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 work 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's a palpable gap in understanding how to quantify occlusion in the scene and how this characteristic influence the performance of advanced semantic segmentation softwares like the Minkowski Engine. To bridge the research gap, we proposed a novel metric to quantify the occlusion level of a scene with viewpoint information. 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 is inversely proportional to the quality of semantic segmentation. This finding is crucial for advancing the state of the art in semantic segmentation and for paving the way for more robust applications in autonomous driving, robotics, and 3D scene analysis.

Contents

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 [S3dis], 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.

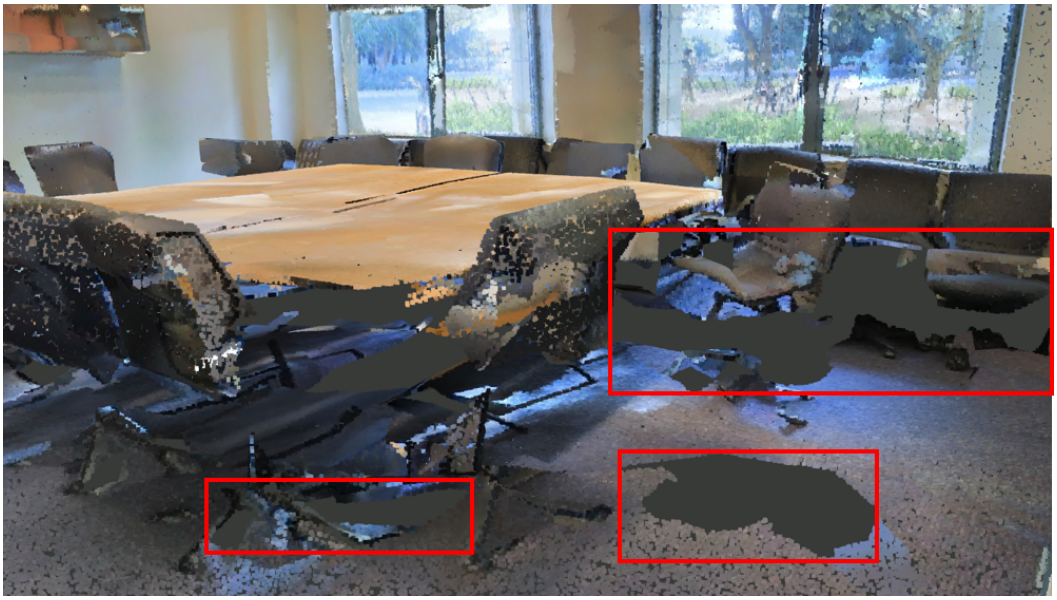


Figure 1.1: Occlusion in a scene

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 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's important to make an accurate occlusion prediction. Therefore, they evaluate the performance of occlusion estimation on dataset FlyingThings3D, 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, the authors proposed $OcclusionCompletion$, 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 weighths 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 occluison 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 decide to create occlusion in the scene by placing viewpoints in different locations. With the dataset we created, we can propose a metric to quantify the occlusion level of a scene. If we think about occlusion level of a scene in real world, our intuition is that the more viewpoints we use to look at a scene, the more area of the scene we can see. Therefore, we propose a metric that is based on the ratio between visible area and total area of the scene. Since there's no surface area information in a point cloud, we would first validate our metric on mesh. To extend the concept $VisibleAreaRatio$ to point cloud, we would use $OcclusionRay$, which means the ray hit occlusion in the scene, to estimate occlusion level.

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.[wiki]

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 ascended as a pivotal research domain, given its profound implications in a myriad of applications. From the intricate pathways navigated by autonomous vehicles 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 discern the true 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 setting, a chair might be obscured by a table. These occlusions can lead to misclassifications, reducing the overall accuracy of the segmentation process.

This project proposal is rooted in the quest to unravel the intricacies of occlusion within point cloud data. Specifically, we aim to delve deep into understanding how varying levels of occlusion in a scene impact the performance and quality of semantic segmentation methods. By systematically analyzing the effects of occlusion, we aspire to shed light on potential strategies to enhance segmentation accuracy, even in highly occluded environments.

Minkowski Engine. The Minkowski Engine stands out as a state-of-the-art tool in this domain. It 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.

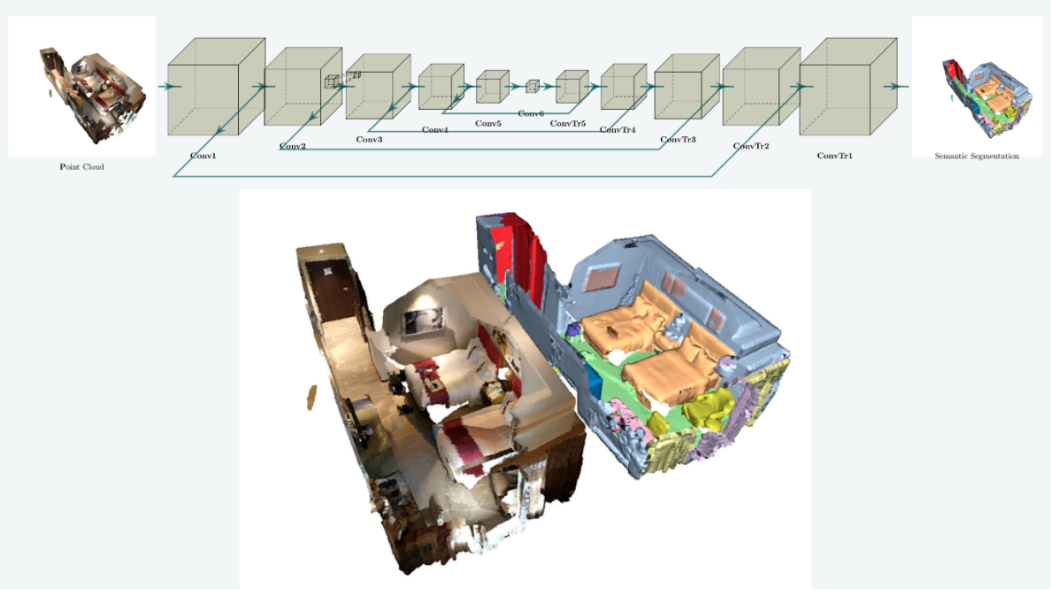


Figure 1.2: Minkowski Engine Indoor Scene Segmentation

1.4 Motivation

Among the myriad factors influencing the semantic segmentation of point cloud data, the level of occlusion stands as a paramount challenge. 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, conventional semantic segmentation approaches might falter, leading 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's 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 occlusion levels and delve deep into their implications on point cloud data semantic segmentation. Through this investigation, we aim to offer more precise semantic segmentation methodologies, especially in environments with high levels of occlusion. In essence, our objective is to forge a nexus between occlusion levels and the efficacy of semantic segmentation, providing invaluable insights for future research and applications.

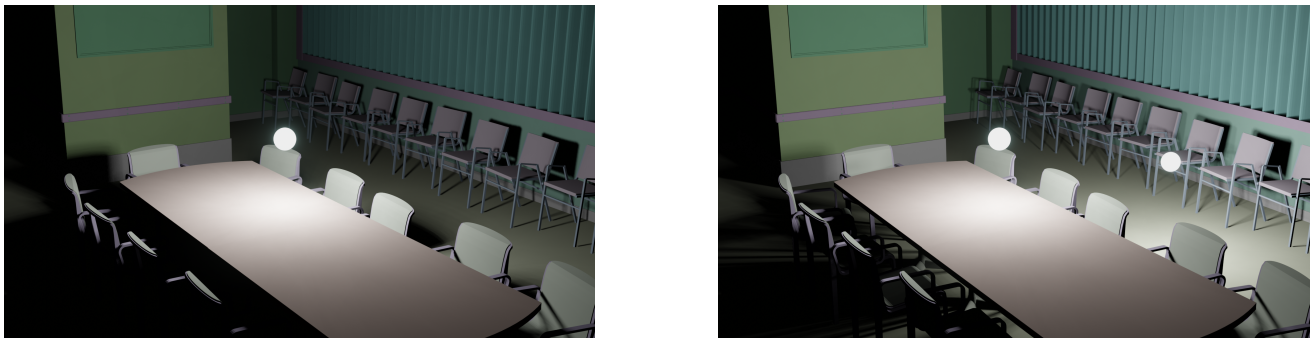
1.5 Outline

In this work we use ray-tracing based methods to scan ground truth cloud and compute occlusion level for the scanned cloud. Before we doing this, we apply similar methods to estimate occlusion level of mesh to validate that ray-based methods are reliable. Finally, an interactive web application is developed to visualize the point cloud together with a backend is developed to compute the occlusion level of the point cloud.

2 Problem Statement

2.1 Occlusion Level Computation of Mesh

We have to validate here that more viewpoints lead to lower occlusion level of the interior scene.



(a) 1 viewpoint

(b) 2 viewpoints

Figure 2.1: Scene with Light Sources

2.1.1 Ground Truth Mesh

We should compute the occlusion level of ground truth mesh.

2.1.2 Estimated Mesh from Point Cloud

We also want to estimate the occlusion level of the mesh generated from ground truth point cloud.

Why do we wanna do thi

2.2 Occlusion Level Computation of Point Cloud

After validation in previous steps, we should directly compute the occlusion level of the ground truth point cloud.

2.3 Evaluation of Segmentation Result

It's essential to evaluate result of segmentation of point cloud so that we can maybe find correlation between occlusion level and segmentation performance.

3 Technical Solution

In this chapter we will introduce our technical solution to the problem stated in 2.

3.1 Ray Tracing Based Estimation of Visible Area Ratio

Ray tracing is a powerful technique used in computer graphics to simulate the way light interacts with objects to generate realistic images. In the context of our study, ray tracing plays a pivotal role in estimating the visible area ratio of triangles in a scene. This estimation is crucial for understanding how much of a triangle's surface is directly illuminated by a light source, especially when other objects in the scene might occlude it.

A flow chart of the whole pipeline is shown here.

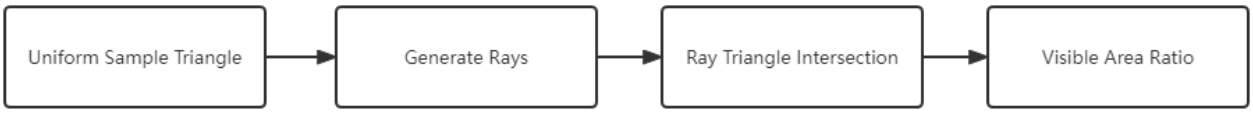


Figure 3.1: Flow chart of visible area ratio estimation

3.1.1 Uniform Sample Triangles

To accurately represent a triangle's area using samplings, it's imperative that these samplings comprehensively cover the triangle. For this purpose, we employ the random uniform sampling method. This method ensures that each point within the triangle has an equal chance of being selected, leading to a fair representation of the triangle's entire area.

The process begins by randomly generating two parameters, r_1 and r_2, both of which lie in the interval [0, 1]. These random parameters are then used to compute the barycentric coordinates of the sampled points within the triangle. The barycentric coordinates, denoted as α, β, and γ, allow us to express any point within the triangle as a linear combination of the triangle's vertices.

To derive the sampled point's coordinates using r_1 and r_2, we use the following equations:

$$\alpha = 1 - \sqrt{r_1}$$

$$\beta = \sqrt{r_1} \times r_2$$

$$\gamma = 1 - \alpha - \beta$$

$$P = \alpha V_1 + \beta V_2 + \gamma V_3$$

Where P is the sampled point, and V_1, V_2, and V_3 are the triangle's vertices. Below is a visualization of the uniform sampling method.

