Using NeRF- and Mesh-Based Methods to Improve Visualisation of Point Clouds

Oscar Montelin, Vilma Ylvén

Master's thesis
2024:E6

LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

CENTRUM SCIENTIARUM MATHEMATICARUM

Abstract

In recent years, the field of generating synthetic images from novel view points has seen some major improvements. Most importantly with the publication of Neural Radiance Fields [9] allowing for extremely detailed and accurate 3D novel views. Usage of LiDAR products to collect actual depth data has also seen an increase as it is immensely useful for achieving high resolution 3D mapping of a space. However, these point clouds can be hard to read as they give a discrete sample of surfaces and lack colour and texture. In this thesis we explore various ways of improving visualisation and human understanding of scenes and objects generated from a stationary camera-LiDAR pair. We do this by first isolating individual rigid moving objects in a scene and constructing denser point clouds of these objects by projecting them on the camera video and aggregating over time. By utilising the novel view synthesis method Point-NeRF [15] we also improve visualisation of these dense point clouds further. This is done by training a point-based neural network on the aggregated point clouds and the corresponding video frames. Lastly two methods for surface reconstruction of objects and the backgrounds are tested. With this we achieve accurate and understandable renders of a variety of vehicles. We believe that with a well calibrated camera this method shows significant promise for reconstructing scenes in 3D in post-processing well.

Under de senaste åren har stora framsteg gjorts inom fältet för *novel view synthesis*, främst tack vare publikation av *Neural Radience Fields* [9] som möjliggjort enormt detaljerade, tidigare osedda 3D-vyer. Användningen av LiDAR-produkter för insamling av djupdata har också sett en ökning i popularitet på senare tid, då de kan generera en enormt högupplöst 3D-mappning av ett utrymme. Dessa punktmoln kan dock vara mycket svårlästa då de representerar ett diskret urval av ytor och saknar både färg och textur. I det här examensarbetet undersöker vi metoder för att förbättra visualisering och mänsklig förståelse för scener och objekt som genereras från ett stationärt kamera/LiDAR-par. Vi åstadkommer detta genom att först isolera individuella rigida objekt i rörelse, och sedan konstruera tätare punktmoln av dessa objekt genom att projicera ner dem på videon och aggregera över tid. Genom att utnyttja den nya metoden Point-NeRF [15] förbättrar vi också visualiseringen av dessa moln ytterligare. Detta genom att träna ett punkt-baserat neuralt nätverk på det aggregerade punktmolnet och den korresponderande video-sekvensen. Till sist testas två metoder för återskapning av ytor på både objekt och bakgrund. Med detta uppnår vi exakta och förståeliga resultat av en mängd olika fordon. Vi tror att med en väl kalibrerad kamera kan denna metod visa lovande resultat för att rekonstruera scener i 3D vid efterbearbetning.

Acknowledgements

This thesis has, barring the obligatory frustration that comes with writing a thesis, been a delight. The people that we have worked with, at both the company and at LTH, are not only great engineers and scientists, but also wonderful people.

For those that assisted us in the workplace, you where an enormous help in answering all our questions and swiftly providing help or giving us the data we needed to proceed. You assisted us immensely by always providing us with an alternate angle of attack to solve the problems we where facing.

We would also like to specifically thank our LTH supervisors: Kalle Åström and Johanna Engman. Thank you for acting as our counsellors and assuring us, once a week, that we were in fact were well on our way to a finished thesis. Thank you also for all the guiding nudges and interesting papers, it's been a delight to work with you.

No master students were, significantly, harmed during the making of this thesis. Which quite frankly is something to acknowledge and and that we are grateful for as well.

Glossary

COLMAP - [13] [14],
general purpose SfM and MVS pipeline, see 2.7.3.

DBSCAN - ***D****ensity-****B****ased* ***S****patial* ***C****lustering of* ***A****pplications with* ***N****oise* [6],
algorithm for data point clustering, see 2.2.

GPU - ***G****raphical* ***P****rocessing* ***U****nit.*

ICP - ***I****terative* ***C****losest* ***P****oint* [2],
algorithm for point cloud registration, see 2.4.3.

I-NGP - ***I****nstant* ***N****eural* ***G****raphics* ***P****rimitives* [11]

LiDAR - ***Li****ght* ***D****etection* ***A****nd* ***R****anging*,
optical sensor for recording distances in 3D environments.

