Pusan National University

Computer Science and Engineering

Technical Report 2022-17

# 포인트 클라우드를 이용한

# 3D 공간 생성/복원

기태욱

서준오

이해성

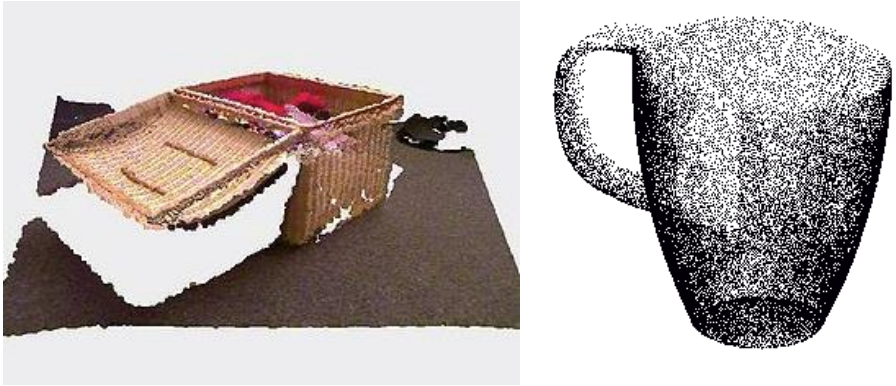
지도교수 : 이명호

# 목차

# 1. 과제 배경 및 목표

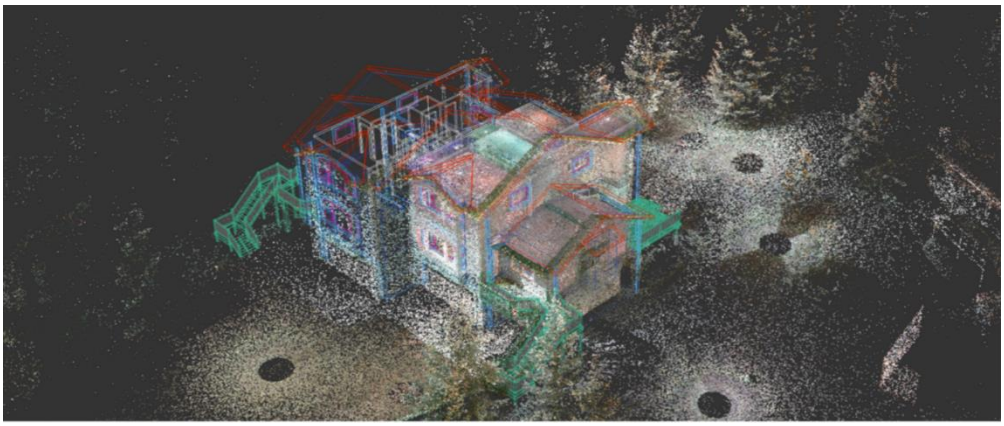
## 1.1 과제배경

Point Cloud란 Lidar 센서, RGB-D센서 등으로 수집되는 3차원 공간상에 퍼져 있는 여러 포인트(Point)의 집합(set cloud)를 의미한다. 센서들은 물체에 빛/신호를 보내서 돌아오는 시간을 기록하여 각 빛/신호 당 거리 정보를 계산하고, 하나의 포인트를 생성한다. 그렇게 수집된 데이터는 각각 아래 그림과 같이 나타내어진다.



<그림1> - LiDAR센서로 수집된 Point Cloud(좌), RGB-D센서로 수집된 Point Cloud(우)

<그림1>과 같이 측정 장비에서 얻어진 PointCloud는 제조 부품, 계측/품질 검사 및 시각화, 애니메이션, 렌더링 및 3D CAD 모델을 만드는 등 다양한 목적으로 사용된다. 특히 LiDAR 센서를 사용하는 자율주행과, 건축물의 3차원 가상 건설 환경 데이터인 BIM(Building Information Modeling)에서 그 활용이 두각 되고 있다.



<그림2> - Point cloud to BIM(Building Information Modeling)

## 1.2 과제 목표

일반적으로 PointCloud는 직접 사용할 수 없다. 그래서, 대개 다각형 메쉬 또는 삼각형 메쉬 모델, NURBS 표면 모델, CAD 모델로 변환해야 한다. 본 졸업 과제에서는 한 가구의 실내 3차원 PointCloud 데이터에서 천장, 벽, 바닥을 탐색하여 메쉬로 변환하는 작업을 진행한다. 그리고 실내의 작은 물체, 노이즈 등 복원이 필요 없는 물체들을 검출하여 제거하고 제거된 영역을 다시 채우는 방법을 연구하는 데에 목표를 둔다.

- 천장, 벽, 바닥을 추출하기 위한 알고리즘 구현
- 작은 물체, 가구들을 분류 및 처리 기능 구현
- 텍스쳐 처리 및 제거된 영역 복원 기능 구현
-

## 1.3 기존 사항 분석

PointCloud의 각 포인트는 3 개의 위치 정보를 가지고 있으며 이 포인트들이 모여 하나의 형태를 형성하게 된다. PointCloud는 3차원 데이터 형태 중 가장 원본 센서 데이터와 부합한다는 특성을 가지고 있으며, 3개의 속성만을 지닌 포인트들의 집합이기 때문에 효율적인 저장 공간 사용을 가능하게 하며 재질이나 색과 같은 객체가 지닌 특징을 부분 단위로 추가하기 쉽다는 장점을 가지고 있다. 또, PointCloud는 아래와 같은 특징을 가진다.

**1) 순서가 존재하지 않음**

PointCloud 데이터는 이미지와 같은 픽셀 배열과 다르게 세 개의 축에 대한 위치만을 포함하기 때문에 순서가 존재하지 않는다.

**2) 각 포인트들의 관계**

PointCloud 상에 있는 포인트들은 격리(isolate) 된 상태가 아닌 서로 간의 측정 가능한 거리를 지닌 이웃들을 구성하며 존재한다. 이는 이웃들과의 관계를 학습 과정에서 로컬 구조를 파악하는 과정에서 유의미하게 활용할 수 있음을 의미한다.

**3) 변환에 강한 특징**

회전과 같은 변환을 하였을 때 이웃 포인트과의 거리나 전체적인 구조가 변경되는 것이 아니기 때문에 PointCloud가 지닌 구조적인 특징을 그대로 보존할 수 있다. 정리하면 PointCloud는 3D 센서와 같은 3차원 데이터 생성 과정에서 가장 낮은 단계(low level)의 원본과 유사한 데이터를 제공해 준다. 각 포인트는 각 축에서 포인트의 위치를 표현하는 3개의 수치로 표현되며 추가로 속성을 부여할 수 있다. 각 포인트들의 관계가 거리 측정이 가능한 형태로 존재하며 변환에 강하나 순서가 존재하지 않는다.

# 2. 연구 개요

## 2.1 개발 환경

PointCloud 데이터는 수 많은 좌표와 normal을 값으로 가지고 있다. 이를 다루기 위해 numpy, open3D와 같은 라이브러리를 사용하기 위해 python을 사용할 것이다. Anaconda는 python, R언어의 패키지 관리를 쉽게 수행하는 오픈 소스 배포판이다. Conda를 통해 가상환경을 사용하기 위해 이를 선택하였다.

