

쉬운설명_KMZ에서_영상표출까지_전과정

KMZ 파일에서 영상 위 이름표까지 — 전 과정 그림 설명서

이 글은 **초등학생도 이해할 수 있게** 그림과 이야기로 쓴 설명서예요. "지도 파일(KMZ) 하나가 어떤 여행을 거쳐, 드론 영상 속 **정확한 자리에** 다리·터널 이름표로 나타날까?" 그 여정을 **8단계**로 하나씩 따라가 봅니다. (파일·줄 번호는 2026-07-03 기준)

한눈에 보는 전체 여정

KMZ는 "지도 위 핀들을 모아 둔 **선물 상자**"예요. 이 상자가 열려서, 안의 핀 하나하나가 영상 화면의 정확한 픽셀에 이름표로 붙기까지 — 전체 흐름은 이렇습니다.

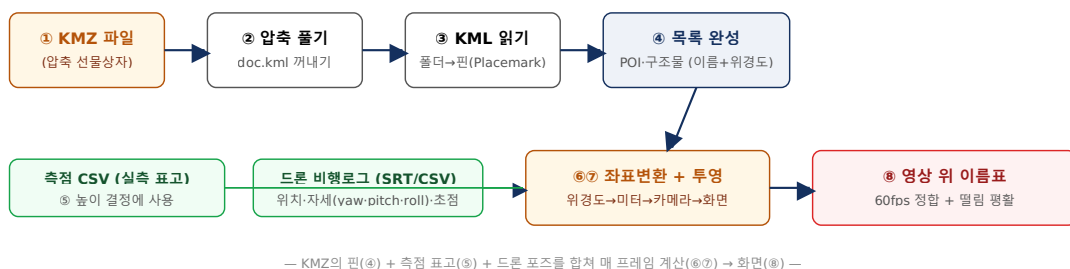


그림 1. KMZ → 압축해제 → 파싱 → 목록 → (표고·드론 포즈 결합) → 좌표변환·투영 → 영상 표출

1단계 — KMZ는 '압축된 선물 상자'

- **KMZ = KML을 ZIP으로 압축한 것**이에요. 상자(KMZ)를 열면 안에 설명서(doc.kml)가 들어 있어요.
- 사용자가 폴더를 선택하면, 프로그램은 폴더 안에서 **.kml** (맨 설명서)이 있으면 그걸 먼저 쓰고, 없으면 **.kmz** (상자)를 찾아 엽니다.
- 압축 풀기는 브라우저 안에서 **fflate** 라이브러리의 **unzipSync** 로 처리해요 — **서버에 보내지 않고 내 컴퓨터에서 바로** 풉니다.

코드: [geoData.ts:295](#) `parseKml()` — `.kml` 우선, 없으면 `.kmz` 를 `unzipSync` 로 해제해 `doc.kml` 을 꺼냄. 실패하면 경고 후 건너뛴.

2단계 — 설명서(KML) 속은 '폴더 나무 + 핀'

KML은 XML이라는 형식의 글이에요. 그 안은 **폴더 나무** 모양이고, 앞서귀마다 **핀(Placemark)** 이 달려 있어요.



그림 2. (왼쪽) 폴더 나무를 따라 내려가며 핀 수집 · (오른쪽) 핀 1개에서 이름·속성표·좌표를 추출

프로그램은 나무를 **재귀적으로**(가지 끝까지) 걸어 내려가면서, 각 핀이 **어느 폴더에 속했는지**를 함께 기억해요.

코드: [geoData.ts:376](#) `walk()` — 폴더 트리 재귀. [geoData.ts:276](#) `parseKmlDescProps()` — 속성표(HTML `<tr><td>`) 파싱. 구글어스 재저장본의 태그 변형(`<tbody>`, `>` 엔티티 등)도 견고하게 처리.