MVS - ***M****ulti-****V****iew* ***S****tereo*,
method of determining depth from multiple 2D views of an object or environment.

NeRF - ***Ne****ural* ***R****adiance* ***F****ields* [9],
see 2.7.

NVIDIA - Company that primarily produces GPUs.

Point-NeRF - ***Point-****based* ***Ne****ural* ***R****adiance* ***F****ields* [15],
see 2.7.6.

SfM - ***S****tructure* ***f****rom* ***Mo****tion*,
method of estimating structure from a series of images and the motion information of the camera.

Voxel - **Vo**lumetric equivalent of a pi**xel**.

Contents

1 Introduction

The field of computer vision has seen significant progress as of late. The introduction of LiDAR has meant that correct depth images can be produced without relying on photogrammetry. However, LiDAR contains little to no information about visual elements such as colour and texture, making the raw data hard to interpret for humans. This can be compensated for by adding an ordinary camera to the setup, thereby allowing for the coloration of the point cloud by re-projection.

A recent area of interest in the field of computer vision is the problem of novel view synthesis. The question posed has been how to best render an object from an arbitrary view point. In 2020 a new method, Neural Radiance Fields (NeRF) [9], for novel view synthesis was published by Ben Mildenhall et al. This led to a renaissance in the field, resulting in multiple methods based on the proposed NeRF-model.

In this thesis we use a camera-LiDAR pair, described in Section 3.1 and visualised in Figure 12, in conjunction with mesh and NeRF methods to improve the visualisation of point clouds. While we improve the visualisation in general, we focus on improving the visualisation of rigid movable objects, i.e. moving vehicles. Cars have two distinctive properties that this thesis aims to exploit, the fact that they almost always are in motion and the fact that they are rigid objects. This allows for collection and aggregation of data over time, something that we have yet to see in other papers in the area.

1.1 Task

The main goal of this thesis is to improve visualisation of objects in 3D point clouds. This will be done by aggregating data over time and fusing it with a video stream of the same objects. The aggregated data will then be given to a Neural Radiance Field to further improve details. The aim is to post-recording present a 3D point cloud scene that is as close to reality as possible and that can easily be understood by a human.

A secondary goal is to improve the visualisation of point clouds in general. This will be done by exploring various meshing techniques together with re-projection of LiDAR point onto images in order to collect colour data.

These improvements are done with the goal of aiding LiDAR's integration into security surveillance, by allowing operating personnel to interpret LiDAR data without having to be intimately acquainted with LiDARs or point clouds. The added depth data of the LiDAR could then work as a complement to the video data from cameras, giving more exact measurements of, for example, peoples heights or the distances between objects.

1.2 Limitations

The main limitation we will be working under is that we have access to one LiDAR and one video camera. These will be stationary and at a defined distance from each other, simplifying both our thesis and any potential future commercial applications.

Working with data that changes over time is both the basis for this thesis and a significant limitation. Therefore we have restrained ourselves to only attempt to work with rigid object

over time as their structural integrity is crucial for aggregation several frames.

To try to minimise unknown variables we have limited ourselves to working with slowly constantly moving vehicles as that was the most easily available data. The low speed limitation is due to artefacts in the LiDAR scanning that appears when the objects are moving too fast.

1.3 Previous Works

This thesis is part of a sequence of works done on LiDAR point clouds and various ways of processing them. It mainly builds upon the recent works of Afsén and Boye Frick [1], who similarly to this thesis attempted to aid a human user in the viewing of a LiDAR point cloud. Instead of doing this with the help of neural networks Afsén and Boye Frick took a more mathematical approach to the reconstruction. The main methods applied in their work, alpha shapes and Poisson mesh reconstruction, have also been applied in this thesis and has acted as a baseline for the point cloud surface reconstruction. We recommend looking at the results of their thesis as something to contrast the results of our thesis against.

Previous relevant thesis papers also include the works of Doyle and Nilsson [4] and the works of Lind and Bernstråle [8] who both have dealt with similar data sets and used many of the methods applied early on in this thesis. The latter concerned itself with the segmentation and tracking of items found in the point clouds, something that this thesis only briefly touches on as a stepping stone for point cloud aggregation. Doyle and Nilsson focused on testing different methods for background segmentation of a scene. By using more classical methods as well as neural networks they achieved segmentation of ground, vegetation and other objects in outdoor LiDAR scenes.

Both of these works are relevant to this one, and their work could most certainly be worked into our data pre-processing to achieve more precise results. Since the focus of this thesis is the actual visualisation, more simple methods have been applied.