Open3D 라이브러리는 3차원 데이터를 다루며 Python, C++을 지원하며 오픈소스로 열려 있다. 이 라이브러리는 Mesh, Polygon, PointCloud와 같은 3차원 자료구조와 각종 가공 및 시각화 기능을 제공한다. 이번 과제에서 PointCloud에 대한 편집을 수행하기 위해 해당 라이브러리를 사용하게 되었다.

Unity 엔진은 C#언어를 지원하는 게임 개발환경이자 3D 시각화 콘텐츠를 제작하는 통합 제작 프로그램이다. 이 엔진은 프로젝트 내에서 사용가능한 3D 모델, 오디오 파일, 이미지 등을 제공하는 Unity Asset을 다운로드하는 스토어를 지원한다는 특징이 있다. 이번 과제에서는 PointCloud 데이터의 시각화하기 위해 사용할 것이다.

## 2.2 목표 기능

### 2.2.1 사물 및 공간 분류

주어진 실내 공간 PointCloud 데이터에서 open3D 라이브러리의 RANSAC을 사용한 plane 검출, DBSCAN등을 활용하여 벽, 바닥, 천장, 사물을 분류하여 작은 PointCloud 데이터로 표현하고자 한다.

### 2.2.2 작은 사물 제거

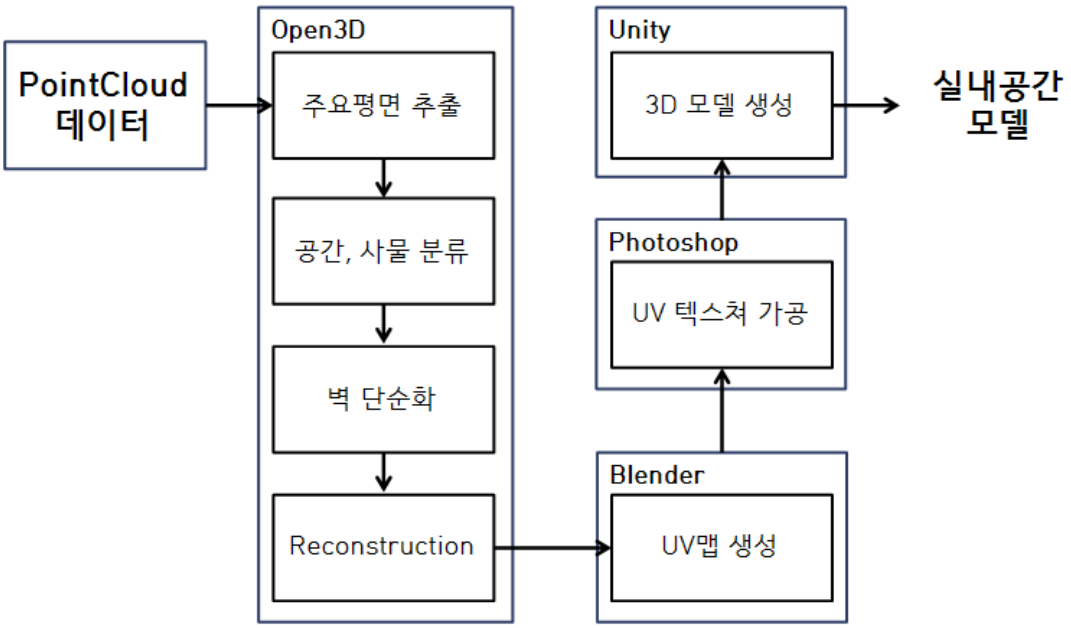
나누어진 사물을 나타내는 PointCloud 데이터 중, 특정 크기 이하의 사물을 데이터에서 제거하는 기능을 구현하고자 한다.

### 2.2.3 Reconstruction

작은 사물이 제거된 PointCloud 데이터를 mesh화 하여 Unity를 사용해 표현한다. 추가로, 사물이 제거됨에 따라 발생하는 공백을 mesh로 복원하거나 해당 사물과 유사한 Unity Asset로 대체하는 방법을 구현하고자 한다.

# 3. 연구 내용

## 3.1 개요



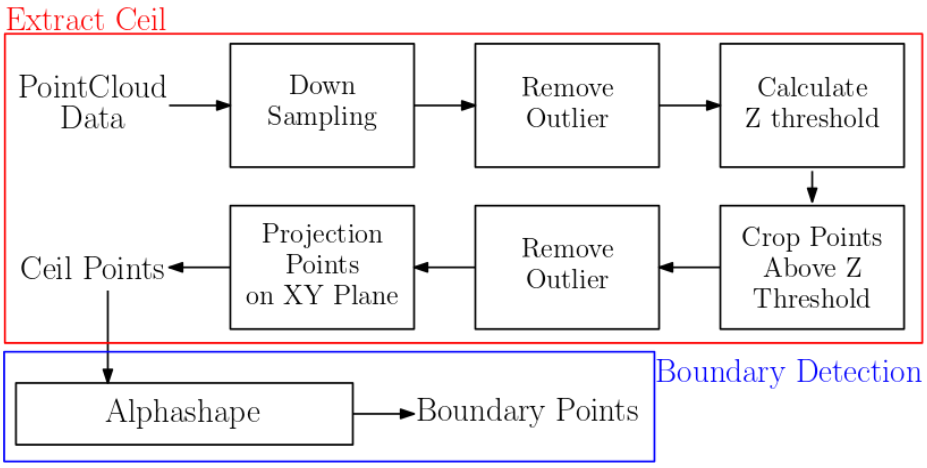

<그림3> - PointCloud 데이터를 가공하는 과정을 간략히 나타낸 흐름도

구현하고자 하는 시스템은 <그림3>와 같이 구성될 것이다. 먼저 Open3D( 3차원 데이터 처리 관련 라이브러리)를 통해 Point Cloud Data를 기반으로 공간을 새로 생성한 뒤, 텍스쳐 작업을 수행한다..

천장과 바닥면을 탐색해 추출하고, 바운더리 포인트들을 알아낸다. 바운더리 포인트는, 첫째, 공간과 사물을 구분하는 목적으로 사용되고, 둘째, 울퉁불퉁한 벽을 단순화하기 위해 필요하다. 전체 공간에서 바운더리 포인트의 일정 범위 안에 포함되는 점들을 제외하면 구조물을 걷어낸 사물 포인트들만 얻을 수 있다. 그리고 바운더리 포인트를 활용해 새로 선분을 그어, 단순화된 벽을 생성하고 천장면과 바닥면을 더해 단순화된 벽과 빈공간이 복원된 새로운 포인트 클라우드 공간을 생성하였다. 데이터를 mesh로 Reconstruction 하여 파일로 출력한다.

텍스쳐링을 위해 먼저 Blender 툴을 이용해 UV맵을 생성한다. 360 이미지를 사용하기 위해 Photoshop으로 왜곡을 감소시켜, 앞서 생성한 UV맵과 평탄화된 360이미지를 합성해 텍스쳐를 얻는다. 마지막으로 시각화를 위해 Unity로 생성한 공간 모델을 가져오고 이미지 텍스쳐링 작업을 통해 실내 공간 모델을 완성 시킨다.