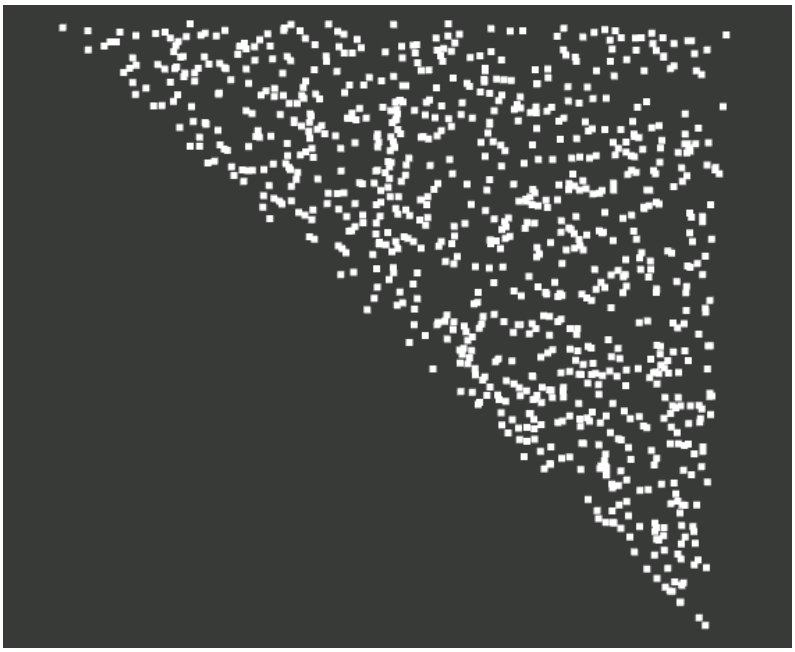


Figure 3.2: Uniform Sample Triangle

3.1.2 Generate Rays

In computer graphics, especially in the context of ray tracing, generating rays is a fundamental step. These rays simulate the path of light as it interacts with objects in a scene. In our approach, we generate rays from the sampled points on the triangle to a light source. This is crucial for determining how light interacts with the triangle, especially when considering factors like shading and occlusion.

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 look-at point and the origin. The direction vector is then normalized to ensure its magnitude is 1, which simplifies subsequent calculations.

The direction of the ray, Direction, can be computed as:

$$\text{Direction} = \frac{\text{LookAt} - \text{Origin}}{\|\text{LookAt} - \text{Origin}\|} \tag{3.1}$$

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.

By generating rays in this manner, we can accurately simulate the behavior of light as it travels from the triangle to the viewpoint or light source. This is essential for producing realistic renderings and for analyzing the effects of various factors, such as occlusion, on the final image.

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 intersection algorithm. This algorithm determines the intersection of a ray and a triangle in a 3D space without any need for pre-computed plane equations.

Given a ray represented by its origin O and direction D, and a triangle defined by its vertices V_1, V_2, and V_3, the algorithm computes the intersection using barycentric coordinates.

The intersection point P can be represented as:

$$P = (1 - u - v)V_1 + uV_2 + vV_3$$

Where u and v are the barycentric coordinates. The ray intersects the triangle if $0 \leq u \leq 1$, $0 \leq v \leq 1$, and $u + v \leq 1$.

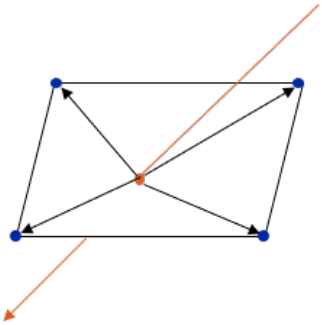


Figure 3.3: Ray Intersect Triangle

The algorithm uses the following equations to compute u, v, and t (where t is the distance from the ray origin to the intersection point):

- **Edge Vectors:** These vectors represent two edges of the triangle.

$$e_1 = V_2 - V_1$$

$$e_2 = V_3 - V_1$$

- **Vector h:** The cross product of the ray direction and e_2.

$$h = D \times e_2$$

- **Determinant a:** It's used to check if the ray is nearly parallel to the triangle.

$$a = e_1 \cdot h$$

- **Factor f:** Used for subsequent calculations.

$$f = \frac{1}{a}$$

- **Vector s:** Represents the vector from the ray's origin to one vertex of the triangle.

$$s = O - V_1$$

- **Barycentric Coordinate u:** The first computed barycentric coordinate.

$$u = f(s \cdot h)$$

- **Vector q:** The cross product of vector s and e_1.

$$q = s \times e_1$$

- **Barycentric Coordinate v:** The second computed barycentric coordinate.

$$v = f(D \cdot q)$$

- **Distance t to Intersection:** Represents the distance from the ray's origin to the intersection point.

$$t = f(e_2 \cdot q)$$

The algorithm first checks if a is close to zero, which means the ray is nearly parallel to the triangle and thus, likely does not intersect it. If a is not close to zero, the algorithm proceeds to compute u, v, and t to determine the intersection.

This algorithm is both efficient and robust, making it a popular choice for ray-triangle intersection tests in various graphics applications.

3.1.4 Visible Area Ratio

In the context of ray tracing and light interaction, understanding the visible area of a triangle is crucial. The visible area ratio provides insight into how much of a triangle's surface is directly illuminated by a light source without being occluded by other objects in the scene.

From the previous steps, we have computed samplings on each triangle and generated rays originating from these samplings directed towards the light source. To determine the visible area, we first need to compute the visibility weight for each sampling. This weight represents the ratio of visible samplings to the total number of samplings on the triangle.

Mathematically, the visibility weight, w, can be defined as:

$$w = \frac{\text{Number of visible samplings}}{\text{Total number of samplings}} \tag{3.2}$$

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

$$A_{\text{visible}} = w \times A_{\text{total}} \tag{3.3}$$

where A_{total} is the total area of the triangle.

A sampling is deemed visible if at least one ray originating from it does not intersect any triangle other than the one from which the sampling was generated. To ascertain this, we must examine all intersections related to the ray emanating from the sampling. If no intersection is closer to the sampling than the distance between the sampling and the light source, then the sampling is considered visible.

This method ensures a precise computation of the visible area, accounting for occlusions and the intricate play of light within the scene.

3.2 Segmentation Based Estimation of Mesh

3.2.1 Region Growing Segmentation

In our approach, we utilize the region growing [pcl region growing] algorithm for segmentation. This algorithm works by iteratively expanding a region by adding neighboring points that are similar based on certain criteria, such as geometric proximity and surface normals. Starting from a seed point, the region grows by incorporating neighboring points that meet the similarity criteria until no more points can be added.

The region growing algorithm can be summarized in the following steps:

1. Select a seed point from the point cloud that has not been assigned to any cluster.
2. Identify neighboring points of the seed based on a predefined distance threshold.
3. Evaluate the similarity of neighboring points based on criteria like curvature or surface normals.
4. Add the similar neighbors to the current region and mark them as visited.
5. Repeat the process for the newly added points.
6. Continue the growth until no more points can be added to the current region.
7. Start a new region with another unvisited seed point and repeat the process until all points are assigned to clusters.

By the end of this process, we obtain a number of clusters that represent distinct structures or objects in the original point cloud. These segmented clusters facilitate further analysis, such as object recognition, surface reconstruction, and other advanced 3D processing tasks.

3.2.2 Build Triangles From Clusters

After segmenting the point cloud into clusters using the region growing algorithm, the next step is to construct a mesh from these clusters. This mesh representation is essential for applying ray tracing techniques, as described in Section 3.1. To achieve this, we need to build triangles from the clusters.

Convex Hull Estimate Polygons

The first step in this process is to compute the convex hull for each cluster. The convex hull can be thought of as the ”tightest” polygon that encloses all the points in a cluster. It provides a simplified representation of the cluster’s shape, eliminating any concavities. Various algorithms, such as the Graham’s scan or the QuickHull algorithm, can be employed to compute the convex hull efficiently.

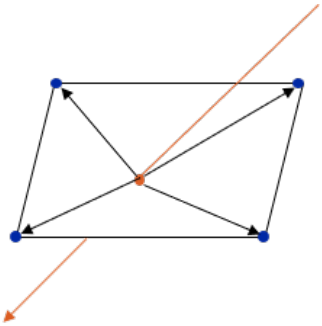


Figure 3.4: Convex Hull

Compute Polygon Centroid

Once the convex hull is determined, the next step is to compute the centroid of each polygon. The centroid, often referred to as the ”center of mass” or ”balance point”, represents the average position of all the points in the polygon. For a given polygon with vertices (x_i, y_i), the centroid (C_x, C_y) is computed using the following method:

First, the area A of the polygon is computed using the formula:

$$A = \frac{1}{2} \sum_{i=1}^{n} x_i y_{i+1} - x_{i+1} y_i$$

where n is the number of vertices in the polygon, and the indices are taken modulo n to ensure that the last vertex wraps around to the first.

The coordinates of the centroid are then given by:

$$C_x = \frac{1}{6A} \sum_{i=1}^{n} (x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$
$$C_y = \frac{1}{6A} \sum_{i=1}^{n} (y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

With the centroid computed, we then connect the centroid to each of the vertices of the polygon. This results in a series of triangles that fan out from the centroid, providing a triangulated representation of the cluster.

Occlusion Level Estimation

With the triangles constructed, we can now estimate the occlusion level of the mesh. By applying the ray tracing method described in Section 3.1, we can determine how much of each triangle is visible from a given viewpoint. This information is invaluable for various applications, including rendering, simulation, and analysis, as it provides insights into the visibility and shading characteristics of the mesh in a scene.

3.3 Ray Tracing Based Point Cloud Scanning

Ray tracing is a powerful technique that simulates the path of rays as they traverse through a scene. In the context of point cloud data, ray tracing can be employed to identify visible and occluded regions within the cloud. By scanning the ground truth point cloud using a ray tracing based method, we can derive a modified point cloud where the retained points represent the visible areas, while the removed points indicate occluded regions.

3.3.1 Spherical Light Source

To simulate a light source that emits rays in all directions, we use a spherical model. Points are uniformly sampled on the surface of this sphere, with each point representing a potential 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 uniform sampling on the sphere is achieved using random sampling in the azimuthal angle θ and the polar angle ϕ. The azimuthal angle θ is uniformly sampled in the range $[0, 2\pi]$, while the polar angle ϕ is sampled such that its cosine is uniformly distributed in the range $[-1, 1]$. This ensures a uniform distribution of points over the sphere's surface.

Given a center point of the sphere and a predefined radius r, the coordinates (x, y, z) of a sampled point on the sphere are computed as:

$$x = x_{\text{center}} + r \sin(\phi) \cos(\theta)$$
$$y = y_{\text{center}} + r \sin(\phi) \sin(\theta)$$
$$z = z_{\text{center}} + r \cos(\phi)$$

This process is repeated for the desired number of samples, resulting in a set of points uniformly distributed on the surface of the sphere. These points can then be used as origins for rays to simulate light emission in all directions.

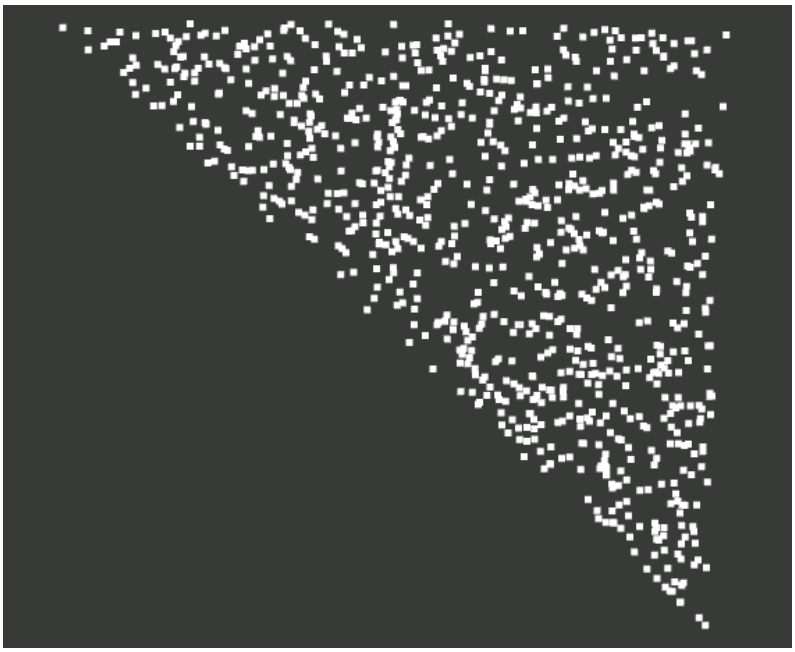


Figure 3.5: Uniform Sample Sphere

3.3.2 Ray Point Intersection

In our model, 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.