Moreover, in-house work has been done at the company. This includes a project where point clouds have only been coloured via projection onto a camera video, something that we in this thesis continue upon. The result of a simple projection can be seen in Figure 1.

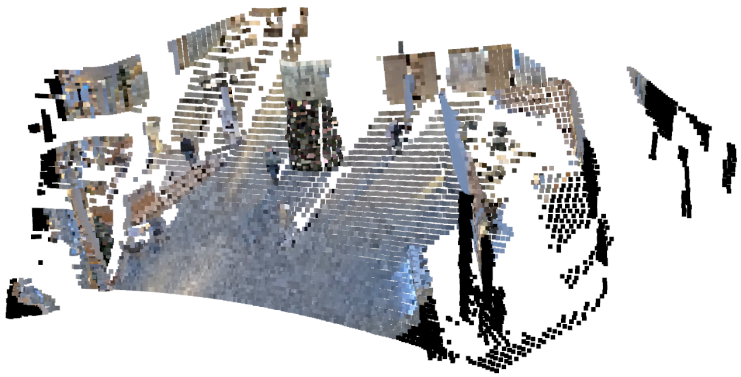


Figure 1: A LiDAR point cloud that has been coloured via projection onto camera frames.

1.4 Statement of Contribution

Throughout the entire project Oscar and Vilma have worked very closely together. However, Vilma has focused more on the processing of the 3D-point clouds while Oscar has worked on the image 2D image processing. In this report Oscar's main focus has also been theory while Vilma has written results and methods.

2 Theory

2.1 Normal Estimation of Point Clouds

Several of the methods applied in this thesis rely on the points in the point clouds having normals. One of the very first steps will thus have to be an estimation of normals, as this is not data the LiDAR gives.

Normal estimation of points in point clouds is classically done by using the k-nearest neighbours to compute the covariance matrix for each point. Through principal component analysis the eigenvalues and eigenvectors of this covariance matrix can then be computed. Essentially the k nearest points are used to estimate a surface that we take the normal of.

This leaves two options for orientation of the normals. Of these two options the normal pointing towards the LiDAR sensor is chosen, since the LiDAR can only detect surfaces with a normal pointing towards it.

2.2 Density-Based Spatial Clustering Of Applications With Noise (DBSCAN)

The standard algorithm used for clustering of points is k-means. It is a simple algorithm that given a number of clusters, k, iteratively assigns each point to the cluster who's centre is the closest. The initial cluster centres are chosen at random and are then updated every iteration.

DBSCAN [6] is a clustering algorithm that relies on the density of the data points to determine clusters. This allows it to be more flexible than other clustering algorithms such as k-means. This as it can handle noise in the data set and allow for clusters to take any shape as long as they have a mostly consistent density. This property can be seen in the example in Figure 2. Note that DBSCAN has the ability to classify data points as noise, but struggles to classify clusters with varying density.

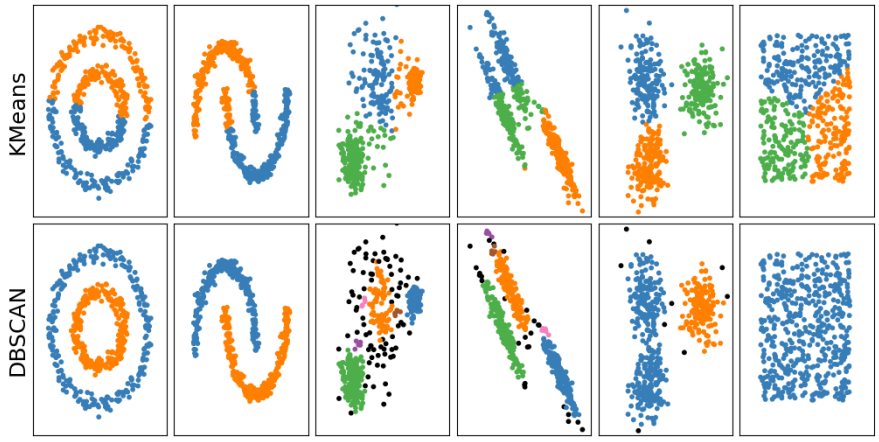


Figure 2: Comparison of DBSCAN and k-means clustering algorithms on six different data-sets. Black data points have been classified as noise. The different colours represent different classes.