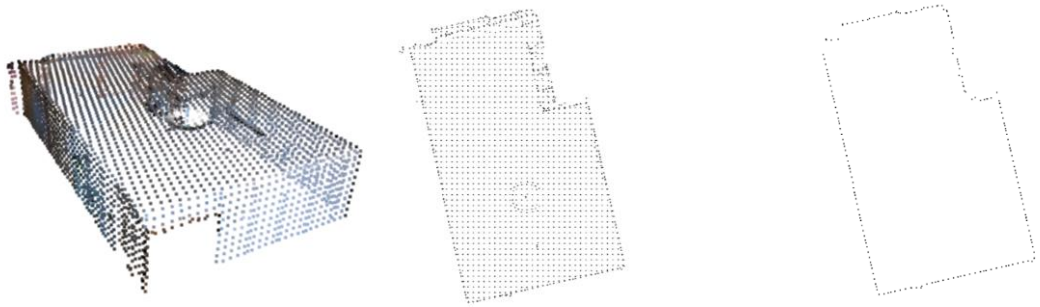
## 3.2 주요 평면 및 Boundary 추출




<그림4> - 공간의 벽, 천장, 바닥을 제거하기 위한 알고리즘의 흐름도

3D 포인트 클라우드 데이터에서 천장면을 도출해내 평면에 투영 어떤 공간의 가구를 제외한 모든 천장, 바닥, 벽을 제거하기 위해 <그림4>와 같은 방법을 사용하였다.

가장 큰 면인 천장을 먼저 탐색하고자 Open3d의 down sampling, 통계적 outlier 제거기능을 통해 노이즈를 제거한다음 데이터의 전체 z범위에 대해 Z threshold를 구하여, 해당 값보다 높은 z값을 가지는 포인트들을 추출하여 다운 샘플링 처리 후 XY 평면에 투사한다. (<그림5>)

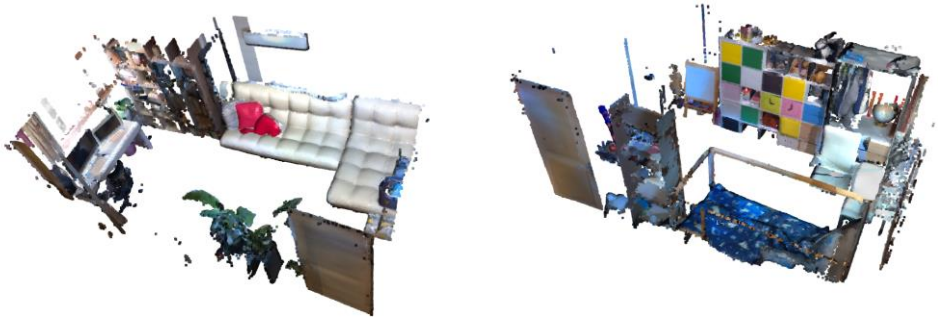


(a) (b) (c)

<그림5> - (a) 전체 z범위에서 75%보다 높은 z값을 가지는 point를 선택하여 나타낸 데이터
(b) 는 xy평면에 투영되어 검출된 천장
(c) 검출된 천장 데이터에서 alphashape를 적용한 모습

천장 및 바닥면을 추출한 후, 윤곽선(Boundary Points)를 추출하기 위해 알파 쉐이프 알고리즘(Alpha Shape Algorithm)을 사용하였다. 알파 쉐이프 알고리즘은 두 포인트를 지나는 지름이 알파(α)인 원안에 다른 포인트가 포함되지 않을 때 두 포인트는 윤곽 기하 정보를 이루는 포인트로 규정하는 개념이다. 보통 흩어져있는 포인트들에서 다각형을 일반화하기 위해 사용하며, 매개 변수인 알파에 적합한 값을 대입하는 것으로 Concave Hull을 얻을 수 있다. 이를 부록의 <코드1>, <코드3>에 나타내었다. 이에 대한 결과는 <그림5-(c)>와 같다.

## 3.3 가구 추출 및 후처리



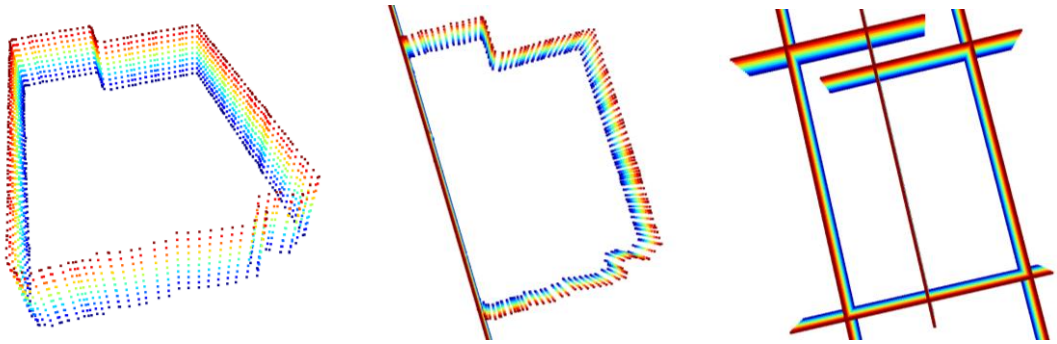
(a) (b)

<그림6> - (a), (b) 는 벽, 천장, 바닥으로 부터 가구를 분리한 결과물이다

가구를 추출하는 것은 공간의 주요 평면인 천장, 바닥, 벽을 제거하는 것과 같다. 이를 위해, 두 개의 Z threshold 값(z_high, z_low)을 사용하여 z_low <= z <= z_high를 만족하는 모든 포인트를 구하여 천장과 바닥을 제거한다. 다음, 벽을 제거하기 위해 위에서 구한 boundary points를 기준으로 해당 공간의 모든 포인트에 대해 z값에 상관없이 해당 포인트으로부터 반지름이 r인 원기둥(또는 길이가 n인 사각기둥)의 범위만큼 x,y 좌표가 겹치는 포인트들을 제거한다. 이를 부록의 <코드4>, <그림6> 에 나타내었다.

## 3.4 벽 단순화 및 공간 생성

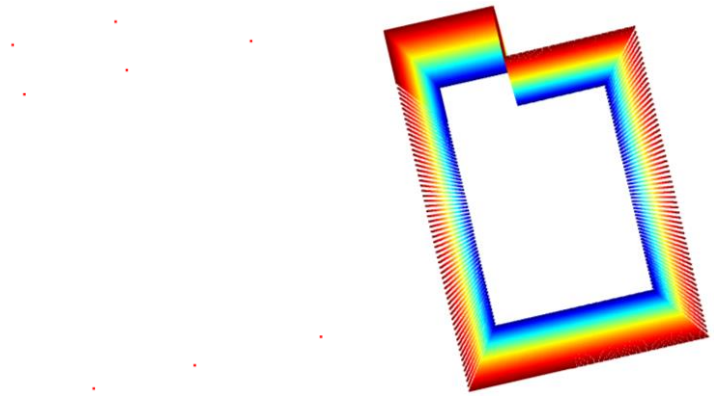
기존 포인트 클라우드 공간에서 가구를 제거하면 가구가 벽면에 붙어있는 영역만큼 빈 공간이 생기는 문제를 가지고 있다. 이러한 이유로, 공간을 메쉬화 하기위한 normal 계산이 정상적으로 이루어지지 않게 된다. 이를 해결하기 위해 단순화된 공간을 생성하고 원본공간과 합쳐 빈 공간을 메우는 방법을 고안하였다.



(a) (b) (c)

<그림7> - (a)는 boundary를 z축에 대해 0.1 간격으로 0~1.0 사이에 배치한 것을, (b)는 plane segmentation을 통해 찾아낸 평면(왼쪽 직선벽)과 boundary를 통해 생성한 벽을 동시에 나타낸 것이다. (c)는 (b)의 과정을 반복하여 나타난 결과이다.

