3D TileSet에서 건물 객체화 방법

**① 평면의 다각형 영역으로 모델을 자르기(CookieCutting 방식)**

원리*\* 이전 PPT참고*

다각형 영역을 아랫면으로 하는 각기둥을 생성하여 collision check

🡪감지된 각 GameObject에 대해서, vertex가 다각형 내외부인지 체크

🡪각 mesh triangle에 대해 삼각형의 세 정점의 내외부 여부에 따라 처리

다각형 영역을 얻는 방법

1) 사용자가 직접 클릭을 통해 다각형 영역 생성 (현실적으로 힘들다)

2) 클릭한 지점의 위치정보를 통해 DB에서 해당 건물의 영역 데이터를 수신

*\*DB에서 얻은 영역 정보를 유니티 씬에 다시 불러왔을 때 어느정도 오차가 존 재할 가능성이 높다. 엄밀하게 오차를 보정하는 프로세스를 도입할 수 있다면 좋겠지만, 일단은 받아온 영역에 padding을 입히는 것으로 목표건물의 일부가 영역에 포함되지 않는 문제를 해결했다.*

3) 클릭한 지점의 항공사진에서 AI segmentation으로 건물 영역 data 생성

하지만 위 방법은 DB에 다각형 영역에 대한 data가 존재하지 않거나, segmentation이 잘 이루어지지 않은 경우 사용하기 힘들다는 단점이 존재.

그리고, 객체화된 오브젝트를 추후에 Simplify 할 때 어차피 각 면(surface)을 추출해야함.

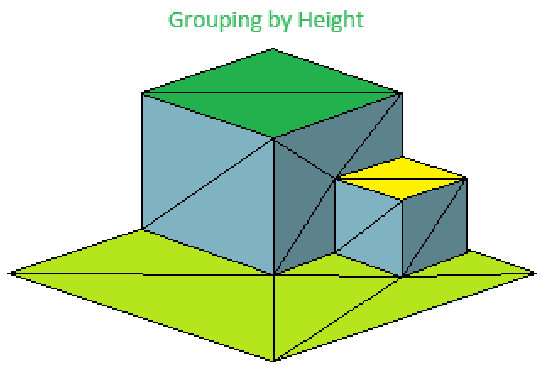
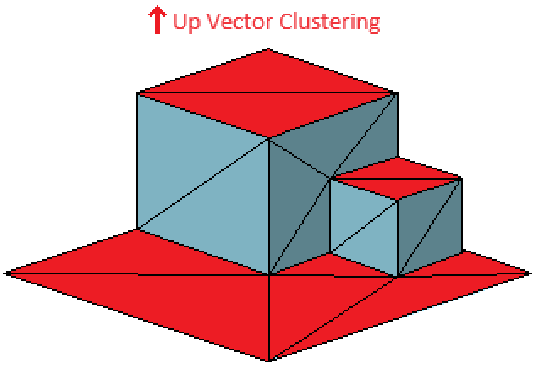
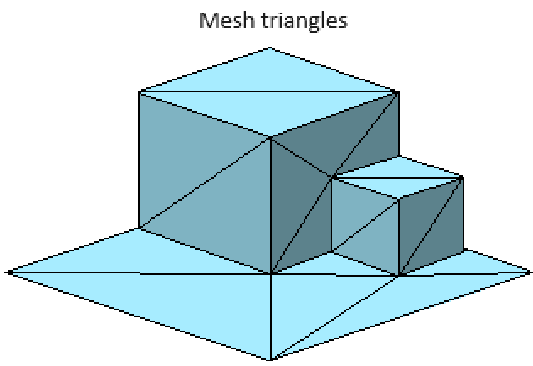
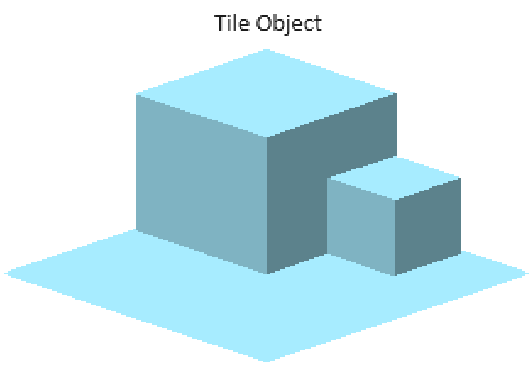
∴ 객체화 단계에서부터 면을 추출하는 방식으로 진행하려고 시도 ↓