3단계 — 핀을 종류별로 나누기

폴더 이름이 곧 분류 기준이에요. 학교에서 반 나누듯이:

폴더	분류	어디에 표시되나
03)교량 · 04)터널 · 06)구교	구조물 (bridge/tunnel)	영상 라벨 + 하단 스테이션바 마커
02)지장물 · 05)출입문	POI (지점)	영상 라벨
철도역 (KAKAO_RAIL)	POI + 역사 구조물	영상 라벨 + 스테이션바 양쪽

- 이때 속성표에서 **연장(m)**, **시설종별(1종/2종/3종)** 같은 값도 같이 담아요 → 나중에 시설등급 필터(1·2종만 보기)와 팝업 정보에 쓰입니다.

- 결과: `pois[]` (지점 목록) + `structures[]` (구조물 목록). **KMZ가 이 데이터의 유일한 원본**이라, 없으면 "데이터 누락" 경고를 띄워요.

코드: [geoData.ts:331](#) `handlePlacemark()` — 폴더명 분기. [geoData.ts:539](#)
`loadGeoData` — KMZ 누락 시 `kmzMissing` 경고.

4단계 — 높이(표고) 정하기: "핀은 몇 m 높이에 있나?"

KMZ의 핀은 **평면 위치(위도·경도)**는 정확하지만, **높이(z)**는 없거나 부정확한 경우가 많아요. 높이가 틀리면 이름표가 하늘에 뜨거나 땅에 파묻혀 보여요. 그래서 이렇게 정합니다.

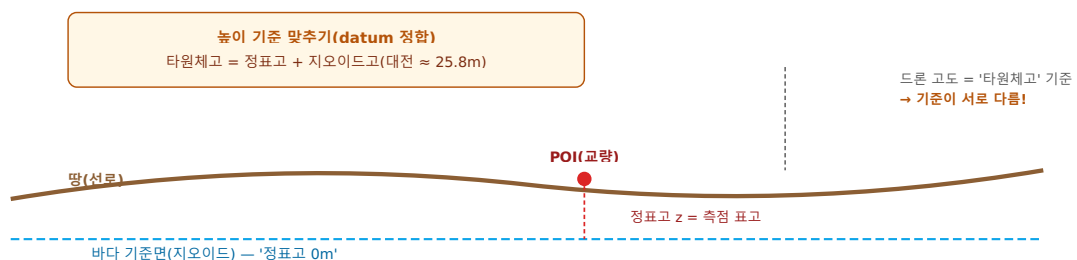


그림 3. 핀의 높이는 **가장 가까운 측정점의 실측 표고**를 빌려 쓰고, 드론 고도와 기준을 맞추기 위해 지오이드고(+25.8m)를 더한다

높이 결정 우선순위 (셋 중 위에서부터):

1. **드래그 보정값** — 사용자가 화면에서 직접 고친 값 (5단계 뒤 '보너스' 참고)
2. **DEM 자동 조회값** — open-meteo API로 받은 그 지점 지면고도 (선택 기능)
3. **최근접 선로(측점) 표고** — 기본값. 측정 CSV의 **실측 정표고**를 빌려 씀

이 "측점 표고 빌려 쓰기" 덕분에 **비싼 지형데이터(DEM) 없이도** 하향각 오차가 42.7°→11.0°로 줄었어요.

코드: [StationOverlay.tsx:767](#) — 지면고도 gz 결정. [geoProjection.ts](#) `geoidOffset` — datum 정합.

5단계 — 좌표 바꾸기: 주소(위경도)를 '미터 자'로

위도·경도는 **둥근 지구 위의 주소**라서, 그대로는 "몇 m 옆에 있나"를 계산하기 어려워요. 그래서 3번 알아탑니다.



그림 4. 위경도 → 한국 TM 평면(m) → 드론 기준 상대좌표 → 카메라 좌표, 3번의 갈아타기

- **위경도 → TM(m)**: proj4 라이브러리로 한국 표준 평면좌표(EPSG:5186)로 변환. 이제 "미터"로 잴 수 있어요.
- **TM → 드론 기준**: 핀 위치에서 드론 위치를 빼면, "드론에서 동쪽 40m, 북쪽 12m, 아래 30m" 같은 상대 위치가 나와요.

