

25.11.21 Pixel-wise Forward Z-buffer 파이프라인 구조와 동작 개요

이 문서는 `C:\Users\jscool\uav_pipeline_codes\25.11.21\` 디렉터리의 **Pixel-wise Forward Z-buffer** 파이프라인을 한눈에 볼 수 있도록 정리한 것이다. 파이프라인의 목적은 UAV 영상에 대한 전력선 등 객체의 위치를 점군에서 찾아내기 위한 **픽셀단위 Forward 투영**을 수행하고, GPU 가속으로 효율을 높이는 것이다.

파일 및 폴더 구성

주요 파일과 역할을 요약하면 다음과 같다.

- **Python 실행 스크립트**
 - `run_pixelwise.py` – 단일 사이트용 실행 스크립트로 모드(single, top-K, test)를 받아 픽셀별 Forward 투영을 수행한다 [12 † L97-L118]. K는 인자로 지정하거나 모드에 따라 1 또는 10으로 설정된다 [10 † L262-L293].
 - `run_all_sites.py` – 여러 사이트를 일괄 처리하는 스크립트로, 사이트 리스트, 별별 처리 개수, 샘플링 크기 및 이미 완료된 사이트 건너뛰기 옵션을 제공한다 [12 † L153-L162]. 각 사이트마다 `run()` 함수를 호출하여 처리하며 완료된 사이트를 건너뛴다 [12 † L171-L188].
- **Windows 배치 파일**
 - `RUN_PIXELWISE_TEST.bat`, `RUN_PIXELWISE_ALL_TEST.bat` – 샘플링된 점군(예: 1 M points)으로 단일 또는 전체 사이트를 테스트 실행한다.
 - `RUN_PIXELWISE_ALL_SITES.bat` – 모든 사이트를 전체 점군으로 처리하는 배치 스크립트.
 - `RUN_PIXELWISE_SINGLE.bat`, `RUN_PIXELWISE_TOP10.bat`,
`RUN_PIXELWISE_COMPARE.bat` – 각각 K=1, K=10 실행 또는 두 모드 비교용 테스트를 제공한다.
- **핵심 모듈**
 - `part3_forward_pixelwise.py` – Forward 투영과 Z-버퍼 투표를 수행하는 핵심 모듈로, GPU 버전과 CPU 버전을 모두 포함한다 [14 † L217-L254].
 - `camera_io.py` – Pix4D의 report.xml 또는 camera_db.json에서 카메라 외부·내부 파라미터를 불러와 `cam_db` 사전을 생성한다 [16 † L566-L584].
 - `camera_calibration.py` – 사이트별 카메라 내부파라미터(K 행렬)를 반환한다 [16 † L638-L646].
 - `gsd_parser.py` – 사이트의 GSD를 기반으로 수평·수직 허용오차(`h_tol`, `v_tol`)를 계산한다 [16 † L659-L677].
 - `color_gate.py` – 전력선 검출 영역만 True로 하는 마스크를 생성하는 함수 `img_mask()`를 제공한다. 예시 구현은 없지만 실제 환경에서는 딥러닝 모델이나 색상 게이트를 사용한다 [16 † L608-L616].
 - `constants.py` – GPU 사용 여부, 출력 디렉터리, 이미지 디렉터리, GSD 기본값, LAS 클래스 코드, 투표 임계값 등을 정의한다 [16 † L694-L750] [5 † L229-L236].
- **문서**
 - `README.md` – 파이프라인 전체 개요.
 - `PIXELWISE_GUIDE.md` – 실행 방법과 옵션 설명.
 - `25.11.20_Forward_Pixelwise_SingleHit_ZBuffer.md` – Z-버퍼 방식 Single-hit 실행에 관한 기술 설명.
 - `CODE_DOCUMENTATION.md` – 코드 구조와 함수 설명.
- **입력 데이터 파일**

- Site_A_Images_EOPs.txt, Site_B_C_Images_EOP.txt - 각 사이트의 이미지 EXIF/EOP 정보 목록.