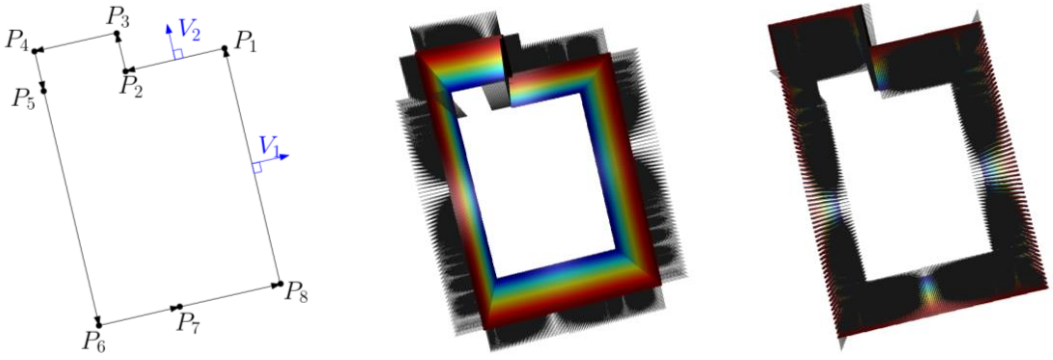
위에서 얻은 Boundary를 z축에 대해 일정한 간격으로 배치하여 벽을 생성한다. 다음, <그림7-a>와 같이 open3d의 Plane Segmentation을 통해 생성된 벽에 존재하는 평면을 탐색한다.(<그림7-b>) 적합한 수의 평면을 찾기 위해, 매번 평면을 발견할 때마다 근처의 벽 포인트를 제거한다. 이를 생성된 벽면이 일정 개수 이하의 포인트를 가질 때까지 반복한다. (<그림7-c>)



(a) (b)

<그림8> - (a)는 평면끼리 생기는 교점을 범위 맞게 잘라낸 것을, (b)는 이를 시계방향으로 정렬후, 보간하여 벽을 단순화한 것이다.

발견된 모든 평면을 xy평면 상에서 무한히 확장했을 때 평면끼리 생기는 교점들을 모두 구한다(<그림8-a>). 다음, 원본 공간의 범위에서 벗어나지 않는 모든 교점을 시계방향으로 정렬한뒤, 각 교점 사이를 보간하는 방식으로 <그림8-b>와 같은 단순화된 공간을 생성한다.



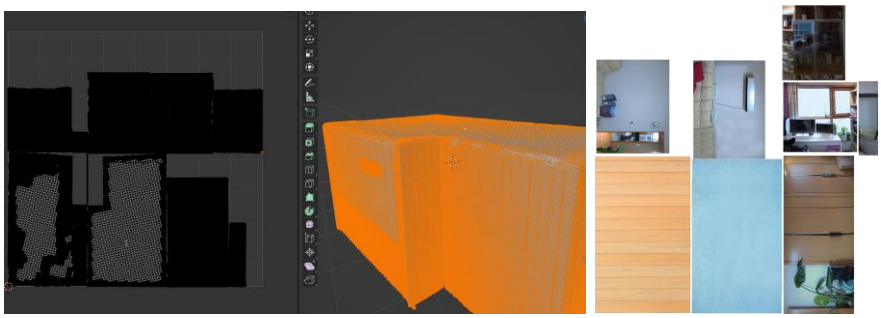

(a) (b) (c)

<그림9> - (a)는 <그림8-(a)>에서 나타난 정렬된 교점사이의 벡터를 구해 해당 벡터에 수직인 벡터 V1, V2를 구해 나타낸 모습이다. (b)는 해당 벡터를 보간된 벽에 적용한 모습이다. (c)는 (b)의 normal을 반전시켜 실제 포인트 클라우드에적용한 것을 나타낸 그림이다.

마지막으로, Reconstruction을 위해 모든 포인트들의 normal을 계산해야 한다. 정렬된 교점 사이의 벡터를 <그림9>와 같이 구한다. 다음, $V_1, V_2$와 같이 해당 벡터에 수직이 되는 벡터를 구한다. 이 벡터를 두 점 사이에 보간되어 생성되는 벽에 적용한다.(<그림9-(b)>) 마지막으로, 이렇게 얻은 벡터는 모두 바깥을 향하므로 실제 포인트 클라우드에 적용하기 위해 벡터 방향을 반대로 적용하여 공간 생성을 마무리한다.(<그림9-(c)>)

## 3.5 메쉬화(Reconstruction) 및 텍스쳐링

3-4에서 처리된 공간과 추출한 천장, 바닥면을 합친 데이터를 Poisson surface reconstruction 방법을 통해 메쉬화 한다. 그리고, open3d를 통해 ply 파일을 mesh화 하여 obj 확장자로 생성한다. 생성된 obj 파일에는 텍스쳐 정보가 포함되어 있지 않으므로 360이미지에서 추출한 각 면을 Blender로 파일을 가져와 uv맵을 생성하여 텍스쳐에 대한 png 파일을 만들고 obj 파일로 저장한다.



<그림10> -블렌더를 이용하여 obj파일의 UV 맵을 생성하고 UV맵 텍스쳐를 가공한다

이후 생성된 uv맵 png 파일에 붙일 텍스쳐를 붙여 uv맵을 완성한 후, Unity로 obj 모델을 불러와 생성한 텍스쳐를 입혀 컬러값을 복원한다.

<그림11> -Unity로 생성된 Mesh의 모습

## 3.6 실행 결과 및 지표

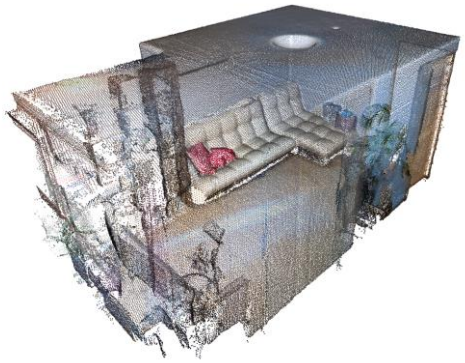
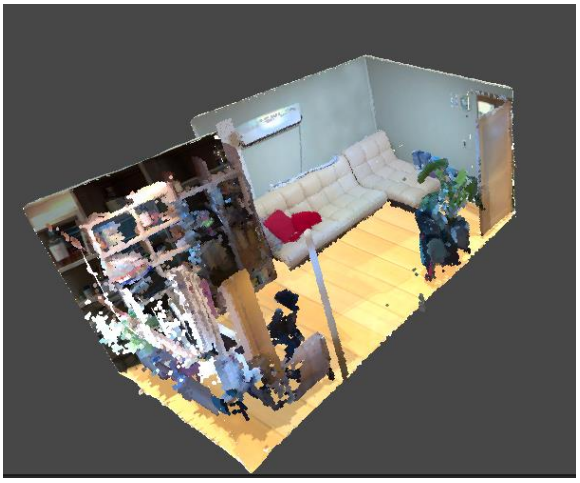
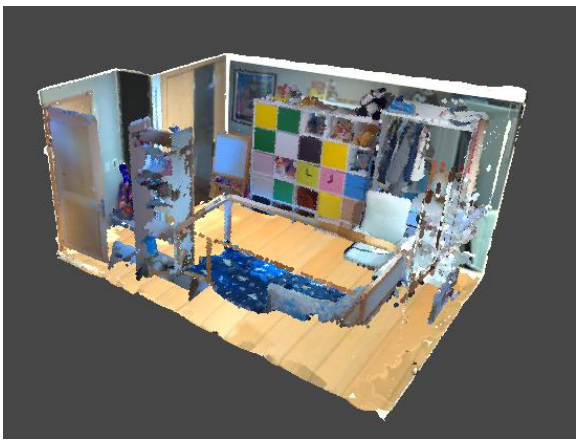
테스트를 위해, 서로 다른 두개의 방을 나타낸 포인트 클라우드 파일인 Cropped_1.ply와 Cropped_2.ply를 사용하였다. 두 파일은 각각 155694, 153916개의 포인트로 구성되어 있다. 프로그램 수행에 사용된 데스크탑은 CPU로 Intel Core i5 6600 @ 3.30GHz, 메모리 DDR4 8Gbytes, 그래픽 카드 NVIDIA GeForce GTX 1060 3GB를 사용하였다.