**② mesh Clustering을 통해 면을 추출하여 객체화**

원리

Mesh 삼각형들을 normal Vector를 기준으로 clustering

🡪각 cluster에서도 위치값을 기준으로 grouping



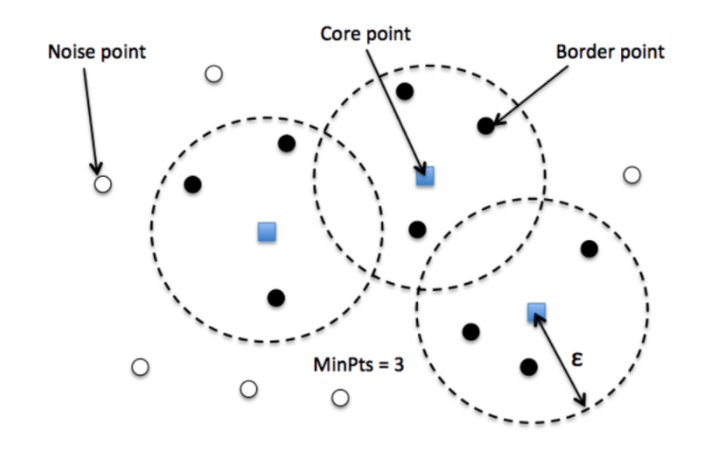
🡪지면에 해당하는 group을 제거, 건물에 해당하는 면만 취합하여 건물 객체화

**주요 논점**

1) clustering 방법?

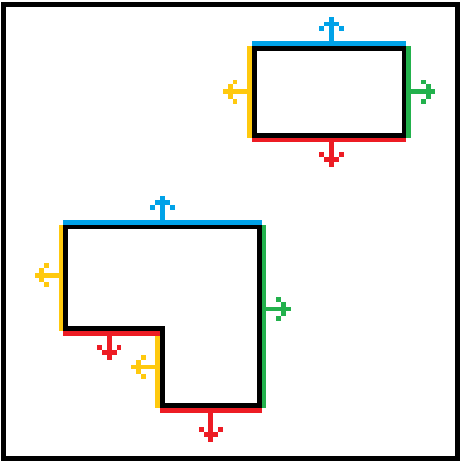
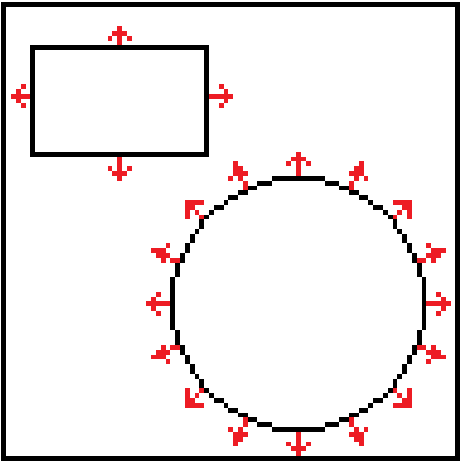
**접근1.** 모든 mesh 삼각형의 normal Vector를 기준으로 DBSCAN을 한다

DBSCAN이란 밀도 기반 군집화 기법이다.



\* 위를 바라보는 면을 clustering 하는것은 쉽지만, 옆면을 clustering 할 때 문 제가 생길 수 있다. 원통형 물체, 또는 구형의 물체가 타일에 포함되어있다면 그 물체에 의해 normalVector끼리의 경계가 이어져버린다. (DBSCAN의 한계)

*예시)*

* *

*왼쪽 타일에서는 cluster가 명확하게 나뉘지만, 오른쪽 타일에서는 경계가 모호 하여 전부 하나의 cluster로 묶일 수 있다.*

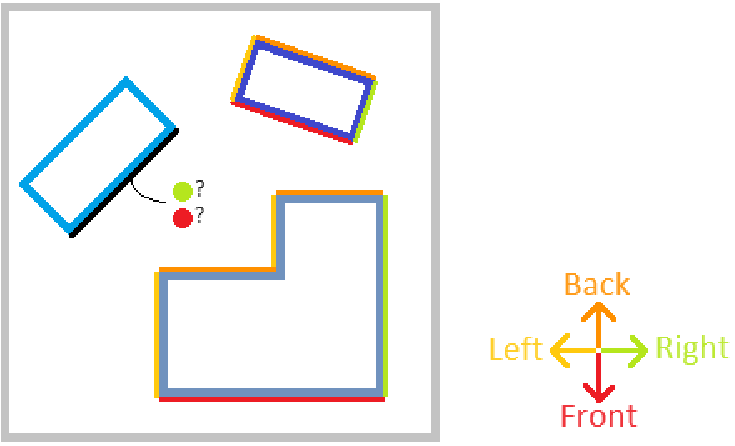
**접근2.** 기준벡터로 clustering

각 삼각형을 세 정점을 통해 구한 노멀벡터와 가까운 기준벡터의 cluster에 포 함된다.

예를들어, 지면과 천장을 모두 포함하는 cluster를 만들기 위해서는 upVector와 의 각도 차이가 15도 이내인 normalVector를 갖는 삼각형들을 clustering한다. (위를 바라보는 cluster를 만드는것은 기준이 명확해서 큰 문제없이 작동한다.)

벽면을 clustering 하기위해서는 frontward, backward, right, left등을 기준으로 각 도 차이가 허용범위 이내인 삼각형들을 clustering을 한다. 하지만 이 방식으로 는 옆면을 완벽하게 clustering 해내지 못할 가능성이 높다.

