dir / "cloud.ply"
            points3d_txt = work_path / "sparse/0/points3D.txt"
            if not points3d_txt.exists():
                raise RuntimeError("points3D.txt not found, cannot generate cloud.ply")
            convert_points3d_to_ply(points3d_txt, ply_path)
            # 공개용 키 저장
            with open(job_dir / "key.txt", 'w') as f:
                f.write(job["pub_key"])
            # 상태 갱신: 완료
            job["status"] = "DONE"
            log_file.write(f"=== Job {job_id} 완료: 성공적으로 처리되었습니다. ===\n")
            log_file.flush()
        except Exception as e:
            # 오류 발생: 상태 표시 및 로그 기록
            job["status"] = "ERROR"
            error_msg = f"[Exception] {str(e)}"
            log_file.write(error_msg + "\n")
```

```
                log_file.write(f"=== Job {job_id} 종료: 오류 발생 ===\n")
                log_file.flush()
                print(error_msg)  # 서버 콘솔에도 출력
            # (세마포어는 async with 블록에서 자동 해제됨)

#  API 엔드포인트 구현

@app.post("/recon/jobs", summary="새 3D 재구성 작업 생성", description="여러 이미지를 업로드하여 3D 재구성 작업을 생성합니다. 작업 ID와 공개 뷰어 키를 반환합니다.")
async def create_reconstruction_job(files: list[UploadFile] = File(...)):
    # 새로운 job 식별자 생성 (UUID4 기반 8문자 또는 timestamp 등)
    job_id = uuid.uuid4().hex[:8]  # 8자리 16진수 ID
    # 10자리 공개용 랜덤 키 생성 (숫자)
    pub_key = ''.join(random.choice("0123456789") for _ in range(10))
    # 혹시 pub_key 충돌 시 변경
    while pub_key in pub_to_job:
        pub_key = ''.join(random.choice("0123456789") for _ in range(10))
    # 작업 디렉터리들 생성
    job_dir = Path(BASE_DIR / job_id)
    (job_dir / "upload" / "images").mkdir(parents=True, exist_ok=True)
    (job_dir / "colmap").mkdir(parents=True, exist_ok=True)
    # 업로드된 이미지들을 저장
    for file in files:
        # 파일 이름이 중복되면 덮어쓰지 않도록 고유 이름 추가 가능 (여기서는 그대로 사용)
        content = await file.read()
        img_path = job_dir / "upload" / "images" / file.filename
        with open(img_path, 'wb') as f:
            f.write(content)
    # 작업 초기 상태 저장
    jobs[job_id] = {
        "status": "PENDING",  # 아직 대기 상태
        "pub_key": pub_key
    }
    pub_to_job[pub_key] = job_id
    # 백그라운드로 파이프라인 작업 수행 시작
    asyncio.create_task(process_job(job_id))
    # 작업 ID와 공개 키 반환
    return {"job_id": job_id, "pub_key": pub_key}

@app.get("/recon/jobs/{job_id}/status", summary="작업 상태 조회", description="지정한 job_id에 대한 현재 상태와 로그 일부, 결과 확인 URL 등을 반환합니다.")
async def get_job_status(job_id: str):
    if job_id not in jobs:
        raise HTTPException(status_code=404, detail="Job not found")
    job_info = jobs[job_id]
    status = job_info.get("status", "PENDING")
    # run.log에서 마지막 N줄 읽기
    log_path = Path(BASE_DIR / job_id / "run.log")
    log_tail = []
    if log_path.exists():
        try:
```

```
                with open(log_path, 'r') as f:
                    lines = f.readlines()
                    # 최근 10줄만 추출
                    if len(lines) > 0:
                        log_tail = lines[-10:]
            except Exception as e:
                log_tail = [f"(log read error: {e})"]
        response = {
            "job_id": job_id,
            "status": status,
            "log_tail": log_tail
        }
        # 완료된 경우 결과 뷰어 URL 포함
        if status == "DONE":
            response["viewer_url"] = f"/v/{job_info['pub_key']}"
        elif status == "ERROR":
            # 오류인 경우 메시지를 로그 tail에 담고, 필요하면 에러 설명 추가
            response["error"] = "An error occurred. Check log for details."
        return JSONResponse(content=response)

