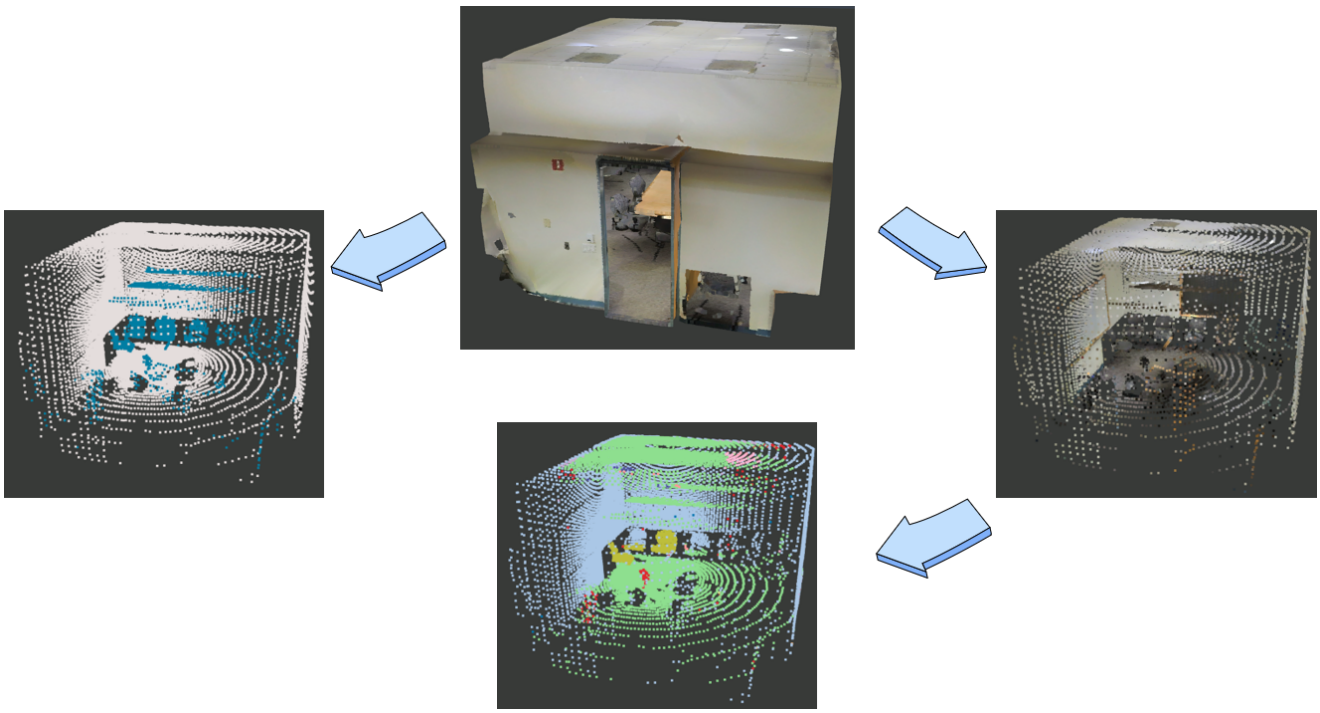
# Analyzing the Impact of Occlusion on the Quality of Semantic Segmentation Methods for Point Cloud Data



**Bachelor Thesis**
13th March 2023 - 13th September 2023

Yufeng Xiao
19-763-663

Supervisors:
Prof. Dr. Renato Pajarola
Lizeth Joseline Fuentes Perez

Visualization and MultiMedia Lab
Department of Informatics
University of Zürich

University of Zurich[UZH]

VISUALIZATIONANDMULTIMEDIALAB

# Abstract

This thesis aims to analyze the impact of occlusion on the quality of semantic segmentation methods for point cloud data. Occlusion is a prevalent phenomenon in 3D scenes, where objects often overlap or obstruct each other. This can significantly compromise the quality and integrity of data, leading to inaccuracies in semantic segmentation. While the issue of occlusion has garnered attention in 3D data processing, current research on how different occlusion levels impact the quality of semantic segmentation is rare. Specifically, there is a palpable gap in understanding how to quantify occlusion in the scene and how this characteristic influence the performance of advanced semantic segmentation software like the Minkowski Engine. To bridge the research gap, we proposed a novel metric to quantify the occlusion level of a scene. We then applied this metric to analyze the impact of occlusion on the quality of semantic segmentation methods for point cloud data. Our results show that the occlusion level of a scene has limited impact to the quality of semantic segmentation.

# Contents

# List of Figures

# Listings

# 1 Introduction and Related Works

Artificial intelligence has received significant attention in recent research. Fields such as natural language processing, image generation, and autonomous driving have benefited from considerable investments, and some related applications have been successfully implemented. Currently, the practical implementation of autonomous driving is still challenging due to the high demands for model performance and safety. To achieve its goals, AI needs to precisely understand its surrounding environment. Hence, semantic segmentation of scenes has become an essential topic. In this thesis, as opposed to focusing on outdoor scenes commonly associated with autonomous driving, we are more interested in understanding indoor environments. The correct classification and comprehension of indoor objects can assist in various aspects of human life, such as creating intelligent robots to help humans with a series of tasks. To understand scenes accurately, we need to provide AI with high-precision data, typically point clouds, which are a collection of data points defined in a three-dimensional coordinate system, represent the external surface of an object. These data points can capture the shape, and sometimes color, of the physical entities in a scanned environment. They are usually obtained from laser scanners or multi-view reconstructions. However, when collecting this type of data, objects are often obstructed due to the viewpoint of the scanner, leading to occlusions in the data, as shown in Figure 1.1, where part of occlusions in a scene are marked with red box. Therefore, we aim to explore how occlusion affects AI's understanding of a scene. More specifically, we want to propose a metric to reflect the occlusion level of a scene and analyze how this characteristic influences the performance of semantic segmentation methods.

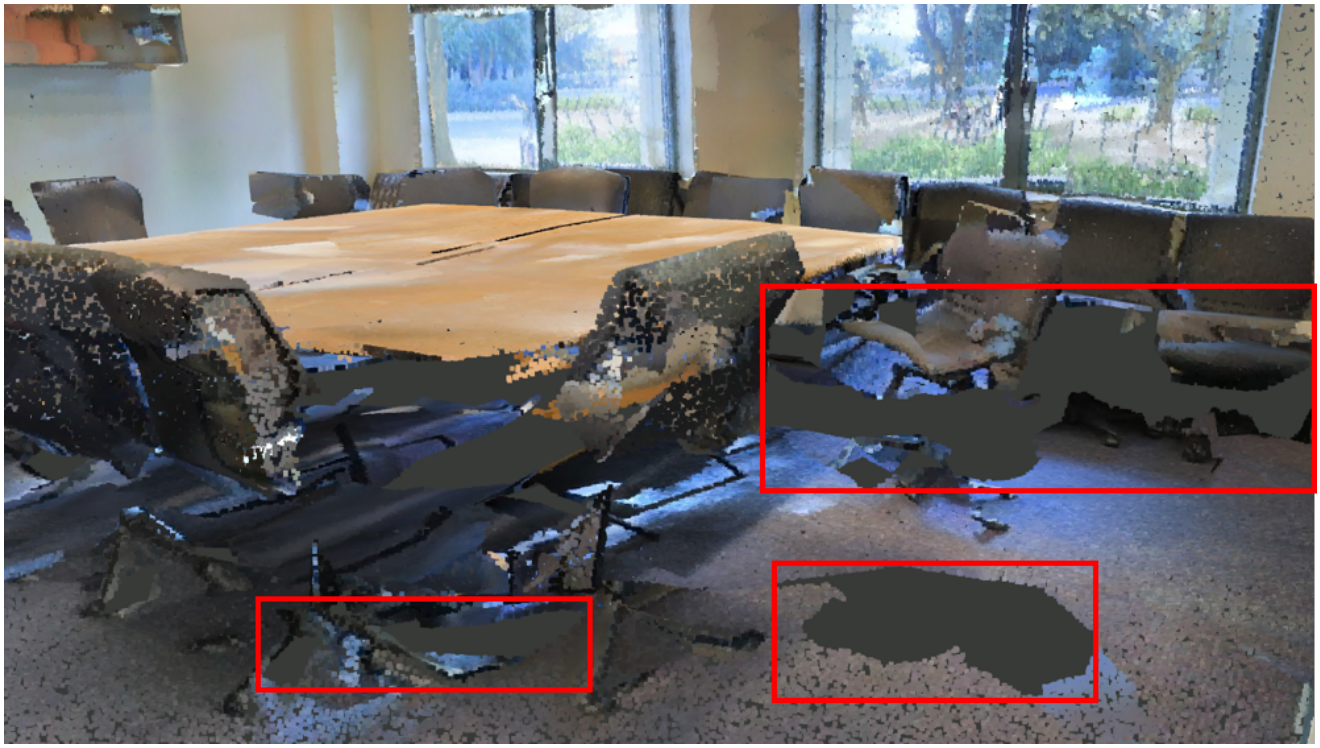


Figure 1.1: Occlusion in a scene

## 1.1 Previous work

To the best of our knowledge currently there is not explicit works that compute occlusion level for an entire indoor scene. There are some related works handling occlusion of point cloud. In this section, we will briefly introduce these works, especially the ones that are related to occlusion.

**Occlusion Guided Scene Flow Estimation on 3D Point Clouds.** This paper [OR21] presents the OGSF-Net, a novel architecture designed to address the challenges of occlusions in 3D scene flow estimation. Occlusions, where certain regions in one frame are hidden in another, can hinder accurate flow estimation. The OGSF-Net uniquely integrates the learning of flow and occlusions, using an occlusion handling mechanism in its Cost Volume layer to measure frame correlations. This approach aims to enhance both flow accuracy and the understanding of occluded regions, marking a pioneering effort in 3D scene flow estimation on point clouds. To make this method robust, it is important to make an accurate occlusion prediction. Therefore, they evaluate the performance of occlusion estimation on data set FlyingThings3D [MIH+16], which provides the ground truth occlusion mask of point cloud. For evaluation, they use accuracy and F1 score as metrics. This inspires us to apply similar metrics to evaluate the result of semantic segmentation.

**OcCo: Unsupervised Point Cloud Pre-training via Occlusion Completion.** In this work [WLY+21], the authors proposed *Occlusion Completion*, an unsupervised pre-training method. The main idea is to generate occlusion in a point cloud, then use an encoder-decoder model to reconstruct occluded points, and apply the encoder weights for downstream tasks. What is interesting for us is how they generate occlusion in a scene. They view point cloud from a camera, which is placed in different viewpoints. At each viewpoint, points are projected to a camera reference frame, if some points share the same pixel coordinates, then there might be occlusion. This could also be an inspiration for us to generate occlusion in a scene.

## 1.2 Positioning and Contributions

Previous work has shown that occlusion of a point cloud can be estimated based on ground truth occlusion information. However, in our work we have to estimate occlusion level of a scene without knowing such information. To achieve this, we will define occlusion as the amount of area occluded from a particular scan viewpoint. To estimate occlusion we defined, we propose the metric *occluded area ratio* to represent occlusion level in the scene which is represented by a triangulated mesh.

This work is focusing on the data set of point cloud. Since there is no property can quantify area in point cloud we cannot directly extend the metric *occluded area ratio* to point-based data structure. Therefore, we have to use a different metric to estimate occlusion level, here we propose *Boundary ray ratio* which is the ray that intersect with boundary points in point cloud. We also defined boundary points based on their semantic labels.

## 1.3 Technical Background

Based on the introduction in previous sections, we would elaborate on some key concepts in the technical background of this work.

### 1.3.1 Point Cloud Data

A point cloud is a discrete set of data points in space. The points may represent a 3D shape or object. Each point position has its set of Cartesian coordinates (X, Y, Z), in some cases it can also include color(RGB) or intensity information. Point clouds are generally produced by 3D scanners or by photogrammetry software, which measure many points on the external surfaces of objects around them.

Point cloud can be used in different areas. One usage is for rendering and modeling. Typically 3D objects are modeled using polygon meshes, and polygons are the rendering primitives in the graphics pipeline. However, representing all objects with point sampling allows easy mixing of objects in a scene without specialized algorithms for different geometry types. Other applications include depth sensing, perception, scientific computing etc.

### 1.3.2 Semantic Segmentation

Semantic segmentation for point cloud data has rapidly became a pivotal research domain, given its profound implications in many applications. From the intricate pathways navigated by autonomous driving to the precise movements of robotics and the detailed analysis of 3D scenes, the ability to accurately segment and categorize each data point in a 3D environment is important.

At the heart of this research lies the challenge of dealing with occlusions. In real-world scenarios, objects within a scene often overlap or obstruct each other, leading to partial or even complete occlusions. Such occlusions can significantly distort the spatial distribution of data points, making it challenging to distinguish the structure and category of the obstructed objects. For instance, in an urban driving scenario, a pedestrian might be partially hidden behind a parked car, or in an indoor scene, a chair might be obscured by a table. These occlusions can lead to misclassifications, reducing the overall accuracy of the segmentation.

**Minkowski Engine.** The Minkowski Engine [CGS19] is an auto-differentiation library specifically designed for sparse tensors. In the area of deep learning, where dense tensors are commonly used, the Minkowski Engine brings a fresh perspective by focusing on sparse tensors. This is particularly beneficial for 3D data, which often exhibits spatial sparsity. The engine supports all standard neural network layers, including convolution, pooling, unpooling, and broadcasting operations, but tailored for sparse tensors. Such capabilities make it an ideal choice for semantic segmentation tasks, especially when dealing with point cloud data. [Minkowski]

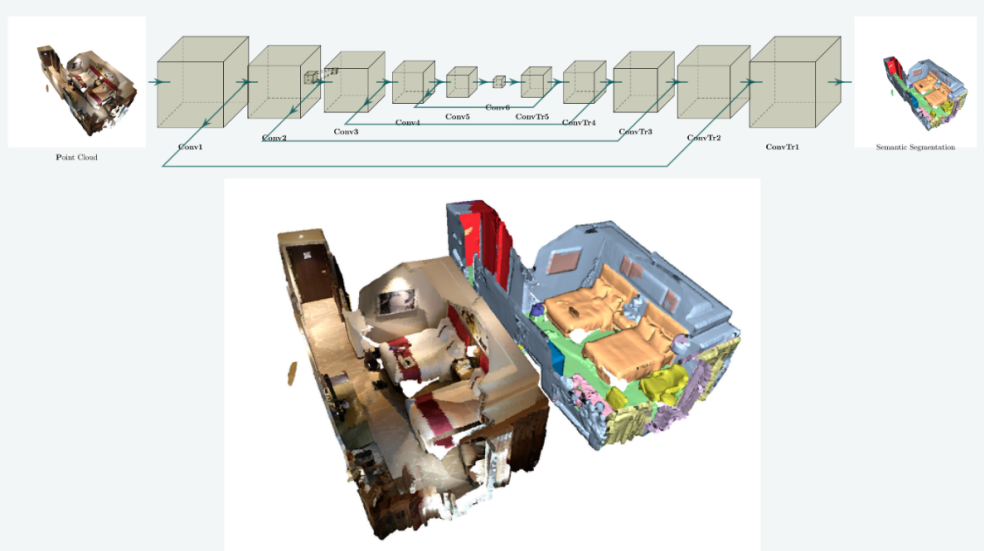



Figure 1.2: Minkowski Engine Indoor Scene Segmentation

## 1.4 Motivation

Among the many factors influencing the semantic segmentation of point cloud data, the level of occlusion stands as one of the major challenges. Occlusions, a prevalent phenomenon in 3D scenes, can significantly compromise the quality and integrity of data. When objects are partially or entirely obscured by others, semantic segmentation

might lead to inaccuracies in segmentation. While the issue of occlusion has garnered attention in 3D data processing, current research on how different occlusion levels impact the quality of semantic segmentation remains fragmented. Specifically, there is a palpable gap in understanding how to quantify occlusion levels and how these levels influence the performance of advanced tools like the Minkowski Engine. Thus, the primary motivation behind this research is to systematically evaluate level of occlusion and delve deep into their impact on point cloud data semantic segmentation.

## 1.5 Outline

In this thesis we will first briefly mention related works and discuss the part which bring us inspirations. We will also introduce related technical concepts to give reader richer background information. Then our contributions and motivations will be stated. In the next Chapter we elaborate on problems which we have to solve in order to compute metrics proposed before correctly. Our technical solution are listed and explained in detail with figures and formulas in Chapter 3. Once we have the solution, we will start to implement concrete codes. Thus, in Chapter 4 implementation details of our algorithm are explained together with the technical stack applied to support our computational pipeline. In the end of this part we will introduce the structure of our software to give insights on how the whole pipeline works.

For this thesis it is crucial to present our experimental results in a meaningful way. Hence, in Chapter 5 we need to first validate that those metrics we proposed can accurately reflect the level of occlusion of a point cloud. With successful validation we can then apply them to correlate with the performance of semantic segmentation. In the final step, we discuss the result of experiments and conclude this thesis.

# 2 Problem Statement

In this thesis we aim to find the correlation between occlusion level and performance of semantic segmentation. Given a data set include mesh and point cloud, we have to first prove that *occluded area ratio* can be computed correctly in a mesh. To extend this metric to point cloud, we would estimate a mesh from the cloud and we compute its occluded area ratio. Then we would directly generate a set of occluded point clouds to apply the *Boundary ray ratio* to estimate level of occlusion. To validate *Boundary ray ratio* is a reliable metric in estimation, its value should be close to the value of *occluded area ratio*. After the validation we can compute occlusion level of point cloud together with the performance of semantic segmentation, then we compare performance of point cloud with different occlusion level to see if these metrics can correlated with each other.

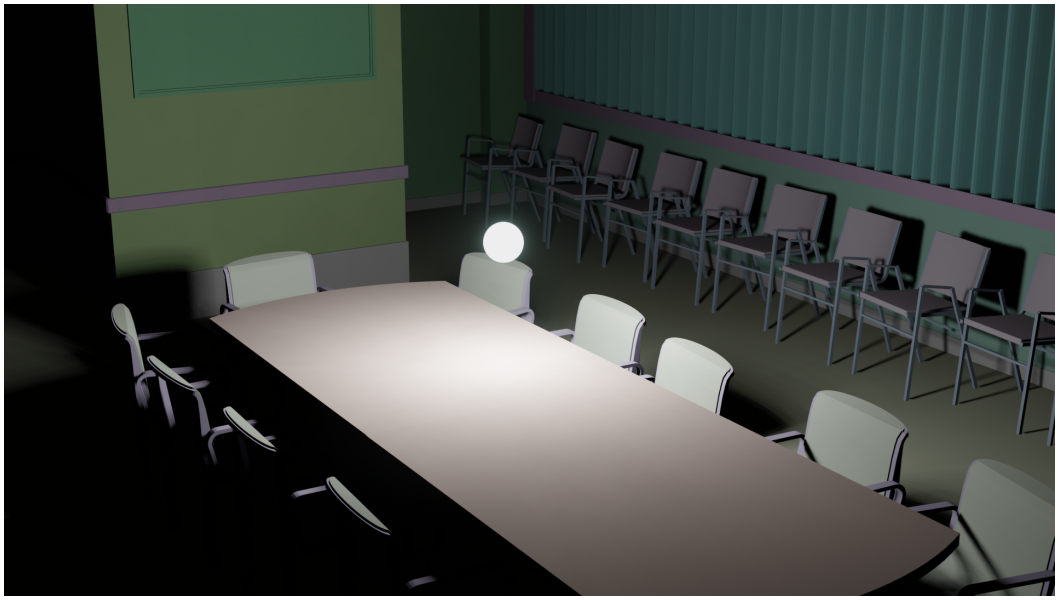
These problems are addressed in following points. Each of them will be provided with more detailed explanation in the next chapter where our technical solutions are given.

- Compute *occluded area ratio* of mesh.
- Estimate a mesh from the given point cloud.
- Compute the *occluded area ratio* for the estimated mesh and generate a new point cloud by sub-sampling the original point cloud.
- Apply the *Boundary ray ratio* metric to the new point cloud to estimate the level of occlusion and validate the reliability of the *Boundary ray ratio* metric by comparing its value with the *occluded area ratio*
- Compute the occlusion level and the performance of semantic segmentation of the point cloud, then compare these metrics between different clouds to find correlation.

## 2.1 Occluded Area Ratio of Mesh

In this part we have to prove that as the number of viewpoints increase, there will be more visible area. In another word, there is less occluded region and thus lower occlusion. As shown in figure 2.1, which is generated by simulating a scene with multiple light sources in blender [Ble23], the scene with 2 viewpoints has more bright area and less shadow, this also matches our intuition in real life.

To determine which area is visible and which is not is the key point here to compute occluded area ratio correctly. Obviously, the visible region to a viewpoint is where rays from viewpoint can reach. Hence, ray-tracing based algorithm will play an important role in computation of occlusion level.