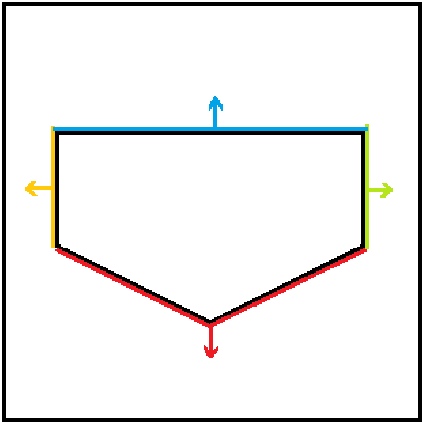
*ex)*

**

*검은색 면의 경우 Right 클러스터에 포함되어야 하는지 Front 클러스터에 포함 되어야 하는지 애매하다.*

*\* 옆면 간의 모서리가 둔각인 경우, 실제로는 서로 다른 옆면이 같은 cluster로 묶일 것이고, 이후 grouping 단계에서도 같은 group으로 묶일 가능성이 크다..*

*예시)*

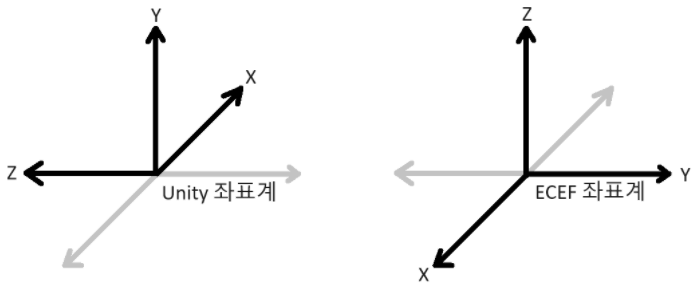


*빨간 두 면은 다른 group이 되어야 하는데 frontVector기준으로 각도차이가 허용치를 만족해서 같은 cluster로 묶였다. 게다가 실제로 인접해 있기 때문에 grouping 단계에서도 같은 group으로 묶일 것.*

(up방향을 제외한) 8방향 또는 6방향 벡터를 기준으로 하면 이런 현상을 어느 정도 방지할 수 있을 것으로 기대한다.

\* *upVector 설정시 주의할 점 : cesiumGeoreference하의 tile들은 upVector 값이 (0,-1,0)이다. tile을 외부 b3dm뷰어를 통해 obj화 하고 다시 Unity에 임포트해보 면, UpVector의 방향을 ECEF좌표계 변환을 통해 구한 다음, 아래 그림에 맞게 ECEF to Unity로 변환해줘야 한다.*

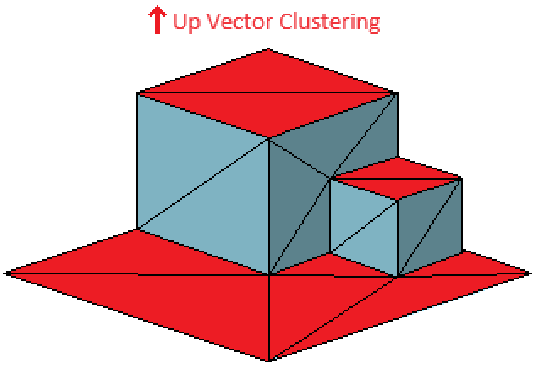
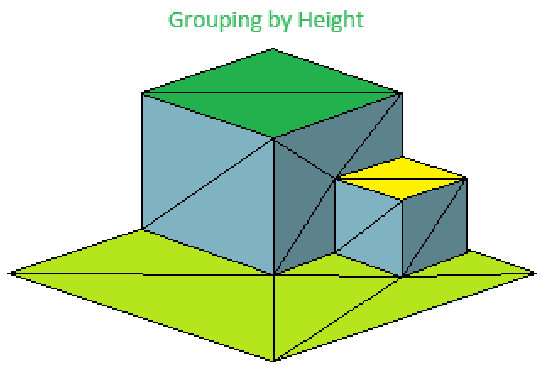
*\*일반적인 유니티 오브젝트나 Cut의 결과로 생성된 오브젝트는 (0,1,0)이 up방 향이다. (CesiumGeoreference하에 있지 않아야한다.)*



또한 DBSCAN 알고리즘의 특성상 어느 cluster에도 속하지 않는 Noise Point에 해당되는 삼각형들이 발생할 것이다.

2) cluster 내에서 group을 나누는 방법

노멀벡터의 방향을 기준으로 삼각형들을 clustering한 후, 실제로 인접한 면들끼리만 group을 형성해야한다.

이를 위해 DBSCAN 알고리즘을 사용하는데, 삼각형이 이웃함을 판정하는 방법을 선정해 야했다.