@app.get("/v/{pub_key}", summary="3D 결과 뷰어 페이지", response_class=HTMLResponse)
async def view_result_page(pub_key: str):
    # 주어진 공개 키에 해당하는 job 찾기
    job_id = pub_to_job.get(pub_key)
    if job_id is None or job_id not in jobs:
        raise HTTPException(status_code=404, detail="Invalid viewer key")
    # 작업이 완료되었는지 확인
    if jobs[job_id].get("status") != "DONE":
        # 아직 완료되지 않은 경우 대기 메시지 표시 (간단한 HTML 반환)
        return HTMLResponse(content="<html><body><h3>Result is not ready yet. Please check again later.</h3></body></html>")
    # Three.js 기반 PLY 뷰어 HTML 페이지 생성 및 반환
    html_content = f"""
  <!DOCTYPE html>
  <html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>3D Point Cloud Viewer</title>
    <style> body {{ margin: 0; background-color: #000; }} canvas {{ display: block; }} </style>
  </head>
  <body>
    <div id="viewer"></div>
    <!-- Three.js 및 PLYLoader 로드 (CDN 이용) -->
    <script src="https://unpkg.com/three@0.153.0/build/three.min.js"></script>
    <script src="https://unpkg.com/three@0.153.0/examples/js/controls/OrbitControls.js"></script>
    <script src="https://unpkg.com/three@0.153.0/examples/js/loaders/PLYLoader.js"></script>
    <script>
      // Three.js 장면, 카메라, 렌더러 초기화
      const scene = new THREE.Scene();
      scene.background = new THREE.Color(0x000000);
      const camera = new THREE.PerspectiveCamera(75, window.innerWidth/window.innerHeight,
```

```
0.1, 1000);
    const renderer = new THREE.WebGLRenderer({ antialias: true });
    renderer.setSize(window.innerWidth, window.innerHeight);
    document.getElementById('viewer').appendChild(renderer.domElement);
     // 궤도 컨트롤 (OrbitControls) 추가
    const controls = new THREE.OrbitControls(camera, renderer.domElement);
    controls.target.set(0, 0, 0);
    controls.update();
     // 조명 설정 (Directional Light)
    const light = new THREE.DirectionalLight(0xffffff, 0.8);
    light.position.set(1, 1, 1);
    scene.add(light);
    scene.add(new THREE.AmbientLight(0xffffff, 0.2));
     // 윈도우 리사이즈 이벤트 처리
    window.addEventListener('resize', function() {{
      camera.aspect = window.innerWidth / window.innerHeight;
      camera.updateProjectionMatrix();
      renderer.setSize(window.innerWidth, window.innerHeight);
    }});
    // PLY 포인트클라우드 로드
    const loader = new THREE.PLYLoader();
    loader.load('/recon/pub/{pub_key}/cloud.ply', function (geometry) {{
      geometry.computeBoundingBox();
      const material = new THREE.PointsMaterial({{ size: 1.5, vertexColors: true }});
      const points = new THREE.Points(geometry, material);
      scene.add(points);
        // 카메라를 포인트클라우드에 맞게 위치 조정
      const center = geometry.boundingBox.getCenter(new THREE.Vector3());
      const size = geometry.boundingBox.getSize(new THREE.Vector3());
      const maxDim = Math.max(size.x, size.y, size.z);
      camera.position.copy(center.clone().add(new THREE.Vector3(maxDim, maxDim, maxDim)));
      camera.lookAt(center);
      camera.far = maxDim * 10;
      camera.updateProjectionMatrix();
    }});
     // 애니메이션 루프
    function animate() {{
      requestAnimationFrame(animate);
      controls.update();
      renderer.render(scene, camera);
    }}
    animate();
  </script>
 </body>
 </html>
 """
    return HTMLResponse(content=html_content, status_code=200)

@app.get("/recon/pub/{pub_key}/cloud.ply", summary="생성된 포인트클라우드 PLY 파일 다운로드")
async def download_ply(pub_key: str):
    # 공개 키로부터 job 검색
```