(a) 1 viewpoint

(b) 2 viewpoints

Figure 2.1: Scene with Light Sources

## 2.2 Mesh Estimation from Point Cloud

If we want to compute *occluded area ratio* from a point cloud, we have to first generate mesh from it. This mesh is certainly only an estimation. Algorithm such as region growing [Rus23b] can segment points into clusters based on their properties. With these clusters we can estimate a plane for each of them, then we have a mesh composed by a set of planar primitives. The accuracy of estimation in this step will also affect subsequent process, hence, it is crucial for us to improve the result of mesh estimation.

In case of region growing, we need to tweak its input parameters. An example of a code snippet is shown in List 2.1, with many parameters to adjust, the result of region growing can be unstable, which also makes subsequent process difficult to control. Therefore, it is essential to find a more accurate way to estimate mesh.

**Listing 2.1: Region Growing**

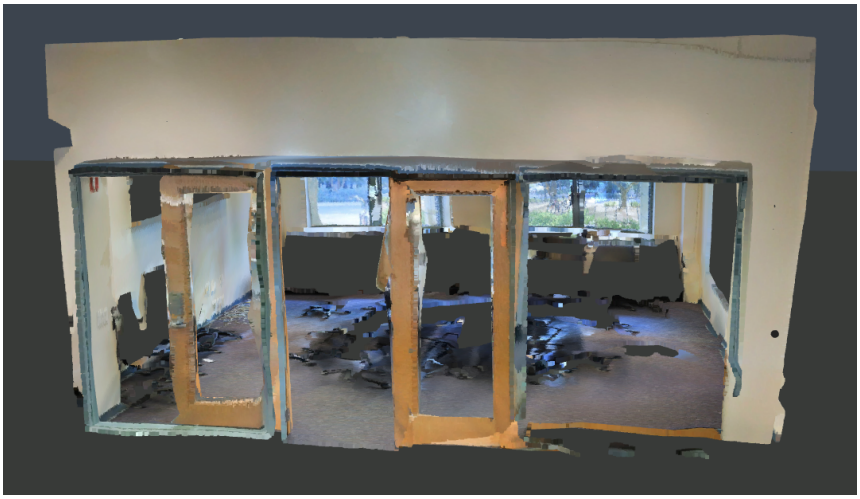
```
pcl::RegionGrowing<pcl::PointXYZ, pcl::Normal> reg;
reg.setMinClusterSize(min_cluster_size);
reg.setMaxClusterSize(max_cluster_size);
reg.setSearchMethod(tree);
reg.setNumberOfNeighbours(num_neighbours);
reg.setInputCloud(cloud);
reg.setInputNormals(normals);
reg.setSmoothnessThreshold(smoothness_threshold / 180.0 * M_PI);
reg.setCurvatureThreshold(curvature_threshold);
reg.extract(rg_clusters);
```

## 2.3 Sub-sample Original Point Cloud

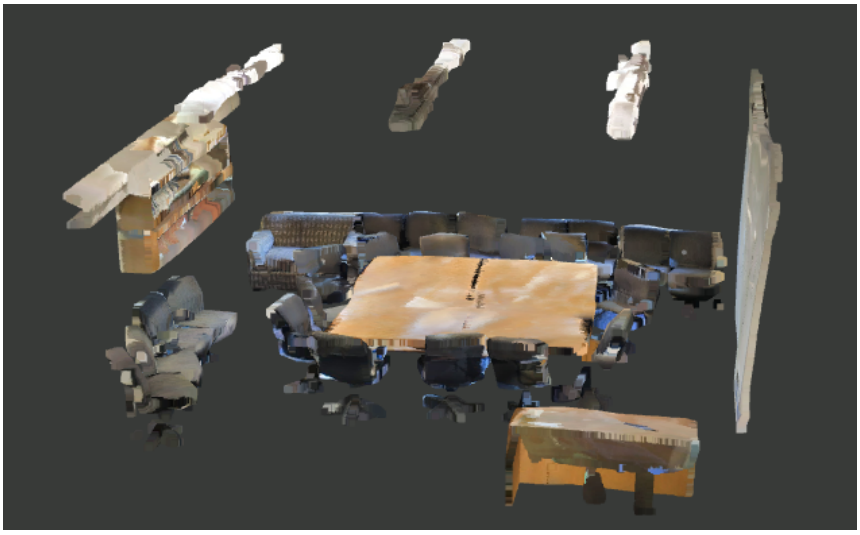
We would like to generate point clouds from the original cloud since for the validation of our metric we will need a set of comparable point clouds in terms of the proposed metric, where the *comparable* means that these clouds come from the same scene and thus they share the same structural information, hence, it's comparable and can be used for computation of the metric. This can be done by applying different strategies to sub-sample the original point cloud.

## 2.4 Boundary Ray Ratio of Point Cloud

A common way to generate point cloud is conducted by casting lasers from scanner. Since lasers cannot pass through objects, occlusion is then generated on those obscured surfaces. And they are mostly shown on the exterior structure of an indoor scene such as walls, floors and ceilings etc, as they are located in the outermost layer of the room. Items such as chairs and tables, they can also be obscured by other things, but it is difficult to estimate how much they are occluded since their structure is more complex compared to large structures like walls. An example of exterior and interior structure of point cloud is shown in Figure 2.2.



(a) Exterior



(b) Interior

Figure 2.2: Scene with Light Sources

Based on what we stated above, we choose *Boundary ray ratio* as the metric to estimate occlusion in point cloud. A set of rays will be cast to detect if it intersect with points identified as exterior structure, in this work we define these external points as boundary. With ground truth labeling of points, the classification of points can easily be done. Finally, we can compute a ratio related to ray that intersects boundary to represent occlusion level of a point cloud.

## 2.5 Performance Evaluation of Semantic Segmentation

This step is to find correlation between occlusion level and performance of semantic segmentation. From previous steps we can already compute occlusion level of a point cloud. The problem here is to use some metrics to evaluate the performance of segmentation. In Section 1.1, the author of the related work about scene flow, applied F1 score to evaluate their results. This will be considered as one of the options for evaluation. With metrics we computed before we can finally investigate how would the evaluation metrics change with different occlusion level.

# 3 Technical Solution

In this chapter we will introduce our technical solution to the problem stated in Chapter 2 in a detailed way. The overall workflow is shown below in Figure 3.1.

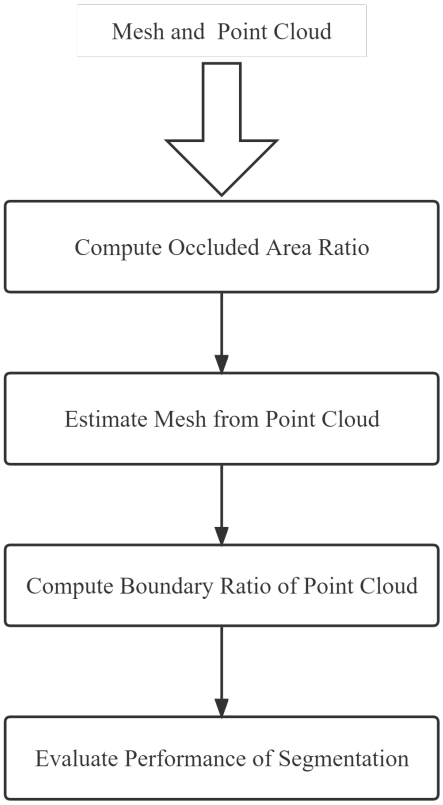



Figure 3.1: Overall Flowchart of Technical Solution

## 3.1 Compute Occluded Area Ratio

It is difficult to accurately compute occluded area of a triangle, since we don't exactly know which part is visible or occluded geometrically. The method applied here is to sample triangle, with samplings we can have a good coverage of the area. Next step is to compute each sampling's visibility to viewpoints in the scene. Then we are able to compute the ratio by counting how many points are occluded. A flow chart of the whole pipeline is shown below in Figure 3.2.

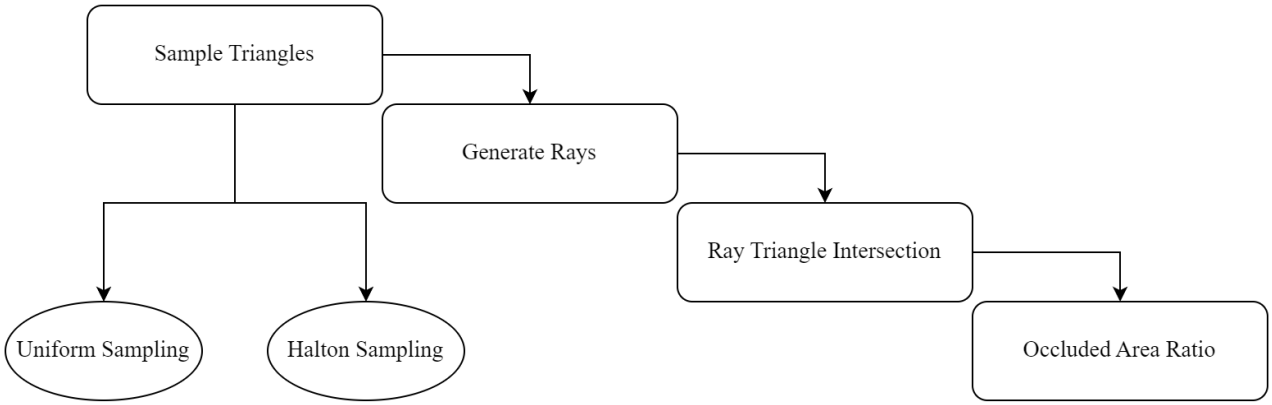



Figure 3.2: Flowchart of Computing occluded area ratio

### 3.1.1 Sample Triangles

In order to make samplings cover the area of triangle uniformly, we take 2 sampling algorithms into consideration. There are uniform sampling and halton sampling. The only difference of these methods is the way they generate random numbers. With these number we calculate barycentric coordinates based on vertices of triangle.

**Uniform Sampling** The process begins by randomly generating two parameters, $r_1$ and $r_2$, both of which lie in the interval [0, 1] with a uniform distribution, ensuring that each point within the interval has an equal probability of being selected. These random parameters are then used to compute the barycentric coordinates of the sampled points within the triangle. The barycentric coordinates, denoted as $\alpha$, $\beta$, and $\gamma$, allow us to express any point within the triangle as a linear combination of the triangle's vertices.

The algorithm of this part is shown below:

**Algorithm 1** Uniform Sampling within a Triangle

**Require:** Triangle vertices $V_1$, $V_2$, $V_3$
**Ensure:** Sampled point $P$ within the triangle
Generate two random numbers $r_1$ and $r_2$ with a uniform distribution in the interval [0, 1]
Compute the barycentric coordinates using $r_1$ and $r_2$ as follows:
$\alpha \leftarrow 1 - \sqrt{r_1}$
$\beta \leftarrow \sqrt{r_1} \times r_2$
$\gamma \leftarrow 1 - \alpha - \beta$
Compute the Cartesian coordinates of the sampled point $P$ as:
$P \leftarrow \alpha V_1 + \beta V_2 + \gamma V_3$
**return** $P$

**Halton Sampling** In the part of uniform sampling, we have introduced how to compute barycentric coordinates on a triangle. Therefore, here we focus on the generation of random numbers using the Halton sequence, a method that generates quasi-random numbers which are known to fill the space more uniformly compared to purely random numbers.

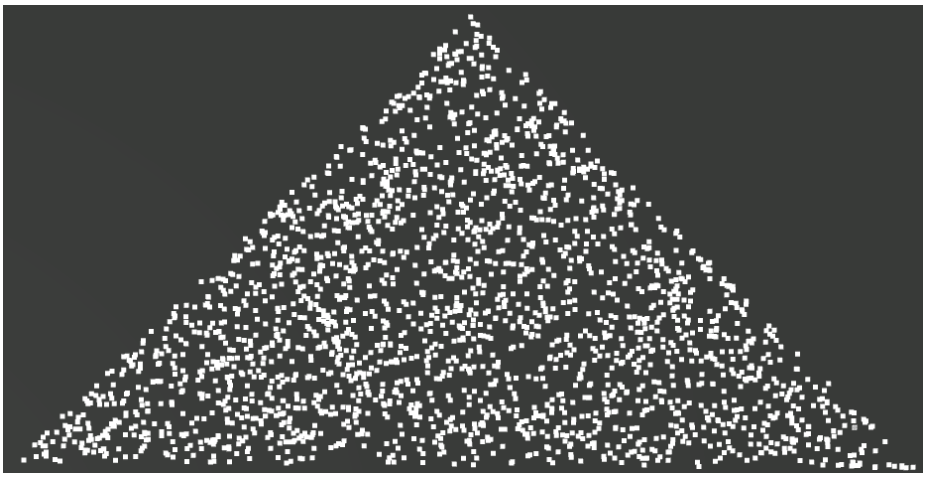
The algorithm to generate a number in the Halton sequence can be represented as:

**Algorithm 2** Halton Sequence Generation

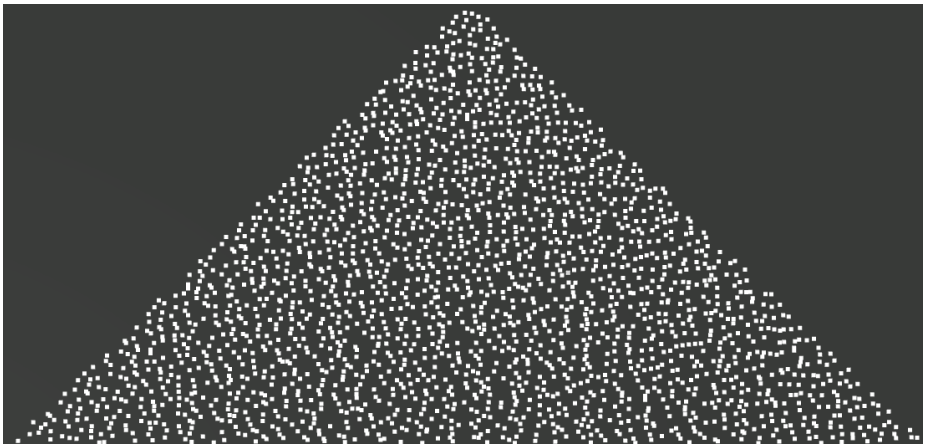
**Require:** $index \geq 0$, $base > 1$ (a prime number)
**Ensure:** Halton number corresponding to the given index and base

```
result ← 0.0
f ← 1.0/base
i ← index
while i > 0 do
    result ← result + f · (i mod base)
    i ← i/base                                  ▷ Integer division
    f ← f/base
end while
return result
```

Using this method, we generate two sequences of Halton numbers with different bases such as 2 and 3, respectively, which are then used to compute the barycentric coordinates for uniform sampling within a triangle. These quasi-random numbers, $r1$ and $r2$, provide a more evenly distributed set of sample points within the triangle compared to purely random sampling, facilitating a more uniform sampling process. This can be verified in Figure 3.3, where halton sampling shows a result which is closer to even distribution.



(a) Uniform



(b) Halton

Figure 3.3: Sample Triangle

### 3.1.2 Generate Rays

Rays simulate the path of light as it interacts with objects in a scene. In our approach, while it is an usual case to cast ray from viewpoint, here we generate them from the sampled points on the triangle to the light source as shown in Figure 3.4, we need to pay attention that the viewpoint is abstracted as a point instead of an object with volume. The sampled points serve as the origin of each ray, while the view point (or light source) acts as the look-at point. The direction of each ray is computed based on the difference between the viewpoint and the origin. The direction vector is then normalized to ensure its magnitude is 1, which simplifies subsequent calculations.

The direction vector can be calculated using the formula

$$Direction = \frac{viewpoint - Origin}{\|viewpoint - Origin\|}$$

Where:

- *Origin* is the starting point of the ray, which in our case is the sampled point on the triangle.
- *Destination* is the end point of the ray, typically representing the viewpoint or light source.

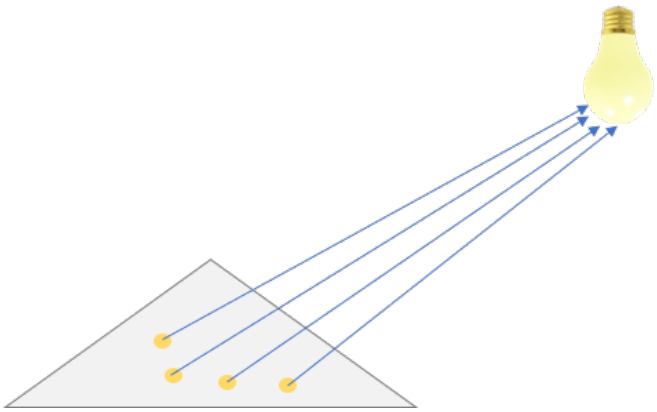


Figure 3.4: Generate Ray from Sampling to Viewpoint

When there are multiple viewpoints in the scene, we have to generate same amount of rays as viewpoints for each sampling.

### 3.1.3 Ray Triangle Intersection

Ray-triangle intersection is a fundamental operation in computer graphics, especially in the context of ray tracing. Determining whether a ray intersects a triangle and finding the intersection point are crucial for rendering scenes composed of triangular meshes.

One of the most efficient and widely used algorithms for this purpose is the *Möller–Trumbore* [MT97] intersection algorithm. It operates by representing the triangle in barycentric coordinates, which allows it to avoid the computation of explicit plane equations. It then uses these coordinates to find the intersection point of the ray and the triangle, if it exists. The algorithm performs a series of vector operations including dot products and cross products to compute the barycentric coordinates and the distance from the ray's origin to the intersection point. More detail is shown below:

**Algorithm 3** Möller–Trumbore Ray-Triangle Intersection Algorithm

**Require:** Ray with origin $O$ and direction $D$
**Require:** Triangle with vertices $V_1$, $V_2$, and $V_3$
**Ensure:** Intersection point $P$ or indication of no intersection
  Compute edge vectors:
    $e_1 \leftarrow V_2 - V_1$
    $e_2 \leftarrow V_3 - V_1$
  Compute vector $h$ as the cross product of $D$ and $e_2$:
    $h \leftarrow D \times e_2$
  Compute determinant $a$:
    $a \leftarrow e_1 \cdot h$
  **if** $a$ is close to zero **then**
    **return** No intersection ▷ Ray is nearly parallel to the triangle
  **end if**
  Compute factor $f$:
    $f \leftarrow \frac{1}{a}$
  Compute vector $s$:
    $s \leftarrow O - V_1$
  Compute barycentric coordinate $u$:
    $u \leftarrow f \cdot (s \cdot h)$
  **if** $u < 0$ or $u > 1$ **then**
    **return** No intersection
  **end if**
  Compute vector $q$ as the cross product of $s$ and $e_1$:
    $q \leftarrow s \times e_1$
  Compute barycentric coordinate $v$:
    $v \leftarrow f \cdot (D \cdot q)$
  **if** $v < 0$ or $u + v > 1$ **then**
    **return** No intersection
  **end if**
  Compute distance $t$ to the intersection point:
    $t \leftarrow f \cdot (e_2 \cdot q)$
  Compute intersection point $P$:
    $P \leftarrow (1 - u - v)V_1 + uV_2 + vV_3$
  **return** Intersection point $P$ and distance $t$

### 3.1.4 Occluded Area Ratio

From previous steps, we have computed samplings on each triangle and generated rays originating from these samplings directed towards the viewpoints. As opposed to directly compute occluded area, we detect visible samplings here. We use visible weight represents the ratio of visible samplings to the total number of samplings on the triangle. With this weight we compute visible area for each triangle, then we sum up them to get the total visible area, and the ratio can be easily calculated with total area. Mathematically, the visibility weight, $w$, can be defined as:

$$w = \frac{\textit{Number of visible samplings per triangle}}{\textit{Total number of samplings per triangle}}$$

The visible area $A_{\textit{visible}}$ of the triangle can then be computed as:

$$A_{\textit{visible}} = w \times A_{\textit{total}}$$

The visible area ratio, denoted as $R_{visible}$, is calculated as the ratio of the total visible area $A_{total\ visible}$ to the total area $A_{total}$ of the triangle. It can be mathematically represented as:

$$R_{visible} = \frac{A_{total\ visible}}{A_{total}}$$

Conversely, the occluded area ratio, denoted as $R_{occluded}$, can be calculated as the complement of the visible area ratio. It is given by:

$$R_{occluded} = 1 - R_{visible}$$

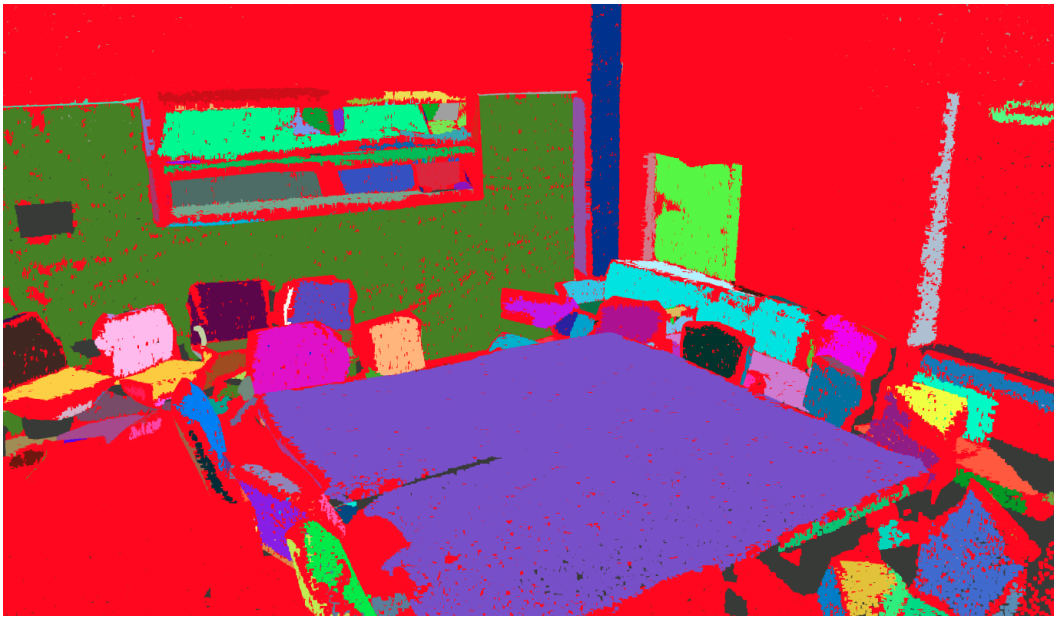
A sampling is deemed visible if at least one ray originating from it does not intersect any triangle between its origin and the viewpoint other than the one from which the sampling was generated. To ascertain this, we should examine all first hit intersection related to the ray cast from the sampling. If the distance from origin to viewpoint is shorter than the distance between origin and first hit intersection, which means no intersection can obscure the viewpoint, then this sampling is considered visible.