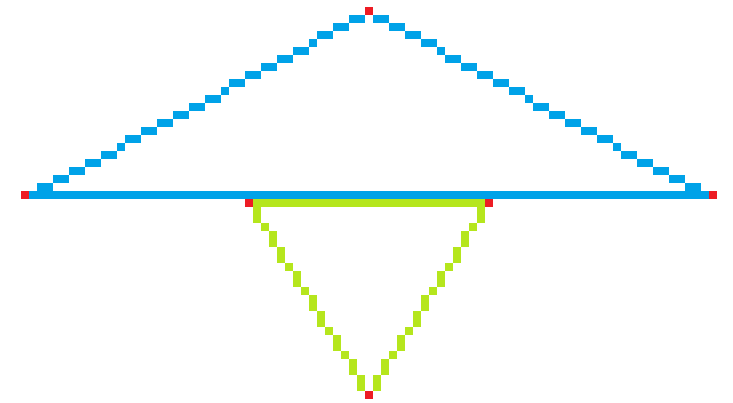
**접근1**. 삼각형의 세 정점으로 무게중심을 구하고, 무게중심간의 거리로 이웃함을 판정

문제점) 삼각형의 모양에 따라 이웃함을 잘 판정하지 못할 수 있다.

**접근2**. 한 삼각형의 세 정점과 다른 삼각형의 세 정점간의 거리의 최소값을 기준으로 이웃함을 판정

문제점) 실제로는 인접해있지만 vertex간의 거리가 있는경우 이웃함을 잘 판정 하지 못할 수 있다.

예시)



두 삼각형은 인접해있지만 vertex간의 거리가 epsilon보다 크다면 다른 group으 로 묶일것이다.

**접근3**. 3차원 공간상의 삼각형간의 거리를 구하여 이웃함을 판정.

계산량이 접근 1,2에 비해 많긴 하지만 오류없이 실제로 인접한 삼각형들을 이 웃하다고 판정할 가능성이 가장 높다.

추가로, DBSCAN 알고리즘을 이용할때 생기는 group에 포함되지 못한 원소들에 대한 처 리가 필요하다.

접근1. 가장 가까운 group으로 편입한다..

...group과 삼각형간의 거리는 어떻게 판정하나? --> 미해결된 부분이다.

3) 지면을 어떻게 구분?

**접근1**. Up 방향 cluster의 산하 group중 높이값이 낮은 group을 지면으로 판단한다.

group의 높이값을 어떻게 계산할것인가?

① 그룹내 모든 vertex 높이값의 평균

② 그룹내 각 삼각형 무게중심의 높이값의 평균

③ 임의로 선정한 vertex 하나의 높이값

--> 어느 방법을 채택하던 큰 차이 없을 것이라고 예상됨.

주의점)

- clustering 할 때 upVector와의 각도 차이 허용범위를 적절하게 조절해야한다.

- 비슷한 높이의 삼각형들을 세밀하게 군집화하기 위해 DBSCAN에서 distance 를 구할 때, upVector 방향 성분에 가중치를 적절하게 부여해야한다. 가중치가 너무 크면 한 surface가 여러 group으로 쪼개져 버릴 수 있고, 가중치가 낮으면 구분되야할 그룹이 묶여버릴 수 있다.

- 지면이 없는 타일을 선택한 경우 엉뚱한 group이 제외될 것이다. (뒤에서 다 룸)

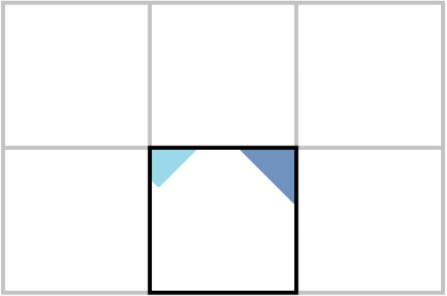
- 지면이 두개 이상이고, 지면간의 높이 차이가 있는 경우, 높은 지면은 지면으 로 판단되지 않을 것이다. (일반적으로 대부분의 타일에선 지면간 높이차가 별 로 없기 때문에 큰 문제는 없을것이라고 생각됨.)

**접근2.** 사용자가 수동으로 지면 그룹을 구분해준다.

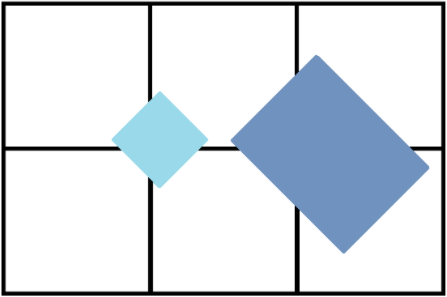
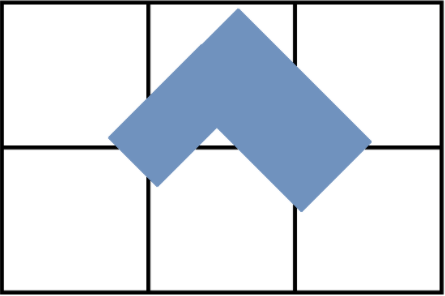
잘못된 지면이 선택되는 일은 없겠지만, 현실적으로 힘들다.