```
    job_id = pub_to_job.get(pub_key)
    if job_id is None or job_id not in jobs:
        raise HTTPException(status_code=404, detail="Invalid key or job not found")
    if jobs[job_id].get("status") != "DONE":
        raise HTTPException(status_code=400, detail="Result not ready")
    ply_file = Path(BASE_DIR / job_id / "export" / "cloud.ply")
    if not ply_file.exists():
        raise HTTPException(status_code=404, detail="cloud.ply not found")
    # PLY 파일을 반환 (다운로드 가능 또는 뷰어에서 직접 로드)
    return FileResponse(path=ply_file, filename="cloud.ply", media_type="application/octet-stream")

# (선택) 애플리케이션 실행 설정
if __name__ == "__main__":
    import uvicorn
    # uvicorn 서버 실행: 필요에 따라 host/port 조정
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

주요 구현 설명: 위 `main.py`는 FastAPI 기반 웹 서버로 이미지 업로드부터 3D 모델 생성 파이프라인, 상태 조회 API, 그리고 웹 뷰어 제공까지 모든 기능을 포함합니다. 비동기 `asyncio`와 세마포어(`sem`)를 사용하여 최대 2개의 재구성 작업만 동시에 실행되도록 제어하고 있습니다. COLMAP 명령어(특징 추출, 매칭, 재구성, 언디스토션)와 Gaussian Splatting(`train.py`, `metrics.py`) 스크립트 실행은 모두 백그라운드 태스크로 처리되며, 각 단계의 출력 로그는 `/data/jobs/{job_id}/run.log`에 기록됩니다. 작업 완료 시 COLMAP의 sparse 포인트 클라우드를 PLY 형식으로 변환하여 저장하고, `/v/{pub_key}` 경로로 접근 가능한 Three.js 기반 웹 뷰어 HTML을 통해 결과를 시각화합니다.

2. `setup.sh` – 환경 구축 및 설치 스크립트

```
#!/bin/bash
# Ubuntu 22.04 + CUDA 12.8 환경에서 COLMAP 및 Gaussian Splatting 설치 스크립트
# 이 스크립트를 root 권한으로 실행하거나, sudo를 사용하여 필요한 부분을 설치합니다.

echo "=== [1/5] 시스템 패키지 업데이트 및 필수 패키지 설치 ==="
sudo apt-get update
# COLMAP 빌드에 필요한 개발 라이브러리 설치 (Qt5 GUI 비활성화 가능하지만, 여기서는 포함하여 설치)
sudo apt-get install -y git build-essential cmake ninja-build \
  libboost-program-options-dev libboost-graph-dev libboost-system-dev \
  libeigen3-dev libfreeimage-dev libmetis-dev libgoogle-glog-dev \
  libgtest-dev libgmock-dev libsqlite3-dev libglew-dev \
  qtbase5-dev libqt5opengl5-dev libcgal-dev libceres-dev \
  libcurl4-openssl-dev libmkl-full-dev

# (참고) Ubuntu 22.04의 기본 CUDA 툴킷(GPU 가속)과 GCC 호환 문제 해결:
echo "=== [2/5] GCC 컴파일러 설정 (필요 시 gcc-10 사용) ==="
sudo apt-get install -y gcc-10 g++-10
export CC=/usr/bin/gcc-10
export CXX=/usr/bin/g++-10
export CUDAHOSTCXX=/usr/bin/g++-10
```

```
echo "=== [3/5] COLMAP 소스 다운로드 및 빌드 ==="
# COLMAP 저장소 클론 (v3.8 안정 버전 체크아웃 가능; 여기서는 최신 버전 사용)
git clone https://github.com/colmap/colmap.git ~/colmap
cd ~/colmap
git submodule update --init --recursive # 필요한 서브모듈 (glog, gtest 등) 초기화
mkdir build && cd build
# CMake 구성: GUI 비활성화 및 CUDA 아키텍처 설정 (native: 현재 머신 GPU에 최적화)
cmake .. -GNinja -DCMAKE_CUDA_ARCHITECTURES=native -DBUILD_GUI=OFF
# 컴파일 및 설치
ninja -j$(nproc)
sudo ninja install # colmap 명령을 /usr/local/bin 에 설치

# COLMAP 명령어 확인
if ! command -v colmap &> /dev/null; then
  echo "COLMAP 설치 실패: 'colmap' 명령을 찾을 수 없습니다."
  exit 1
fi
echo "COLMAP 설치 완료. 버전: $(colmap --version)"

echo "=== [4/5] Python 가상환경 및 Gaussian Splatting 설치 ==="
# Python3.10 venv 생성 및 활성화
python3 -m venv ~/gs-env
source ~/gs-env/bin/activate

# Python 패키지 업그레이드 및 PyTorch 설치 (CUDA 12.8 지원 버전)
pip install --upgrade pip
# PyTorch (CUDA 12.8 용) 설치 - PyTorch 공식 휠 경로 사용 (버전 2.x 가정)
pip install torch torchvision torchaudio --index-url https://
download.pytorch.org/whl/cu128

# Gaussian Splatting 레포지토리 클론 (서브모듈 포함)
git clone --recursive https://github.com/graphdeco-inria/gaussian-splatting.git
~/gaussian-splatting
cd ~/gaussian-splatting
# 요구 Python 패키지 설치
pip install -r requirements.txt
# (주의) pycolmap 설치 문제 시 특정 버전으로 재설치
pip install pycolmap==0.4.0

# Gaussian Splatting CUDA 확장 빌드
python setup.py build_ext --inplace

echo "=== [5/5] 환경 구성 완료 ==="
echo "가상환경 'gs-env'이 생성되었으며, COLMAP 및 Gaussian Splatting이 설치되었습니다."
echo "FastAPI 서버 코드(main.py) 실행 전에 'source ~/gs-env/bin/activate'로 가상환경을 활성화하세요."
echo "또한, /data/jobs 디렉토리가 존재하고 애플리케이션이 해당 경로에 쓸 수 있는지 확인하십시오."
```