To determine if a ray intersects with a sphere, we first normalize the direction vector of the ray. Then, we compute the vector $\mathbf{L}$ from the ray's origin to the center of the sphere. If the squared magnitude of $\mathbf{L}$ is less than the squared radius of the sphere, it indicates that the ray's origin is inside the sphere, and thus, an intersection exists.

Next, we compute the projection t_{ca} of $\mathbf{L}$ onto the ray's direction. If t_{ca} is negative, it means the sphere is behind the ray's origin, and no intersection occurs.

We then compute the squared perpendicular distance d^2 from the center of the sphere to the ray. If d^2 exceeds the squared radius of the sphere, the ray does not intersect the sphere.

Mathematically, the steps can be summarized as:

$$\begin{aligned} \mathbf{L} &= \text{center of sphere} - \text{ray origin} \\ \text{originDistance}^2 &= \mathbf{L} \cdot \mathbf{L} \\ t_{ca} &= \mathbf{L} \cdot \text{normalized ray direction} \\ d^2 &= \text{originDistance}^2 - t_{ca}^2 \end{aligned}$$

If the ray's origin is not inside the sphere, t_{ca} is positive, and d^2 is less than the squared radius of the sphere, then the ray intersects the sphere.

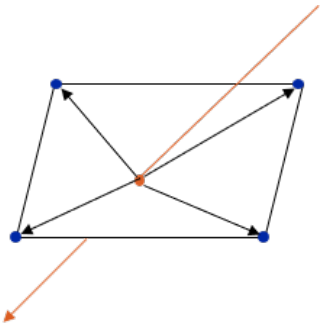


Figure 3.6: Ray Intersect Point

3.3.3 Ray Openings Intersection

Openings or gaps in the scene, such as windows or doorways, should not contribute to occlusion. 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's 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 this plane 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.

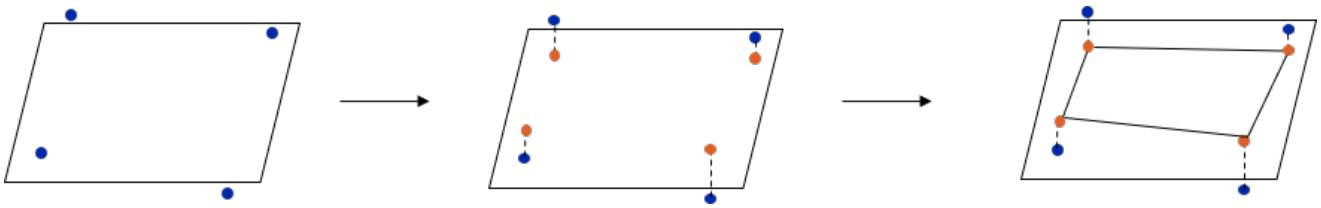


Figure 3.7: Estimate Polygon from Points

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.

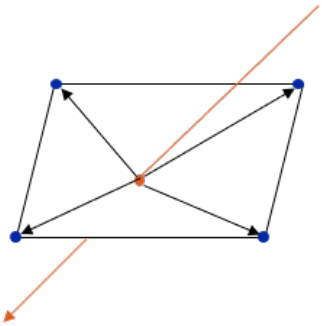


Figure 3.8: Ray Intersect Polygon

3.3.4 Ray Tracing Based Occlusion Computation

To assess occlusion within the point cloud, we employ a ray tracing method where rays emanate from various points on a spherical light source. The degree of occlusion for each point in the cloud is deduced from the count of rays that neither intersect any point in the cloud nor any predefined openings in the scene.

Within this framework, we introduce the notion of "occlusion rays". These rays, once cast from the light source, bypass any intersections with the point cloud or the designated scene openings.

To provide a quantitative measure of occlusion, we calculate the ratio of occlusion rays to the overall number of rays cast. This metric, termed the "occlusion ray ratio", offers an insight into the occlusion level within the point cloud. A higher ratio signifies a pronounced occlusion presence.

3.3.5 Exterior Structure Based Occlusion Computation

Exterior structures, such as walls, ceilings, and floors, frequently experience occlusion, primarily due to obstructions posed by interior objects. We operate under the assumption that a scene with a higher density of interior objects will result in increased occlusion of exterior structures. While interior objects themselves can be occluded, pinpointing such occlusions is challenging. Consequently, we use the occlusion level of exterior structures as a proxy for the scene's overall occlusion level. Our aim is to discern a correlation between the occlusion level and the ratio of interior to exterior objects, enhancing the reliability of our results.

Exterior Structure Extraction

We directly extract exterior structures from the ground truth point cloud. In our scenario, this involves isolating walls, ceilings, and floors.

Ratio Computation

The next step involves computing the ratio of interior points relative to exterior points.

Ray Tracing Based Occlusion Computation

Echoing the methodology delineated in Section 3.3.4, rays are generated from various points on the spherical light source. The occlusion level for each point is then determined based on the number of rays that avoid intersections with any point on the exterior structure.

3.4 Evaluate Performance with Metrics

In the realm of semantic segmentation, evaluating the performance of a model 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. By using these metrics, researchers and practitioners can gauge the strengths and weaknesses of their models, facilitating improvements and ensuring optimal performance. In this section, we discuss the 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 dataset. Due to discrepancies between the ground truth point cloud dataset, denoted as [S3dis], and the dataset used for the pre-trained model, we have defined the following semantic classes for our segmentation task:

Semantic Class	Class ID
Wall	0
Floor	1
Ceiling	1
Chair	4
Sofa	5
Table	6
Door	7
Window	8
Bookcase	9
Beam	20
Board	21
Clutter	25
Column	26

Table 3.1: Mapping of semantic classes to their respective class IDs.

It's worth noting that certain classes, such as the ceiling, have been mapped to the same class ID as other classes (in this case, the floor). This decision was made based on the similarities between the datasets and to ensure a more streamlined segmentation process.

3.4.2 Metrics

We use the following metrics to evaluate the performance of our model:

IoU

Inttersection over Union, also known as the Jaccard Index, IoU measures the overlap between the predicted segmentation and the ground truth. It's calculated as the area of overlap divided by the area of union of the two sets.

$$IoU = \frac{\text{Intersection}}{\text{Union}} \tag{3.4}$$

A higher IoU indicates better segmentation accuracy.

F1 Score

The F1 Score is the harmonic mean of precision and recall. It provides a balance between the two, ensuring that both false positives and false negatives are taken into account.

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.5}$$

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

Accuracy

Accuracy measures the proportion of correctly predicted segmentation pixels to the total number of pixels.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{3.6}$$

While a useful metric, accuracy can sometimes be misleading, especially if the classes are imbalanced.

Recall

Recall measures the proportion of actual positives that were correctly identified.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{3.7}$$

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

Precision

Precision measures the proportion of positive identifications that were actually correct.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{3.8}$$

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

4 Implementation

In this chapter, we delve deeper into the nuts and bolts of our solution as presented in Chapter 3, shedding light on the tools and methodologies employed and highlighting their critical roles in our system.

4.1 PCL Serves as Main Component in Computational Pipeline

The Point Cloud Library (PCL) stands as the cornerstone of our implementation. We predominantly rely on PCL for an array of vital tasks, ranging from reading and segmenting point cloud data to evaluating its occlusion levels.

Parallel to PCL, the Eigen library emerges as an invaluable asset. It is our preferred choice for handling intricate mathematical operations that are indispensable to our pipeline. To ensure a comprehensive and cohesive system, we also incorporate auxiliary libraries, such as JsonCpp and Websocketpp, which furnish our framework with supplemental functionalities.

4.1.1 Uniform Sampling

In many computer graphics and simulation applications, it's often necessary to sample points uniformly from geometric shapes like spheres or triangles. To achieve this, we often use random number generators to create random points that fall within the shape.

We'll start by understanding the methodology for each shape, followed by C++ code samples.

Spherical Light Source

In order to simulate a spherical light source or any uniformly distributed sampling on a sphere, the 'UniformSamplingSphere' method is introduced. The sampling process commences by initializing a random number generator with a uniform distribution:

Listing 4.1: Spherical Sampling Initialization

```
static std::default_random_engine generator;
static std::uniform_real_distribution<double> distribution(0.0, 1.0);
```

With the random number generator established, azimuthal and polar angles are computed for each sample. These angles are then transformed into Cartesian coordinates, centered around a provided point:

Listing 4.2: Sampling on a Sphere

```
for (size_t i = 0; i < num_samples; ++i) {
  double theta = 2 * M_PI * distribution(generator);  // Azimuthal angle
  double phi = acos(2 * distribution(generator) - 1); // Polar angle

  pcl::PointXYZ sample;
  sample.x = center.x + radius * sin(phi) * cos(theta);
  sample.y = center.y + radius * sin(phi) * sin(theta);
  sample.z = center.z + radius * cos(phi);

  samples.push_back(sample);
}
```

This method returns a vector of sample points surrounding the given center.

Triangle

For uniform sampling on a triangle, the primary idea revolves around barycentric coordinates. The number of samples per triangle is proportionate to its area, which is multiplied by a pre-defined '$samples_per_unit_area$' to decide the sample count:

Listing 4.3: Triangle Sampling Initialization

```
    for (auto& tri : t_triangles) {
        double area = calculateTriangleArea(tri.second);
        size_t num_samples = static_cast<size_t>(area * samples_per_unit_area);
```

If only one sample is required, it's directly chosen as the triangle's center:

Listing 4.4: Triangle Center Sampling

```
if(num_samples == 1) {
  Sample sample;
  sample.point = tri.second.center;
  t_samples[sample.index] = sample;
}
```

For a general case, random barycentric coordinates are generated using two random numbers, r_1 and r_2:

Listing 4.5: Barycentric Coordinate Sampling

```
for (size_t i = 0; i < num_samples; ++i) {
  double r1 = distribution(generator);
  double r2 = distribution(generator);

  double sqrtR1 = std::sqrt(r1);
  double alpha = 1 - sqrtR1;
  double beta = r2 * sqrtR1;

  Eigen::Vector3d sample_point = alpha * tri.second.v1 + beta * tri.second.v2 + (1 -
      alpha - beta) * tri.second.v3;

  Sample sample;
  sample.point = sample_point;
  t_samples[sample.index] = sample;
}
```

This method ensures that the samples are uniformly distributed across the surface of the triangle, which can be especially crucial for graphics, simulations, or light transport calculations.

4.1.2 Mesh Estimation

Estimating the mesh necessitates an initial handling of point cloud data. As such, leveraging the Point Cloud Library (PCL) becomes an intuitive decision, and we seamlessly integrate it into the core of our computational pipeline.

Region Growing

To segment the point cloud into discernable clusters, we employ PCL's normal estimation and region-growing algorithm. Achieving an accurate segmentation hinges upon meticulous parameter tuning.