4) group화된 면들 중 특정 건물의 면인지 어떻게 구분?

지면을 제외한 면들에 대해서 목표 건물의 면인지 다른 건물의 면인지 판단하기 어려움.



하늘색 면을 클릭했고, 한 타일에 대해서 group화를 완료했을 때,

하늘색과 파란색 건물이 하나의 건물인지 아닌지는 건물이 연결된 타일을 전부 검사해보아야 알 수 있다.

🡪목표 건물이 연결되어 있을 타일을 선정하는 기준이 필요하다.

**기준1)** 단순하게 주변 8개 (위아래 타일까지 고려하면 26개) 타일을 선정

문제점)

* 충분하지 않을 수 있다. 건물이 더 많은 타일 범위에 걸쳐 있을 수 있음.
* 주변 타일 오브젝트를 감지하는 방법?

ⅰ) 이름을 통해서 찾는 방법 …X

오브젝트 이름이 네이밍 규칙에 맞지 않을 가능성 존재

ⅱ) tileset.json을 통해 찾아가는 방법

ⅲ) collision check로 찾는 방법

**기준2)** group들에 인접한 타일들을 선정. (여기서 group이란 group내의 삼각형들로 이루 어진 GameObject를 의미한다. 물리적으로 인접한지 찾아야하기에 그룹을 오브젝트화 시킨 것이다.)

타일이 group과 인접했는지 판단하는 방법(인접 타일 탐색)

ⅰ) group의 각 vertex에서 구 형태로 collision check한다.

유니티에서 mesh collider끼리의 collision check를 하는것이 어 렵기 때문에 vertex를 중심으로한 구체형태로 coliision check 를 한다.

group내 모든 vertex에 대해서 충돌체크를 하므로 연산량이 적지 않을 것

ⅱ) group내 vertex들에 대해서 minX maxX minZ maxZ minY maxY를 구하고 그 값이 다른 타일 범위에 속하는지 체크하여 인접함을 판단.

타일간 연결부위가 오버랩 되어있기 때문에 vertex가 인접 타 일 범위에도 포함될 것임.

게다가 찾아야할 타일이 오브젝트상태로 로드되어있지 않을 수 있기 때문에 연결부의 vertex 좌표를 통해 tileset.json 파일 구조에서 해당 타일을 찾아내는 방법을 사용하는것이 좋다.

타일 그리드 범위에 대한 정보를 알고 있어야지 가능한 방법 이다.

정사각형 형태가 아닌 타일같이 규격 외의 타일들에 대해서는 이런 방법으로 처리하기 힘들 것.

타일 탐색시 주의할 점) 목표 건물이 아닌 다른 건물에 속하는 group에 의해서 도 tile들이 선정될 수 있음. 엉뚱한 타일에 대해서도 연산할 가능성 존재

따라서 그룹화(지면 group제외 과정 포함)를 마친 후, 그룹들에 대해서 startGroup (클릭한 지점을 포함하는 그룹)과 연결되어 있는지 확인해야함.

startGroup 그룹과 연결된 그룹에 한해서 인접타일 탐색을 실시

다음 타일에 대해서 그룹화를 마친 후, 모든 타일의 group에 대해서 또다시 startGroup과의 연결성 판단하고 연결된 그룹에 한해서 인접타일 검색

재귀적으로 반복한다.

\* group간의 연결성 판단하는 방법

같은 tile내에 있다면 vertex를 공유하는지 여부로 판단이 가능하지만,

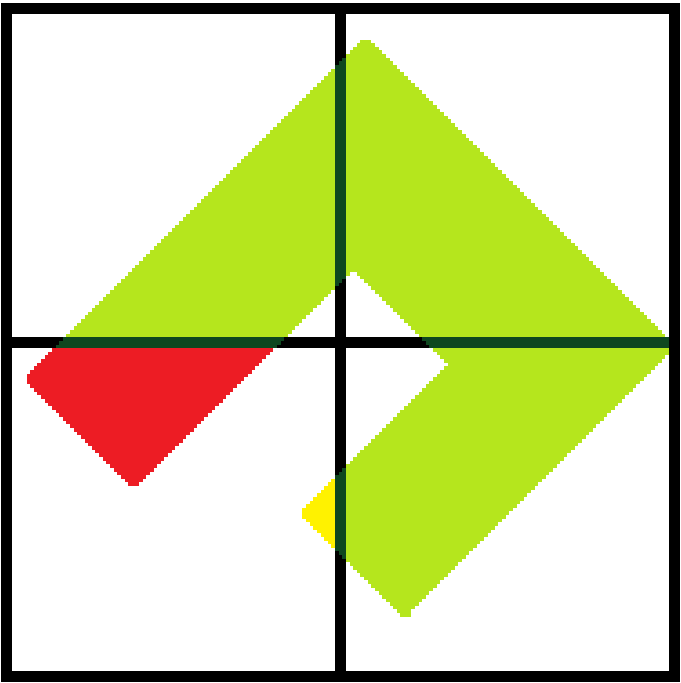
서로 다른 tile에 속한 group인 경우 overlap 되는지 물리적으로 check 해야함.