**Algorithm 4** Determining the Visibility of a Sampling

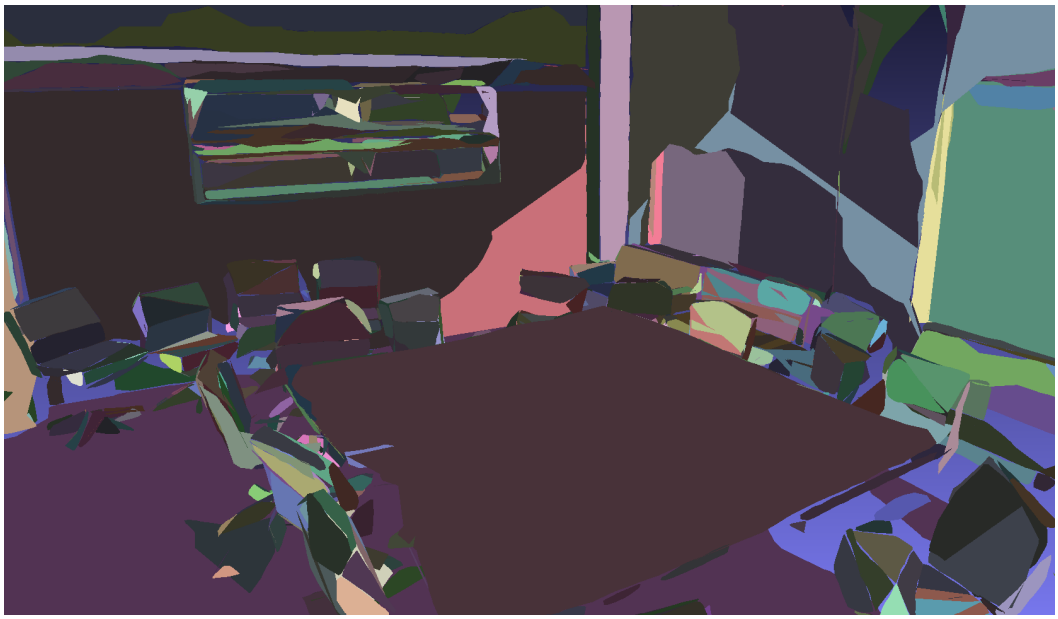
**Require:** Ray originating from sampling with origin $O$ and direction $D$
**Require:** Viewpoint $V$
**Require:** Set of triangles $T$ excluding the triangle from which the sampling was generated
**Ensure:** Visibility status of the sampling
  Compute the distance from the origin to the viewpoint: $d_{\text{viewpoint}} = \|V - O\|$
  Initialize variable $d_{\text{intersection}}$ to infinity
  Initialize variable isVisible to false
  **for** each triangle $t$ in $T$ **do**
    Compute the intersection of the ray with triangle $t$ using the Möller–Trumbore algorithm
    **if** there is an intersection at distance $d$ from $O$ **then**
      Update $d_{\text{intersection}}$ to the minimum of $d_{\text{intersection}}$ and $d$
    **end if**
  **end for**
  **if** $d_{\text{viewpoint}} < d_{\text{intersection}}$ **then**
    Set isVisible to true ▷ No intersection obscures the viewpoint
  **end if**
  **return** isVisible

## 3.2 Estimate Mesh from Point Cloud

In this part we estimate mesh for subsequent computation of occluded area ratio. As discussed in Section 2.2, region growing is an option here to segment points and to fit planar primitives based on segmented clusters of points, but the result is normally difficult to control, and the process of fitting plane from clusters might increase its instability. An alternative could be the software *GoCoPP* [YL22], which can help us on finding good configurations of planar primitives in unorganized point cloud. Regarding the paper behind this software, algorithm in *GoCoPP* outperforms region growing on different evaluation metrics. Below in Figure 3.5 shows the result of 2 methods where in case of region growing we only segmented the point cloud, while in *GoCoPP* they already triangulated planar primitive by applying alpha shape algorithm [AEF$^+$95]. With the triangulated mesh we can directly feed this data into our pipeline. Therefore, we choose to apply *GoCoPP* to generate mesh from point cloud.



(a) Region Growing



(b) Segmented via GoCoPP

Figure 3.5: Segmentation Methods

We use the same pipeline introduced in Section 3.1 to compute its occluded area ratio to represent occlusion level. The result depends on which strategy we use to place viewpoints. We can randomly place them in the scene or with fixed position such as center of the scene's bounding box, this will be explained more detailed in Chapter 5.

## 3.3 Compute Boundary Ray Ratio of Point Cloud

We define boundary ray as the ray which intersects with boundary point. Hence, it's also important to categorize each point as either *boundary* or *non-boundary*. Normally, we regard wall, floor, ceiling and window as boundary, in the context of our input data set, we will also add board, column and beam to this class. The exact classification

will be shown in Section 3.3.4. With all points classified we can then input these data into computational pipeline of *boundary ray ratio*. An overall workflow of this part is shown in Figure 3.6.

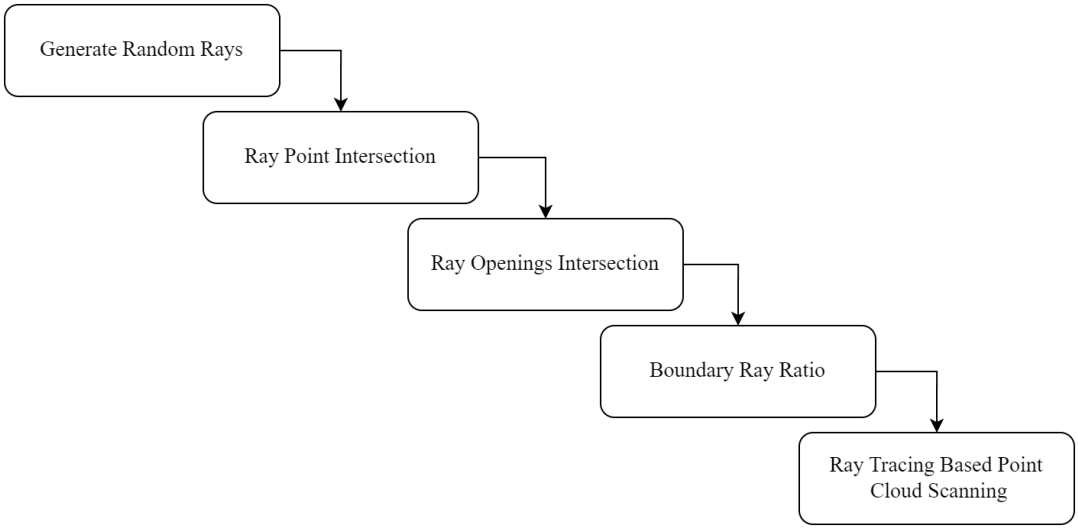
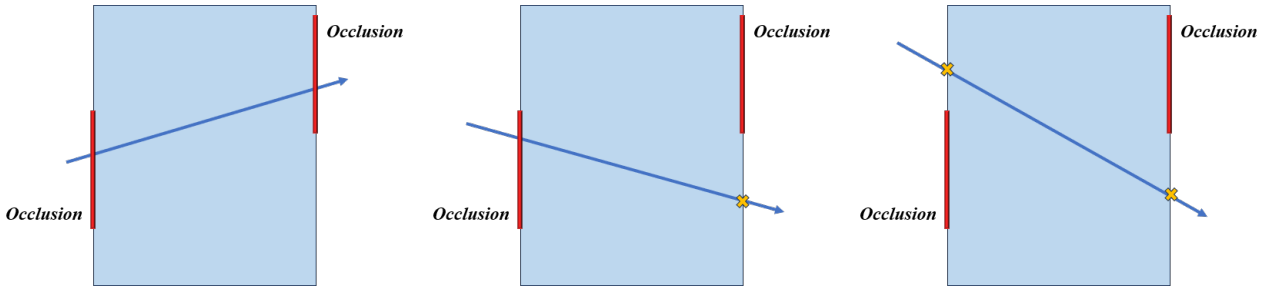



Figure 3.6: Flowchart of Computing Boundary Ray Ratio

### 3.3.1 Generate Random Rays

We want to use a ray from outside of the scene and directed towards the cloud, then we check how many boundaries the ray can intersects with, since this indicate how much occlusion a ray can contribute to the overall occlusion level. In case of a room, its boundary can be represented as a cuboid, which can at most has 2 faces intersect with one ray. Based on that, our rays can be classified as *non-boundary ray*, *1-boundary ray* and *2-boundary ray* as shown in Figure 3.7.




(a) Non-boundary Ray (b) 1-boundary Ray (c) 2-boundary Ray

Figure 3.7: Classification of Boundary Rays

In our work, it is more feasible for us to create rays within the room which is always presented as point cloud. We first randomly generate two points within the bounding box of the scene. One as origin of the ray and the other is the look-at direction. To simulate a ray originates from outside and pass through the room we also take its opposite direction into consideration. Namely, for one ray we have to compute its intersection with points in both directions.

A ray can intersect multiple boundary and non-boundary points at the same time, here we define that the ray has intersection with boundary on one direction when there is at least one boundary point intersect with it.

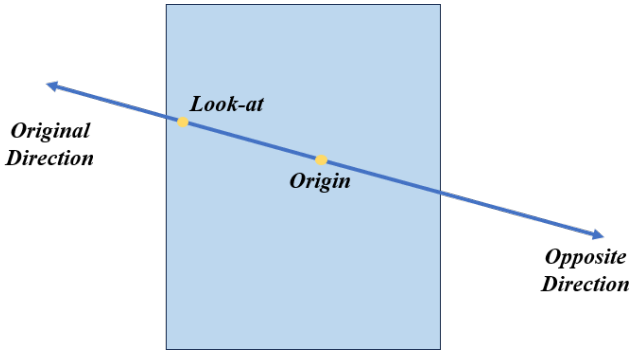



Figure 3.8: Ray with 2 Directions

### 3.3.2 Ray Point Intersection

Each point within the point cloud is represented as a small sphere with a defined radius. This simplifies the ray-point intersection check, as we can treat each point as a volumetric entity rather than a singular coordinate in space. The way ray intersects with a sphere is illustrated in Figure 3.9.

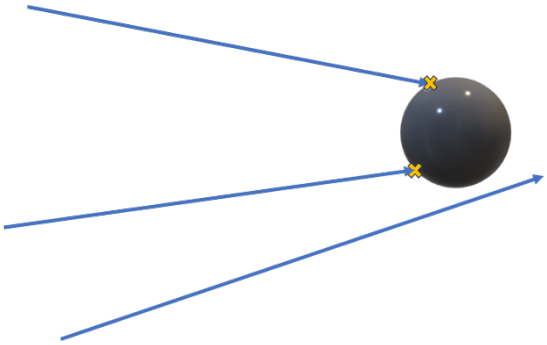


Figure 3.9: Ray Intersect Point

To determine if a ray intersects with a sphere, we follow the algorithm [Scr23b] stated below:

**Algorithm 5** Ray-Sphere Intersection Algorithm

**Require:** Ray origin, Ray direction, Sphere center, Sphere radius

Normalize the ray direction vector if not normalized
Compute vector $\mathbf{L} =$ Sphere center $-$ Ray origin
Compute originDistance$^2 = \mathbf{L} \cdot \mathbf{L}$
Compute $t_{ca} = \mathbf{L} \cdot$ Normalized ray direction
**if** originDistance$^2 <$ Sphere radius$^2$ **then**
    **return** Intersection exists (Ray origin is inside the sphere)
**end if**
**if** $t_{ca} < 0$ **then**
    **return** No intersection (Sphere is behind the ray)
**end if**
Compute $d^2 =$ originDistance$^2 - t_{ca}^2$
**if** $d^2 >$ Sphere radius$^2$ **then**
    **return** No intersection (Ray misses the sphere)
**else**
    **return** Intersection exists (Ray intersects the sphere)
**end if**

### 3.3.3 Ray Openings Intersection

Openings or gaps in the scene, such as windows or doors, should not contribute to occlusion. In the context of boundary rays, we regard openings as boundary. To account for these openings, we need an efficient method to determine if a ray intersects with any of the openings in the scene. Given the typically limited number of openings in a scene, it is feasible to manually select a few points that represent the boundary of each opening. Using these points, we can construct a polygonal representation of the opening. The ray-openings intersection check can then be performed by testing if the ray intersects with any of these polygons.

For a set of points $P = \{p_1, p_2, ..., p_n\}$ that define a polygon, we first estimate a plane $\mathbf{N}$ that best fits these points. Once the plane is determined, all points are projected onto it to ensure they lie within the same plane. The next step involves calculating the intersection of the ray with this plane. If an intersection exists, we need to determine if this intersection point lies inside the polygon. This workflow is shown in Figure 3.10.

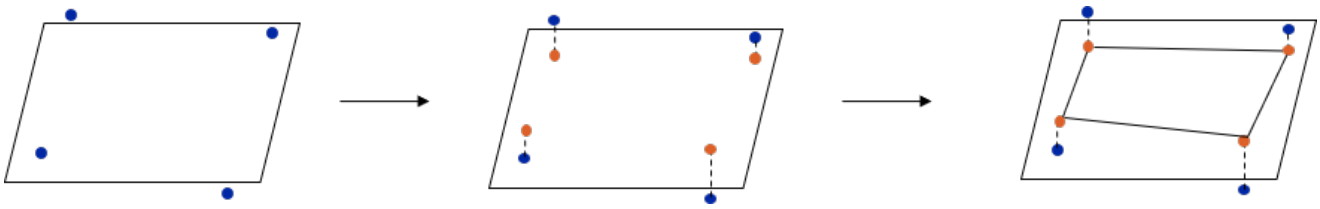


Figure 3.10: Estimate Polygon from Points

To check the position of the intersection point relative to the polygon, we construct vectors between the intersection point and each vertex of the polygon. For every pair of neighboring vectors, we compute their cross product. If all resulting cross products point in the same direction (i.e., they have the same sign), then the intersection point is inside the polygon. Otherwise, the intersection point lies outside the polygon.

Mathematically, given two vectors $\mathbf{v_1}$ and $\mathbf{v_2}$, their cross product is given by:

$$\mathbf{v_1} \times \mathbf{v_2} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ v_{1x} & v_{1y} & v_{1z} \\ v_{2x} & v_{2y} & v_{2z} \end{vmatrix}$$

Where $\mathbf{i}, \mathbf{j}$, and $\mathbf{k}$ are the unit vectors in the x, y, and z directions, respectively. By iterating over all the vertices of the polygon and computing these cross products, we can determine the position of the intersection point with respect to the polygon. In Figure 3.11 the ray intersects with polygon. To verify the algorithm with this illustration, we can apply right-hand rule to every 2 neighbouring vectors in clockwise order, then the cross product should always point downwards.

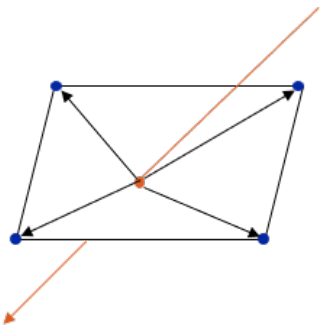


Figure 3.11: Ray Intersect Polygon

### 3.3.4 Boundary Ray Ratio

Before we compute ray-boundary intersection, it is required to classify all points properly. Then we can categorize rays into different classes.

**Classification of Points** To compute *boundary ray ratio* correctly, we have to define the class of each point. The exact classification is shown in Table 3.1.

| Boundary | Non-Boundary |
|---|---|
| Wall | Chair |
| Floor | Sofa |
| Ceiling | Table |
| Sofa | Bookcase |
| Table | Clutter |
| Door | |
| Window | |
| Beam | |
| Board | |
| Column | |

Table 3.1: Classification of Points

**Computation** After classifying all points in the scene, we compute ray-point intersection to determine which class the ray belongs to. Subsequently, we compute the boundary ray ratio using the following formula:

$$\textit{Occlusion Level} = \sqrt{\frac{\left(\frac{2}{3}\right) \cdot \textit{non-boundary ray count} + \left(\frac{1}{3}\right) \cdot \textit{1-boundary ray count}}{\textit{total rays}}}$$

We assign different weights to *non-boundary ray* and *1-boundary ray* based on the principle that a ray with less boundary intersection contribute more to occlusion level. To make this ratio comparable with *occluded area ratio*, we apply the square root here since the ray fills the volume of the space and the measurement grows at cubic rate while the area based ratio grows at quadratic rate.

### 3.3.5 Ray Tracing Based Point Cloud Scanning

To validate that *boundary ray ratio* is a reliable metric to represent occlusion level we would compare it with *occluded area ratio*. Sufficient data is needed in this step as we cannot ensure the robustness of validation with only 2 or 3 comparisons. Besides, the comparison should be conducted within the same scene since the structure of different scene can differ significantly. Therefore, it is necessary to create a set of occluded point clouds for each scene.

We would perform it in a manner similar to sampling a real scene using a laser scanner. The points stored in the data acquired from the scanner are those that are visible to it. If we apply this to original point cloud, we are actually creating a sub-sampled data set based on visibility to light sources.

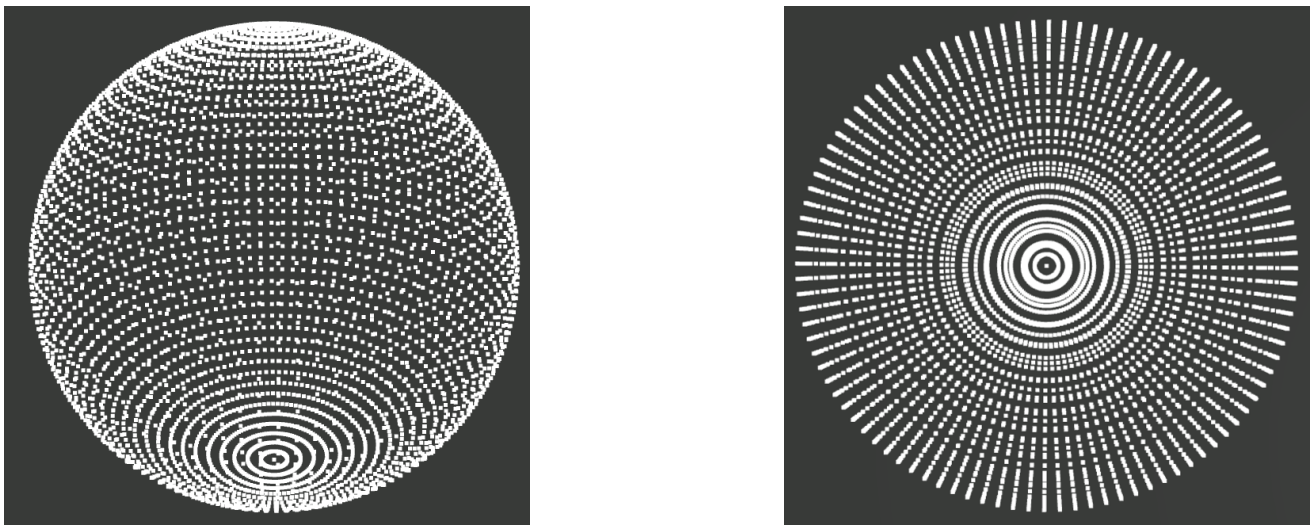
**Spherical Light Source** To simulate a light source that emits rays in all directions, we use a spherical model. Points are sampled on the surface of this sphere, and each point represents a direction for a ray. The number of rays (or sampled points) is predetermined and can be adjusted based on the desired resolution or accuracy of the scan. The exact sampling method is inspired by the work which is dealing with parameterizing surface of spheres [FH05]. The specific algorithm is as follows:

**Algorithm 6** Surface Parameterization Based Sampling

**Require:** Number of horizontal samplings $sampling\_hor$ and number of vertical sampling $sampling\_ver$
**Ensure:** A vector of directions constituting the scanning pattern.