DBSCAN has two adjustable parameters, ε and *min_samples*. The algorithm starts by calculating for each point, the number of other points within a radius of ε. If the number of points within this radius is greater or equal to *min_samples* that point is considered a core point. DBSCAN then randomly picks a core point and starts building a cluster by iteratively including all core points within the ε radius of the growing cluster. When no more core points can be added to the cluster the method does a similar search over the set of non-core points. Non-core points that are within a radius of ε of a cluster are added to said cluster. The process is then repeated until the set of core points is emptied and the remaining elements of the set of non-core points are considered noise.

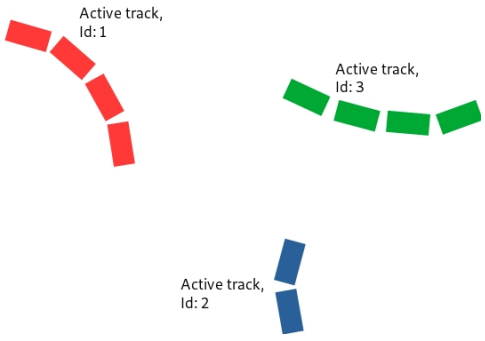
The issue of varying density has been solved in the succeeding algorithm HDBSCAN [3] by dynamically varying the ε. Depending on the implementation this can also come with lower time complexity, and is thus a more ideal method for this thesis. However, an implementation of this algorithm did not exist in the graphics library used, and thus DBSCAN will be the method used for clustering throughout.

2.3 Tracking

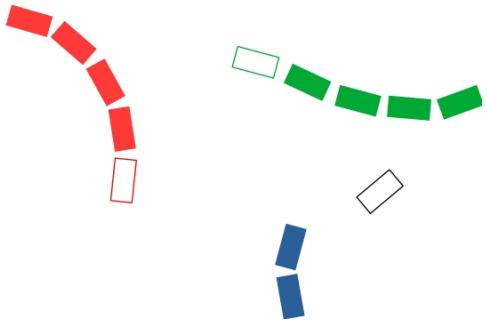
In this thesis we will use a relatively simple tracking algorithm since object tracking is not the main focus.

The algorithm used in this thesis is an adaptation of the algorithm presented by Yin, Zhou and Krähenbühl [16]. In their paper the centre-point of the cluster is used for comparison between frames, instead of the more traditional method for 2-D tracking where a bounding box is used.

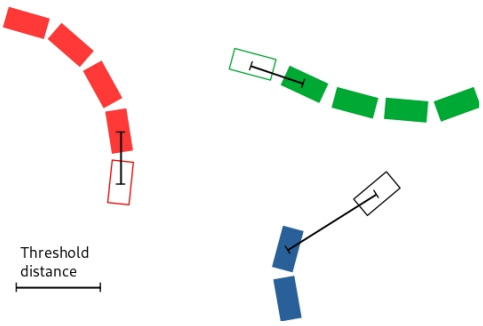
In both their paper and in our thesis, for each frame the centres of the clusters are compared to the closest cluster-centres in the previous frame that are not already matched up. The clusters are declared a match if the distance between them is less than a set threshold value.



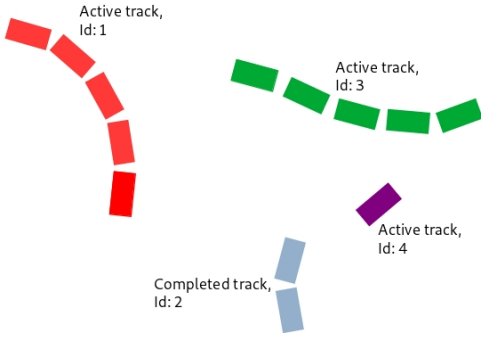
(a) A scenario where we have 3 active tracks.



(b) A new frame of clusters is introduced, represented in white outlined boxes.



(c) The centres of the new clusters are compared to the closest centre of a cluster in the last frame.



(d) The new clusters that were close enough (red and green) have been matched up with an existing track. The cluster that was too far away from the last blue cluster has been marked as a new track. The blue track has been marked as completed, not having been continued by any new cluster.

Figure 3: Illustration of the centre-based tracking algorithm used.

Once a track is marked as completed it cannot become active again, thus if a track has a single frame where the cluster is not registered by the clustering algorithm, it will be split into two tracks.

An outline of the entire tracking algorithm can be found in Figure 3.