스크립트 설명: `setup.sh` 스크립트는 Ubuntu 22.04 서버에서 COLMAP과 Gaussian Splatting 환경을 자동으로 설정합니다.

- 1단계에서는 COLMAP 컴파일에 필요한 의존 패키지(CMake, Boost, Eigen, FreeImage, Glog, GFlags, Ceres 등)를 apt로 설치합니다.
- 2단계에서는 CUDA 12.8 환경에서의 컴파일 호환을 위해 GCC-10을 설치하고 환경변수를 설정합니다 (Ubuntu 22.04 기본 GCC-11과 CUDA 빌드 호환성 이슈 대응).
- 3단계에서는 COLMAP 소스를 GitHub에서 클론하고, Ninja를 이용해 GUI 비활성화 옵션으로 빌드한 뒤 `ninja install`로 `colmap` 실행파일을 시스템에 등록합니다.
- 4단계에서는 Python 3.10 가상환경(`gs-env`)을 생성/활성화한 후, PyTorch (CUDA 12.8용), Gaussian Splatting 공식 레포지토리를 클론 및 Python 의존성 설치를 진행합니다. `python setup.py build_ext --inplace` 명령으로 Gaussian Splatting의 CUDA 확장 모듈들을 컴파일합니다.
- 5단계에서는 스크립트 완료 메시지와 사용 안내를 출력합니다.

이 스크립트를 실행한 이후에는 FastAPI 서버(`main.py`)를 실행하기 전에 `source ~/gs-env/bin/activate`로 가상환경을 활성화해야 하며, 서버 구동 시 COLMAP과 Gaussian Splatting (`convert.py`, `train.py`, `metrics.py` 등)이 정상적으로 호출될 수 있도록 환경이 갖춰집니다.

3. Three.js 기반 PLY 웹 뷰어 HTML (`/v/{pub_key}` 경로 출력 내용)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>3D Point Cloud Viewer</title>
  <style>
  body { margin: 0; background-color: #000; }
  canvas { display: block; }
 </style>
</head>
<body>
  <div id="viewer"></div>
  <!-- Three.js 및 OrbitControls, PLYLoader 불러오기 (CDN) -->
  <script src="https://unpkg.com/three@0.153.0/build/three.min.js"></script>
  <script src="https://unpkg.com/three@0.153.0/examples/js/controls/OrbitControls.js"></script>
  <script src="https://unpkg.com/three@0.153.0/examples/js/loaders/PLYLoader.js"></script>
  <script>
  // Three.js 장면 구성
  const scene = new THREE.Scene();
  scene.background = new THREE.Color(0x000000);
  const camera = new THREE.PerspectiveCamera(75, window.innerWidth/window.innerHeight, 0.1,
1000);
  const renderer = new THREE.WebGLRenderer({ antialias: true });
  renderer.setSize(window.innerWidth, window.innerHeight);
  document.getElementById('viewer').appendChild(renderer.domElement);
  // 카메라 컨트롤 설정
  const controls = new THREE.OrbitControls(camera, renderer.domElement);
```