Initialize a vector $pattern$ with size $sampling\_hor \times sampling\_ver$
$hor \leftarrow sampling\_hor$
$ver \leftarrow sampling\_ver$
**for** $i = 0$ to $sampling\_hor - 1$ **do**
    $cos\_i \leftarrow \cos\left(\frac{2\pi i}{hor}\right)$
    $sin\_i \leftarrow \sin\left(\frac{2\pi i}{hor}\right)$
    **for** $j = 0$ to $sampling\_ver - 1$ **do**
        $sin\_j \leftarrow \sin\left(\frac{2\pi j}{ver}\right)$
        $cos\_j \leftarrow \cos\left(\frac{2\pi j}{ver}\right)$
        Create a vector $dir$ with elements $cos\_i \cdot sin\_j$, $sin\_i \cdot sin\_j$, and $cos\_j$
        $pattern[j + i \cdot sampling\_ver] \leftarrow dir$
    **end for**
**end for**
**return** pattern

An example of the result of this method is shown in Figure 3.12 with multiple views.



(a) Frontal View

(b) View from Above

Figure 3.12: Sample Sphere

## 3.4 Evaluate Performance of Segmentation

Evaluating the performance of semantic segmentation is crucial to ensure its reliability and effectiveness. Metrics serve as standardized measures to assess the quality of segmentation results, comparing the predicted outputs against the ground truth. In this part, we discuss the semantic classes as well as metrics used to evaluate the performance of semantic segmentation models.

### 3.4.1 Semantic Classes

Semantic segmentation requires a clear definition of the classes that are to be identified within the data set. Due to discrepancies between the ground truth point cloud data set, denoted as S3dis [ASZ+16], and the data set used

for the pre-trained model in *Minkowski Engine*, we have to make sure that their classes are comparable. Classes in S3dis is shown in Table 3.2. When we do the comparison between points' semantic labels, we categorize class which is not included in the table to *Clutter*.

| **Semantic Class** | **ID** |
|---|---|
| Wall | 0 |
| Floor | 1 |
| Ceiling | 2 |
| Chair | 3 |
| Sofa | 4 |
| Table | 5 |
| Door | 6 |
| Window | 7 |
| Bookcase | 8 |
| Beam | 9 |
| Board | 10 |
| Clutter | 11 |
| Column | 12 |

Table 3.2: Mapping of semantic classes to their respective class IDs.

Given a segmented point cloud, we can evaluate the segmentation by calculating various metrics such as precision, and recall for each class. In semantic segmentation of point clouds, each point in the cloud is classified into one of several possible categories, which are True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN) regarding the correctness of segmentation. The Algorithm 7 demonstrates how to classify points into them and counting the values. With these values we can then compute metrics explained in subsequent section.

**Algorithm 7** Computation of TP, TN, FP, and FN for Point

**Require:** Segmented point cloud $P$ with $N$ points, Ground truth labels for each point
**Ensure:** Computation of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) for each class

```
Initialize counters for True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) for each class to zero
for each point p_i in point cloud P do
    Get the ground truth label and predicted label for p_i
    for each class c_j do
        if the ground truth label of p_i is c_j then
            if the predicted label of p_i is c_j then
                TP for class c_j += 1
            else
                FN for class c_j += 1
            end if
        else
            if the predicted label of p_i is c_j then
                FP for class c_j += 1
            else
                TN for class c_j += 1
            end if
        end if
    end for
end for
return Counters TP, TN, FP, and FN for each class
```

### 3.4.2 Evaluation Metrics

To evaluate the performance of our model in the context of point cloud segmentation, we employ several widely-recognized metrics, which includes *Precision*, *Recall* and *F1 Score*.

These metrics provide different perspectives on the model's capabilities and are computed using the values of True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN) which are obtained for each class through the algorithm described earlier. Below, we detail these metrics along with their respective mathematical formulations:

**Precision and Recall** are two complementary metrics that provide insights into the model's performance regarding false positives and false negatives.

**Precision** quantifies the proportion of positive identifications that were actually correct, as given by:

$$\textit{Precision} = \frac{\textit{True Positives}}{\textit{True Positives} + \textit{False Positives}}$$

or equivalently,

$$\textit{Precision} = \frac{TP}{TP + FP}$$

A higher precision indicates a larger percentage of the model's positive predictions were correct.

**Recall**, on the other hand, measures the proportion of actual positives that were identified correctly, as given by:

$$\textit{Recall} = \frac{\textit{True Positives}}{\textit{True Positives} + \textit{False Negatives}}$$

or equivalently,

$$\textit{Recall} = \frac{TP}{TP + FN}$$

A higher recall indicates that fewer actual positives were missed by the model.

**F1 Score** is the harmonic mean of *precision* and *recall*, providing a balance between the two and ensuring that both false positives and false negatives are considered. It is calculated as:

$$F1 = 2 \times \frac{\textit{Precision} \times \textit{Recall}}{\textit{Precision} + \textit{Recall}}$$

or equivalently,

$$F1 = 2 \times \frac{TP}{2 \times TP + FP + FN}$$

An F1 Score closer to 1 indicates better performance, while a score closer to 0 indicates poor performance.

# 4 Implementation

In this chapter, we first delve deeper into the core part of implementation of our computational pipeline as presented in Chapter 3. Then we go through the overall structure of the software which is ready for user to interact with.

## 4.1 PCL and Eigen Serves for Computation

The Point Cloud Library (PCL) [RC11] stands as the backbone of our implementation. We rely on PCL to accomplish a set of tasks, ranging from reading point cloud to evaluating its occlusion levels. Parallel to PCL, the Eigen library [GJ+10] emerges as an invaluable asset. It is our preferred choice for handling mathematical operations that are indispensable to our pipeline.

### 4.1.1 Handling Point Cloud Data

We read point cloud data and load them into different types as follows:

- **pcl::PointXYZ** - Point cloud with only coordinate information.
- **pcl::PointXYZI** - Extra intensity field, in this work it is used mostly to store category information of a point.
- **pcl::PointXYZRGB** - Point cloud with coordinate and color information. If without RGB value attached, we will see point cloud rendered with white color.

Sometimes we need to store points to a new point cloud, which can be done in a typical way shown in code snippet below:

**Listing 4.1: Save Point Cloud Data**

```
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud(new pcl::PointCloud<pcl::PointXYZ>);
for(...) {
    pcl::PointXYZ point;
    cloud->points.push_back(point);
}
cloud->width = cloud->points.size();
cloud->height = 1;
cloud->is_dense = true;
pcl::io::savePCDFileASCII("../files/cloud.pcd", *cloud);
```

### 4.1.2 Estimate Polygon

Computation of ray-opening intersection plays an crucial role in calculating boundary ray ratio. Each opening is represented as a polygon, and the data we can extract from the original point cloud is a set of small clouds where each point inside it can be regarded as a vertex of polygon in 3D space. And these vertices are most likely not in the same plane, therefore, we would estimate a polygon which is able to be used in subsequent computation.

**Plane Estimation** The initial step is fitting a plane to each cluster's point data. For this, we rely on the RANSAC algorithm [FB81] embedded within PCL. In the code snippet shown below, PCL's *SACSegmentation* [Rus23a] is configured to detect planar models. If successful, it retrieves the plane's coefficients.

**Formation of Polygon** With the plane's coefficients, our next move is to project every point in the cluster onto this plane. This ensures that all points lie in a co-planar fashion. Once our points sharing a planar space, we compute the convex hull of each cluster. This results in a 2D polygon within a 3D space, effectively enveloping all the cluster points. Here, PCL's *ProjectInliers* projects the points onto the detected plane, thereby creating a co-planar point cloud. The usage of PCL's *ConvexHull* [CLRS01] class create a cloud representing the polygon.

### 4.1.3 Ray Tracing Based Intersection Computation

Ray tracing based methods are pivotal to our computational pipeline. We use ray to check if it intersects with any basic element in the scene, such as point, triangle, polygon, bounding box, etc.

**Database-like Structure** To compute and record all intersections between all rays and all elements in the scene, we build different structure for each type of element. Their interactions are also stored, which makes the system appears to be a relational database. We can query this database to get all the information we need.

**Listing 4.2: Data Structures**

```
struct Ray3D {      // ray structure used for point cloud ray tracing
    size_t index;
    pcl::PointXYZ origin;
    pcl::PointXYZ direction;
    std::vector<size_t> first_dir_bound_intersection_idx;
    std::vector<size_t> second_dir_bound_intersection_idx;
    ...
};
struct Intersection {
    size_t index;
    size_t triangle_index;
    size_t ray_index;
    Eigen::Vector3d point;
    bool is_first_hit;
    ...
};
struct Ray {
    size_t index;
    size_t first_hit_intersection_idx;
    Eigen::Vector3d origin;
    Eigen::Vector3d direction;
    std::vector<size_t> intersection_idx;
    std::vector<size_t> triangle_idx;
    ...
};
struct Sample {
    size_t index;
    size_t triangle_index;
    Eigen::Vector3d point;
    bool is_visible = false;
    std::vector<size_t> ray_idx;
};
struct Triangle {
    size_t index;
    Eigen::Vector3d v1;
    Eigen::Vector3d v2;
    Eigen::Vector3d v3;
    std::vector<size_t> sample_idx;
    std::vector<size_t> intersection_idx;
    ...
};
```

**Octree Partitioning** In the context of our pipeline, it can be quite overwhelming to iterate over all points for rat-tracing computation. A naive approach, iterating through each data point and ray, would be computationally taxing. Recognizing this challenge, it becomes imperative to partition our data into more manageable chunks. Commonly, data structures such as BVH [Eri05] and octree [Mea80] are used for this purpose. Since there existing an octree implementation in PCL, we employ it for partitioning our data.

The outcome of octree partitioning is a structured tree. Within this tree, there are three types of nodes: leaf nodes, branch nodes, and the root node. Leaf nodes serve as storage units for data, branch nodes encapsulate the bounding box for their respective child nodes, and the root node includes the bounding box of the entire tree. Navigating this structure, if a ray originates inside the root node, we examine its intersection with the root's child nodes. The procedure recursively continues until we reach a leaf node, at which point we verify the ray's intersection with the points stored in the leaf. If there is a data intersection, the intersected point is returned.

**Octree Data Structure** The underlying node structure for our octree is captured succinctly in the following code representation:

**Listing 4.3: Octree Node Structure**

```
struct OctreeNode {
    size_t index;
    size_t parent_index = -1;
    size_t prev = -1;
    size_t next = -1;
    int depth;
    std::vector<size_t> children;
    std::vector<size_t> triangle_idx;
    int diagonal_distance;
    Eigen::Vector3d min_pt;
    Eigen::Vector3d max_pt;
    Eigen::Vector3d min_pt_triangle;
    Eigen::Vector3d max_pt_triangle;
    bool is_leaf = false;
    bool is_branch = false;
    bool is_root = false;
};
```

**Partition Mesh** We can partition mesh with the help of PCL' octree class. However, this requires a preliminary step where the extraction of a point which represent the triangle. To achieve this, we use the triangle's center of gravity as its representative point. By making these points a cloud, an octree can be constructed. PCL offers several traversal methods for octree, here the depth-first iterator is applied by default.

**Build Depth-Size Map** Recognizing that nodes at the same depth exhibit identical bounding box sizes, a traversal of the tree allows us to construct a size-depth map. Here, the bounding box's size (denoted by the distance between *maximal point* and *minimal point*) acts as the key, while the depth is the associated value.

**Build Depth-Node Map** For each node in the tree, we can now find its corresponding depth through *Depth-Size Map* built in former step. Then we build the *Depth-Node Map* where depth is the key and value is composed of indexes of nodes belong to this level.

**Build Connections Between Nodes** Establishing connections between parent and child nodes is an essential step. This phase involves a traversal of the depth-node map, starting from the root and working through to the leaf node level. There are some points are worth highlighting:

- The root node stands alone without a parent, but nodes at level 1 always share the root as their parent.

- For branch nodes, the traversal of their child nodes occurs after the traversal of the left sibling's children but prior to the right sibling's children. This traversal pattern help us in effectively mapping the parent-child relationships.

Following is an example with a tree depth of three. Though an octree typically has eight children for every node, for the sake of clarity, we've reduced its size in this explanation.

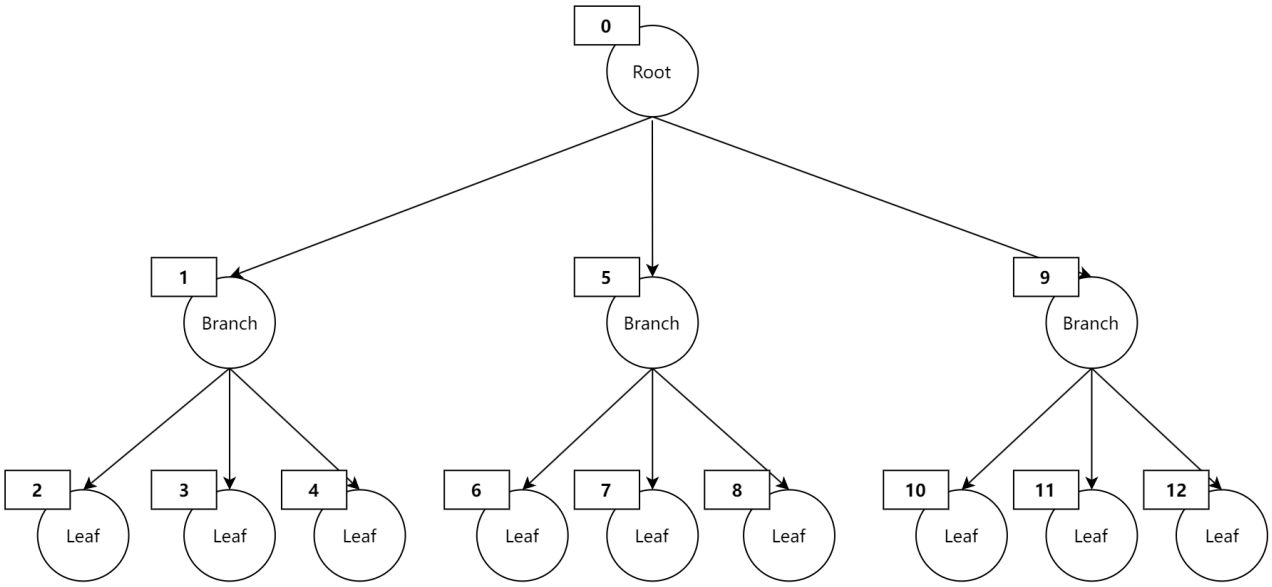



Figure 4.1: Octree Structure

The node index, placed at the upper left corner of each box, equates to its traversal order in a depth-first pattern. Using node 1 as a reference, all its children indices are lesser than 5. Such observations facilitate the accurate connection of children to their parent nodes.

**Compute Bounding Box of Each Node** For each leaf node, its bounding box can be directly returned from methods implemented by PCL's octree class. But inside the box, there only stores center point of triangle. Hence, we have to compute the bounding box based on the real geometry of triangle. Each point is stored in form of *pcl::PointXYZI*, where *I* represent its intensity field. In our case we use this field to store the index of triangle. If we iterate over all points, we are actually iterating over all triangles.

The *minimal point* and *maximal point* can be obtained from the minimal and maximal vertex of all triangles. Once we have built bounding box for each leaf node, we can traverse upwards to compute bounding box for each branch node and root node.

**Partition Point Cloud** For partitioning the point cloud, most steps are the same as mentioned in 4.1.3. The only difference is that we don't need to compute bounding box based on triangle for each node, methods offered by PCL' octree class already meet our needs.

**Ray Intersecting with Bounding Box** To determine whether a ray intersects with a bounding box, there are two primary conditions to be considered:

- If the ray's origin lies inside the bounding box, then it is evident that the ray intersects.

- If the ray's origin is outside the bounding box, a specialized algorithm is employed to check for the intersection.

The underlying principle is to inspect how the ray interacts with the bounding box by calculating potential intersection points for each dimension (i.e., x, y, and z). This method of computation involves the slab technique [Scr23a], which calculates the intersection of the ray with the planes that define the bounding box.

## 4.2 Auxiliary Components

Several auxiliary libraries play a pivotal role in our system, furnishing it with additional functionalities and capabilities. We briefly discuss these libraries and their respective roles in our pipeline.

**JsonCpp - Parameter Parser** JsonCpp is a C++ library that allows for the parsing of JSON files. In our system, we use it to parse the configuration file, which contains all the parameters required for our pipeline. The following snippet illustrates the configuration file's structure:

**Listing 4.4: Json Configuration File**

```
"occlusion": {
    "mesh": {
        "path": "../files/simp_conf.obj",
        "ply_path": "../files/copy_alpha_shape.ply",
        "pattern": 4,
        "octree_resolution": 1.0,
        "enable_acceleration": true,
        "samples_per_unit_area": 100,
        "use_ply": true
    },
```

**Websocketpp - Communication Channel** Websocketpp is a C++ library that facilitates the establishment of a communication channel between the backend and frontend. It is a critical component of our system, enabling the seamless exchange of data between the two components. The following snippet illustrates the communication channel's initialization:

**Listing 4.5: Websocket Server**

```
typedef websocketpp::server<websocketpp::config::asio> server;
void on_message(server& s, websocketpp::connection_hdl hdl, server::message_ptr msg,
    DataHolder& data_holder) {
        ...
    s.send(hdl, msg->get_payload(), msg->get_opcode());
    std::string payload = msg->get_payload();
    if (payload.substr(0, 3) == "-i=") {
      data_holder.setFileName(payload.substr(3, payload.length()));
    }
    ...
int main{
  print_server.set_message_handler([&print_server, &data_holder](websocketpp::
      connection_hdl hdl, server::message_ptr msg) {
            on_message(print_server, hdl, msg, data_holder);
        });
        ...
  print_server.listen(8080);
  print_server.run();
}
```

## 4.3 Web-Based User Interface

Our software is encapsulated within the framework of a web application. The PCL-based backend is responsible for computation, whereas the frontend enhances user interaction. Detailed structures of the backend were previously introduced in Section 4.1. In this section, we will briefly introduce usage of the user interface.

The user interface's visual representation is displayed in Figure 4.2.


Figure 4.2: Web-Based User Interface

### 4.3.1 Three.js Serves for Visualization

Three.js [Mr.21] is a cross-browser JavaScript library built upon WebGL, complemented with an API designed to exhibit animated 3D computer graphics directly in a web browser. This dynamic library provides the interactive visualization pivotal for our application.

**Web Technology Stack** Given our choice of three.js for visualization, a group of complementary web technologies is integrated to build our frontend. The technologies used in our web tech stack are as follows:

- **TypeScript** - An enhanced version of JavaScript, TypeScript introduces a stricter syntax and the convenience of optional static typing, fortifying code robustness and clarity.
- **TailwindCSS** - A utility-first CSS framework, it grants developers the tools to rapidly carve custom user interfaces without the redundancy of routine styling.
- **Vite** - Tailored for modern web projects, Vite is a forward-thinking build tool that streamlines the development process.
- **Websocket** - Elevating communication protocols to the next level, Websocket delivers full-duplex communication channels over a single TCP connection, encouraging instantaneous data exchange.

### 4.3.2 User Interface Usage

In this part we will briefly introduce our user interface and explain how to use it.

**Stats Panel** Stats panel is located at the top left corner of the user interface. It shows three different metrics of the scene in different color.

- **Frame Rate** - Blue, frame rate of the visualization.
- **Network Latency** - Green, network latency.
- **Cache Size** - Red, cache size of the point cloud.

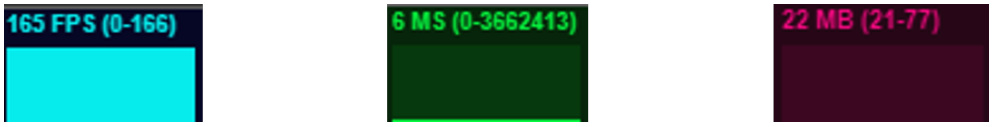



Figure 4.3: Stats Panel

**GUI** GUI is located at the top right corner of the user interface. It shows different parameters of the scene.

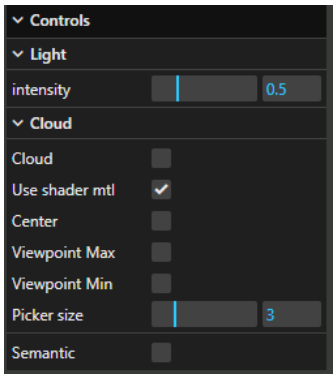



Figure 4.4: GUI

- **Light Intensity** - Control the intensity of the light source.
- **Show Cloud** - Show the point cloud.
- **Use Shader Material** - Use shader material to visualize the sphere generated by point picker in a different way. Here it is blinking between white and black.
- **Center** - Show center viewpoint.
- **Viewpoint Max** - Show max-mid viewpoint.
- **Viewpoint Min** - Show min-mid viewpoint.
- **Picker Size** - Size of the point picker.
- **Semantic** - When this enabled, the semantic label of a point will be displayed after right clicking with mouse.