The first step in this process involves setting up a k-d tree, which allows for efficient neighbor searches when estimating normals for the point cloud:

Listing 4.6: Setting up k-d tree and Normals Estimation

```
pcl::search::KdTree<pcl::PointXYZ>::Ptr tree(new pcl::search::KdTree<pcl::PointXYZ>());
pcl::PointCloud<pcl::Normal>::Ptr normals(new pcl::PointCloud<pcl::Normal>);
pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal> normal_estimation;
normal_estimation.setInputCloud(cloud);
normal_estimation.setSearchMethod(tree);
normal_estimation.setKSearch(50);
normal_estimation.compute(*normals);
```

Subsequently, we proceed with region growing, which is contingent on the previously computed normals. Key parameters like the minimum and maximum cluster sizes, number of neighbors, and thresholds for smoothness and curvature play pivotal roles in determining the quality and granularity of segmentation:

Listing 4.7: Region Growing Parameters Setting and Execution

```
pcl::RegionGrowing<pcl::PointXYZ, pcl::Normal> reg;
reg.setMinClusterSize(100);
reg.setMaxClusterSize(100000);
reg.setSearchMethod(tree);
reg.setNumberOfNeighbours(40);
reg.setInputCloud(cloud);
reg.setInputNormals(normals);
reg.setSmoothnessThreshold(4.0 / 180.0 * M_PI);
reg.setCurvatureThreshold(1.0);

reg.extract(rg_clusters);
```

Through this approach, the point cloud data is effectively fragmented into coherent clusters, forming a solid foundation for subsequent mesh estimation and reconstruction steps.

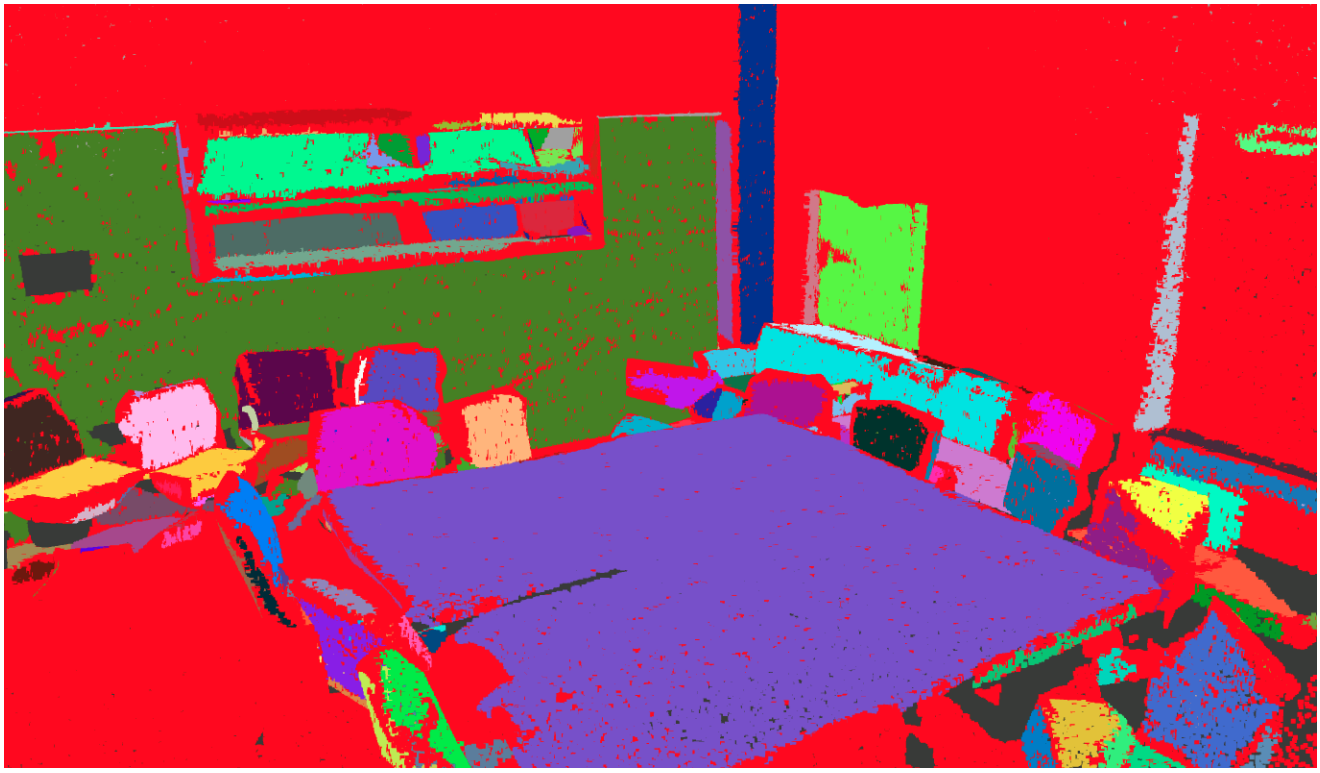


Figure 4.1: Region Growing Segmented Interior Scene

Generate Triangles from Clusters

The generation of triangles from clusters is a nuanced process that harnesses the capabilities of PCL. We navigate through this process in stages: fitting clusters to planes, projecting points, and computing the convex hull of each cluster.

Plane Estimation: The initial step revolves around fitting a plane to each cluster's point data. For this, we rely on the RANSAC algorithm embedded within PCL. The function `computePlaneCoefficients` captures this essence:

Listing 4.8: Estimating Plane Coefficients

```
pcl::ModelCoefficients::Ptr coefficients = computePlaneCoefficients(points);
```

In this snippet, PCL's `SACSegmentation` is configured to detect planar models. If successful, it retrieves the plane's coefficients, essential for subsequent operations.

Projection onto Plane: Armed 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. The function `estimatePolygon` accomplishes this feat:

Listing 4.9: Projecting Points onto Plane

```
pcl::PointCloud<pcl::PointXYZ>::Ptr polygon_cloud = estimatePolygon(points, coefficients
    );
```

Here, PCL's `ProjectInliers` projects the points onto the detected plane, thereby creating a co-planar point cloud, which we refer to as '$cloud_projected$'.

Convex Hull Computation: With our points now sharing a planar space, we strive to compute the convex hull of each cluster. This results in a 2D polygon within a 3D space, effectively enveloping all the cluster points. The usage of PCL's `ConvexHull` class, as seen in the 'estimatePolygon' function, is pivotal in this stage.

Triangle Generation: On establishing the polygon, our final venture is to segment it into triangles. This process involves using the polygon's centroid, combined with successive boundary points, to define each triangle. In the `generateTriangleFromCluster` function, every triangle's geometric attributes and supplementary properties, like its area, are meticulously computed and stored.

Listing 4.10: Triangle Generation from Polygons

```
void Occlusion::generateTriangleFromCluster() { ... }
```

The `computeCentroid` function calculates the centroid of the polygon by traversing through its points. Subsequently, triangles are generated with the centroid and polygon vertices, with each triangle's center and area meticulously computed.

Collectively, this workflow ensures a methodical transformation of raw clusters into structured triangular meshes, laying the foundation for advanced analysis and visualization.

4.1.3 Ray Tracing 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.

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 recorded, which makes the system looks like a database. We can query the database to get all the information we need.

Database-like Structure

Listing 4.11: Data Structures

```
struct Intersection {
    size_t index;
    size_t triangle_index;
    size_t ray_index;
    Eigen::Vector3d point;
    double distance_to_look_at_point;
    double distance_to_origin;
    bool is_first_hit;
};
struct Ray {
    size_t index;
    size_t source_triangle_index;
    size_t source_sample_index;
    size_t first_hit_intersection_idx;
    Eigen::Vector3d origin;
    Eigen::Vector3d look_at_point;
    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;
    Eigen::Vector3d center;
    double area;
    std::vector<size_t> sample_idx;
    std::vector<size_t> intersection_idx;
    std::vector<size_t> ray_idx;
};
```

4.1.4 Octree Partitioning

In the context of our pipeline, the sheer magnitude of input data can be quite overwhelming. It's not uncommon for us to deal with data sets comprising millions of points and thousands of intersecting rays. 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, kdTree, and octree are used for this purpose. Leveraging the existing 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 distinct 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 safeguards the bounding box encompassing the entire tree. Navigating this structure, if a ray originates inside the root node, we examine its intersection with the root's child nodes. Typically, if our tree depth surpasses two, this should return a branch node. The procedure recursively continues until we pinpoint a leaf node, at which point we verify the ray's intersection with the data housed in the leaf. If there's a data intersection, the intersection point is duly returned.