```
    controls.update();
    // 조명 추가 (직사광 + Ambient)
    const dirLight = new THREE.DirectionalLight(0xffffff, 0.8);
    dirLight.position.set(1, 1, 1);
    scene.add(dirLight);
    scene.add(new THREE.AmbientLight(0xffffff, 0.2));
    // 창 크기 변경 대응
    window.addEventListener('resize', () => {
     camera.aspect = window.innerWidth / window.innerHeight;
     camera.updateProjectionMatrix();
     renderer.setSize(window.innerWidth, window.innerHeight);
    });
    // PLY 포인트 클라우드 로드
    const loader = new THREE.PLYLoader();
    loader.load('/recon/pub/【pub_key】/cloud.ply', geometry => {
     geometry.computeBoundingBox();
     const material = new THREE.PointsMaterial({ size: 1.5, vertexColors: true });
     const points = new THREE.Points(geometry, material);
     scene.add(points);
     // Bounding box 중심과 크기에 맞게 카메라 세팅
     const center = geometry.boundingBox.getCenter(new THREE.Vector3());
     const size = geometry.boundingBox.getSize(new THREE.Vector3());
     const maxDim = Math.max(size.x, size.y, size.z);
     camera.position.copy(center.clone().add(new THREE.Vector3(maxDim, maxDim, maxDim)));
     camera.lookAt(center);
     camera.far = maxDim * 10;
     camera.updateProjectionMatrix();
     controls.target.copy(center);
     controls.update();
    });
    // 애니메이션 루프 시작
    function animate() {
     requestAnimationFrame(animate);
     controls.update();
     renderer.render(scene, camera);
    }
    animate();
   </script>
  </body>
  </html>
```

위 HTML 코드는 FastAPI의 `/v/{pub_key}` 엔드포인트에서 반환되는 Three.js 기반 웹 뷰어 페이지의 내용입니다. 사용자가 업로드한 이미지들로부터 생성된 포인트클라우드(`cloud.ply`)를 Three.js의 `PLYLoader`를 통해 불러와 화면에 렌더링합니다.

- **구성 요소:** Three.js 기본 라이브러리와 OrbitControls(사용자가 마우스로 3D 모델을 회전/줌 조작) 및 PLYLoader(PLY 파일 파싱 로더)를 CDN에서 불러옵니다.
- **렌더링:** `THREE.PointsMaterial`을 사용하여 포인트 클라우드를 컬러 버텍스로 표시하며, 배경을 검정색으로 설정하고 기본 조명을 추가했습니다.

- **카메라 설정:** 로드된 포인트들의 bounding box(경계 상자)를 계산하여, 그 중심을 orbit controls의 타겟으로 설정하고 bounding box 대각선 길이에 기반해 카메라를 적절한 거리로 배치합니다. 이렇게 함으로써 모델이 화면에 전체가 보이도록 자동 프레이밍합니다.
- **반응형:** 창 크기가 변경되면 `resize` 이벤트를 처리하여 카메라와 렌더러 크기를 업데이트합니다.

해당 HTML은 `{pub_key}`에 해당하는 공개 키의 PLY 데이터를 `/recon/pub/{pub_key}/cloud.ply` 경로로 요청하여 로딩하므로, FastAPI 서버에서 이 경로를 통해 정적 PLY 파일이 제공되어야 합니다. (우리 `main.py`에서는 `download_ply` 함수로 구현되어 있습니다.)

4. COLMAP points3D.txt → PLY 변환 함수