**Buttons** Below are the buttons on the user interface.

- **Original Cloud** - Upload and visualize original point cloud.
- **Semantic** - Specify file path of point cloud generated from semantic segmentation.

- **Ground Truth** - Specify file path of ground truth point cloud.
- **Mesh** - Upload and visualize mesh.
- **Compute Occlusion** - Compute occlusion of the point cloud.
- **Extract Polygons** - Extract the polygon which is specified by interactively selecting points in the rendering of point cloud.
- **Evaluate** - Evaluate the performance of semantic segmentation.

### 4.3.3 Command Line Usage

This project is designated to be used in a web application. However, we can also manipulate the computational pipeline in command line environment. In addition to functionalities to compute occlusion level and evaluation metrics directly through user interface, there are commands performs intermediate operations within our workflow, such as point cloud scanning, point cloud reconstruction and computation of occluded area ratio etc.

**Arguments and Corresponding Functionality** Below shows arguments used for different usages. To get proper results, it is needed to modify parameters in the configuration file *config.json*.

- **-b** - Start the backend server.
- **-moc** - Compute occluded area ratio.
- **-bounoc** - Compute boundary ray ratio.
- **-fscan** - Scan cloud with fixed viewpoint.
- **-recon** - Reconstruct point cloud with ground truth labels from .txt file.
- **-eval** - Evaluate performance of segmentation.
- **-t2ply** - Transfer the format of .pcd files to .ply.

# 5 Experimental Results

Experiment is one of the most important steps to evaluate the approaches presented in Chapter 3. In this chapter we will first validate the reliability of our proposed metrics, then we apply them to subsequent workflow to investigate the impact of occlusion on semantic segmentation of point cloud.

## 5.1 Validation

The validation phase starts by computing occlusion level of ground truth mesh. Different strategies of the placement of viewpoints will be applied to this scene.

### 5.1.1 Occlusion Level of Ground Truth Mesh

**Setup** We prepare a ground truth mesh which present a conference room [McG17] as our input. Since there are a large quantity of triangles in this mesh, this could potentially result in a significant computational workload, thus we use the software Meshlab [CCC+08] to help us reducing its size. We have 3 viewpoints in this scene, their positions are center of the scene, midpoint between center and maximal point and midpoint between center and minimal point. The mesh scene is shown in Figure 5.1 where all viewpoints are marked with red box.

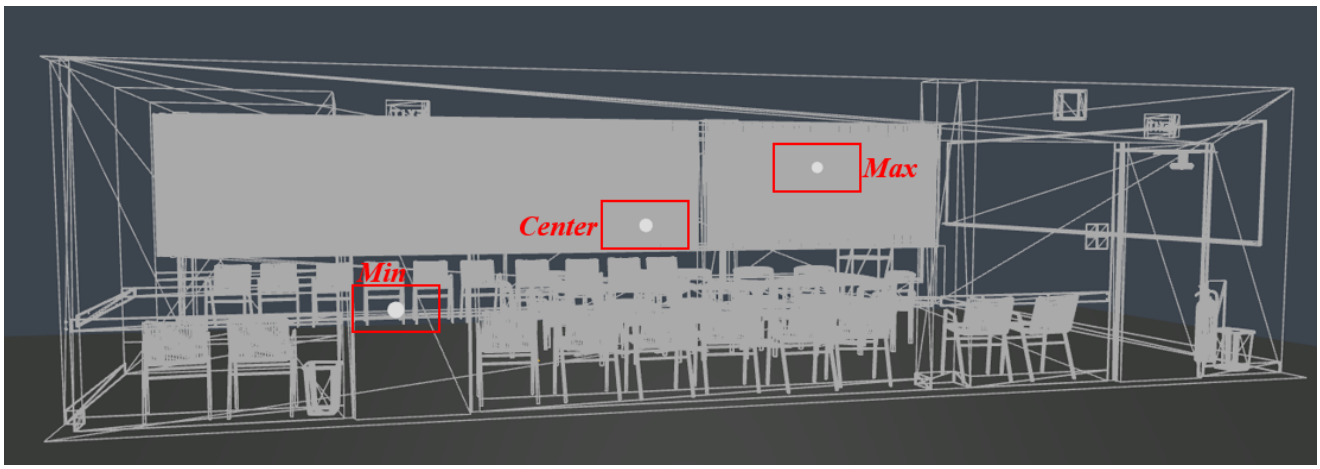



Figure 5.1: Mesh with Viewpoints

We will conduct this experiment on different pattern of viewpoints as shown in Figure 5.2. There are around 10000 triangles in this mesh, to simplify the computation, we set number of samples per unit area to 10, which result in around 20000 samples in total. In case there is one viewpoint in the scene, we have the same amount of rays as samples since we create rays from sampling to viewpoints. Therefore, number of rays will be doubled if we add one more viewpoint, tripled with 3 viewpoints and so on.

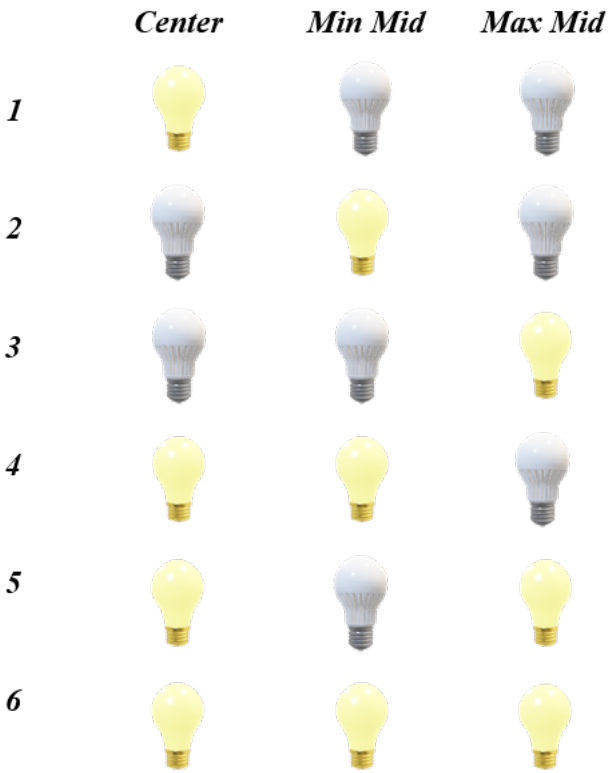



Figure 5.2: Pattern of Viewpoints

**Results** Our results are shown in Table 5.1 and Figure 5.3. We can see that the occlusion level decreases as the number of viewpoints increases. This is because with more viewpoints we can cover more area of the scene, which results in less occlusion. It is important to note that there is only 1 viewpoint in pattern 1, 2 and 3, 2 viewpoints in pattern 4 and 5, and 3 viewpoints in pattern 6.

| Pattern | Occluded Area Ratio |
|---|---|
| 1 | 0.570 |
| 2 | 0.780 |
| 3 | 0.571 |
| 4 | 0.484 |
| 5 | 0.453 |
| 6 | 0.391 |
| Under Table | 0.914 |
| Pure Cube | 0.000 |

Table 5.1: Result of Ground Truth Mesh

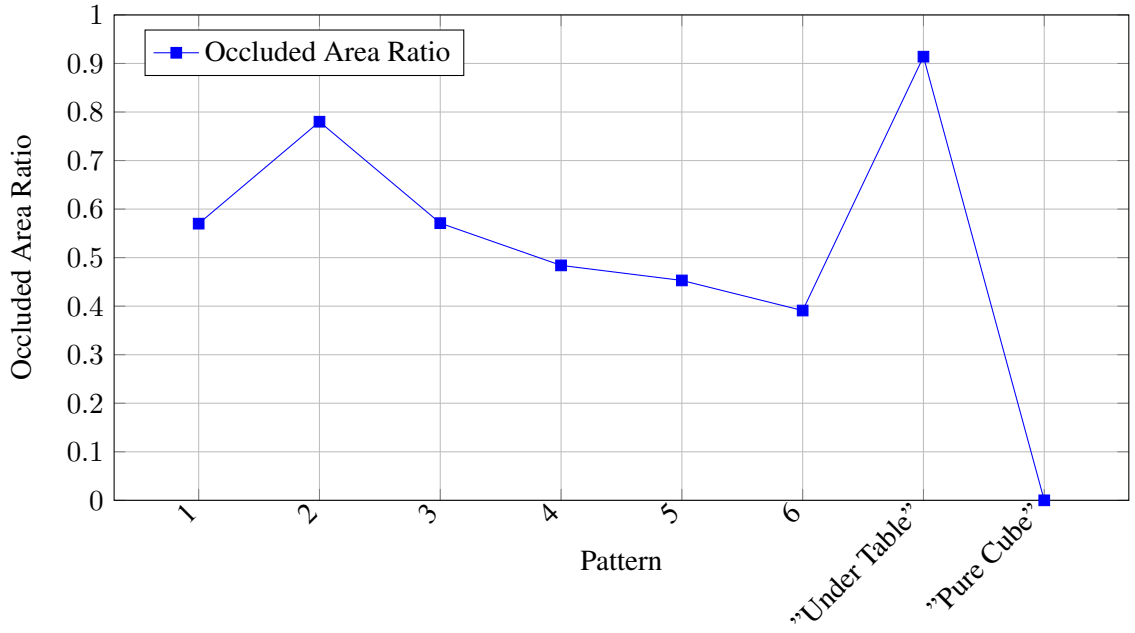



Figure 5.3: Result of Ground Truth Mesh

Except for patterns we mentioned above, we also added 2 corner cases where in the first case we put one viewpoint right below the center as shown in Figure 5.4. The only difference is that the Z-value of this viewpoint is equal to the minimal vertex of the scene. The center viewpoint is also shown since we want to display the relative position of the viewpoint used in corner case to it, but we don't consider it for computation. Based on its special position, most of the rays cast from samplings should be blocked by the table. The value *0.914* can verify that in this case most samplings are occluded to this viewpoint,

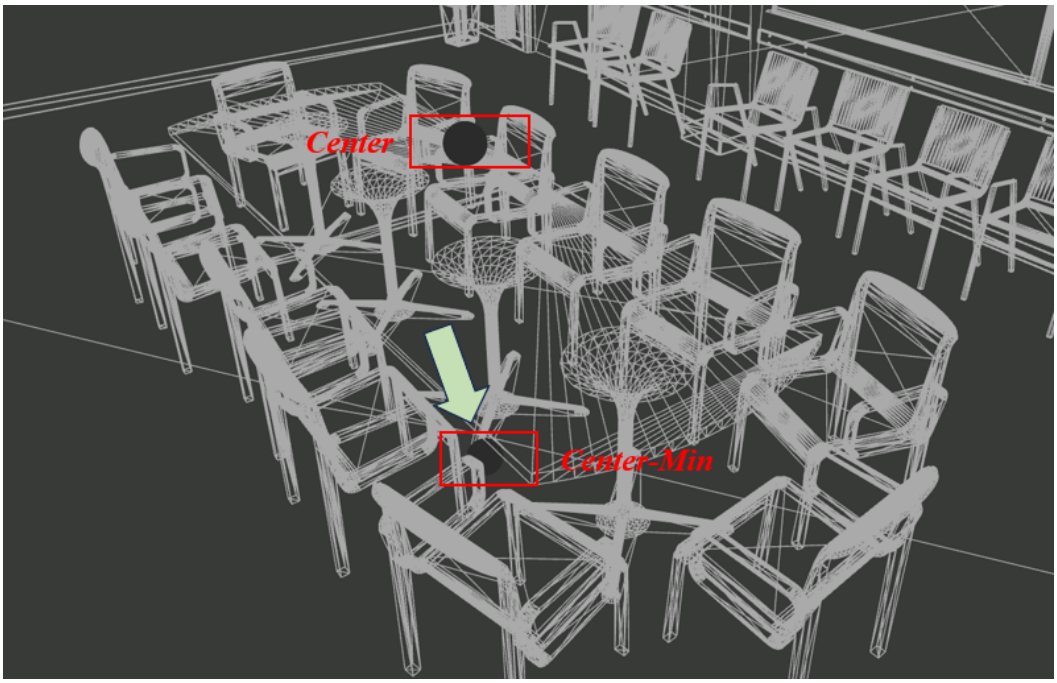



Figure 5.4: Corner Case - Under Table

In the other corner case we change the scene to a pure cube where there is no clutter inside it. Thus, all samplings should be visible to a viewpoint inside the cube regardless of its position. The resulting visible sampling cloud is shown in Figure 5.5. Our pipeline output the value *0*, which can also validate the correctness of computation.

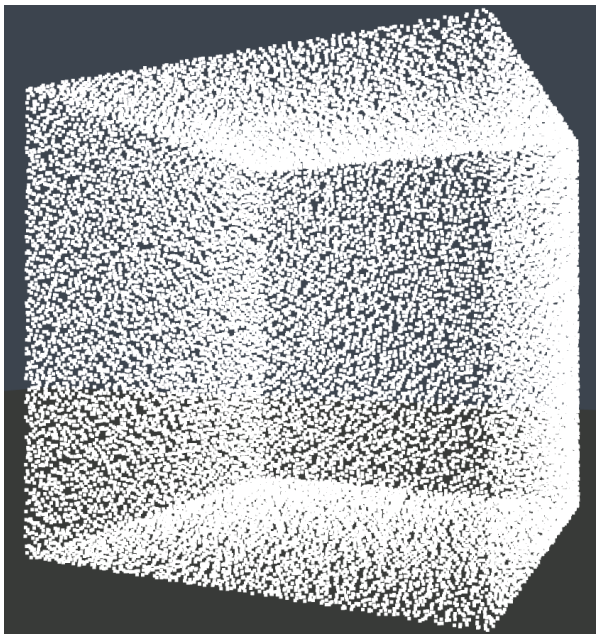


Figure 5.5: Corner Case - Pure Cube

Based on results presented above, we can take the *Occluded Area Ratio* as a reliable metric to represent the occlusion level of a mesh.

### 5.1.2 Estimated Mesh and Scanned Cloud

In this section we are going to compute occlusion level of estimated mesh and scanned point cloud respectively.

**Setup** We prepare 2 original point clouds with name *Conference Room 1* and *Conference Room 2* respectively. For each of them, we estimate a mesh via *GoCoPP*. Then for each mesh, we conduct the same workflow described in Section 3.1 to get *Occluded Area Ratio*. The next step is to apply the computation pipeline of *Boundary Ray Ratio* on scanned clouds, which will be generated by adopting the same patterns illustrated in Figure 5.2 to place spherical light sources as described in Section 3.3.5.

Above workflow will be conducted for each scene 6 times as we would keep the same viewpoint strategy presented in Figure 5.2, and each time we cast 10000 rays in the scene randomly for calculating *Boundary Ray Ratio*.

**Conference Room 1** We present the visualization of the scene *Conference Room 1* in each step. Figure 5.6 shows the result of mesh estimation, where point clouds was transferred into a set of alpha-shape triangulated planar primitives.

To detect boundary ray we have to know each point's category in terms of boundary from scanned point cloud, where we use white color to represent boundary point and blue for non-boundary point as shown in Figure 5.7 (a). This boundary cloud is generated together with the ground truth scanned cloud displayed in Figure 5.7 (b). The resulting visualization will be displayed for scanned clouds under each pattern.

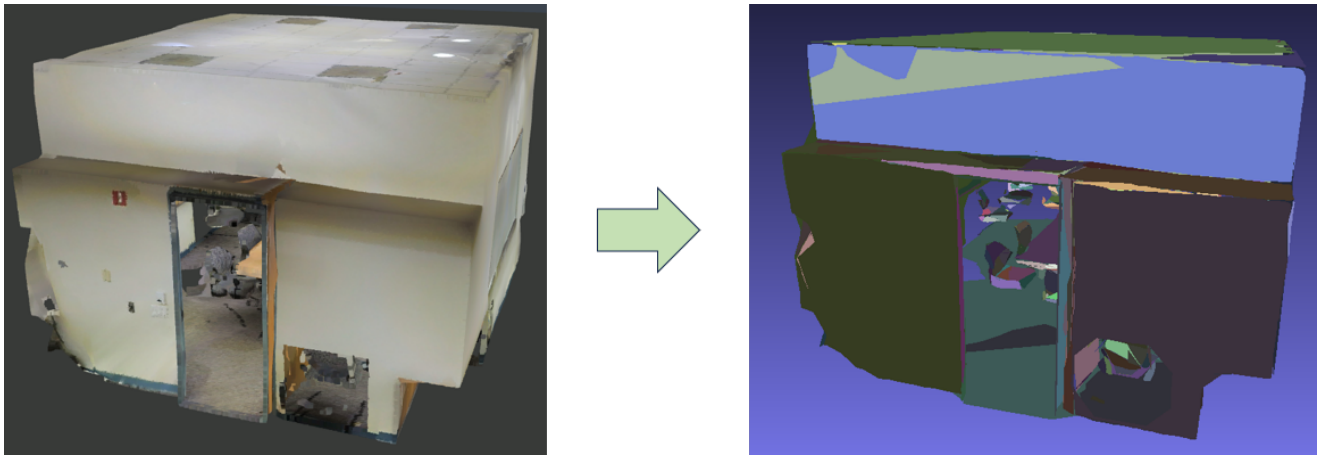
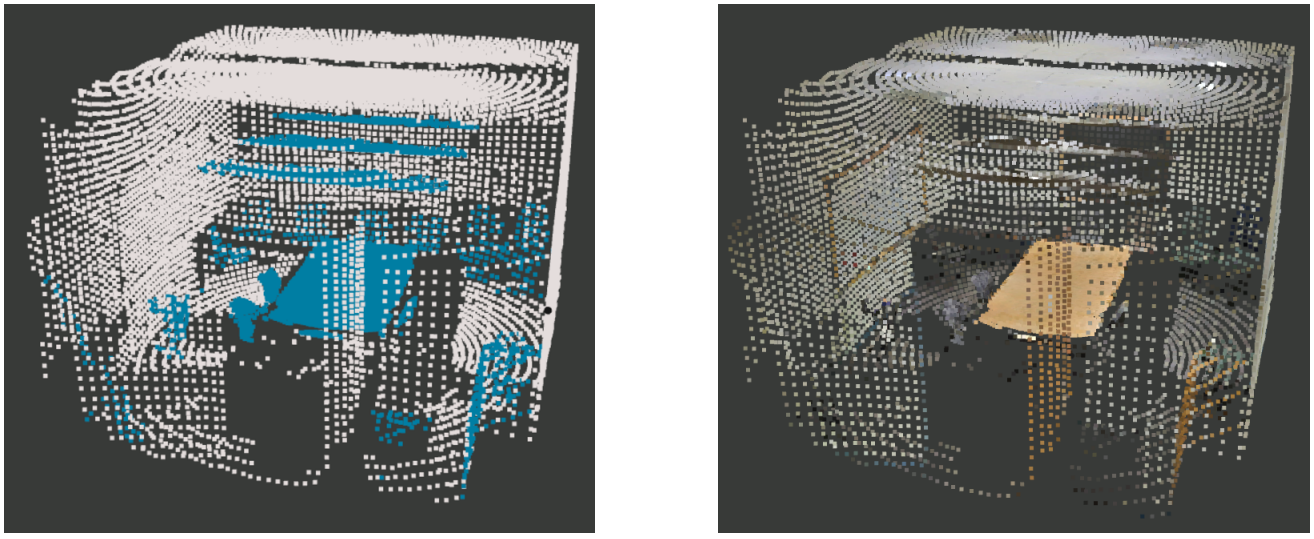


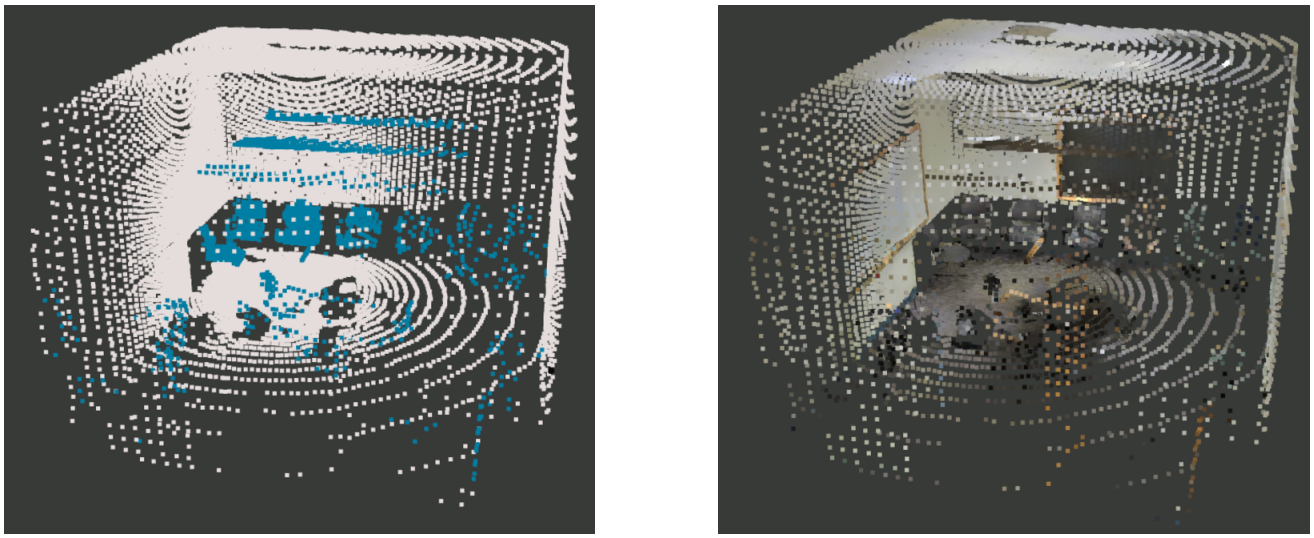
Figure 5.6: Estimate Mesh from Conference Room 1

In the end of this part, result of computation for *Occluded Area Ratio* and *Boundary Ray Ratio* is shown in Table 5.2 together with line graph illustrated in Figure 5.13.