실행 시 작업 단계별로 소요된 시간(ms)은 <표1>과 같다. Cropped_1의 포인트 수가 더 많음으로 약 30초, Cropped_2는 약 26초로 Cropped_1에 비해 약 4초 가량 빠른 결과를 보였다. 또, 가구 추출, 메쉬화에서 주로 많은 시간이 소요된 것을 확인할 수 있다.

| 작업 내용(시간: ms) | Cropped_1.ply | Cropped_2.ply |
|---|---|---|
| 주요 평면 및 Boundary 추출 | 1082.2 | 1200.2 |
| 가구 추출 및 후처리 | 21104.8 | 17452.0 |
| 벽 단순화 | 851.3 | 770.1 |
| 메쉬화 | 7323.5 | 6669.8 |
| 총 소요 시간(단위: s) | 30.3 | 26.1 |

<표1> - Cropped_1과 Cropped_2에 대한 작업 소요시간을 ms 로 나타낸 표이다.

프로그램을 실행하여 얻은 메쉬 파일과 가구를 나타낸 포인트 클라우드 파일을 Unity에서 출력한 결과는 아래와 같다.

| 원본(포인트 클라우드) | 실행 후(포인트 + 메쉬) |
|---|---|
|  <br>cropped_1.ply |  |
|  <br>cropped_2.ply |  |

# 4. 결론 및 향후 개발 방향

## 4.1 결론

기존의 원격 견본주택에서 360도 파노라마 사진을 활용하여 만든 공간의 경우 사진을 촬영했던 장소가 기준점으로 잡히기 때문에, 해당 기준을 벗어나 공간을 살펴볼 수 없었고, 사진이 촬영될 때의 가구 배치가 사진에 남기 때문에 가구를 지운 방의 모습을 보여 주고 싶다면 가구를 치우고 사진을 다시 찍어야 한다는 문제가 있었다.

그러나 Point Cloud Data를 기반으로 공간을 재구성하여 mesh로 가공하게 되면, 실제와 유사한 3D 공간을 생성하게 되고, 이 공간을 Point Cloud Data를 생성했던 기준을 벗어나서도 돌아볼 수 있으며, 가구 또한 360도 파노라마 사진과 달리 해당 포인트들을 지우고 Mesh를 만들면 가구가 비워진 방의 크기와 같은 공간이 나오므로 훨씬 간단하게 해당 문제를 해결할 수 있다.

## 4.2 향후 개발 방향

다른 가구의 3d 모델로 원래 공간의 존재하던 가구를 지우고, 원하는 가구를 원하는 위치에 채워 볼 수도 있을 것이다. 그러기 위해 먼저 가구들을 분류하여 지우고 지워진 공간을 매끄럽게 기존에 존재하는 공간과 어울리도록 재구성하는 식으로 만들 수 있다면 사용자가 직접 가구를 치우고 배치하지 않아도 어떤 모습이 나오는지를 화면으로 간단하게 확인할 수 있다.

이 방법을 활용한다면 견본 주택 내부의 가구 배치뿐만 아니라 넓은 지역의 Point Cloud Data에 건물 모델을 원하는 방식으로 올려보는 식으로 3D 건축 조감도를 만들어 실제와 비슷한 3d 이미지를 구축하는 데에도 사용할 수 있을 것이다.

## 4.3 Future Work

- **가구 모델 재배치**

[Model-based furniture recognition for building semantic object maps] 논문을 기반으로 가구를 구별하여, 기존 가구와 유사한 새로운 모델로 조정 및 배치

- **벽 생성 강화**

벽 생성 시 정확도를 향상시켜 공간을 직각으로 생성되게 하여 틈을 제거하는 기능

- **텍스쳐 정교화**

텍스쳐링 과정에서 더욱 정교한 이미지 샘플링을 통해 텍스쳐 퀄리티를 향상시킨다

# 5. 개발 일정 및 분담

## 5.1 개발일정

<table>
<tr><td colspan="3">5월</td><td colspan="4">6월</td><td colspan="4">7월</td><td colspan="4">8월</td><td colspan="4">9월</td></tr>
<tr><td>2</td><td>3</td><td>4</td><td>1</td><td>2</td><td>3</td><td>4</td><td>1</td><td>2</td><td>3</td><td>4</td><td>1</td><td>2</td><td>3</td><td>4</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td colspan="3">Point cloud 관련 스터디</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td colspan="4">주요 평면 및 Boundary Points 추출 기능 구현</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="3">가구 추출 및 후처리</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="2">중간보고서</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="4">벽 단순화 및 공간 투영</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="2">메쉬화 및 텍스쳐링</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="2">최종보고서</td></tr>
</table>

## 5.2 역할 분담

| 이름 | 역할 분담 |
|---|---|
| **기태욱** | 이미지 텍스쳐 추출<br>공간 스케일 설정 및 테스트<br>벽, 천장, 바닥 법선 편집 기능 구현<br>외곽선 추출을 위한 alphashape 기능 구현 |
| **서준오** | 벽, 천장, 면 메쉬화<br>천장, 바닥의 정확한 Z값 검출 및 압축 기능 구현<br>Corner Detection 알고리즘 구현<br>이미지 텍스쳐링 및 Unity를 활용한 시각화 |
| **이해성** | Boundary Point Detection 알고리즘 구현<br>Boundary Point를 이용한 벽 제거 기능 구현<br>주요 평면을 추출하는 기능 구현<br>포인트들의 normal 정렬 기능 구현<br>교차점을 이용한 벽 평탄화/생성 기능 구현<br>가구 분류 및 재배치 연구 |

# 6. 부록

```python
def alpha_concave(points, alpha=10.0):
    boundary = alphashape.alphashape(points, alpha)  # 3.44

    x = []
    y = []

    some_poly = boundary

    # Extract the point values that define the perimeter of the polygon
    x, y = some_poly.exterior.coords.xy
    out = [None] * (len(x) - 1)
    for i in range(0, len(x) - 1):
        out[i] = [x[i], y[i]]

    result = np.array(out)
    pcd = o3d.geometry.PointCloud()
    pcd.points = o3d.utility.Vector3dVector(np.pad(np.array(result), (0, 1),
'constant', constant_values=0)[:-1, :])

    return pcd
```