코드: [geoProjection.ts:75](#) EPSG:5186 정의, [geoProjection.ts:334](#) `projectPoint()` 1부 — ENU 상대좌표 계산.

6단계 — 카메라의 눈으로 회전: "드론이 보는 방향은?"

드론이 어느 쪽을 보고 있는지(비행로그의 **yaw·pitch·roll**)에 따라, 같은 핀도 화면 왼쪽에 보일 수도, 오른쪽에 보일 수도 있어요. 그래서 상대좌표를 **회전행렬**로 돌려서 "카메라가 보는 세계"의 좌표 (X_c, Y_c, Z_c)로 바꿉니다.

$R = R_z(-yaw) \times R_x(pitch) \times R_y(roll)$ ← 드론의 고개 돌림·고덕임·가웃을 그대로 재현

$(X_c, Y_c, Z_c) = \text{카메라축변환} \times R^T \times (\text{동}, \text{북}, \text{위})$

- **Z_c** 는 "카메라 앞쪽으로 몇 m인가"예요. **$Z_c \leq 0$ 이면 카메라 뒤** → 그리지 않아요.
- 너무 멀거나(거리 필터) 화각(시야) 밖이어도 걸러냅니다.

코드: [geoProjection.ts:301](#) `toCameraCoords()`, 회전 수식 주석은 [geoProjection.ts:319~](#).

7단계 — 핀홀 투영: 3D 세상을 2D 사진으로

카메라는 **바늘구멍 사진기**와 같아요. 3D 점이 구멍(렌즈)을 지나 필름(화면)에 맺힙니다.

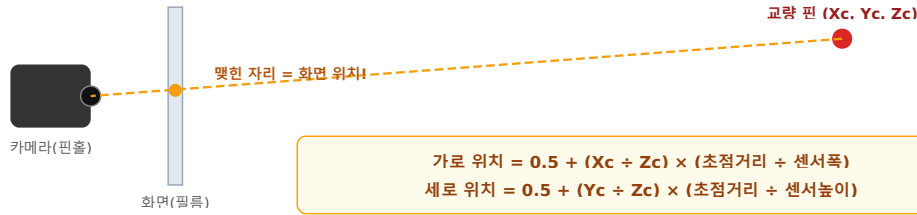


그림 5. 핀홀 투영 — "멀수록($Z_c \uparrow$) 가운데로 모이고, 옆에 있을수록($X_c \uparrow$) 가장자리로" 라는 원근법 그 자체

- 결과는 **0~1 사이의 정규 좌표**예요. (0.5, 0.5)면 화면 정중앙.
- $\div Z_c$ 가 바로 **원근법**이에요 — 멀리 있는 것일수록 화면 가운데 근처로 작게 모여요.
- 초점거리(focal)는 비행로그에서 프레임마다 읽고, 센서 크기는 16:9 영상 기준(36×20.25mm)을 써요. 영상마다 화각이 다르면 **라벨 1개 드래그로 세로 화각을 역산**해 맞출 수도 있어요 (FOV 1점 보정).

코드: [geoProjection.ts:126](#) `pixelFromCamera()` — 위 공식 그대로.

8단계 — 화면에 붙이기: 액자 맞추기 + 떨림 잡기

마지막으로 "0~1 정규 좌표"를 진짜 **화면 픽셀**로 바꿔 이름표를 놓아요. 여기엔 두 가지 마무리가 있어요.