```
def convert_points3d_to_ply(points3d_txt: Path, ply_path: Path):
    """
    COLMAP에서 생성된 points3D.txt 파일을 읽어 PLY (ASCII 포인트 클라우드) 파일로 저장합니다.
    """
    with open(points3d_txt, 'r') as f_txt, open(ply_path, 'w') as f_ply:
        # PLY 파일 헤더 작성
        points = []
        for line in f_txt:
            if line.startswith('#') or line.strip() == '':
                continue
            parts = line.split()
            if len(parts) < 7:
                continue
            # COLMAP points3D.txt: ID, X, Y, Z, R, G, B, error, track...
            X, Y, Z = map(float, parts[1:4])
            R, G, B = map(int, parts[4:7])
            points.append((X, Y, Z, R, G, B))
        f_ply.write("ply\nformat ascii 1.0\n")
        f_ply.write(f"element vertex {len(points)}\n")
        f_ply.write("property float x\nproperty float y\nproperty float z\n")
        f_ply.write("property uchar red\nproperty uchar green\nproperty uchar blue\n")
        f_ply.write("end_header\n")
        # 포인트 좌표 및 색상 데이터 작성
        for (x, y, z, r, g, b) in points:
            f_ply.write(f"{x:.6f} {y:.6f} {z:.6f} {r} {g} {b}\n")
```

함수 설명: 이 Python 함수는 COLMAP의 `points3D.txt` 파일을 읽어서 3D 포인트 좌표와 색상을 추출한 뒤, 이를 PLY 형식의 ASCII 파일로 저장합니다. COLMAP의 sparse 재구성 결과(`points3D.txt`)에는 각 3D 포인트의 좌표(X, Y, Z)와 색상(R, G, B)이 포함되어 있으므로, 이러한 정보를 이용해 표준 PLY 헤더와 포인트 리스트를 작성합니다. 생성된 `cloud.ply` 파일은 Three.js 등의 외부 뷰어에서 바로 읽어들일 수 있는 포맷입니다.

이 함수는 `main.py`의 파이프라인 마지막 단계에서 호출되며, `/data/jobs/{job_id}/work/sparse/0/points3D.txt` → `/data/jobs/{job_id}/export/cloud.ply` 변환에 사용됩니다.

5. README.md – 사용 예시 및 간단 안내

 # Gaussian Splatting 기반 3D 재구성 API 서버

본 프로젝트는 업로드된 다수의 이미지로부터 COLMAP (Structure-from-Motion)을 통해 3D 점군을 추출하고, **3D Gaussian Splatting** 기법으로 신경 방사장 모델을 학습한 뒤 결과를 웹에서 시각화하는 FastAPI 기반의 백엔드 서버입니다.

기능 개요

- **이미지 업로드 및 3D 재구성** – `/recon/jobs [POST]`: 사용자가 여러 장의 사진을 업로드하면 새로운 재구성 작업이 생성됩니다. 응답으로 `job_id` (작업 식별자)와 `pub_key` (공개 결과 조회 키)가 반환됩니다.
- **작업 진행 상태 조회** – `/recon/jobs/{job_id}/status [GET]`: 특정 작업의 현재 상태 (`PENDING`, `RUNNING`, `DONE`, `ERROR`), 최근 로그 내용 및 결과 뷰어 URL(완료 시)을 제공합니다.
- **3D 결과 웹 뷰어** – `/v/{pub_key} [GET]`: Three.js를 사용한 WebGL 페이지로, 해당 작업의 3D 포인트클라우드 결과를 브라우저에서 인터랙티브하게 볼 수 있습니다.
- **포인트클라우드 다운로드** – `/recon/pub/{pub_key}/cloud.ply [GET]`: 결과로 생성된 PLY 포인트클라우드 파일을 직접 다운로드하거나 뷰어에서 불러올 수 있는 엔드포인트입니다.

설치 및 실행

1. **환경 설정**: Ubuntu 22.04 시스템에서, `setup.sh` 스크립트를 실행하여 의존성 라이브러리, COLMAP, Gaussian Splatting 등을 모두 설치합니다.
   ```bash
   sudo bash setup.sh
   ```
   이 스크립트는 Python 가상환경 `gs-env`도 생성하므로, 설치 완료 후 다음을 통해 활성화하세요:
   ```bash
   source ~/gs-env/bin/activate
   ```

2. **서버 실행**: 가상환경 활성화 후, `main.py`를 실행하여 FastAPI 서버를 시작합니다.
   ```bash
   python main.py
   ```
   서버가 uvicorn을 통해 `http://0.0.0.0:8000` (기본)에서 실행됩니다. Swagger UI를 통해 API 문서를 확인할 수 있습니다 (브라우저에서 `/docs` 접속).

3. **API 사용 예시**:
   - **이미지 업로드 및 재구성 요청**: 예를 들어 `images` 폴더 내 JPG 파일들을 업로드하려면:
     ```bash
     curl -X POST "http://localhost:8000/recon/jobs" \
       -F "files=@images/img1.jpg" \
       -F "files=@images/img2.jpg" \
       -F "files=@images/img3.jpg"
     ```
     응답:

```json
{"job_id": "abcd1234", "pub_key": "0123456789"}
```
- **상태 확인**:
```bash
curl http://localhost:8000/recon/jobs/abcd1234/status
```
응답 예:
```json
{
 "job_id": "abcd1234",
 "status": "RUNNING",
 "log_tail": [
  ">> [3/6] Sparse reconstruction (SfM) 시작...\n",
  "... (중략) ...",
  ">> [4/6] Undistorting images 및 변환된 모델 생성...\n"
 ]
}
```
(작업이 `DONE` 상태가 되면 `viewer_url` 필드에 `/v/0123456789`와 같은 뷰어 URL이 포함됩니다.)
- **웹 뷰어 확인**: 상태 응답에서 `viewer_url`을 확인하거나, 직접 브라우저에서 `http://localhost:8000/v/0123456789`로 접속하면 Three.js 기반의 3D 뷰어에서 포인트클라우드 결과를 볼 수 있습니다. 좌클릭 드래그로 회전, 우클릭 드래그로 팬, 스크롤로 줌인이 가능합니다.

디렉토리 구조

작업 별로 `/data/jobs/{job_id}` 폴더가 생성되며, 아래와 같은 구조로 데이터가 저장됩니다:
```plaintext
/data/jobs/abcd1234/
├── upload/
│   └── images/              # 업로드된 원본 이미지들
├── colmap/
│   ├── database.db          # COLMAP 데이터베이스 (특징, 매칭 저장)
│   └── sparse/0/            # COLMAP SfM 결과 (binary 모델)
├── work/
│   ├── images/              # COLMAP undistorter로 생성된 보정 이미지들
│   └── sparse/0/            # COLMAP undistorter 결과 (텍스트 파일: cameras.txt, images.txt, points3D.txt)
├── model/                   # Gaussian Splatting 모델 결과 (예: 체크포인트 등) ※학습 스크립트 기본동작에 따라 생성
├── metrics/
│   └── report.json          # metrics.py에서 출력된 성능 리포트 (PSNR/SSIM 등)
├── export/
│   └── cloud.ply            # 시각화용으로 변환된 포인트 클라우드 (PLY ASCII)
├── key.txt                  # 공개 조회 키 (pub_key)을 저장
└── run.log                  # 실행 로그 (COLMAP 및 GS 출력 내용)
```

참고 및 추가 정보

- COLMAP와 Gaussian Splatting의 실행 과정에서 시간이 오래 걸릴 수 있습니다. 특히 Gaussian Splatting (`train.py`) 학습은 수 분에서 수 시간까지 소요될 수 있으며, 본 서버 구현에서는 기본 설정으로 전체 학습을 수행합니다.
- 동시 실행 제한: 서버는 내부적으로 **세마포어**로 제어되어 동시에 최대 2개의 재구성 작업만 처리합니다. 대기 중인 작업은 이전 작업이 끝날 때까지 `PENDING` 상태로 대기합니다.
- 오류 처리: 재구성 파이프라인 도중 오류 발생 시 해당 작업의 상태는 `ERROR`로 표시되며, `/recon/jobs/{job_id}/status` 응답의 `log_tail` 또는 `run.log` 파일을 통해 상세 원인을 확인할 수 있습니다.
- 웹 뷰어는 Three.js로 구현되어 포인트만을 렌더링하며, Gaussian Splatting으로 생성된 방사장(gaussian splats)은 포인트 클라우드로 단순화하여 보여줍니다. 보다 정교한 시각화(예: 광택 표현)는 GS의 전용 뷰어나 NerfStudio 등의 도구를 활용해야 합니다.

```

---
```