\* *연결성을 매번 처음부터 다시 check하기보다 그래프 구조로 연결 정보 를 저장하면 좋을 것 같다.*

*\* 한번 clustering & group화 한 타일은 다시 방문할 필요 없으므로 방 문 여부를 저장하는것이 좋다. HashSet 이용하면 효율적으로 구현 가 능할것으로 예상.*

연결성 판단 후, 아직 연결되지 않았다고 판단된 group들의 정보들도 보존해야 한다. 다른 타일의 group들을 통해 연결성을 추후에 다시 입증할 가능성이 있 기 때문

*예시)*

**

*좌하단 타일의 빨간 group을 클릭하여 시작한 경우, 노란 group은 초 기에 연결성이 없다고 판단되지만, tile을 확장해나가면서 추후에 연결 성이 입증된다.*

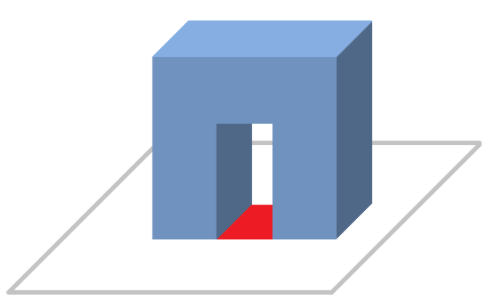
**③ CookieCut + Clustering 합친 방법**

: 대략적인 범위로 CookieCut 알고리즘을 먼저 적용한 후, Clustering & group화를 통한 지면제거를 하여 객체화를 하는 방법

장점)

(0) CookieCut을 단독으로 사용하여 좋은 결과물을 얻으려면, 다각형 영역이 매 우 정교해야하는데, Clustering방법과 합치면 다각형 영역의 퀄리티가 다소 낮아 도 괜찮다. + CookieCut에서 제거할 수 없던 지면도 추가로 제거할 수 있다.

*예시)*



*지면으로부터 떠있는 부분의 지면(빨간영역)은 CookieCut으로 지울 수 없다.*

(1) clustering 대상이 될 tile오브젝트가 CookieCut과정에서 전부 선별된다.

(2) 미리 CookieCut을 통해 주변 건물을 제거할 수 있기 때문에 지면을 제외한 group들 중에서 목표 건물인지 아닌지 확인하는 과정이 필요 없어져서 Clustering 알고리즘이 훨씬 간단해진다.

(3) 미리 CookieCut을 했기 때문에 처리해야할 vertex와 triangle 수가 대폭 줄 어들어서 계산시간 단축에 도움이 된다.

단점)

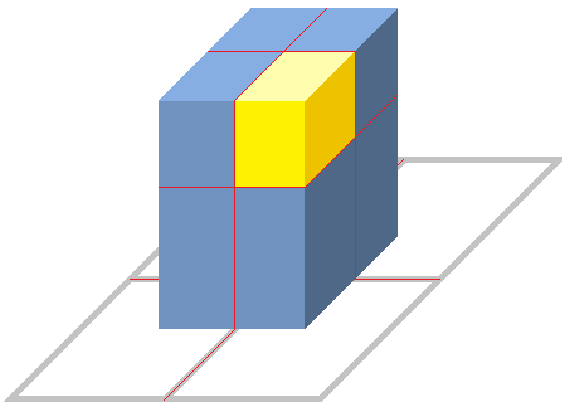
(1) CookieCut 알고리즘을 사용하기 때문에 다각형 영역에 대한 data가 반드시 필요하다.

(2) 다각형 영역의 퀄리티 문제로 CookieCut과정에서 주변 건물이 걷어내지지 못할 수도 있다.(뒷내용의 SelectOnlyBuilding 부분에서 어느정도 해결이 된다.)

지면 제거는 한 타일(오브젝트)에 대해서 normalVertor가 upVector와 가까운 삼각형들을 clustering하여 여러 그룹을 만든 다음, 그 그룹들 중 평균 높이값이 max/10 + min\*9/10 미만인 그룹들을 지면으로 판단하여 제거하는 방식으로 이루어진다.

하지만 지면 제거를 수행할 때, 실제로 지면이 없었던 타일(예를 들어 건물의 윗부분)이 라면 필요한 mesh가 손실될 가능성이 있다.