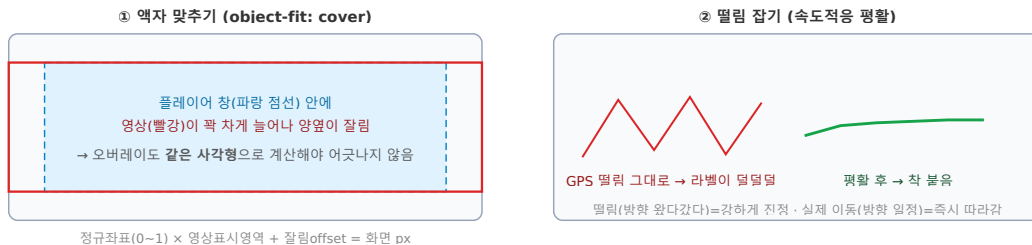


그림 6. (왼쪽) 영상이 창에 '딱 차게' 잘려 보이므로 오버레이도 같은 규칙으로 정렬 · (오른쪽) 떨림만 골라 진정시키는 평활

1. **액자 맞추기(cover 정렬)** — 영상은 플레이어 창을 비율 유지한 채 **딱 채우며 잘려** 보여요. 그래서 이름표도 "창 기준"이 아니라 "**잘린 영상 사각형 기준**" 으로 놓아야 정확히 붙어요.
2. **떨림 잡기(평활)** — GPS·자세값은 미세하게 떨려요. "방향이 왔다갔다(=떨림)"면 강하게 진정시키고, "방향이 일정(=실제 이동)"하면 즉시 따라가는 **속도적응 평활**로, 지연 없이 떨림만 없앱니다.
3. 이 계산 전부를 매 화면 프레임(**60fps, requestAnimationFrame**)마다 다시 해요 — 그런 데도 빠른 이유는, 무거운 준비(가시성 선별)는 미리 해 두고 프레임마다 **가벼운 재투영**만 하기 때문이에요.

코드: [StationOverlay.tsx:948](#) cover 정렬, [StationOverlay.tsx:49](#) `smoothStep()` 평활.

보너스 — 그래도 어긋나면? "드래그 한 번"

지오코딩(주소→좌표 변환) 자체에 오차가 있으면 이름표가 조금 비껴 붙을 수 있어요. 그럴 땐 **이름표를 마우스로 끌어 제자리에 놓기만 하면** 돼요.

- 화면에서 끈 위치(2D)를 7단계 공식의 **역방향**으로 계산해(역투영), 실제 좌표(위경도+높이)를 되찾아 저장해요.
- 2D→3D는 원래 답이 무한히 많지만(깊이를 모르므로), "**카메라와의 거리는 그대로**" 라는 약속을 뒤서 드래그 한 번으로 가로·세로·높이가 **동시에** 맞춰집니다.
- 고친 값은 데이터셋별로 저장돼 다음에 열어도 유지돼요. (왕복 검증 오차 ≈ 0)

코드: [geoProjection.ts:149](#) `worldFromPixel()` — 역투영.

단계별 요약표

단계	하는 일	비유	코드 위치
1	KMZ 압축 해제 → doc.kml	선물상자 열기	geoData.ts <code>parseKml</code>
2	KML 폴더 나무 걷기, 핀 수집	설명서 읽기	geoData.ts <code>walk</code>
3	폴더명으로 POI/구조물 분류	반 나누기	geoData.ts <code>handlePlacemark</code>
4	높이 결정(측점 표고) + 기준 맞춤(+25.8m)	키 재기, 자 통일	StationOverlay + <code>geoidOffset</code>
5	위경도 → 평면 m → 드론 기준 상대좌표	주소를 미터 자로	geoProjection.ts <code>projectPoint</code>
6	yaw·pitch·roll 회전 → 카메라 좌표	카메라 눈으로 보기	geoProjection.ts <code>toCameraCoords</code>
7	핀홀 투영 → 화면 정규좌표 (0~1)	바늘구멍 사진기	geoProjection.ts <code>pixelFromCamera</code>
8			StationOverlay.tsx

단 계	하는 일	비유	코드 위치
	cover 정렬 + 평활 + 60fps 렌더	액자 맞추고 떨림 잡기	
★	어긋나면 드래그 1회 역투영 보정	이름표 손으로 옮기기	geoProjection.ts <code>worldFromPixel</code>

더 깊은 수식·구현 근거: 「기술명세_GhiVideo_종합기술문서」·「구현상세_드론좌표_영상투영_소스코드매칭」 참조.