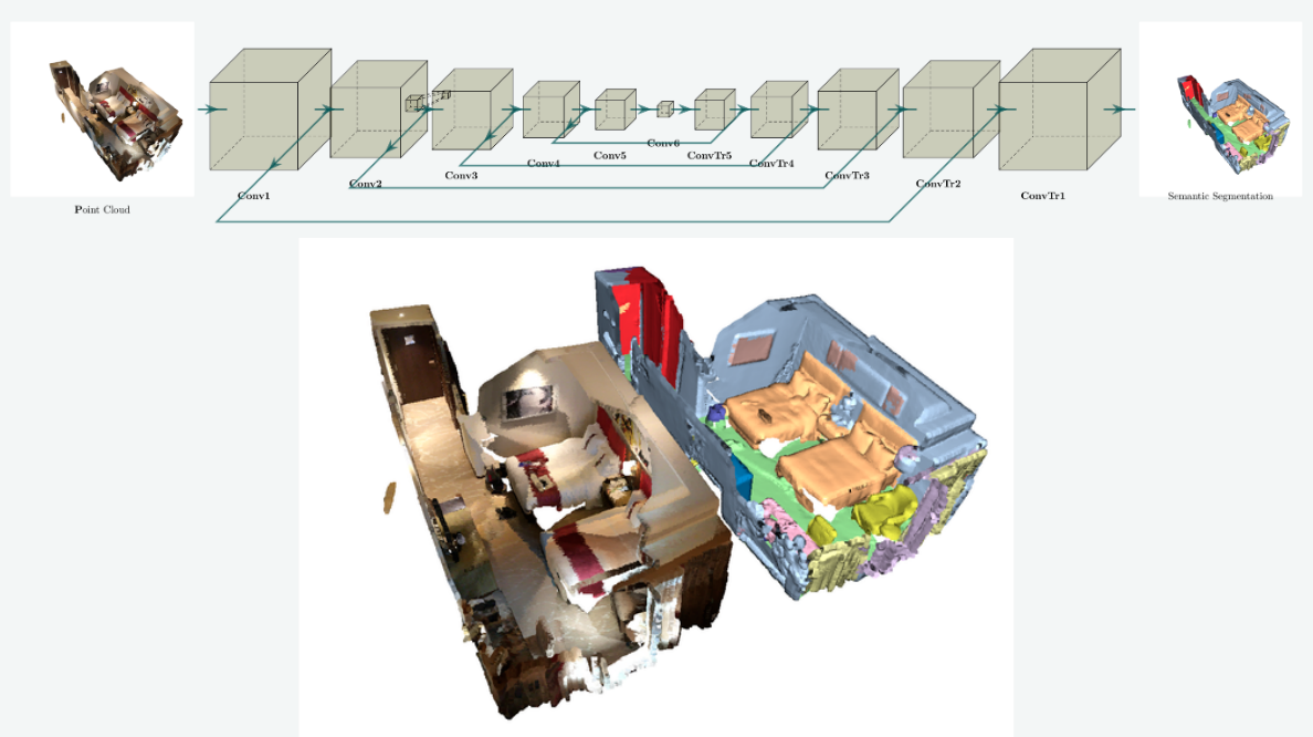


Figure 4.2: Flowchart of the Octree Partitioning Process

Octree Data Structure

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

Listing 4.12: 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;
};
```

Mesh

Harnessing PCL's octree class, we facilitate mesh partitioning. However, this requires a preliminary step: the extraction of a point which epitomizes the triangle. To achieve this, we use the triangle's center of gravity as its representative point. By gathering these representative points, an octree can be constructed.

PCL offers versatility with traversal methods – we have the liberty to opt for either a depth-first or breadth-first approach. Our actual implementation favors the depth-first traversal. The ensuing steps outline our tree-building process:

Build Depth 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 min_pt and max_pt) acts as the key, while the node's depth is the associated value. Post traversal, the map elucidates which sizes correlate to specific depths.

Build Depth Node Map: For individual nodes, a quick reference to the size-depth map reveals their respective depth. We then proceed to compile a map to document nodes within each depth level. Here, the depth is the key, and a vector, comprising node indices at that depth, forms the value. The order of these nodes in the vector mirrors the depth-first traversal sequence.

Build Connections Between Nodes: Establishing connections between parent and child nodes is paramount. This phase involves a traversal of the depth-node map, starting from the root and working through to the penultimate leaf node level. Certain nuances are worth highlighting:

- The root node stands alone without a parent, but nodes at level 1 unanimously share the root as their parent.
- For branch nodes, the traversal of their child nodes occurs post the traversal of the left sibling's children but prior to the right sibling's children. This traversal pattern aids in effectively mapping the parent-child relationships.

Following is a simplistic 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.

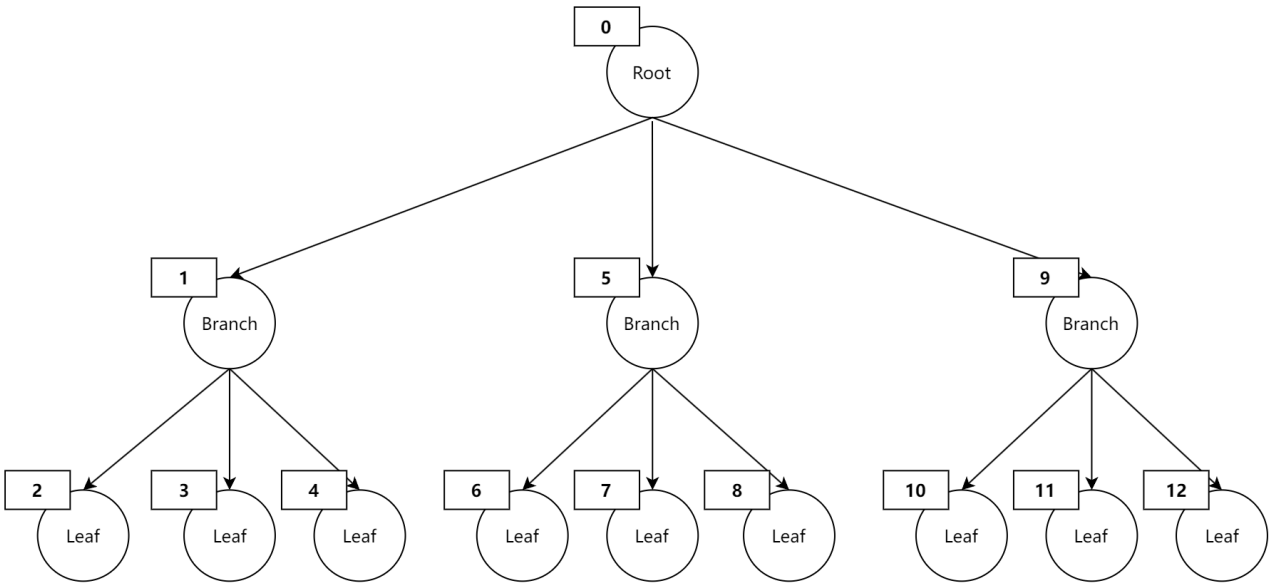


Figure 4.3: Octree Structure

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 linkage of children to their parent nodes.

Compute Bounding Box: With the connections established, we can now compute the bounding box for each node from leaf level. Although the bounding box for points of a leaf node is readily available, here we have to consider triangles. 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 can iterate over all triangles. The min_pt and max_pt 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.

Point Cloud

For partitioning the point cloud, most steps are the same as mentioned in Section 4.1.4. The only difference is that we don't need to compute bounding box based on triangle for each node, bounding box offered by PCL should be enough for this part.

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, which calculates the intersection of the ray with the planes that define the bounding box.

The method is materialized in the 'rayIntersectOctreeNode' function. The procedure commences by verifying if the ray's origin is nestled within the bounding box by leveraging the 'min_pt_triangle' and 'max_pt_triangle' attributes of the octree node. If the origin is confirmed to be within the bounds, the function instantly returns true.

Nevertheless, if the origin is exterior, the function gauges potential intersection points (or t-values) for each of the x, y, and z axes. It inspects the intersection of the ray with the 'slabs' (or bounding faces) of the bounding box. The terminal phase ensures that these intersection points lie within a consistent range across all three dimensions, thereby validating the intersection of the ray with the bounding box.

A distilled representation of the function is provided below:

Listing 4.13: Simplified Ray Intersect Bounding Box

```
bool Occlusion::rayIntersectOctreeNode(Ray& ray, OctreeNode& node) {
    ...
    // Check if ray's origin is within bounding box
    if (...) return true;

    // Compute intersection points (t-values) for x, y, and z dimensions
    double tmin, tmax, tymin, tymax, tzmin, tzmax;
    ...
    // Check intersection with x slabs
    ...
    // Check intersection with y slabs
    ...
    // Check intersection with z slabs
    ...

    // Ensure intersection points are within a valid range for all dimensions
    if ((tmin > tzmax) || (tzmin > tmax)) return false;

    return true;
}
```

It is pivotal to highlight the adoption of a threshold value (in this context, $1e-8$) to ascertain numerical stability. When the ray's directional component for a distinct axis approaches zero, the ray becomes nearly parallel to the planes of the bounding box on that specific axis. This peculiarity is addressed by the function by attributing 't-values' to positive or negative infinity as required.

This algorithm is optimized for efficiency and enables swift determinations of ray-bounding box intersections, a property that's indispensable for maintaining high performance in graphical applications.

4.1.5 Other Libraries Serve as Supporting 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 4.14: Json Configuration File

```
"occlusion": {
  "mesh": {
    "path": "../files/conf_m.obj",
    "pattern": 4,
    "octree_resolution": 64.0,
    "enable_acceleration": true,
    "samples_per_unit_area": 0.01
  },

  "point_cloud": {
    "path": "../files/conf1.pcd",
    "num_rays_per_vp": 30000,
    "num_vp": 9,
    "point_radius": 0.01,
    "polygon_path": "../files/polygon_conf.txt",
    "octree_resolution": 0.5
  }
},
```

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 4.15: 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) {
  try {
    s.send(hdl, msg->get_payload(), msg->get_opcode());
    std::string payload = msg->get_payload();
    if (payload.substr(0, 3) == "-i=") {
```

```
7         data_holder.setFileName(payload.substr(3, payload.length()));
8         data_holder.setInputPath("../files/" + payload.substr(3, payload.length()));
9       }
10      ...
11    }
12  int main{
13    print_server.set_message_handler([&print_server, &data_holder](websocketpp::
          connection_hdl hdl, server::message_ptr msg) {
14              on_message(print_server, hdl, msg, data_holder);
15          });
16    print_server.init_asio();
17    print_server.listen(8080);
18    print_server.start_accept();
19    print_server.run();
20    ...
21  }
```

4.2 Web-Based User Interface

Our software is elegantly encapsulated within the framework of a web application. The PCL-based backend is the computational powerhouse, whereas the frontend enhances user interaction. Detailed structures of the backend were previously introduced in Section 4.1. In this section, our focus shifts to the frontend components and how they seamlessly communicate with the backend.

The user interface's visual representation is captured in Figure 4.4.

Figure 4.4: Web-Based User Interface

4.2.1 Role of Three.js in Visualization

Three.js emerges as a versatile, cross-browser JavaScript library, complemented with an API designed to craft and exhibit animated 3D computer graphics directly in a web browser. This dynamic library provides the interactive visualization pivotal for our application.

4.2.2 Chosen Web Technology Stack

Given our choice of three.js for visualization, an ensemble of complementary web technologies is integrated to shape our frontend. The roster for our web tech stack is 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.

Collectively, these technologies synergize to produce a sleek, responsive, and visually captivating frontend, ensuring users are treated to an intuitive and immersive browsing experience.

4.2.3 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.

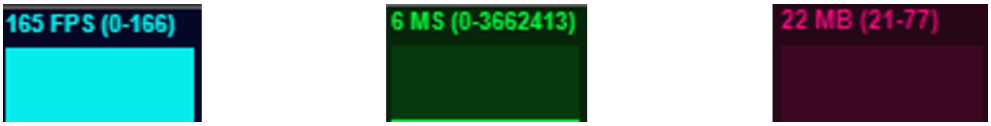


Figure 4.5: Stats Panel

GUI

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

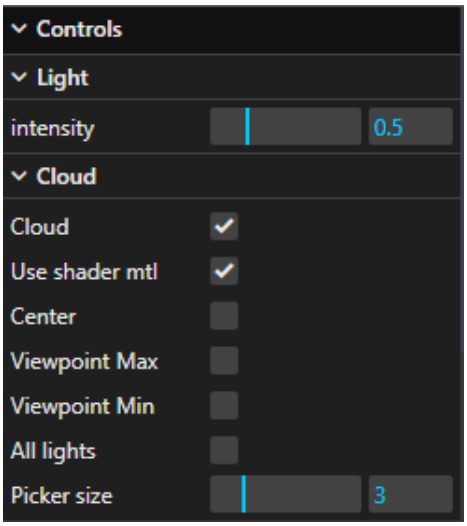


Figure 4.6: GUI

- **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's bliking between white and black.
- **Center** - Center viewpoint.
- **Viewpoint Max** - Max viewpoint.
- **Viewpoint Min** - Min viewpoint.
- **All lights** - Show all light sources in the scene.
- **Picker Size** - Size of the point picker.

Buttons

Below are the buttons on the user interface.

- **Original Cloud** - Upload and visualize original point cloud.
- **Segmented Cloud** - Upload and visualize segmented point cloud.
- **Semantic Cloud** - Upload and visualize semantic point cloud.
- **Mesh** - Upload and visualize mesh.
- **Compute Occlusion** - Compute occlusion of the scene.
- **Extract Polygons** - Extract polygons from the scene.
- **Evaluate** - Evaluate the performance of segmentation.