*예시)*

**

*노란 영역에 해당하는 타일은 지면을 포함하고 있지 않지만, 지면 제거 알고리즘에 의해 엉뚱한 천장 면이 날아갈 가능성이 있다.*

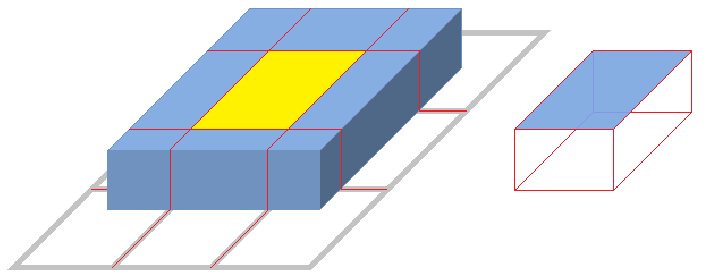
따라서 해당 타일이 지면을 포함하는지 판단하는 방법이 필요하다.

타일이 지면을 포함하는 지 판단하는 방법?

접근1) tileset.json파일의 boundingVolume값을 참고하여 땅과 붙어있는 타일인 지 판단한다

-->굉장히 복잡해진다. 타일 트리의 네이밍 규칙이 항상 맞다는 보장이 없음.(+ 지면 높이의 타일이더라도 지면을 포함하고 있지 않을 수 도 있다.)

*예시)*

**

*노란 영역에 해당하는 타일은 지면을 포함하고 있지 않지만, 지면 제거 알고리 즘에 의해 천장이 날아갈 가능성이 있다.*

접근2) 물리적으로 위,아래에 다른 타일이 있는지 체크하여 건물 윗부분인지 아 랫부분인지 분간한다. --> 알고리즘이 매우 복잡해짐

개선된 접근 방법 : CookieCut을 적용하여 오브젝트 조각들을 만들고, **조각들을 합쳐서 하나의 오브젝트로 만든다**. 그 후, 합쳐진 오브젝트에 대해서 지면 제거 알고리즘을 수 행한다.

--> tile들 중 지면을 포함하는 타일을 선별할 필요가 없어지기 때문에 알고리즘이 간단 해진다!

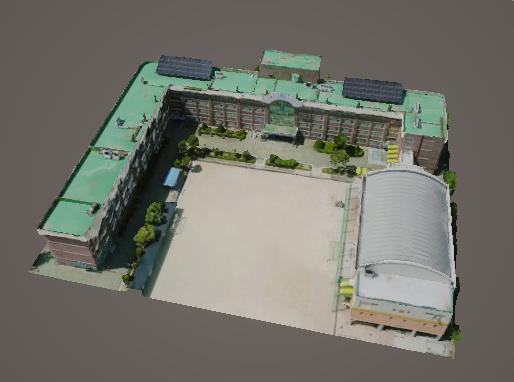
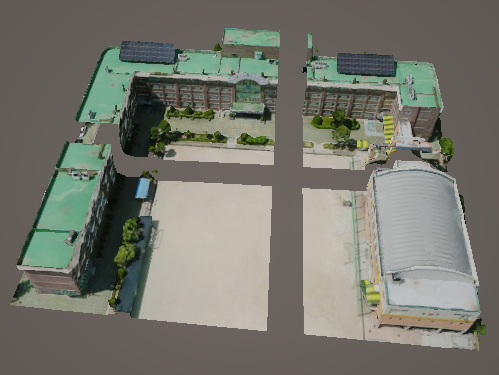
Unity 환경에서 객체화 프로세스를 구현한 모습

(0) 다각형 영역 지정.



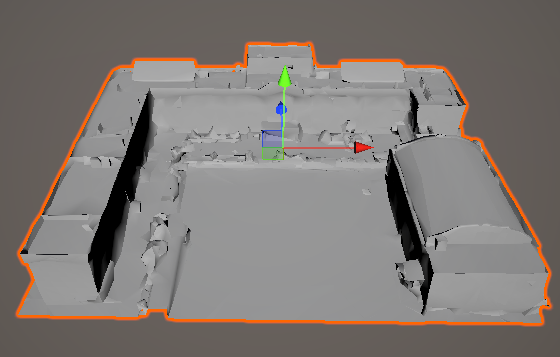
직접 tile표면에 vertex들을 찍어서 영역을 생성했다.

(1) CookieCut

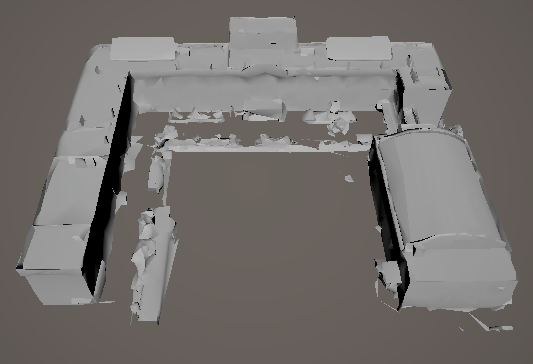
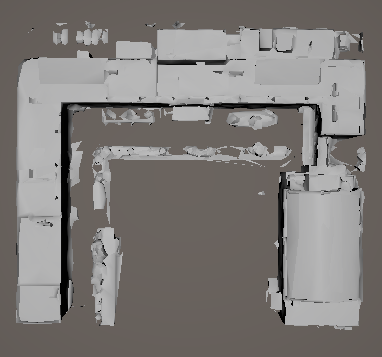
CookieCut 결과물의 모습이다. 4개 타일을 잘랐기 때문에 4개의 오브젝트로 이루어져있다.