<코드1> alpha_concave.py

```python
import numpy as np
import open3d as o3d
import plane_extractor
from skspatial.objects import Plane

# Input data type ---------------------------------------------
# points : <class 'numpy.ndarray'>
def make_wall(boundary_points, min_z, high_z, sample_rate=0.075) :
    big = max(min_z, high_z)
    small = min(min_z, high_z)

    result = np.zeros((1,3))

    for i in np.arange(small, big, sample_rate) :
        temp = np.pad(np.array(boundary_points), [(0,0),(0, 1)], 'constant',
constant_values=i)
        result = np.concatenate((result, temp), axis=0)

    # return type is <class 'numpy.ndarray'>
    return result[1:]
```

<코드2> - 벽을 생성하는 코드 make_wall.py

```
# Input data type ----------------------------------------------
# points : <class 'numpy.ndarray'>
def remove_boundary_noise(points):
    pb = o3d.geometry.PointCloud()
    pb.points = o3d.utility.Vector3dVector(np.pad(np.array(points), (0, 1),
'constant', constant_values=0))
    cl, ind = pb.remove_statistical_outlier(nb_neighbors=3,
                                                    std_ratio=2.0)

    inlier_cloud = pb.select_by_index(ind)
    return inlier_cloud
    # return type is <class 'open3d.cpu.pybind.geometry.PointCloud'>

# Input data type ----------------------------------------------
# point_cloud : <class 'open3d.cpu.pybind.geometry.PointCloud'>
# boundary : <class 'numpy.ndarray'>
def generate_wall_points(point_cloud, boundary, save_filename=None):
    plane, high_z, min_z = plane_extractor.extract_ceil(point_cloud)
    improve_boundary = remove_boundary_noise(boundary)

    pre_arr = np.array(improve_boundary.points)

    my_arr = []
    for i in range(len(pre_arr)):
        arr_n = []
        for j in range(len(pre_arr[i])-1) :
            arr_n.append(pre_arr[i][j])
        my_arr.append(arr_n)

    wall = make_wall(my_arr, min_z, high_z)

    if save_filename != None:
        np.savetxt(save_filename, wall[1:])

    pf = o3d.geometry.PointCloud()
    pf.points = o3d.utility.Vector3dVector(wall[1:])
    o3d.visualization.draw_geometries([pf])
    return wall[1:]

def extend_wall(boundary_pc, min_z, high_z, sample_rate=0.001, min_points=10) :
    result = []
    save = []
    wall_points = make_wall(np.asarray(boundary_pc.points)[:,:2], 0, 1,
sample_rate=0.1)
    wall_pc = o3d.geometry.PointCloud()
    wall_pc.points = o3d.utility.Vector3dVector(wall_points)
    o3d.visualization.draw_geometries([wall_pc])
    plane_models = []
```

<코드2> - 벽을 생성하는 코드 make_wall.py

```
while( (np.asarray(wall_pc.points).shape[0]/10) > min_points ):
        print(np.asarray(wall_pc.points).shape[0]/10)
        # 1. wall_pc 에서 segment 를 구한다.
        [a, b, c, d], inliers = wall_pc.segment_plane(distance_threshold=0.01,
                                                      ransac_n=3,
                                                      num_iterations=4000)
        plane_models.append([a, b, c, d])
        inlier_cloud = wall_pc.select_by_index(inliers)

        if (-1.03 <= a <= -0.97 or 0.97 <= a <= 1.03) and -0.08 < b < 0.08 :
            a = 1.0
            b = 0.0
        print(f"Plane equation: {a:.2f}x + {b:.2f}y + {c:.2f}z + {d:.2f} = 0")
        #plane_model 을 가공한다.
        # (ax + by + cz + d = 0) 꼴에서, c=0 으로 생각하고, y = -(a/b)x - d/b 로
구한다.
        a = -(a/b)
        d = -(d/b)

        #2. 구한 segment 를 wall_pc 의 점을 제거하며 확장한다.
        inlier_points = np.asarray(inlier_cloud.points)
        min_x, min_y = np.min(inlier_points[:, 0]), np.min(inlier_points[:, 1])
        max_x, max_y = np.max(inlier_points[:, 0]), np.max(inlier_points[:, 1])

        # 2-1. segment 의 plane_model 과 , 최소 좌표 point 부터 최대 좌표 point 까지
서치하며 해당 점 주변에 점이 있는지 확인한다.
        # 있는 경우, 주변 점을 제거하고, 평면을 확장한다.
        # 없는 경우, 설정된 오차 허용 횟수에 따라 더 2-1 를 수행한다.
        #A. min_x ~ max_x 사이의 값은 모두 포함한다.
        for x in np.arange(min_x-1, max_x+1, sample_rate) :
            y = a*x + d
            save.append([x,y,0])
            for i in np.arange(0, 1, 0.1) :
                result.append([x,y,i])

        pc = o3d.geometry.PointCloud()
        pc.points = o3d.utility.Vector3dVector(np.asarray(result))
        #o3d.visualization.draw_geometries([pc])
        #o3d.visualization.draw_geometries([pc, wall_pc])

        dists = np.asarray(wall_pc.compute_point_cloud_distance(pc))
        ind = np.where(dists > 0.10)[0]
        wall_pc = wall_pc.select_by_index(ind)
        """
        error = 0
        #B. min_x 보다 작은 값에서 조사한다.
        while(error < error_tolerance) :
            for i in boundary_pc :
                #
                if () :
                    error = error + 1
```

<코드2> - 벽을 생성하는 코드 make_wall.py

```
error = 0
        #C. max_x 보다 큰 값에서 조사한다.
        while (error < error_tolerance):
            if ():
                error = error + 1

        z = np.arange(min_z, high_z, radius * 0.7)
        pillar_points = np.tile(boundary[i], (z.shape[0], 1))

        pillar_points = np.hstack((pillar_points, z.reshape(z.shape[0], 1)))
        pillar_pc = o3d.geometry.PointCloud()
        pillar_pc.points = o3d.utility.Vector3dVector(pillar_points)

        dists = np.asarray(pc.compute_point_cloud_distance(pillar_pc))
        ind = np.where(dists > radius)[0]
        pc = pc.select_by_index(ind)
        print(i)
        """
    cross = get_cross(plane_models)
    cpc = o3d.geometry.PointCloud()
    cpc.points = o3d.utility.Vector3dVector(np.asarray(cross))

    dists = np.asarray(cpc.compute_point_cloud_distance(boundary_pc))
    ind = np.where(dists > 0.4)[0]
    cpc = cpc.select_by_index(ind, invert=True)
    cpc.paint_uniform_color([1, 0, 0])
    o3d.visualization.draw_geometries([cpc])
    cross = np.unique(np.round(np.asarray(cpc.points),8), axis=0)

    print(cross)

    sort_cross_index = sort_ccw(cross)
    print("sort_cross")
    print(sort_cross_index)
    result = make_wall(interpolate_wall(cross, sort_cross_index), min_z, high_z,
sample_rate=0.04)
    return result

    #3. 확장이 완료되면, result 에 해당 평면을 넣고 1 을 반복한다.
    #4. 이를 점의 갯수가 충분히 제거될때까지 반복하고 result 를 리턴한다.
    #5. 다른 함수에서 result 의 코너를 체크한다. (확장된 벽면이 서로 교차하기 때문)
    #6. 얻은 코너를 시계방향으로 정렬하여(topological sort) 해당 점끼리 연결한다.
    #7. 천장과 바닥을 붙인다.
    #8. 빈공간을 비운다.
    #9. 메쉬화를 수행한다.
```