(a) Boundary (b) Scanning

Figure 5.7: Boundary and Scanning of Conference Room 1 under Pattern 1 with 38372 Points

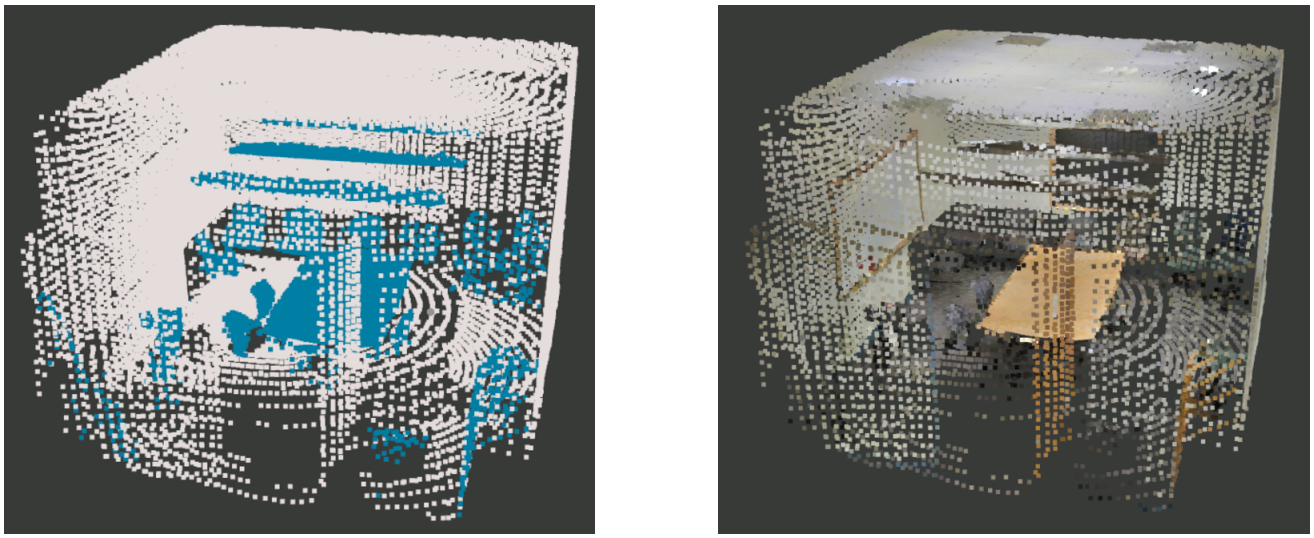


(a) Boundary (b) Scanning

Figure 5.8: Boundary and Scanning of Conference Room 1 under Pattern 2 with 36152 Points

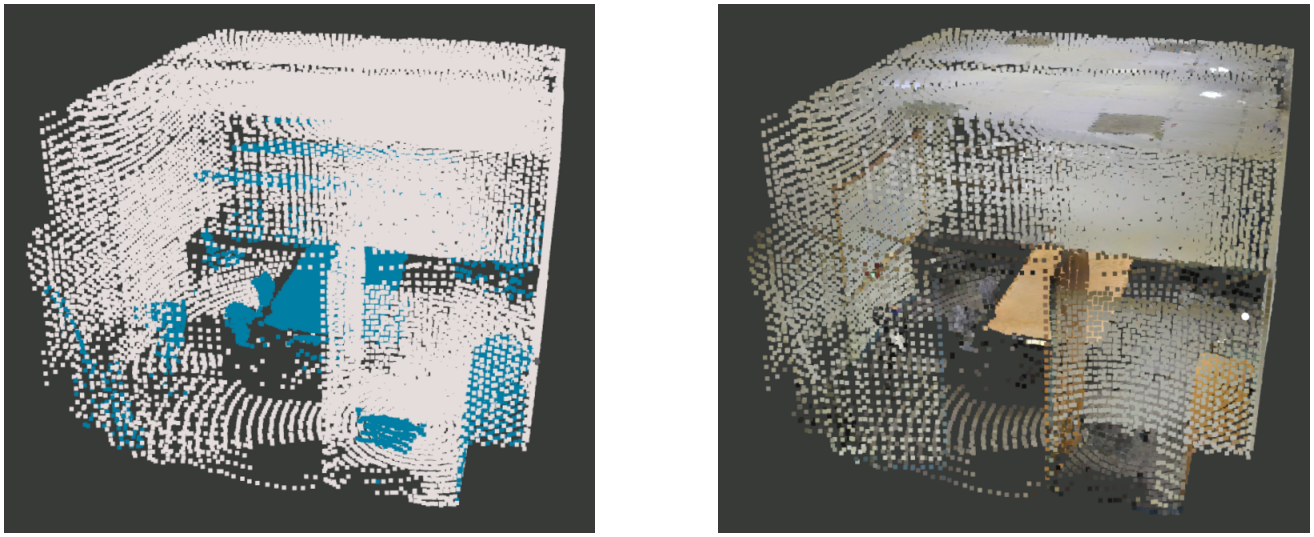
(a) Boundary (b) Scanning

Figure 5.9: Boundary and Scanning of Conference Room 1 under Pattern 3 with 38834 Points



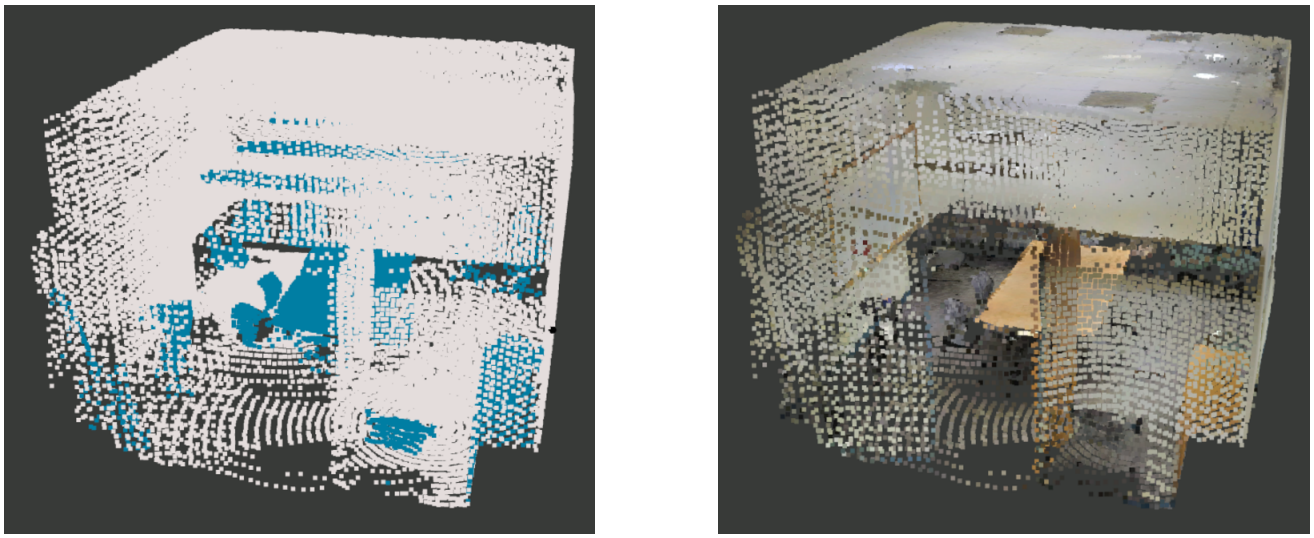
(a) Boundary (b) Scanning

Figure 5.10: Boundary and Scanning of Conference Room 1 under Pattern 4 with 74524 Points



(a) Boundary (b) Scanning

Figure 5.11: Boundary and Scanning of Conference Room 1 under Pattern 5 with 77206 Points



(a) Boundary (b) Scanning

Figure 5.12: Boundary and Scanning of Conference Room 1 under Pattern 6 with 113358 Points

| Pattern | Occluded Area Ratio | Boundary Ray Ratio |
|---|---|---|
| 1 | 0.291 | 0.248 |
| 2 | 0.178 | 0.214 |
| 3 | 0.290 | 0.332 |
| 4 | 0.109 | 0.173 |
| 5 | 0.153 | 0.194 |
| 6 | 0.071 | 0.165 |

Table 5.2: Result of Conference Room 1

The result shows that as the number of viewpoints increase, the *Occluded Area Ratio* decrease. There are 2 viewpoints in both pattern 4 and 5, thus the increase of data in pattern 5 is not against our conclusion. Values of *Boundary Ray Ratio* describe the same tendency, and they do not differ too much from data in *Occluded Area Ratio*. Therefore, based on these results we consider *Boundary Ray Ratio* as a reliable metric to represent the occlusion level of the scene presented by point cloud *Conference Room 1*.

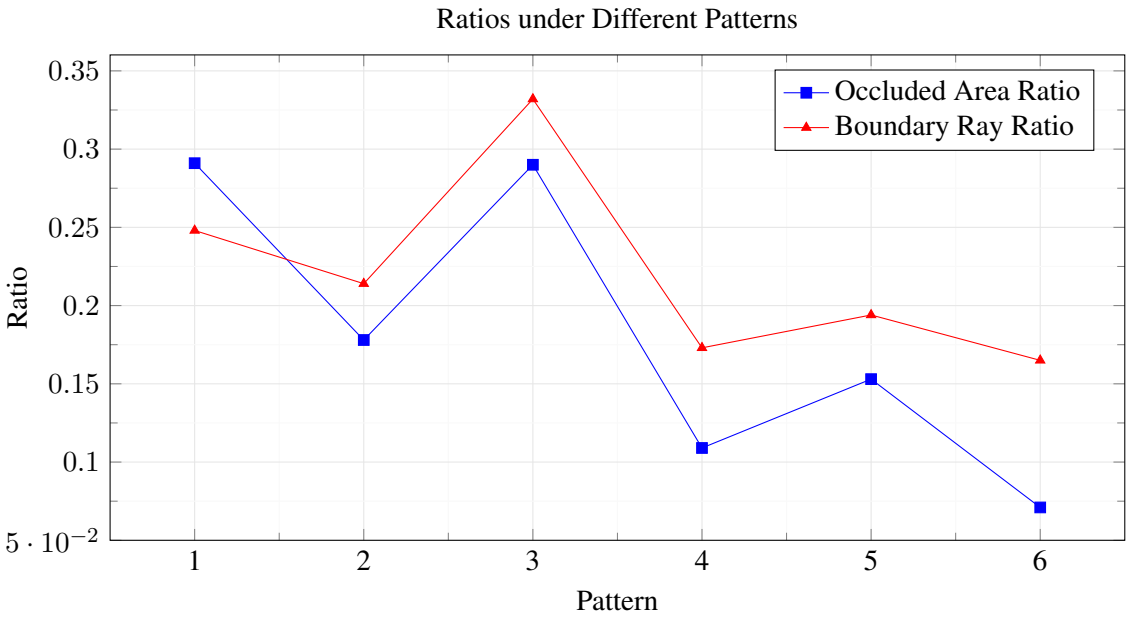



Figure 5.13: Result of Conference Room 1

**Conference Room 2** The resulting mesh estimation is shown in Figure 5.14. Then we present the visualization of scanned point cloud together with its boundary cloud. The workflow here is the same as explained in 5.1.2 and conducted with same patterns displayed in Figure 5.2. Results are shown in the end of the part in Table 5.3 and Figure 5.21.

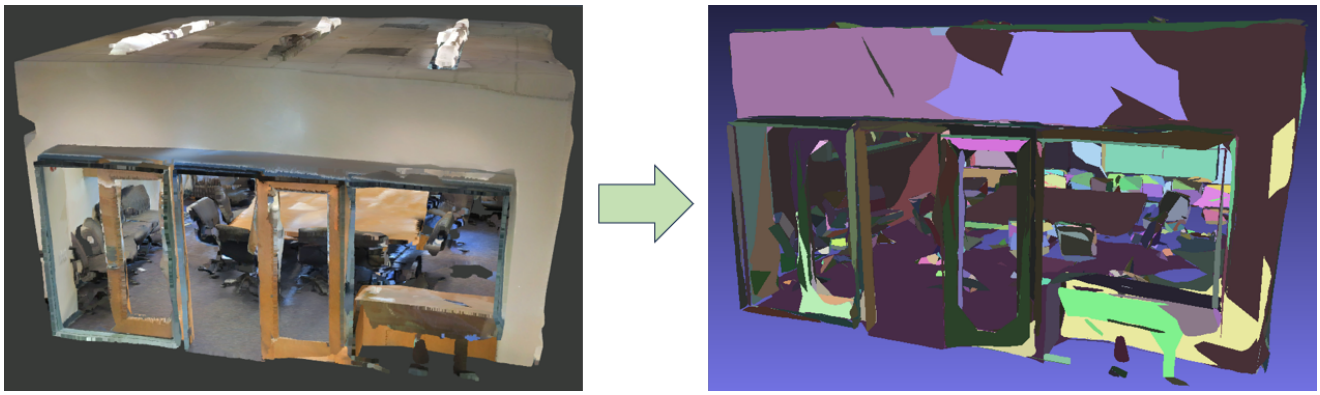
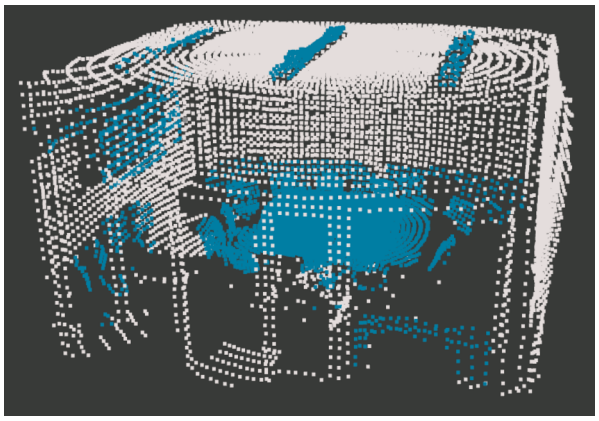
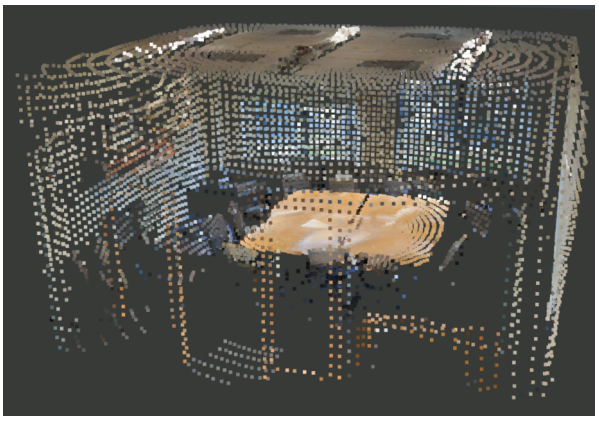


Figure 5.14: Estimate Mesh from Conference Room 2

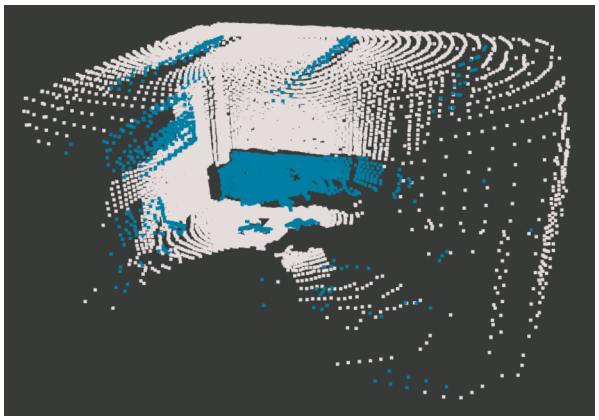


(a) Boundary

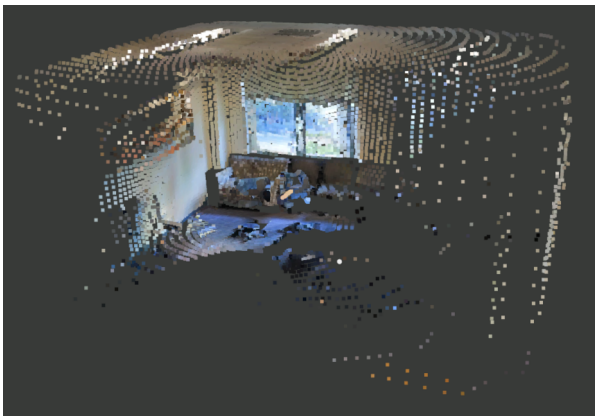


(b) Scanning

Figure 5.15: Boundary and Scanning of Conference Room 2 under Pattern 1 with 36742 Points

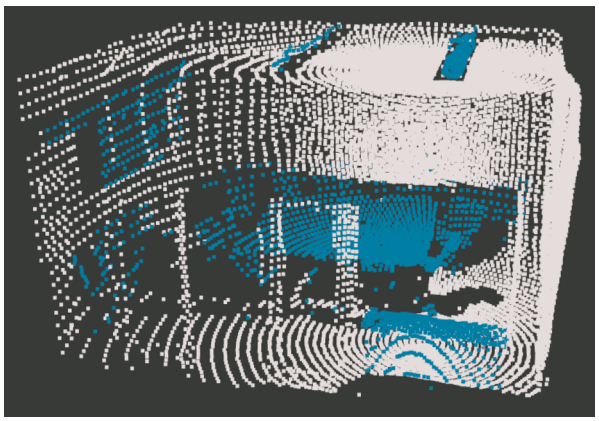


(a) Boundary

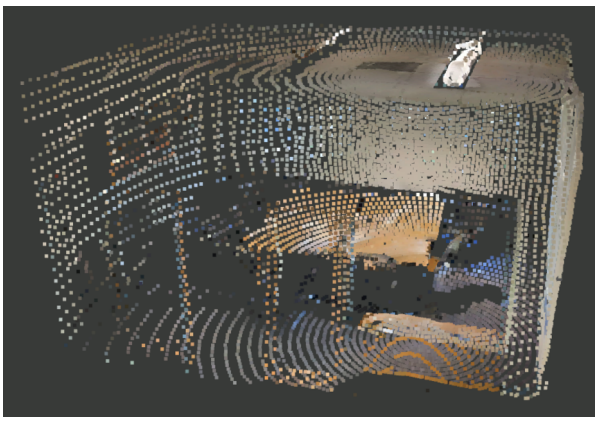


(b) Scanning

Figure 5.16: Boundary and Scanning of Conference Room 2 under Pattern 2 with 34270 Points

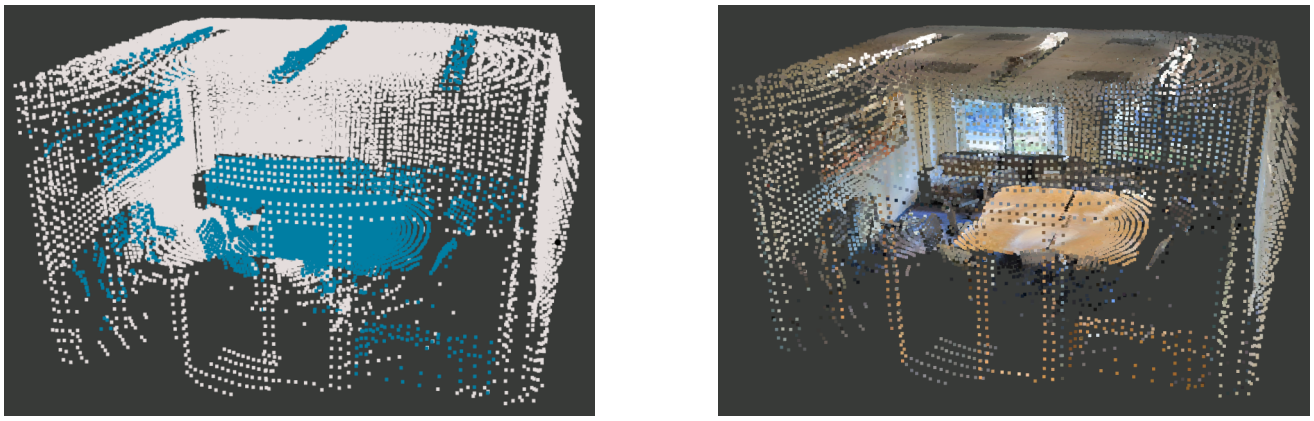


(a) Boundary



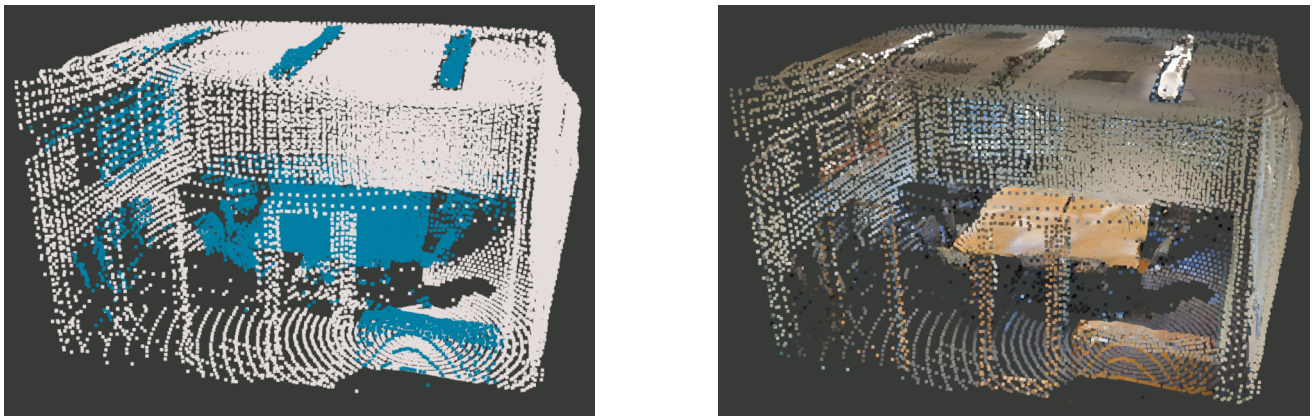
(b) Scanning

Figure 5.17: Boundary and Scanning of Conference Room 2 under Pattern 3 with 35260 Points



(a) Boundary (b) Scanning

Figure 5.18: Boundary and Scanning of Conference Room 2 under Pattern 4 with 71012 Points



(a) Boundary (b) Scanning

Figure 5.19: Boundary and Scanning of Conference Room 2 under Pattern 5 with 72002 Points

(a) Boundary (b) Scanning

Figure 5.20: Boundary and Scanning of Conference Room 2 under Pattern 6 with 106272 Points

| Pattern | Occluded Area Ratio | Boundary Ray Ratio |
|---|---|---|
| 1 | 0.371 | 0.319 |
| 2 | 0.525 | 0.473 |
| 3 | 0.313 | 0.321 |
| 4 | 0.294 | 0.282 |
| 5 | 0.264 | 0.265 |
| 6 | 0.218 | 0.223 |

Table 5.3: Result of Conference Room 2

Both ratios display the same trend and also present a minor difference on value. Thus we can conclude that *Boundary Ray Ratio* is able to assess the occlusion level of *Conference Room 2*.