파이프라인 아키텍처 및 데이터 흐름

1. **실행 인터페이스** - 사용자는 run_pixelwise.py 또는 run_all_sites.py를 통해 파이프라인을 호출한다. 명령행 인자로 K_max(픽셀당 최대 히트 수), 사이트 이름, 샘플링 크기 등을 지정한다 【10 † L262-L293】.
2. **포인트클라우드 로드 및 샘플링** - part2 단계에서 생성된 LAS를 NPY로 캐시한 점군을 불러오고, --sample 인자가 0보다 크면 무작위 샘플링을 적용한다 【14 † L234-L242】.
3. **카메라 파라미터 로드** - camera_io.load_camera_db(det_dir) 가 Pix4D report.xml 또는 camera_db.json 을 파싱하여 각 이미지의 카메라 중심(C), 회전 행렬(R)과 내부파라미터(K)를 읽어온다 【16 † L566-L584】.
4. **GSD 허용오차 계산** - gsd_parser.get_tolerance(site_name) 이 사이트의 GSD에서 수평 1×GSD, 수직 3×GSD 값을 반환한다 【16 † L659-L677】. 이 오차는 광선과 점의 인접 판정 기준으로 사용된다.
5. **Uniform Grid 생성** - GPU 모드에서 create_uniform_grid_gpu() 가 점군을 격자 셀로 분할한다. 각 셀에 속한 점 인덱스 목록을 만들어 광선과 교차하는 셀만 탐색할 수 있게 한다 【14 † L343-L352】.
6. **픽셀별 처리 루프** - 각 이미지에 대해 검출 마스크를 적용하여 관심 픽셀 좌표들을 추출한 뒤, 카메라 내부파라미터 K⁻¹와 회전행렬 R을 사용해 픽셀별 광선(ray)을 계산한다 【14 † L307-L315】.
7. **Z-버퍼 Top-K 선택** - process_rays_with_zbuffer_gpu() 가 광선과 Uniform Grid를 이용해 가장 가까운 K개 점을 선택한다. K=1이면 Single-Hit 모드, K>1이면 Top-K 모드이다 【14 † L312-L319】. 각 광선 별로 선택된 점의 인덱스를 반환한다.
8. **투표 집계** - 선택된 점들의 투표 수를 cp.add.at 를 이용해 GPU 배열에 누적하고, 모든 이미지가 끝나면 CPU로 가져와 votes.npy 로 저장한다 【14 † L319-L323】. 최종적으로 vote ≥7, ≥15, ≥30에 해당하는 점들을 필터링하여 vote_7.las, vote_15.las, vote_30.las 파일로 출력한다.

주요 알고리즘 및 설계

- **Ray 생성** - 픽셀 좌표 (u, v, 1)에 대해 카메라 내부행렬의 역행렬 K⁻¹을 곱하고, 회전행렬 R^T를 적용하여 월드 좌표계의 방향벡터를 얻는다 【14 † L307-L315】. 이를 정규화하여 ray 방향벡터 rays_world 를 구한다.
- **Uniform Grid** - 3차원 공간을 cell_size=h_tol 크기의 격자로 나누고, 각 점의 셀 인덱스를 계산하여 cell_id → point_indices 사전을 만든다 【14 † L343-L352】. 광선은 DDA 방식으로 셀을 이동하며 후보 포인트를 찾는다.
- **Z-버퍼 Top-K** - 광선과 인접한 셀 내의 점들 중 광선과 수평거리 h_tol, 깊이거리 v_tol 이내의 점을 찾고, 거리순으로 정렬해 최대 K개를 유지한다 【14 † L312-L319】. K=1일 경우 첫 번째 점을 찾으면 종료한다.
- **GSD 기반 허용오차** - 사이트별 GSD값을 기준으로 하여 수평 1×GSD, 수직 3×GSD 오차를 허용함으로써 포인트와 픽셀 사이의 위치 오차를 보완한다 【16 † L659-L677】.

출력 파일 및 결과 해석

- forward_votes.npy - 모든 점군에 대해 투표 횟수를 저장한 1차원 numpy 배열. 배열 인덱스는 점군의 순서와 일치한다. 예: votes[10] = 5는 10번쨰 점이 5표를 받았음을 의미한다 【18 † L797-L804】.
- forward_timing_k{k}.csv - 이미지별 처리 시간을 기록한 CSV 파일로, 이미지 인덱스, 파일명, 처리 픽셀 수, 소요 시간이 포함된다 【18 † L790-L797】.
- vote_7.las, vote_15.las, vote_30.las - 각각 7표, 15표, 30표 이상 받은 포인트만 추출한 LAS 파일로, 원본 점의 XYZ 및 RGB 정보를 포함한다. 이 LAS들의 분류코드는 constants.LAS_CLASS_POWERLINE 에 정의된 값(예: 14)으로 설정된다 【5 † L234-L237】.

설정 및 조정

- **GPU 사용 여부** – `constants.py`에서 CuPy 초기화를 시도하여 GPU 사용 가능 시 `USE_GPU=True`로 설정한다 【16 † L694-L714】 . 실패하면 CPU 모드로 폴백하며 속도는 크게 느려진다.
- **출력 경로** – 결과는 기본 출력 디렉터리 `C:\Users\jscool\uav_pipeline_outputs\part3_las\{site_name}`에 저장된다 【16 † L715-L723】 .
- **투표 임계값** – `constants.VOTE_THRESHOLDS = [7, 15, 30]`으로 정의되어 있어 각 임계값별로 LAS 파일을 생성한다 【5 † L229-L236】 . 필요하다면 `thresholds`를 수정하여 필터 강도를 변경할 수 있다.

결론

25.11.21 풀더는 GPU 가속 **픽셀별 Forward 투영** 파이프라인을 구성하는 실행 스크립트와 핵심 모듈, 도우미 모듈, 문서 파일들로 구성되어 있다. 사용자는 `run_pixelwise.py` 또는 `run_all_sites.py`를 통해 사이트별 또는 전체 사이트를 처리할 수 있으며, GSD 기반 허용오차와 Uniform Grid, Z-버퍼 Top-K 알고리즘을 활용하여 포인트 클라우드에서 객체를 효율적으로 식별한다. 결과는 투표 배열과 임계값별 LAS 파일로 저장되어 후속 분석이나 시각화에 활용된다.