4.2.4 Software Structure

Flowchart

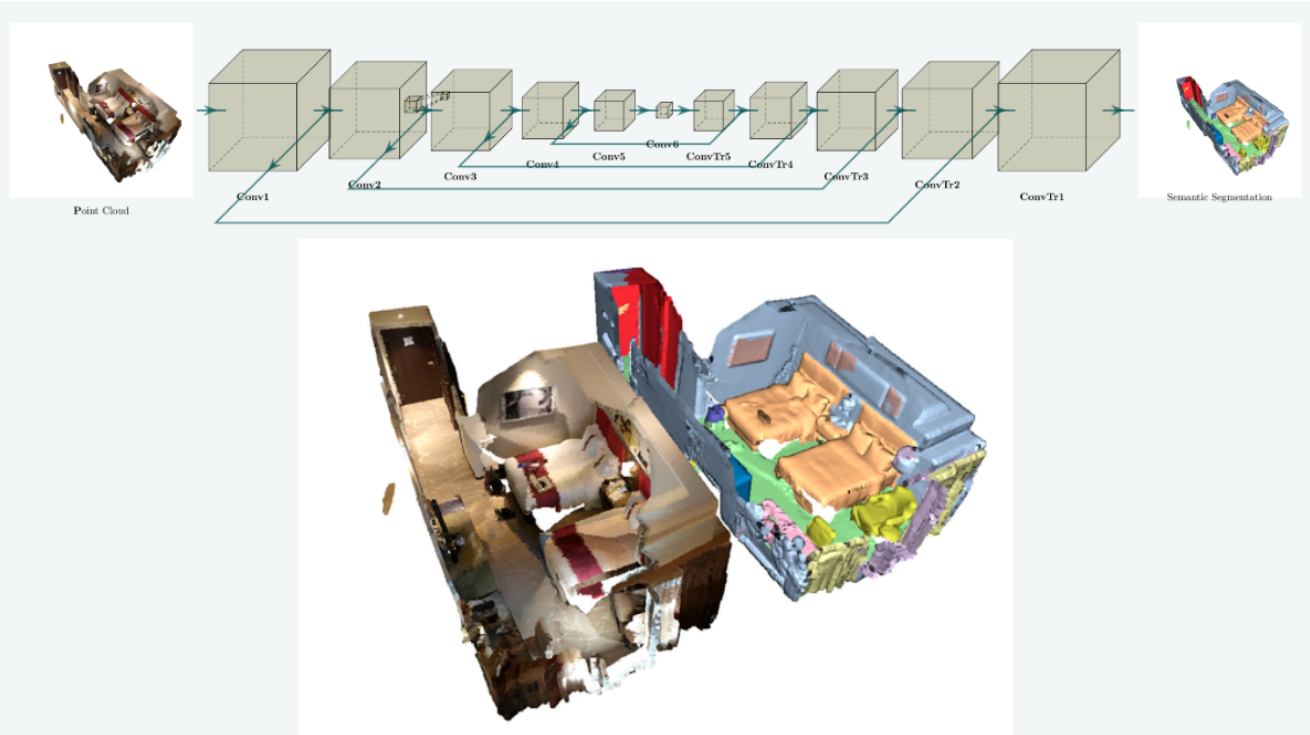


Figure 4.7: Flowchart of the Software

Class Diagram

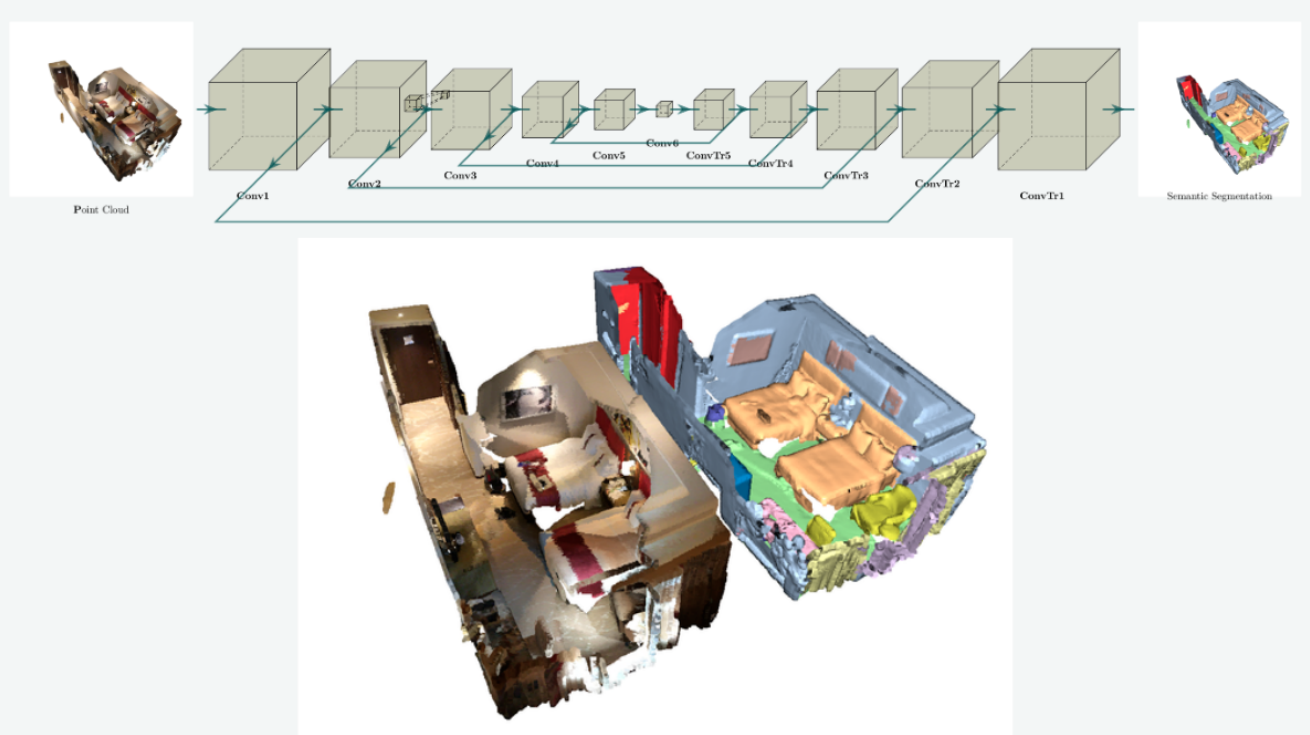


Figure 4.8: Class Diagram of the Software

Sequence Diagram

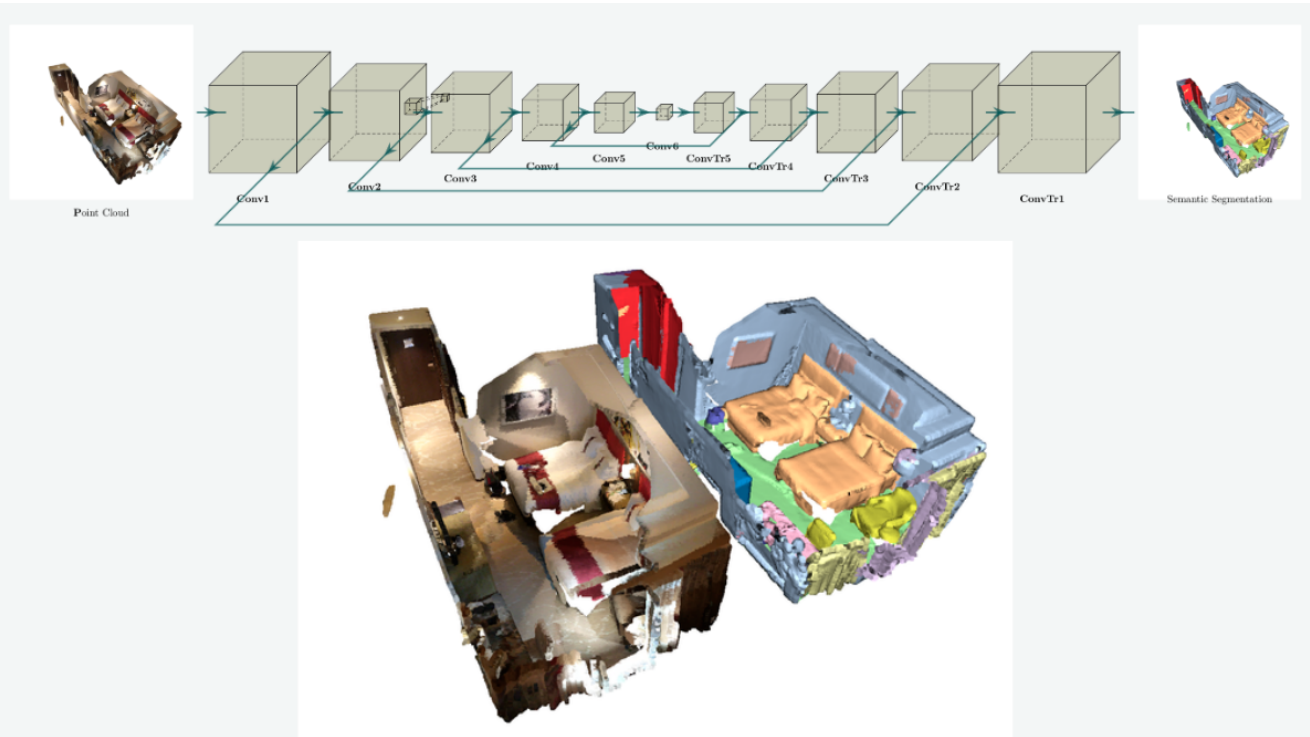


Figure 4.9: Sequence Diagram of the Software

4.2.5 Command Line Only Mode

This project is designated to be used in a web application. However, we also provide a command line only mode for users who want to use this project in a command line environment. In this mode, we use a simple command line interface to interact with the backend.

Arguments and Corresponding Functionality

- **-moc** - Mesh Occlusion Computation.
- **-rgoc** - Region Growing Occlusion Computation.
- **-poc** - Point Cloud Occlusion Computation.
- **-fscan** - Scan cloud with fixed viewpoint.
- **-recon** - Reconstruct point cloud from .txt file.
- **-eval** - Evaluate performance of segmentation.

5 Experimental Results

5.1 Mesh Based Occlusion Level

5.1.1 Ground Truth Mesh

Setup We prepare a ground truth mesh which shows a conference room[McGuire Archives] as our input. 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.

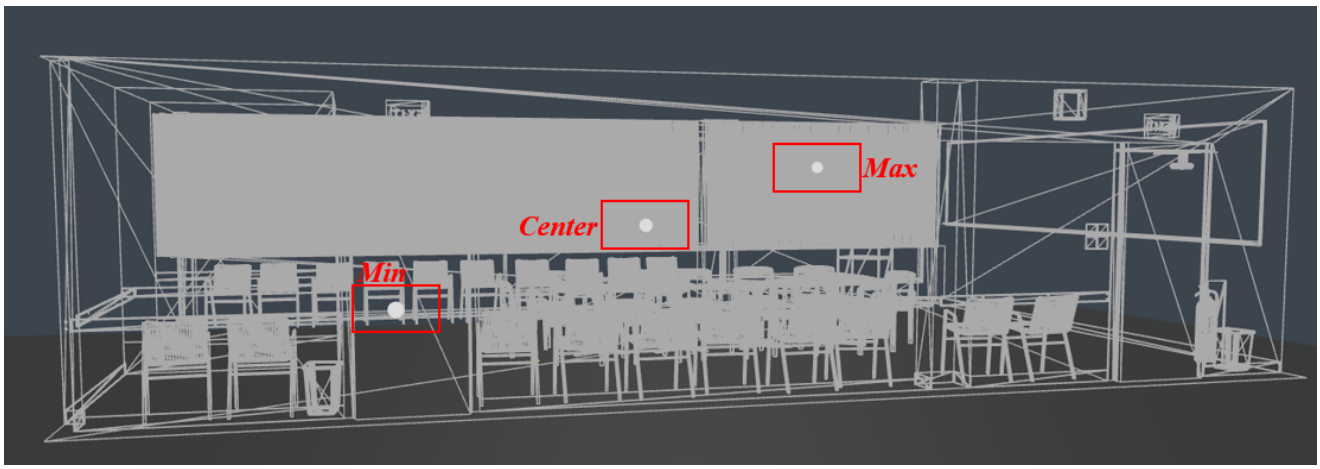


Figure 5.1: Mesh with Viewpoints

We will conduct this experiment on different pattern of viewpoints as shown in Figure 5.2. Each time we will apply different pattern of viewpoints on the same mesh scene. There are 124277 triangles in this scene, to simplify the computation, we set sample per unit area to be 0.01, which result in 125752 samples as well as rays in total.

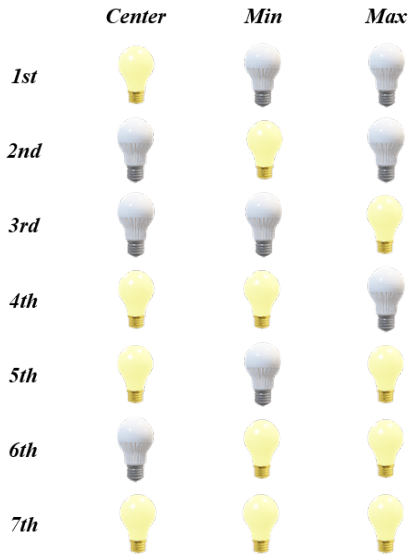


Figure 5.2: Pattern of Viewpoints

Results Our results are shown in Table 5.1. We can see that the occlusion level decreases as the number of viewpoints increases. This is because more viewpoints can cover more area of the scene, which results in less occlusion.