<코드2> - 벽을 생성하는 코드 make_wall.py

```
#여러 평면 모델 (ax+by+cz+d=0)을 넣어 평면의 교점들을 찾아 리턴
def get_cross(plane_models) :
    result = []
    for i in range(len(plane_models)) :
        a1, b1, c1, d1 = plane_models[i]

        for j in range(len(plane_models)) :
            if i != j :
                a2, b2, c2, d2 = plane_models[j]

                if a1*b2 - a2*b1 == 0 :
                    pass

                x = (b1*d2 - b2*d1)/(a1*b2 - a2*b1)
                y = -(a1/b1 * x) - d1/b1
                print([x,y])
                result.append([x,y,0])

    return np.asarray(result)

#주어진 점을 반시계방향으로 정렬하여 오름차순으로 해당 점을 가리키는 index 리스트 리턴
def sort_ccw(points) :
    center = [np.mean(points[:, 0]), np.mean(points[:, 1]), 0]

    angle = []
    for i in range(points.shape[0]):
        delta = points[i] - center
        if delta[1] < 0:
            deg = 360 - np.degrees(np.arccos(delta[0] / np.linalg.norm(delta)))
        else:
            deg = np.degrees(np.arccos(delta[0] / np.linalg.norm(delta)))

        dist = np.linalg.norm(delta)
        angle.append([i, deg, dist])


    angle.sort(key=lambda v: (v[1], v[2]))

    print(angle)
    angle = np.int64(np.asarray(angle))
    print(points)

    for i in range(len(points)) :
        print(points[angle[i, 0]])
        print(angle[i])
        _pc = o3d.geometry.PointCloud()
        _pc.points = o3d.utility.Vector3dVector(points[angle[:i, 0]])
        #o3d.visualization.draw_geometries([_pc])

    #o3d.visualization.draw_geometries([_pc])
    return angle[:, 0]
```

<코드2> - 벽을 생성하는 코드 make_wall.py

```python
def interpolate_wall(points, sorted_index, split_count=60) :
    result = []
    points_length = len(points)
    for i in range(points_length+1) :
        temp = []
        curr_point = points[sorted_index[i % points_length]]
        next_point = points[sorted_index[(i+1) % points_length]]
        delta = (next_point - curr_point) / split_count

        for j in range(split_count+1) :
            temp.append(curr_point + delta * j)

        result += temp

    result = np.concatenate((result, points), axis=0)

    return np.asarray(result)[:, :2]
```

<코드2> 벽을 생성하는 코드 make_wall.py

```python
import numpy as np
import open3d as o3d
import texture

def display_outlier(points, ind):
    inlier = points.select_by_index(ind)
    outlier = points.select_by_index(ind, invert=True)
    inlier.paint_uniform_color([0.7, 0.7, 0.7])
    outlier.paint_uniform_color([1, 0, 0])
    o3d.visualization.draw_geometries([inlier, outlier])

def extract_ceil(pointcloud, show_progress=False, voxel_size=0.1,
nb_neighbors=30, std_ratio=2.0, z_ratio=0.25,
                 save_filename=None):
    if show_progress:
        o3d.visualization.draw_geometries([pointcloud])

    # remove outlier
    pc_down = pointcloud.voxel_down_sample(voxel_size=voxel_size)
    cl, ind = pc_down.remove_statistical_outlier(nb_neighbors=nb_neighbors,
std_ratio=std_ratio)
    if show_progress:
        display_outlier(pc_down, ind)

    pc_down = pc_down.select_by_index(ind)

    # extract ceil
    high_z = -10000
    min_z = 1000000
    high_ind = 0
    low_ind = 0
    points = pc_down.points
    for i in range(len(points)):
        if points[i][2] > high_z:
            high_z = points[i][2]
            high_ind = i

        if points[i][2] < min_z:
            min_z = points[i][2]
            low_ind = i

    if show_progress:
        print(f'{high_z} , {min_z}')
        high_point = pc_down.select_by_index([high_ind])
        low_point = pc_down.select_by_index([low_ind])
        temp = pc_down.select_by_index([high_ind, low_ind], invert=True)
        temp.paint_uniform_color([0.7, 0.7, 0.7])
        high_point.paint_uniform_color([1, 0, 0])
        low_point.paint_uniform_color([0, 1, 0])

        o3d.visualization.draw_geometries([low_point, high_point, temp])
```

<코드3> - 평면을 추출하는 코드 plane_extractor.py

```
# Parameter
    z_threshold = abs(high_z - min_z) * z_ratio
    inliers = []
    for i in range(len(points)) :
        if points[i][2] >= z_threshold :
            inliers.append(i)

    # remove outlier in ceil
    segments = pc_down.select_by_index(inliers)
    if show_progress :
        o3d.visualization.draw_geometries([segments])

    cl, ind = segments.remove_statistical_outlier(nb_neighbors=nb_neighbors,
std_ratio=std_ratio)
    if show_progress:
        display_outlier(segments, ind)
    segments = segments.select_by_index(ind)

    #get plane Z value
    plane_model, inliers = segments.segment_plane(distance_threshold=0.01,
                                                  ransac_n=3,
                                                  num_iterations=2000)
    [a, b, c, d] = plane_model
    print(f"Plane equation: {a:.2f}x + {b:.2f}y + {c:.2f}z + {d:.2f} = 0")

    ceil_cloud = segments.select_by_index(inliers)

    # project to highest Z value
    points = np.array(segments.points)
    points[:, 2] = -d/c
    segments.points = o3d.utility.Vector3dVector(points)

    if show_progress:
        o3d.visualization.draw_geometries([segments])

    points = np.array(segments.points)

    if save_filename != None:
        np.savetxt(save_filename, points[:, :2])

    return points[:, :2], -d/c, min_z, ceil_cloud
```

<코드3> - 평면을 추출하는 코드 plane_extractor.py

```
def extract_floor(pointcloud, show_progress=False, voxel_size=0.1,
nb_neighbors=30, std_ratio=2.0, z_ratio=0.5,
                  save_filename=None):
    if show_progress:
        o3d.visualization.draw_geometries([pointcloud])

    # remove outlier
    pc_down = pointcloud.voxel_down_sample(voxel_size=voxel_size)
    cl, ind = pc_down.remove_statistical_outlier(nb_neighbors=nb_neighbors,
std_ratio=std_ratio)
    if show_progress:
        display_outlier(pc_down, ind)

    pc_down = pc_down.select_by_index(ind)

    # extract ceil
    high_z = -10000
    min_z = 1000000
    high_ind = 0
    low_ind = 0
    points = pc_down.points
    for i in range(len(points)):
        if points[i][2] > high_z:
            high_z = points[i][2]
            high_ind = i

        if points[i][2] < min_z:
            min_z = points[i][2]
            low_ind = i

    if show_progress:
        print(f'{high_z} , {min_z}')
        high_point = pc_down.select_by_index([high_ind])
        low_point = pc_down.select_by_index([low_ind])
        temp = pc_down.select_by_index([high_ind, low_ind], invert=True)
        temp.paint_uniform_color([0.7, 0.7, 0.7])
        high_point.paint_uniform_color([1, 0, 0])
        low_point.paint_uniform_color([0, 1, 0])

        o3d.visualization.draw_geometries([low_point, high_point, temp])

    # Parameter
    z_threshold = (max(high_z, min_z) - min(high_z, min_z)) * z_ratio
    inliers = []
    for i in range(len(points)) :
        if points[i][2] <= z_threshold :
            inliers.append(i)
```