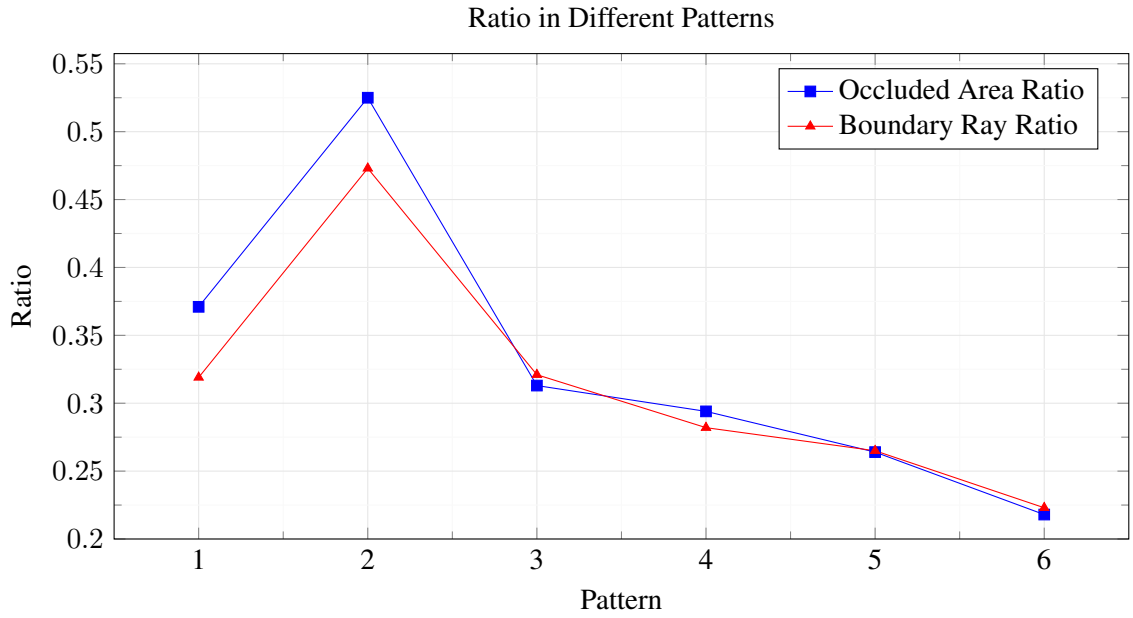



Figure 5.21: Result of Conference Room 2

In this section we conducted 2 experiments to compute and compare occlusion level for different scenes. The result proves that *Boundary Ray Ratio* can be applied for computation in subsequent experiments.

## 5.2 Correlation

The major task in this part is to find correlation between occlusion level and performance of segmentation. Due to the difference in terms of complexity and structure of the 2 scenes, it is obviously not a feasible way to directly compare the metrics of them. Thus, We apply scanned point clouds generated in previous experiment as input data to *Minkowski Engine* for semantic segmentation. In the final step, we calculate evaluation metrics for each segmented cloud. With occlusion level and result of evaluation metrics of this data set, we may correlate them and analyze the impact of occlusion on semantic segmentation. The comparison should be done between data generated from the same scene.

### 5.2.1 Setup

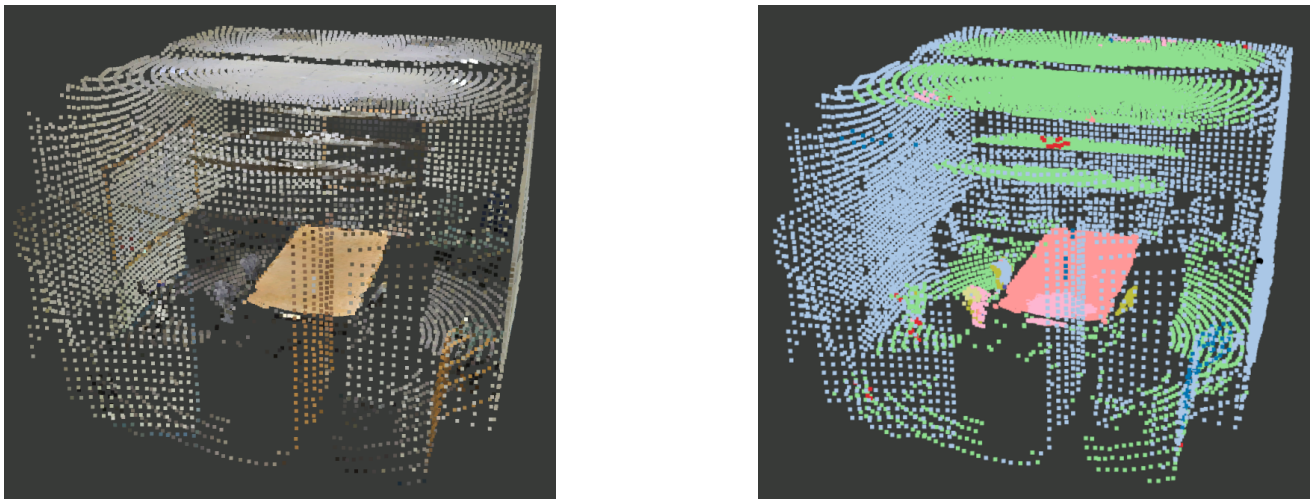
Since we have generated the set of occluded point clouds which also refer to scanned point cloud in previous experiment, we directly use it for semantic segmentation.

Point cloud with predicted semantic information is then generated for each occluded point cloud via *Minkowski Engine*. We calculate evaluation metrics in a pair-wise manner as shown in Figure 5.22 as we have to check the correctness of predicted labels point by point.

### 5.2.2 Results

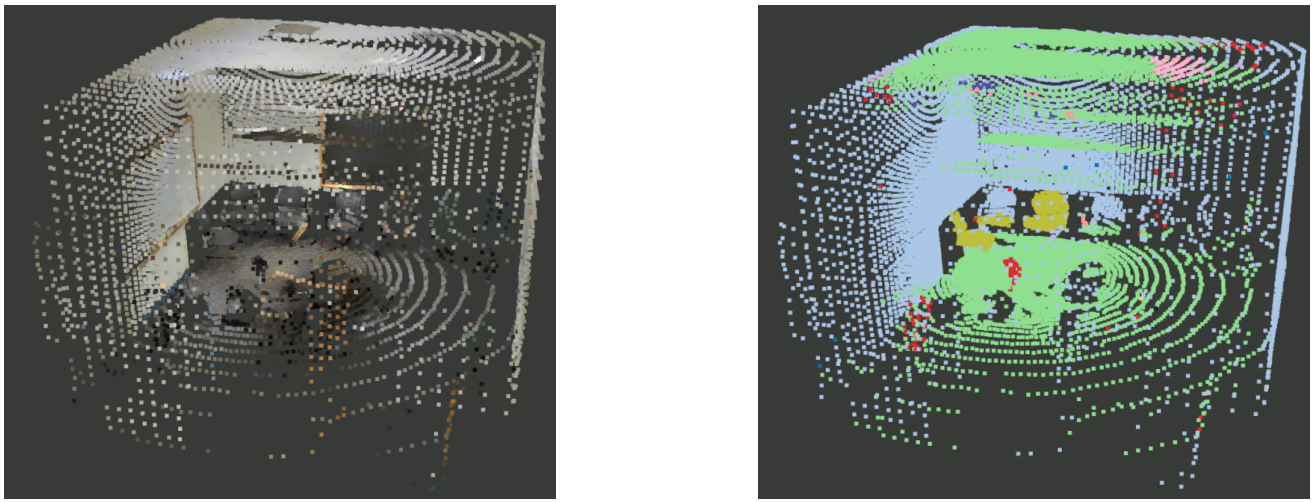
In this part, we display visualization for each pair of clouds which includes one occluded cloud with ground truth RGB information and one cloud with semantic label represented by specific colors. In the end of each part, output of the computational pipeline will be shown in tables and graphs where we can compare data more conveniently.

**Conference Room 1** We first exhibit visual output of scanned point cloud again and then its corresponding segmentation of *Conference Room 1*.



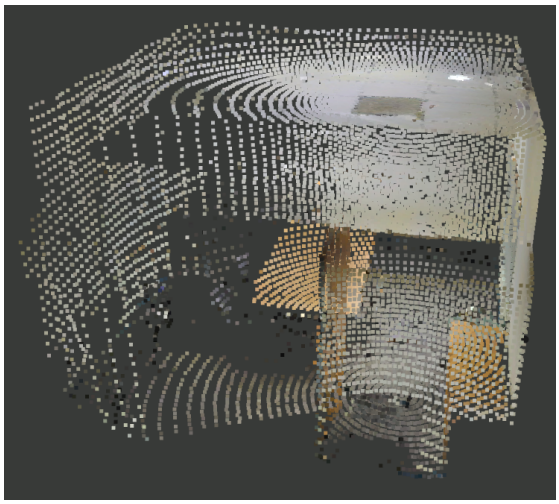
(a) Scanning (b) Segmentation

Figure 5.22: Scanning and Corresponding Segmentation under Pattern 1 of Conference Room 1

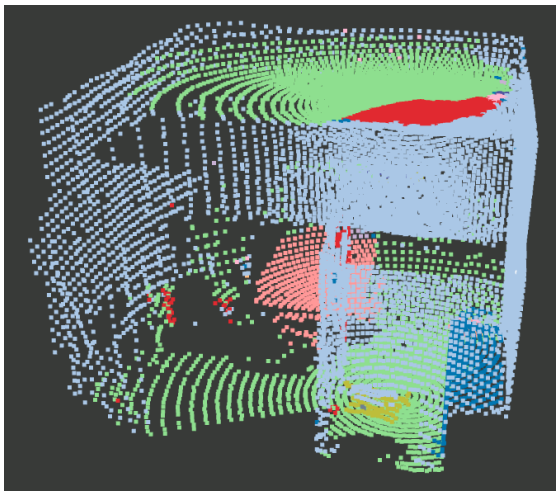


(a) Scanning (b) Segmentation

Figure 5.23: Scanning and Corresponding Segmentation under Pattern 2 of Conference Room 1

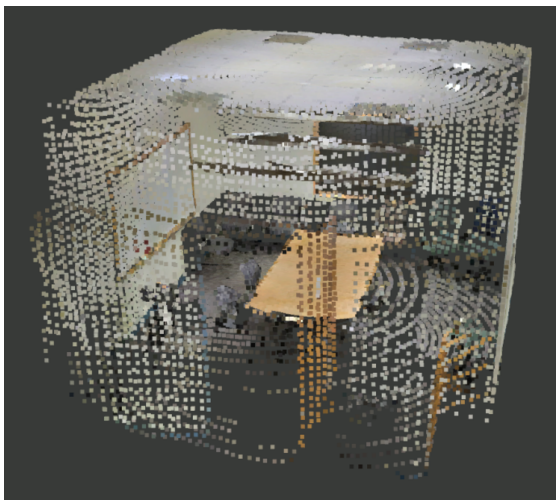


(a) Scanning

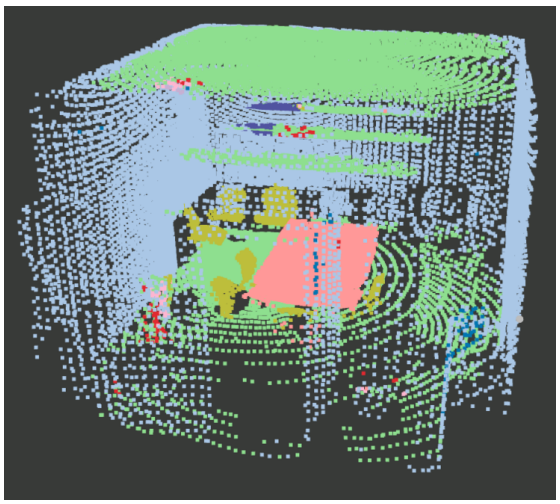


(b) Segmentation

Figure 5.24: Scanning and Corresponding Segmentation under Pattern 3 of Conference Room 1

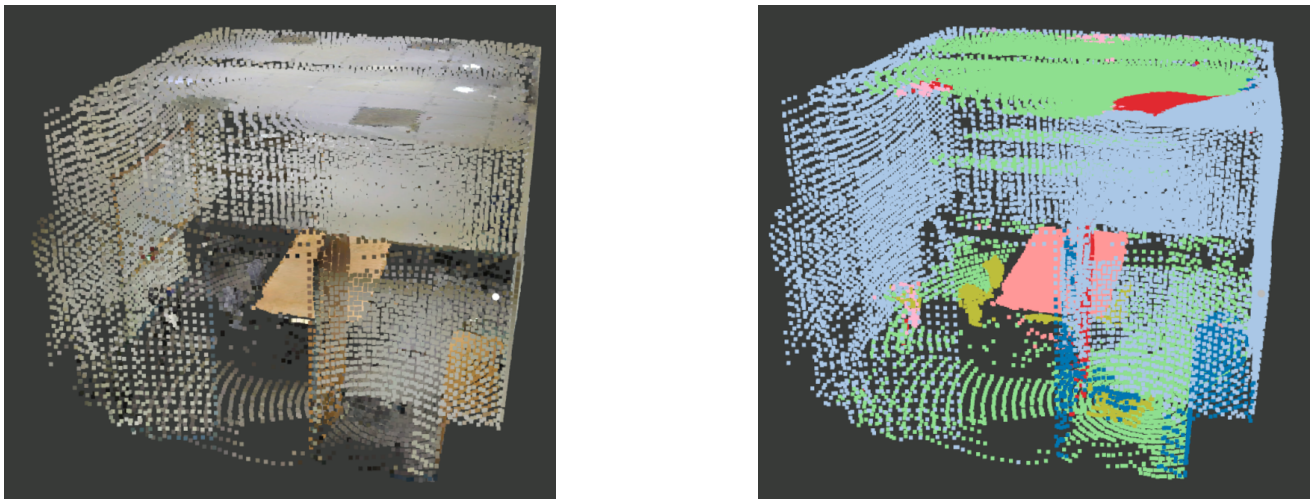


(a) Scanning



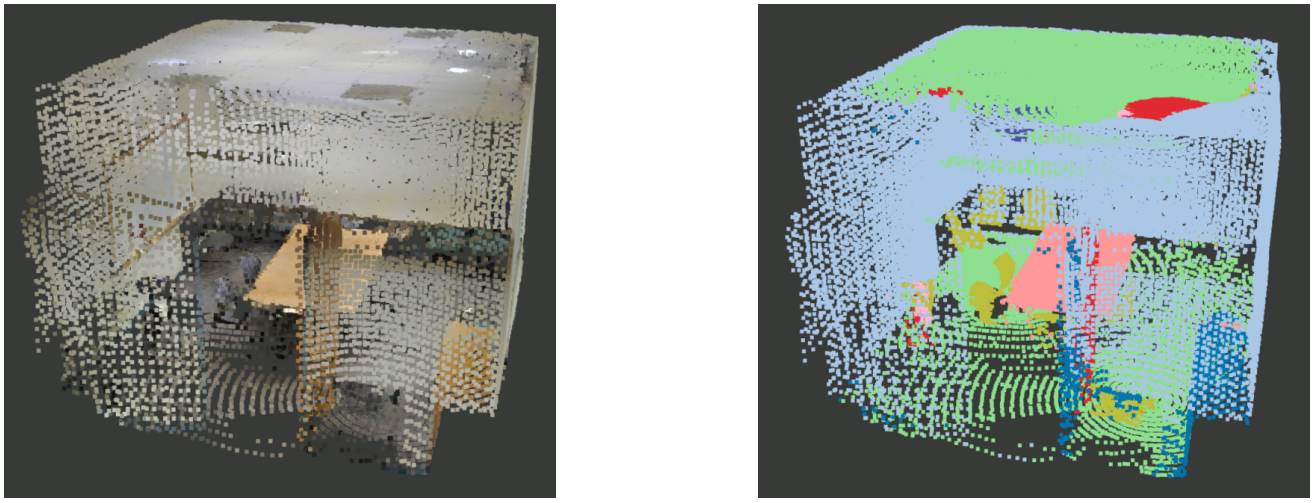
(b) Segmentation

Figure 5.25: Scanning and Corresponding Segmentation under Pattern 4 of Conference Room 1



(a) Scanning (b) Segmentation

Figure 5.26: Scanning and Corresponding Segmentation under Pattern 5 of Conference Room 1



(a) Scanning (b) Segmentation

Figure 5.27: Scanning and Corresponding Segmentation under Pattern 6 of Conference Room 1

The result is shown below in Table 5.4 and Figure 5.28.

| Pattern | Occlusion | F1 Score |
|---|---|---|
| 1 | 0.248 | 0.826 |
| 2 | 0.214 | 0.887 |
| 3 | 0.332 | 0.680 |
| 4 | 0.173 | 0.886 |
| 5 | 0.194 | 0.737 |
| 6 | 0.165 | 0.807 |

Table 5.4: Evaluation Metrics of Conference Room 1

From the graph we can see opposite trends between occlusion level and evaluation metrics. This indicates an inversely proportional relationship. If we compare data of clouds under different patterns pair-wisely, the negative

correlation does not always hold. For example, occlusion level of cloud under pattern 4 is higher than the value in pattern 6, but F1 score in pattern 4 is still higher. Based on that, we conclude that the occlusion level and performance of semantic segmentation are inversely proportional related to each other in terms of *Conference Room 1*.