Experiment No.	Occlusion Level
1st	0.493
2nd	—
3rd	—
4th	0.376
5th	—
6th	—
7th	0.287

Table 5.1: Result of Each Experiment

5.1.2 Estimated Mesh from Point Cloud

Setup

Results

5.2 Segmentation Performance Evaluation of Point Cloud

Setup

Results

6 Conclusion and Discussion

The last section puts the results in perspective, discussing it in relationship to other related work in one or maximum two pages. Indicate possible (side-)effects and eventual limitations due to the evaluation, but try to keep yourself short and clear. Give a short discussion about your results where you focus on what your findings mean. E.g., show how your results and interpretations agree with the original problem question and with other published work or if there are any other practical applications for your work.

State the *take home message* of the paper that the reader should remember and provide an outlook on possible future work that extends the given solution or fixes specific limitations. Close with a brief description (that is different from the Abstract) of the proposed solution.

Conclusion From the results we get from last section, we ca nfind correlation bewtween level of occlusion and segmentation performance.

Limitation However, there're also many limitations on our work. We cannot assess the occlusion level of a scene very accurately.

Future Work Some future work can be done to improve our work.

7 Acknowledgements

Acknowledge any data and code sources you used or for help you received.

Document, Writing and Formatting Guidelines

This part of the document uses non-numbered chapter and section headings as they are not part of a regular report structure. In a regular report, `chapters`, `sections` and `subsections` should normally be numbered. Note the use of comment lines and spacings to give more structure to the ASCII text LaTeX document.

In this appendix like part we discuss the structure and formatting rules and guidelines to follow for a written project, research paper, Bachelor or Master thesis report.

Text

Overall Strategies

The appearance, clarity and organizational structure of a paper is as important as its technical content, but the technical content must be there beforehand. The presentation alone, however, can make the difference between a mediocre and great publication. Exploit all suitable mechanical rules that can always be applied and optimized independently of the technical part.

The paper has to be convincing even to the adverse reader, i.e. a critical reviewer evaluating your work. Think of being a reviewer yourself, not really knowing your domain and possibly not specifically interested in your work. All information must be crystal clear to a non-expert reader, and all terms and concepts must be properly introduced in a logical order. Put yourself in the position of reading that topic and your work for the very first time, with the goal of having to reproduce it afterwards. The presentation must be flawless and the length of the paper must match the amount of content.

Your report must present your work and solution, best within a convincing story about a difficult problem challenge and an important application domain. The text has to encompass and sell your work. Build up and identify the key challenges in the introduction and problem description. Show clearly how you solved exactly this very important problem in a great and unique novel way.

Group your ideas and concepts hierarchically into sections, subsections and even paragraphs. Strictly introduce general and common ideas, concepts and techniques before expanding further on them. Use the concepts of *repetition* and *parallelism* in your text and structure. The main concept of repetition is to:

> Introduce what you are going to tell them – then tell them in detail – and finally review what you told them.

This approach of repetition is applied on all levels of a report, overall document, sections, subsections and paragraph, always in an appropriate level of abstraction or detail. Example levels:

Document The introduction briefly describes the main problem, previews your contribution and summarizes the main results. The main technical sections describe your approach in more detail and the experiments show the achieved results. Finally the conclusion, discussion and future work section(s) summarize your contributions.

Section In the first paragraph(s) you introduce the topic or aspect that this section covers, followed by the (technical) details, and the last paragraph typically wraps it up, or leads to the next section.

Paragraph The first sentence of a paragraph leads into the main *message* of this paragraph, and the last sentence concludes it or leads over to the next paragraph.

The concept of parallelism means to apply the same strategy or structuring to different parts. E.g. for each technical problem question or topic covered in one section, first introduce the problem definition and then describe the solution subsequently, possibly in subsections and paragraphs. Or for each type or version of experimental results, or for each data set, describe the data, method parameters, measured variables, report and discuss the results.

It is important to get your text exactly clear to the reader and to avoid the impression that something has been left out or that your contribution is not that significant. But do not over-claim, and more importantly do not under-claim either.

Writting and Structure

Writing is difficult work and usually takes more time than expected. It's beneficial to formalize and write down your progress as early as possible. Generally *you cannot really be sure that you know something until you are able to explain it* well in writing. Hence conveying ideas exactly but in a concise and compact manner is very important and a key to successful writing. Preciseness and compactness are key to be able to describe a large amount of work and results that you have.

Your text must be smooth, forming a clear and logical order of your thoughts and arguments. Use *parallelism* to introduce a set of topics, questions or issues and then elaborate on them subsequently in sections and paragraphs.

Use *repetition*, e.g. introduce the problem, show how to solve it and review the benefits. Do not assume that the reader still remembers what was mentioned only as a passing comment three pages back, use repetition and parallelism.

Do not abruptly jump topics but motivate topic changes. If the topics of text parts change too much, then divide them into subsections or paragraphs. If between paragraphs there are major changes (sequence of different concepts etc.), try to use inline paragraph headings for easy navigation and orientation. One way to integrate such paragraph sequences is to write an introductory paragraph at the beginning of the section and use parallelism.

Sections and Paragraphs

Use `sections`, `subsections` and `paragraphs` to structure your ideas and content into meaningful parts, and make a clear and meaningful order of them. Each section, subsection or paragraph must cover a clearly delineated topic or idea.

Every section must first introduce to the reader what to expect and then tell the details, following the principles of repetition and parallelism. An introductory paragraph is also a good approach to fill the space between the section heading and the first sub heading if subsections are used. The last paragraph of a section can summarize the concepts and lead over to the next topic or section.

Each paragraph should have one clear single message, there should only be one consistent topic or idea what the paragraph is about. The first paragraph of a section should clearly lead into the main topic of that section, and the first line of a paragraph should state or at least clearly lead into the main message of that paragraph.

Every single sentence should be fully comprehensible in its context, taking only minimal preliminary knowledge of previous paragraphs into account.

Wording and Postprocessing

Use single tenses (i.e. the present or present perfect) as much as possible when describing your design, technical or algorithmic solutions. Generally use present perfect for describing implementations and results which were completed, as this implies something that happened as part of this work in the past but continues to be valid in the present as a result of this paper.

"I" versus "we": For a personal opinion, or your specific personal contribution, you can use the first person. In a personal thesis report one can use "I" as this one's own contribution and text.

For most situations a neutral form should be used, passive voice or third person, but be careful to avoid the interpretation of "passive voice" as "someone else did it, and it is not our contribution", e.g. "This mind-boggling observation was made." vs. "We made this mind-boggling observation.".

Read through the text, spell check, and check the text with a grammar tool as well. Obvious errors are unacceptable. Try to take the role of a reviewer or evaluator: Reading your paper or report costs that person significant time, so question everything. The reviewer may not be forgiving if something is not clear. Be the devil's advocate!

Do not use abbreviations (don't -¿ do not).

Formatting and Typesetting

General Rules

The main text area should have a margin of about 2cm on both sides, and the main text should have a top and bottom margin of about 3cm each.

Section and subsection headings have nested numbering and are typeset in sans-serif bold font type, e.g. Helvetica or Arial. Sizes range from 18pt for main sections down to at least one point larger than the main body text size. Some vertical space before and after section headings is placed (automatically according to the LaTeX document class of this template).

The main body text is typeset in 11pt Times font (serif font family). Text body is one-column, justified and single-spaced. First line of a paragraph is generally indented with about 0.8cm, however, first paragraph after any section heading may also be non-indented.

- Avoid extensive and manual spacing
- Use font modifications carefully
- Use proper math and symbol styles

Folder Structure

Use main folder for the main latex .tex and .bib files, the main paper body and bibliography database. Optional: use a folder, e.g. ./sections, to separately manage individual section's latex files.

Use ./images and ./figures subdirectories for the raster images and vector graphics used in the figures and diagrams of the paper. Only use .jpg and .pdf formats respectively for best results. Put source files, e.g. from Illustrator or OmniGraffle, also directly into the ./figures folder along with the PDF versions.

Latex Coding

Follow a clear prologue structure in the LaTeX source file. After the given template components add your `\usepackage{}` block, and `\input{math}` to include our standard math formula definitions. Complete the preamble by any further specific definitions or commands, e.g. `\TODO` and `\FIXME` macros. After the prologue, complete the title and author parts.

If separate .tex files are used for the main report text body, then include them in the main body latex file, and in each of these files indicate the root latex file at the very top:

```
%\!TEX root = ../<main_file_name>.tex!
```

Further LaTeX coding guidelines and recommendations:

- Use the `\emph{}` command fo emphasizing/italicizing not `\textit()`
- Use (our math.tex) style file for math formula consistency
- Use `\mathrm{function()}` for function names
- Use consistent section, equation and figure labelling `\label{sec:introduction}` as well as `eq:rendering`, `fig:system`

- Do not liberally introduce manual horizontal or vertical spacings
- In particular do not use comment codes % to control spacing or indentation before or after elements, e.g. arounc equations, figures or tables

Mathematical Formulas and Equations

Mathematical expressions should follow a consistent set of rules, symbols and formatting as indicated in our `math.tex` style file. LaTeXiT can be used for consistent symbols and formulas in figures. Apply the same scalar, vector and matrix styles as well as use the same variables consistently throughout your text, see also the predefined styles and specific variables in `math.tex`. A few guidelines that are useful are given below, use them as much as possible (but adjust as necessary to make formulas clear):

- use (lower-case) italic letters for normal (scalar) variables such as x or t
- prefer i and j as index variables and m and n to denote number of elements or iterations
- use for example other letters such as a, b and c for constants
- use lower-case bold italic to denote vectors such as $\boldsymbol{u}$ or $\boldsymbol{v}$
- use upper-case bold letters such as $\mathbf{M}$ or $\mathbf{N}$ for sets and matrices
- use upper- or lower-case italic letters such as $f()$ or $G()$ to denote functions
- use regular plain font and decimal point to denote explicit constants such as 2 or 100.12

Use as much formalism, variables and equations, as is useful to clearly understand your description, but not for trivial facts. Use inline equation format also for variables like x and numbers like 12 used in the main text body segments.

Mathematical equations that are important or are referenced should be laid out as a regular equation with consecutive numbering to the right as below:

$$E = m \cdot c^2 \tag{7.1}$$

Such regular free standing equations do not need a punctuation and follow a paragraph that ends with a dot or double-colon, and the following paragraph starts regularly indented.

Equations may also be treated 'inline' with the main flow of the text, thus being part of one regular sentence. Such 'inline' equations as

$$a^2 + b^2 = c^2,$$

typically end with a comma and the text continues below unindented lower-case to finish the sentence. To adjust spacing, the equation can be connected to the paragraph text flow by separating it only using % comments. If an equation ends the sentence of a paragraph it ends with a dot.

Floats, Figures, Screenshots and Graphs

Figures, screenshots, tables, graphs and other floats should support the overall idea of the paper or some description in the text specifically, and are numbered sequentially. Every figure must support a key idea or concept, and it must be meaningful with self-explanatory captions.

Make sure every figure or float is placed correctly and is used as well as referenced in the text.

Placement Each float is centered and *placed in the text directly after* the first paragraph referencing it, never directly after a heading. If formatting constraints are difficult, figure placement may be forced, e.g. to be placed at the top of the next possible new page. In double-column document formats, put large floats spreading both columns at the top of the next following page.

Figures Use a consistent style and appearance for each figure, using clean lines and diagrams. Make careful use of not too bright or disturbing colors, and watch out for grayscale usage when printed.