(2) Combine



4개의 오브젝트를 하나의 오브젝트로 합쳤다. 단순히 모든 vertex와 triangle을 합치는 방식이라 서 오버랩된 부분은 mesh가 두겹이다. Combine 단계는 다른 라이브러리를 통해 대체하는것이 좋을 것이다.

(3) Remove Ground (Clustering & Grouping)

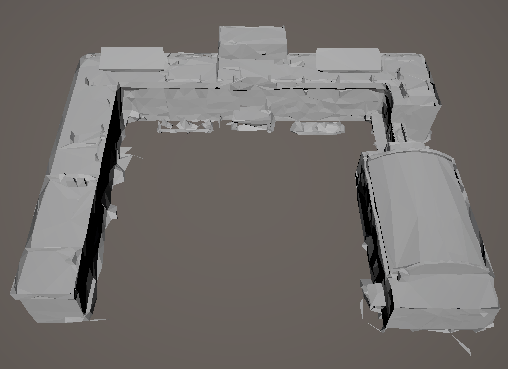
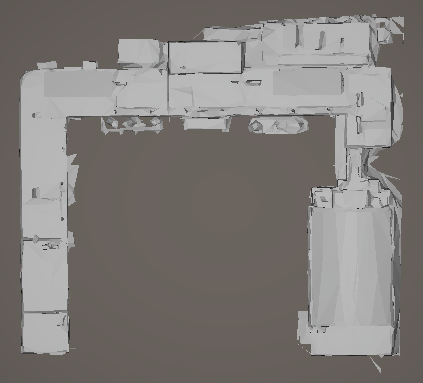
 

RemoveGround를 거친 모습이다. 수풀이나 조그마한 단차 등이 제거되지 않아서 지저분하다.

이 학교 모델만 해도 combine된 mesh의 vertex가 20000개이고 triangle은 16000개여서 지면 제 거 과정이 상당히 오래 걸렸다.(약 8초 소요)

BottleNeck은 두 삼각형이 인접한지 판단할 때 3차원상의 두 삼각형의 거리를 계산하는 과정이 다. 한 삼각형과 인접한 삼각형을 찾을 때 나머지 모든 삼각형에 대해서 인접한지 검사하기 때문 에 O(n^2)만큼 걸린다.

(4) SelectOnlyBuilding : 건물mesh와 연결된 삼각형만 남기는 작업

Remove Ground한 오브젝트에서 가장 많은 삼각형이 연결된 mesh 덩어리만을 남기는 기능을 한다. 마찬가지로 삼각형 간의 인접성을 검사할 때 시간이 굉장히 오래 걸린다.

Octree를 통해 인접성 검사 시간을 단축해본 결과, 원래 10초정도 걸리던 RemoveGround 작업이 5초로 줄었고, 원래 30초정도 걸리던 SelectOnlyBuilding작업이 15초정도로 줄었다.

SelectOnlyBuilding 마지막에서 삼각형이 제거되고 난 후 더이상 사용되지 않는 vertex들을 제거하여 최적화를 한다.

객체화 이후, mesh 간소화 및 개발 진행 방향

MeshSimplify(많은 양의 삼각형으로 이루어진 면을 적은 수의 삼각형으로 재표현하는 작업) 를 하기 위해선 객체화된 건물의 각 면이 구분되어야한다.

일반적인 건물은 사각형 형태이기 때문에 'Cookie Cut과 지면 제거를 통한 객체화가 완 료된 오브젝트'의 면을 구분할 때, normalVector를 기준으로한 DBSCAN 알고리즘을 사용 하여 꽤 잘 구분할 수 있을 것으로 생각한다.

다만 RemoveGound과정에서 보았던 것 처럼 모든 삼각형에 대해 DBSCAN하는것이 OctTree를 이용해도 여전히 오래걸리기 때문에, 이를 완화하기 위한 새로운 접근방법이 필요할 것이라고 생각한다.

건물 객체화에 필요한 타일 오브젝트가 Scene에 로드되어 있지 않은 상태에서도 객체화 가 가능하도록 만드는 것이 좋다. 현재 CookieCut 알고리즘은 물리적으로 tile을 선별하 므로, 객체화에 필요한 오브젝트가 씬에 없다면 tileset.json을 통해 필요한 타일이 무엇 인지 찾고, 만약 씬에 존재하지 않는다면 그 타일을 생성하는 프로세스를 객체화 과정 앞부분에 추가해야 할 것이다.

운동장을 포함하는 건물같은 경우, 지면을 제거하면 안되기때문에 지면 제거를 할지 선 택할 수 있게 하면 좋을것이다.