<코드3> - 평면을 추출하는 코드 plane_extractor.py

```
# remove outlier in ceil
    segments = pc_down.select_by_index(inliers)
    if show_progress :
        o3d.visualization.draw_geometries([segments])

    cl, ind = segments.remove_statistical_outlier(nb_neighbors=nb_neighbors,
std_ratio=std_ratio)
    if show_progress:
        display_outlier(segments, ind)
    segments = segments.select_by_index(ind)

    #get plane Z value
    plane_model, inliers = segments.segment_plane(distance_threshold=0.01,
                                                  ransac_n=3,
                                                  num_iterations=2000)
    [a, b, c, d] = plane_model
    print(f"Plane equation: {a:.2f}x + {b:.2f}y + {c:.2f}z + {d:.2f} = 0")

    floor_cloud = segments.select_by_index(inliers)
    color = texture.get_mean_color(floor_cloud)

    # project to highest Z value
    points = np.array(segments.points)
    points[:, 2] = -d/c
    segments.points = o3d.utility.Vector3dVector(points)

    if show_progress:
        o3d.visualization.draw_geometries([segments])

    points = np.array(segments.points)
    segments.colors = o3d.utility.Vector3dVector(np.tile(color,
reps=[np.asarray(segments.points).shape[0], 1]))
    if save_filename != None:
        np.savetxt(save_filename, points[:, :2])

    return points[:, :2], -d/c, min_z, segments
```

<코드3> - 평면을 추출하는 코드 plane_extractor.py

```python
import numpy as np
import open3d as o3d

"""
@Params : points : numpy array
          boundary : numpy array
          raidus : float
          min_z  : float
          high_z : float
@Return  : pointcloud data
"""
def remove_cylinder(pc, boundary, radius, min_z, high_z) :
    for i in range(boundary.shape[0]) :
        z = np.arange(min_z, high_z, radius * 0.7)
        pillar_points = np.tile(boundary[i], (z.shape[0], 1))

        pillar_points = np.hstack((pillar_points, z.reshape(z.shape[0], 1)))
        pillar_pc = o3d.geometry.PointCloud()
        pillar_pc.points = o3d.utility.Vector3dVector(pillar_points)

        dists = np.asarray(pc.compute_point_cloud_distance(pillar_pc))
        ind = np.where(dists > radius)[0]
        pc = pc.select_by_index(ind)
        print(i)
    return pc

"""
@Params : points : numpy array
          boundary : numpy array
@Return  : numpy array
"""
def remove_wall(pc, boundary, method="edge", radius=0.5, box_size=0.5,
min_z=None, high_z=None) :
    if method == "edge" :
        #use edge algorithm
        pass
    if method == "box" :
        #use box algorithm
        pass
    if method == "cylinder" :
        result = remove_cylinder(pc, boundary, radius=radius, min_z=min_z,
high_z=high_z)

    return result
```

<코드4> - 벽을 제거하는 알고리즘을 구현한 remove_wall.py

```
"""
@Params : pointcloud : point cloud object
          z_high : float
          z_low  : float
@Return  : point cloud object
"""
def remove_ceil_and_floor(pointcloud, z_high, z_low) :
    points = pointcloud.points
    inliers = []
    for i in range(len(points)):
        if points[i][2] <= z_high - 0.08 and points[i][2] >= z_low + 0.04:
            inliers.append(i)

    return pointcloud.select_by_index(inliers)
```

<코드4> - 벽을 제거하는 알고리즘을 구현한 remove_wall.py

```
import open3d as o3d
import numpy as np
import make_wall
import remove_wall
import plane_extractor
import alpha_concave
import matplotlib.pyplot as plt

def concat_pc(pc1, pc2):
    result = o3d.geometry.PointCloud()
    points = np.concatenate((np.asarray(pc1.points), np.asarray(pc2.points)), axis=0)
    colors = np.concatenate((np.asarray(pc1.colors), np.asarray(pc2.colors)), axis=0)
    normals = np.concatenate((np.asarray(pc1.normals), np.asarray(pc2.normals)), axis=0)

    result.points = o3d.utility.Vector3dVector(points)
    result.colors = o3d.utility.Vector3dVector(colors)
    result.normals = o3d.utility.Vector3dVector(normals)

    return result

#for cropped_1
"""
ALPHA = 10.0
MAKE_WALL_SAMPLE_RATE = 0.1
MIN_POINTS = 20
REMOVE_WALL_RADIUS = 0.053
"""

#for cropped_2
ALPHA = 10.0
MAKE_WALL_SAMPLE_RATE = 0.1
MIN_POINTS = 10
REMOVE_WALL_RADIUS = 0.053

pc = o3d.io.read_point_cloud("cropped_2.ply")

#extract ceil, floor
_, high_z, min_z, floor = plane_extractor.extract_floor(pc, show_progress=False)    # 면 뜯어내기(다운샘플 + z축 편집)
plane, high_z, min_z, ceil = plane_extractor.extract_ceil(pc, show_progress=False)    # 면 뜯어내기(다운샘플 + z축 편집)

#extract boundary
boundary = alpha_concave.alpha_concave(plane, alpha=ALPHA)
o3d.visualization.draw_geometries([boundary])

#remove ceil, floor, wall
furniture = o3d.geometry.PointCloud()
removed = remove_wall.remove_wall(pc, np.asarray(boundary.points)[:, :2], method="cylinder",
radius=REMOVE_WALL_RADIUS, min_z=min_z , high_z=high_z)
removed = remove_wall.remove_ceil_and_floor(removed, high_z - 0.07, min_z + 0.07)
```

<코드5> - main.py

```
with o3d.utility.VerbosityContextManager(o3d.utility.VerbosityLevel.Debug) as cm:
    labels = np.array(
    removed.cluster_dbscan(eps=0.05, min_points=30, print_progress=True))
max_label = labels.max()

for i in range(max_label) :
    temp = o3d.geometry.PointCloud()
    index = np.where(labels, )
print(labels)

print(f"point cloud has {max_label + 1} clusters")

o3d.visualization.draw_geometries([removed])
o3d.io.write_point_cloud("furniture.ply", removed)

#extend wall
extend_p, extend_n = make_wall.extend_wall(boundary, min_z, high_z, min_points=MIN_POINTS)

extend_pc = o3d.geometry.PointCloud()
extend_pc.points = o3d.utility.Vector3dVector(extend_p)
extend_pc.normals = o3d.utility.Vector3dVector(extend_n * -1)
o3d.visualization.draw_geometries([extend_pc])

floor.normals = o3d.utility.Vector3dVector(np.tile([0,0,1],
reps=[np.asarray(floor.points).shape[0], 1]))
ceil.normals = o3d.utility.Vector3dVector(np.tile([0,0,-1],
reps=[np.asarray(ceil.points).shape[0], 1]))

extend_pc = concat_pc(extend_pc, ceil)
extend_pc = concat_pc(extend_pc, floor)

#reconstruction
o3d.visualization.draw_geometries([extend_pc])

with o3d.utility.VerbosityContextManager(
        o3d.utility.VerbosityLevel.Debug) as cm:
    mesh, densities = o3d.geometry.TriangleMesh.create_from_point_cloud_poisson(
        extend_pc, depth=8)

o3d.visualization.draw_geometries([mesh])

o3d.io.write_triangle_mesh("mesh.obj",mesh, print_progress=True)

f = o3d.io.read_point_cloud("furniture.ply")
m = o3d.io.read_triangle_mesh("mesh.ply")
```

<코드5> - main.py