Use sans-serif fonts in figures and diagrams unless for math symbols and variables. Enforce consistent upper-lower case usage throughout all figures and terms used in the text, and match variables and math formulas or symbols in figures to the ones in in the text. Also match the size of text in figures to the size of the main paper body text.

For figures and diagrams that include any vector graphics elements (e.g. arrows, lines, boxes, points etc.), use the PDF vector graphics format with white or transparent background, and do not use raster image formats. If raster images are included in vector graphics figures, also use the PDF format for the combined figure.

Screenshots Each screenshot must demonstrate a clear visual effect, or major point of your work, or an example or problem of a standard (prior) approach. Some, very few, images may be mostly visual teasers. Each screenshot must include all details such as (data) statistics, used parameters and experimental settings in the caption or in the accompanying text where the figure is referenced from. Use raster image formats only for image-only figures, and then use the JPG file format not PNGs.

Graphs Graphs and plots must clearly demonstrate some numerical test results or data statistics. Strictly avoid clutter within one graph, and clearly relate parameters and results of experiments. Just listing basic values can be done in simple tables. Use expressive and clearly labelled axis in all graph plots. Graphs should basically be understandable on their own along with their caption also without reading the main text. Graphs and plots are usually based on vector graphics, thus use the PDF vector graphics format with white or transparent background as for figures and diagrams.

Cross-References

Cross-references to numbered items such as sections and figures are capitalized as in the following examples. In Section 1 we provide a brief summary of text formatting instruction and Figure 7.1 shows our group logo. A figure can contain multiple subfigures which can be referenced individually as illustrated in Figure 7.2(b).

Figure 7.1: Example of single figure with caption text.

(a) vmml logo

(b) uzh logo

Figure 7.2: Example subfigure with captions referencing (a) the VMML and (b) the UZH logos.

Use the same full or shortened form for all references, e.g. Sec. 1, Fig. 7.2(a), Eq. 7.1.

Pseudo Code

Example for a pseudo code given below

```
nlevels = log2(I) + 1
for l = nlevels : -1 : 1
  loop over all 2x2x2 blocks (i,j,k)
    A_l(i,j,k) = average(block)
  end
end
```

Commands

You can use the command `\FIXME{}` in order to mark sections in the text, which need to be edited and fixed. E.g. **FIX**: check reference XY.

Bibliography

The bibliography with list of references is placed on a new page at the end of the document, titled “References” or “Bibliography” and typeset similar to the main section headings but without numbering. Bibliography entries should follow standard IEEE or ACM proceedings or transaction journal formatting styles, and be referenced by last name abbreviation and year such as e.g. [Paj07] and [GSSP10], or by number as for [1].

Formatting References

Bibliography entries should be clean, accurate, compact and consistent across all entries. Thus for the entries in the BibTeX bibliography .bib file follow the following basic rules:

- Have complete entries, including full author names (first and last), full conference paper (inproceedings type with name, year, pages) and journal (article type with journal name, volume, number/issue, month) data
- Use capitalized title and special terms, e.g. ”Point Set Processing for Data Analysis on the GPU”.
- Use consistent venue description (name the same conference/journal in the same way).
- Use compact *inproceedings* booktitle for conferences without year, i.e. ”Proceedings IEEE Visualization” instead of ”8th Int. Conference on Visualization (VIS’08), IEEE 2008”.
- Do not use separate organization field if clear from the conference or journal, and usually no publisher, month and address for conferences.
- Keep entries concise, avoid unnecessary or duplicate information such as e.g. address or location for conferences or even journals, redundant numpages fields, or duplicated association/organization data etc.
- Use these fields sparsely if at all necessary: howpublished, publisher (don’t for conferences, only for books or so), series (don’t for conferences, only for LNCS or similar)
- Typically do not use venue, address or location for conferences or even journals
- Keep keywords simple and meaningful (e.g. main first keyword graphics, visualization, geometry, theory, databases, mathematics etc. followed by a few more specific ones)

Example bad BibTeX entry:

```
@ inproceedings{as78439729asf,
  Title = {Tile-based LOD for the Parallel Age},
  Author = {Niski, Krzysztof and Cohen, J.~D.},
  Booktitle = {Proceedings IEEE Visualization Conference (IEEE Vis'10)},
  Volume = {13},
  Pages = {1352},
  Organization = {IEEE},
  Series = {IEEE TVCG journal},
  Publisher = {Computer Society Press},
}
```

Above BibTeX entry will result in a not very nice, incomplete and inconsistent reference:

[NC] Krzysztof Niski and J. D. Cohen. Tile-based lod for the parallel age. In *Proceedings IEEE Visualization Conference (IEEE Vis'10)*, volume 13 of *IEEE TVCG journal*, page 1352. IEEE, Computer Society Press.

The corresponding clean and consistent BibTeX entry would be:

```
@article{NC:07,
  Title = {Tile-based {LOD} for the Parallel Age},
  Author = {Niski, Krzysztof and Cohen, Jonathan~D.},
  Journal = {IEEE Transactions on Visualization and Computer Graphics},
  Month = {November/December},
  Volume = {13},
  Number = {6},
  Pages = {1352--1359},
  Year = {2007}
}
```

This will result in a nice and consistent reference:

[NC07] Krzysztof Niski and Jonathan D. Cohen. Tile-based LOD for the parallel age. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1352-1359, November/December 2007.

More bad examples include the following

[HBC10a, SWF10a, HXS09a, CH09a, Str09a, FM07a, WGS07a, CMF05a, AGL+02a, KCCO00a, CYH+97a],

and the good ones are these here

[HBC10b, SWF10b, HXS09b, CH09b, Str09b, FM07b, WGS07b, CMF05b, AGL+02b, KCCO00b, CYH+97b].

Pages

Cover and Abstract

Title of thesis, author, affiliation, date and any other administrative information should be placed on a separate cover page (see first page).

Main title of work should be typeset in large sans-serif bold font type. Title is to be followed by author and affiliation. At the bottom, separated group affiliation is set.

Abstract follows on separate page after cover page and before the table-of-content page(s).

TOC

The table-of-content (TOC) contains all Section and (Sub-)Sub Section headings and their corresponding pages and is listed on separate pages before the first section. It has its own heading typeset as a section heading without numbering.

Bibliography

[AGL+02a] J. Allard, V. Gouranton, L. Lecointre, E. Melin, and B. Raffin. Netjuggler: Running VR Juggler with multiple displays on a commodity component cluster. In *Proceeding IEEE Virtual Reality*, pages 275–276, 2002.

[AGL+02b] Jérémie Allard, Valérie Gouranton, Loïck Lecointre, Emmanuel Melin, and Bruno Raffin. Net Juggler: Running VR Juggler with multiple displays on a commodity component cluster. In *Proceeding IEEE Virtual Reality*, pages 275–276, 2002.

[CH09a] Clement Courbet and Celine Hudelot. Random accessible hierarchical mesh compression for interactive visualization. In *Proceedings of the Symposium on Geometry Processing*, SGP '09, pages 1311–1318, Aire-la-Ville, Switzerland, Switzerland, 2009. Eurographics Association.

[CH09b] Clement Courbet and Celine Hudelot. Random accessible hierarchical mesh compression for interactive visualization. In *Proceedings Eurographics Symposium on Geometry Processing*, pages 1311–1318, 2009.

[CMF05a] Xavier Cavin, Christophe Mion, and Alain Filbois. COTS cluster-based sort-last rendering: Performance evaluation and pipelined implementation. In *Proceedings IEEE Visualization*, pages 111–118. Computer Society Press, 2005.

[CMF05b] Xavier Cavin, Christophe Mion, and Alain Filbois. COTS cluster-based sort-last rendering: Performance evaluation and pipelined implementation. In *Proceedings IEEE Visualization*, pages 111–118, 2005.

[CYH+97a] Tzi-cker Chiueh, Chuan-kai Yang, Taosong He, Hanspeter Pfister, and Arie E. Kaufman. Integrated volume compression and visualization. In *Proceedings of the 8th conference on Visualization '97*, VIS '97, pages 329–336, Los Alamitos, CA, USA, 1997. IEEE Computer Society Press.

[CYH+97b] Tzi-cker Chiueh, Chuan-kai Yang, Taosong He, Hanspeter Pfister, and Arie E. Kaufman. Integrated volume compression and visualization. In *Proceedings IEEE Visualization*, pages 329–336, 1997.

[FM07a] Nathaniel Fout and Kwan-Liu Ma. Transform coding for hardware-accelerated volume rendering. *IEEE Trans Vis Comput Graph*, 13(6):1600–1607, 2007.

[FM07b] Nathaniel Fout and Kwan-Liu Ma. Transform coding for hardware-accelerated volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1600–1607, 2007.

[GSSP10] Prashant Goswami, Philipp Schlegel, Barbara Solenthaler, and Renato Pajarola. Interactive SPH simulation and rendering on the GPU. In *Proceedings ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 55–64, 2010.

[HBC10a] Mark Howison, E. Wes Bethel, and Hank Childs. Mpi-hybrid parallelism for volume rendering on large, multi-core systems. In James P. Ahrens, Kurt Debattista, and Renato Pajarola, editors, *EGPGV*, pages 1–10. Eurographics Association, 2010.

[HBC10b] Mark Howison, Wes E. Bethel, and Hank Childs. MPI-hybrid parallelism for volume rendering on large, multi-core systems. In *Proceedings Eurographics Symposium on Parallel Graphics and Visualization*, pages 1–10, 2010.

[HXS09a] Chang Hui, Lei Xiaoyong, and Dai Shuling. A dynamic load balancing algorithm for sort-first rendering clusters. In *IEEE International Conference on Computer Science and Information Technology*, pages 515 –519, aug. 2009.

[HXS09b] Chang Hui, Lei Xiaoyong, and Dai Shuling. A dynamic load balancing algorithm for sort-first rendering clusters. In *Proceedings IEEE International Conference on Computer Science and Information Technology*, pages 515 –519, 2009.

[KCCO00a] Vladlen Koltun, Yiorgos Chrysanthou, and Daniel Cohen-Or. Virtual occluders: An efficient intermediate pvs representation. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pages 59–70, London, UK, 2000. Springer-Verlag.

[KCCO00b] Vladlen Koltun, Yiorgos Chrysanthou, and Daniel Cohen-Or. Virtual occluders: An efficient intermediate PVS representation. In *Proceedings Eurographics Workshop on Rendering Techniques*, pages 59–70, 2000.

[Paj07] Renato Pajarola. Efficient data structures. In Markus H. Gross and Hanspeter Pfister, editors, *Point-Based Graphics*, Series in Computer Graphics, pages 148–165. Morgan Kaufmann Publishers, 2007.

[Str09a] Filip Strugar. Continuous distance-dependent level of detail for rendering heightmaps. *journal of graphics, gpu, and game tools*, 14(4):57–74, 2009.

[Str09b] Filip Strugar. Continuous distance-dependent level of detail for rendering heightmaps. *Journal of Graphics, GPU and Game Tools*, 14(4):57–74, 2009.

[SWF10a] Tim Suss, Timo Wiesemann, and Matthias Fischer. In *Proceedings of the 2010 IEEE Fifth International Conference on Networking, Architecture, and Storage*, NAS '10, pages 448–456, Washington, DC, USA, 2010. IEEE Computer Society.

[SWF10b] Tim Suss, Timo Wiesemann, and Matthias Fischer. Evaluation of a c-load-collision-protocol for load-balancing in interactive environments. In *Proceedings IEEE International Conference on Networking, Architecture, and Storage*, pages 448–456, 2010.

[WGS07a] Chaoli Wang, Antonio Garcia, and Han-Wei Shen. Interactive level-of-detail selection using image-based quality metric for large volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13:122–134, 2007.

[WGS07b] Chaoli Wang, Antonio Garcia, and Han-Wei Shen. Interactive level-of-detail selection using image-based quality metric for large volume visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(1):122–134, January/February 2007.