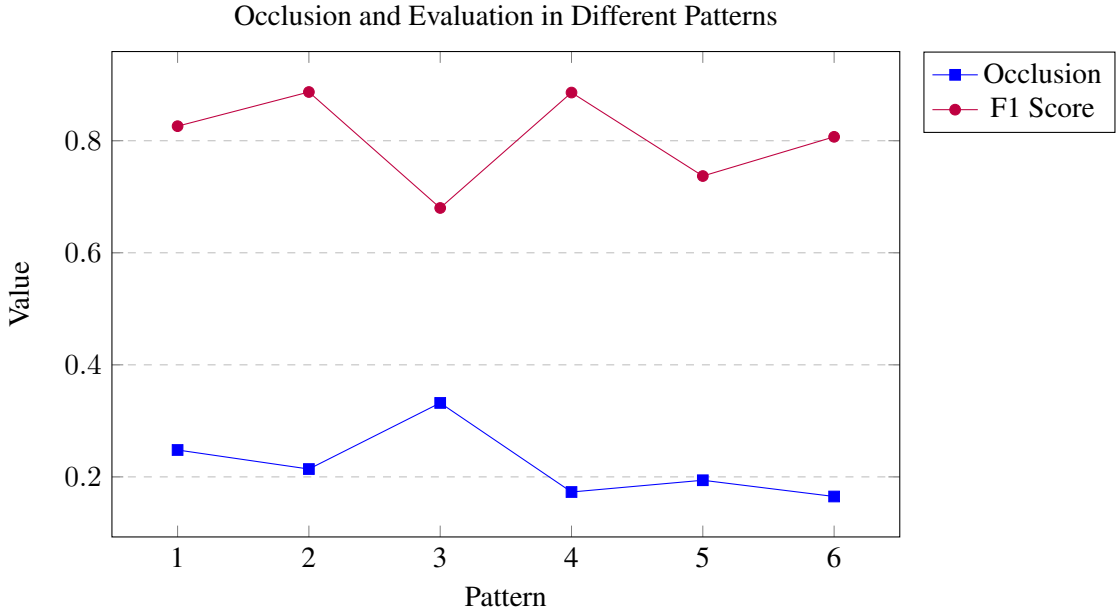
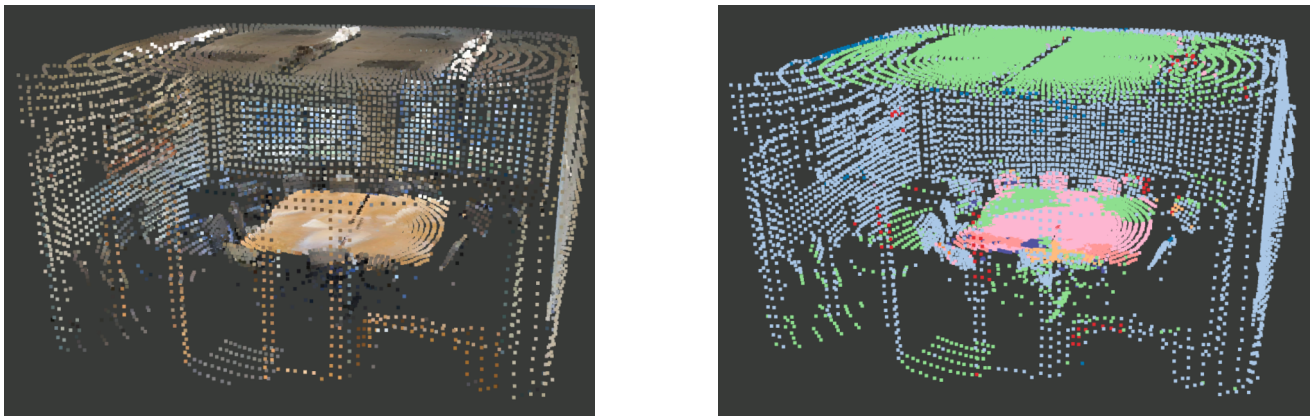


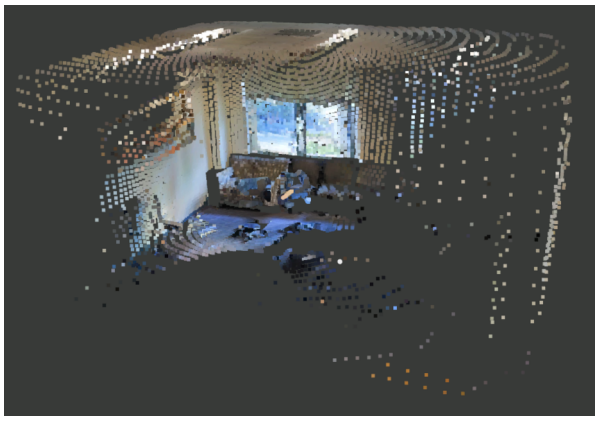

Figure 5.28: Occlusion and Evaluation of Conference Room 1

**Conference Room 2** We apply same workflow as described in *Conference Room 1* to present our results.

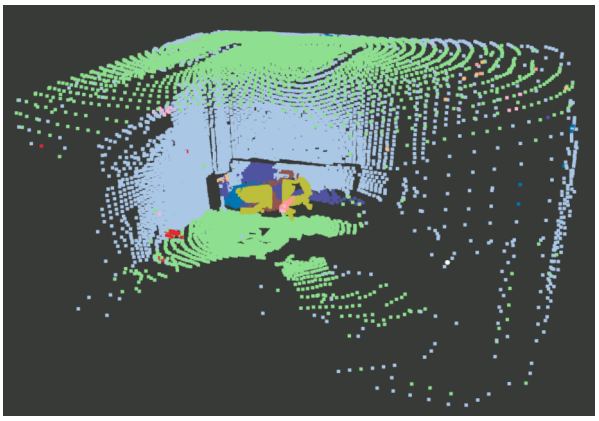


(a) Scanning (b) Segmentation

Figure 5.29: Scanning and Corresponding Segmentation under Pattern 1 of Conference Room 2

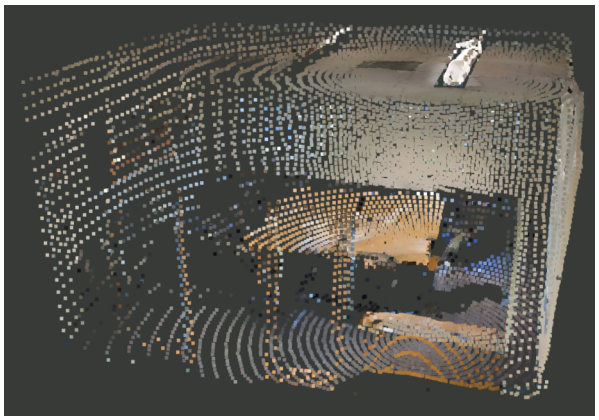


(a) Scanning

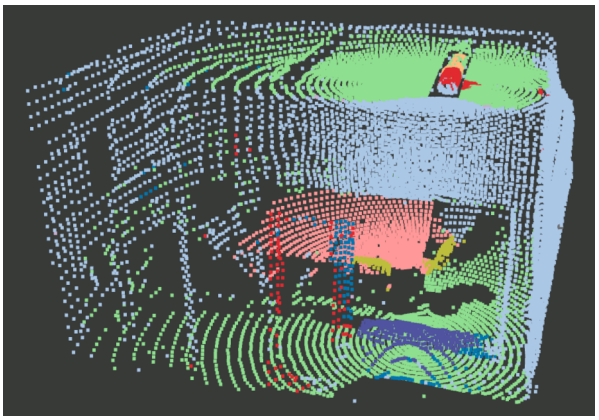


(b) Segmentation

Figure 5.30: Scanning and Corresponding Segmentation under Pattern 2 of Conference Room 2



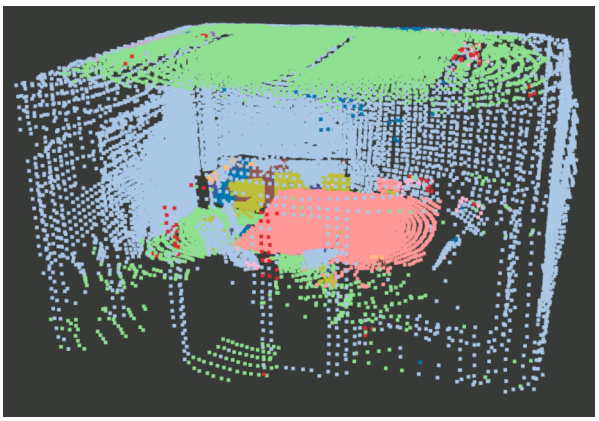
(a) Scanning



(b) Segmentation

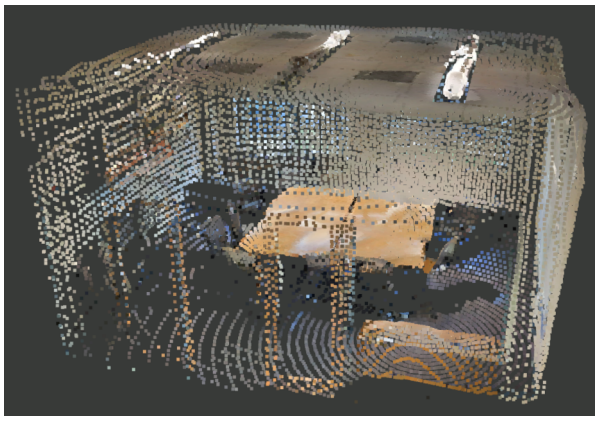
Figure 5.31: Scanning and Corresponding Segmentation under Pattern 3 of Conference Room 2

(a) Scanning

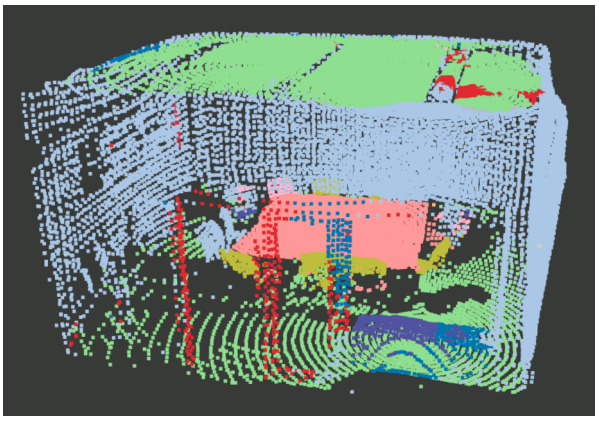


(b) Segmentation

Figure 5.32: Scanning and Corresponding Segmentation under Pattern 4 of Conference Room 2

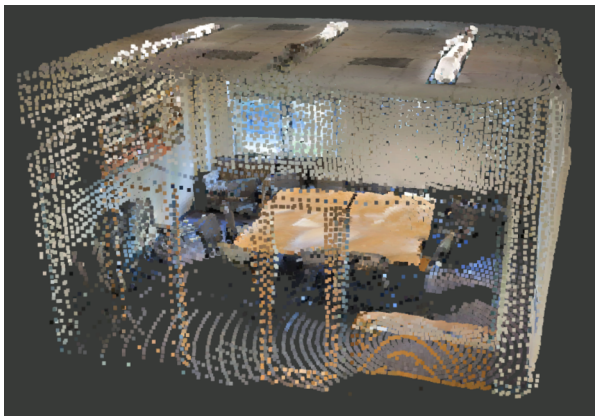


(a) Scanning

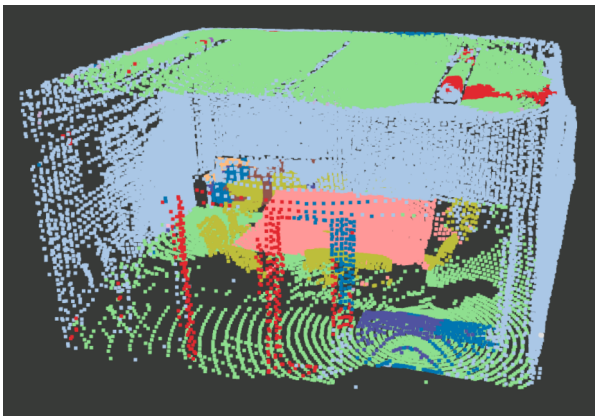


(b) Segmentation

Figure 5.33: Scanning and Corresponding Segmentation under Pattern 5 of Conference Room 2



(a) Scanning



(b) Segmentation

Figure 5.34: Scanning and Corresponding Segmentation under Pattern 6 of Conference Room 2

The result is shown in Table 5.5 and Figure 5.35.

| Pattern | Occlusion | F1 Score |
|---|---|---|
| 1 | 0.319 | 0.765 |
| 2 | 0.473 | 0.888 |
| 3 | 0.321 | 0.772 |
| 4 | 0.282 | 0.858 |
| 5 | 0.265 | 0.797 |
| 6 | 0.223 | 0.837 |

Table 5.5: Occlusion and Evaluation of Conference Room 2

The results in pattern 1, 2 and 3 show the same trends as opposed to the result of *Conference Room 1*. If we compare data in other patterns pair-wisely, both positive and negative relationships can be found. Hence, we came to the conclusion that no obvious correlation can be found between occlusion level and performance of semantic segmentation in terms of *Conference Room 2*.

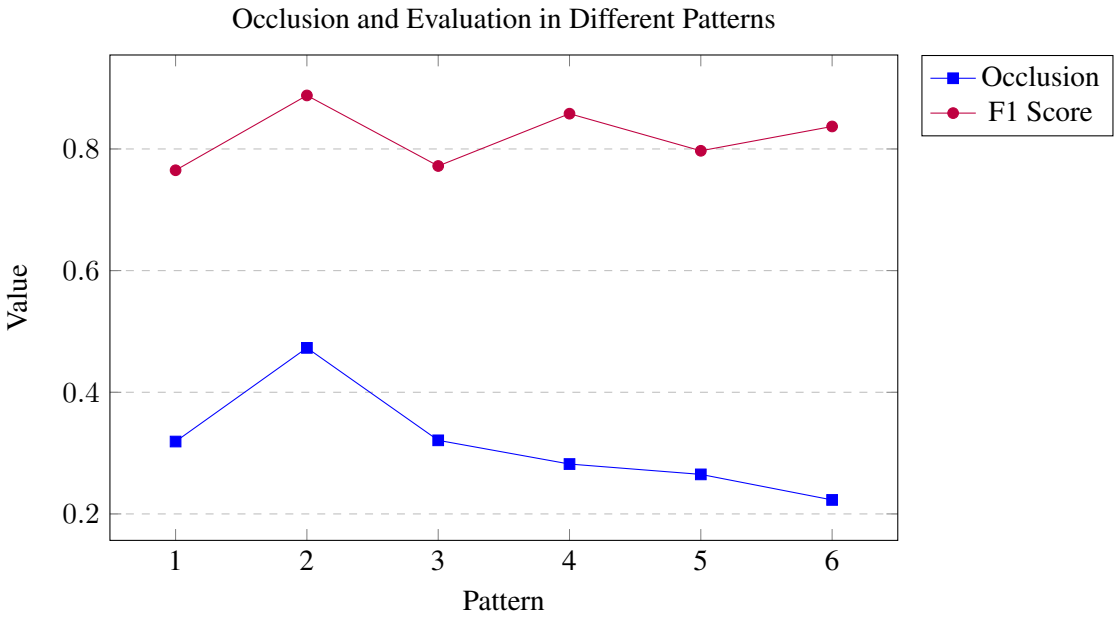



Figure 5.35: Occlusion and Evaluation of Conference Room 2

# 6 Conclusion and Discussion

## 6.1 Conclusion

Semantic segmentation achieved through *Minkowski Engine* shows a good understanding on classification of structure and objects of the point cloud based indoor scene. But the accuracy of the output still needs to be improved in order to extend the application of semantic segmentation. Hence, in this bachelor thesis we focus on occlusion, which is a common feature to represent the loss of information of point cloud acquired by scanning, to investigate its impact on the performance of semantic segmentation. We first proposed the metric *occluded area ratio* to reflect the occlusion level of mesh. Then we extend this concept to point cloud and proposed another metric *boundary ray ratio*. Through the comparison of the 2 ratios of the same scene in Section 5.1.2 we came to the conclusion that *boundary ray ratio* is a reliable metric to estimate occlusion level of a point cloud. Based on that we applied this metric in our experiments to estimate how much the set of scanned point clouds are occluded and compare the output with the result of evaluation metric *F1 score*. From the results described in Section 5.2, we found that in the scene *conference room 1* there exists an inversely proportional relationship between occlusion level and performance of semantic segmentation, while in *Conference Room 2* there is no obvious correlation can be found.

It is worth to mention some edge cases such as the cloud scanned under pattern 2 of *conference room 2* shown in Figure 5.30, where the cloud has the largest loss of information but scores highest on performance. If we inspect this cloud carefully, we can notice that despite the significant incompleteness of the whole scene, the region close to the corner exhibits a dense state, thus the segmentation method might have better understanding there due to high richness of structural information. This indicates that the performance is also affected by structure of point cloud and density of points in certain region. Therefore, occlusion alone is not enough to affect the result of semantic segmentation. It is critical to consider additional metrics to obtain a better evaluation. Other influential factors may include the complexity of the scenes being segmented, variations in placement of light sources, and the diversity of the interior items. Moreover, spatial relationship between different objects and structures may affect segmentation method's understanding on point cloud. According to these findings, we came to the conclusion that occlusion level of point cloud has limited impact on its performance of semantic segmentation.

## 6.2 Discussion

### 6.2.1 Limitation

Although the reliability of the metric for estimating occlusion level has been proven, it is not computing the real occlusion of point cloud. Our way of estimation is achieved through random rays intersecting boundary points, which is highly dependent on the exterior structure of the scene. Due to the limited diversity of data used in our experiments, we cannot guarantee that our metric will remain effective on more complex structures.

### 6.2.2 Future Work

Some future work can be done for improvements. More data with different structures and complexities can be used in experiments to improve robustness and reliability of occlusion metrics. The development of more comprehensive metrics for evaluating occlusion levels should be a priority, with a focus on considering the complex interrelations of elements within the scene. This could alternatively be done by proposing an evaluation system for

assessing the completeness of the scene, where influential factors regarding semantic segmentation are assigned with certain weights for computation.

By addressing these aspects, we can expect advancements in devising better metrics or system to evaluate factors that may influence semantic segmentation.

# 7 Acknowledgements

I would like to extend my heartfelt gratitude to the Visualization and MultiMedia Laboratory for providing me with the opportunity to undertake this challenging yet fascinating topic. I would also express my sincere appreciation to Prof. Dr. Renato Pajarola for the continual assistance and invaluable suggestions throughout the entire process. Special thanks go to Lizeth J. Fuentes Perez, whose detailed guidance and dedication to spending regular time discussing various aspects of this project have been instrumental. Lastly, I have to thank my family and friends, your support has allowed me to concentrate fully on completing this project.

# Bibliography


[AEF+95] N. Akkiraju, H. Edelsbrunner, M. Facello, P. Fu, E. Mücke, and C. Varela. Alpha shapes: definition and software. In *International Computational Geometry Software Workshop (GCG)*, pages 63–66, 1995.

[ASZ+16] Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, 2016.

[Ble23] Blender. Blender (version 3.6.0). `https://www.blender.org`, 2023. Computer software.

[CCC+08] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. In Vittorio Scarano, Rosario De Chiara, and Ugo Erra, editors, *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008.

[CGS19] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, second edition, 2001. Section 33.3: Finding the convex hull, pp. 947–957.

[Eri05] Christer Ericson. Chapter 6 - bounding volume hierarchies. In Christer Ericson, editor, *Real-Time Collision Detection*, The Morgan Kaufmann Series in Interactive 3D Technology, pages 235–284. Morgan Kaufmann, San Francisco, 2005.

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, jun 1981.

[FH05] Michael S. Floater and Kai Hormann. Surface parameterization: a tutorial and survey. In Neil A. Dodgson, Michael S. Floater, and Malcolm A. Sabin, editors, *Advances in Multiresolution for Geometric Modelling*, pages 157–186, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[GJ+10] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[McG17] Morgan McGuire. Computer graphics archive, July 2017.

[Mea80] Donald Meagher. Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer, 10 1980.

[MIH+16] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. arXiv:1512.02134.

[Mr.21] Mr.doob. Three.js - javascript 3d library. `https://threejs.org`, 2021.

[MT97] Tomas Möller and Ben Trumbore. Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools*, 2(1):21–28, 1997.

[OR21] Bojun Ouyang and Dan Raviv. Occlusion guided scene flow estimation on 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 2805–2814, June 2021.

[RC11] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[Rus23a] Radu Bogdan Rusu. Planar segmentation. `https://pcl.readthedocs.io/projects/tutorials/en/latest/planar_segmentation.html`, 2023. Accessed: June 5, 2023.

[Rus23b] Radu Bogdan Rusu. Region growing segmentation. `https://pcl.readthedocs.io/projects/tutorials/en/latest/region_growing_segmentation.html`, 2023. Accessed: June 5, 2023.

[Scr23a] ScratchPixel. Minimal ray tracer: Rendering simple shapes - ray-box intersection. `https://www.scratchapixel.com/lessons/3d-basic-rendering/minimal-ray-tracer-rendering-simple-shapes/ray-box-intersection.html`, 2023. Accessed: June 22, 2023.

[Scr23b] ScratchPixel. Minimal ray tracer: Rendering simple shapes - ray-sphere intersection. `https://www.scratchapixel.com/lessons/3d-basic-rendering/minimal-ray-tracer-rendering-simple-shapes/ray-sphere-intersection.html`, 2023. Accessed on August 10, 2023.

[WLY+21] Hanchen Wang, Qi Liu, Xiangyu Yue, Joan Lasenby, and Matt J. Kusner. Unsupervised point cloud pre-training via occlusion completion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9782–9792, October 2021.

[YL22] Mulin Yu and Florent Lafarge. Finding good configurations of planar primitives in unorganized point clouds. In *Proc. of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, New Orleans, US, 2022.