2.4 Point Cloud Registration

Point cloud registration refers to the method of taking two separate point clouds, that are both depicting pieces of the same object, and aligning them. See for example Figure 4 where the two point clouds, yellow and blue, are measurements of the same car from different angles. In Figure 4a neither the rotation nor the translation of the point clouds are what they would need to be to align the point clouds, whereas in Figure 4b the blue point cloud has been transformed to fit the yellow point cloud.

2.4.1 Rotation and Translation

The two transformations that we will use in this thesis are rotation and translation.

A rotation matrix is a 3×3 matrix causing a rotation around some axis. For example, for a rotation of the point $\mathbf{x}$ around the z-axis with the angle θ, the matrix looks as follows

$$\mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{1}$$

Multiplying this as $\mathbf{x}' = \mathbf{R}\mathbf{x}$ will result in the rotated point $\mathbf{x}'$. A rotation around some arbitrary axis can be achieved by multiplying several rotation matrices around the base axes. A rotation matrix will always fulfil the conditions that $\det(\mathbf{R}) = 1$ and $\mathbf{R}^T\mathbf{R} = \mathbb{I}$.

(a) Before registration (b) After registration

Figure 4: Point cloud registration of two LiDAR frames. Here the blue frame has been transformed to minimise its distance to the yellow cloud.

A translation is simply done by adding the desired vector to the point in question resulting in $\mathbf{x}' = \mathbf{x} + \mathbf{t}$. This can be expressed as a matrix multiplication by having

$$\begin{bmatrix} x_1' \\ x_2' \\ x_3' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_1 \\ 0 & 1 & 0 & t_2 \\ 0 & 0 & 1 & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{bmatrix}. \tag{2}$$

The order of operations is important since a rotation may result in a displacement, and thus the total translation will end up being remarkably different depending on if it was performed before or after the rotation.

A general rigid transformation, combining both rotation and translation can be expressed as

$$\begin{bmatrix} x_1' \\ x_2' \\ x_3' \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{bmatrix}, \tag{3}$$

where $\mathbf{R}$ is a 3×3 rotation matrix and $\mathbf{t}$ is a 3×1 vector.

In this thesis we will at times rely heavily on a rotation matrix that aligns two normalised vectors $\mathbf{a}$ and $\mathbf{b}$ in 3-dimensional space. To compute this we first recognise that it is equivalent to a 2-dimensional rotation in the plane spanned by $\mathbf{a}$ and $\mathbf{b}$. A rotation matrix for such a rotation is as seen in Equation 1 where θ is the angle between $\mathbf{a}$ and $\mathbf{b}$. Moreover, according to trigonometry we have

$$\cos\theta = \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}||\mathbf{b}|} = \mathbf{a} \cdot \mathbf{b}, \qquad \sin\theta = \frac{||\mathbf{a} \times \mathbf{b}||}{|\mathbf{a}||\mathbf{b}|} = ||\mathbf{a} \times \mathbf{b}||,$$

giving us, in the plane coordinates, the rotation matrix

$$\mathbf{G} = \begin{bmatrix} \mathbf{a} \cdot \mathbf{b} & -||\mathbf{a} \times \mathbf{b}|| & 0 \\ ||\mathbf{a} \times \mathbf{b}|| & \mathbf{a} \cdot \mathbf{b} & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

An orthogonal basis for the plane spanned by $\mathbf{a}$ and $\mathbf{b}$ can be given by $(\mathbf{u}, \mathbf{v}, \mathbf{w})$ where $\mathbf{u} = \mathbf{a}$, $\mathbf{v} = (\mathbf{b} - (\mathbf{a} \cdot \mathbf{b})\mathbf{a}) \,/\, (||\mathbf{b} - (\mathbf{a} \cdot \mathbf{b})\mathbf{a}||)$ and $\mathbf{w} = \mathbf{u} \times \mathbf{v} = (w_1, w_2, w_3)^T$, resulting in the basis changing matrix $\mathbf{F} = (\mathbf{u}, \mathbf{v}, \mathbf{w})^{-1}$.

We can then calculate the desired matrix $\mathbf{R} = \mathbf{F}^{-1}\mathbf{G}\mathbf{F}$. Expanding this gives the following formula:

$$\mathbf{R} = \mathbb{I} + \mathbf{K} + \mathbf{K}^2 \frac{1 - (\mathbf{a} \cdot \mathbf{b})}{||\mathbf{w}||^2}, \tag{4}$$

where $\mathbb{I}$ is the identity matrix and

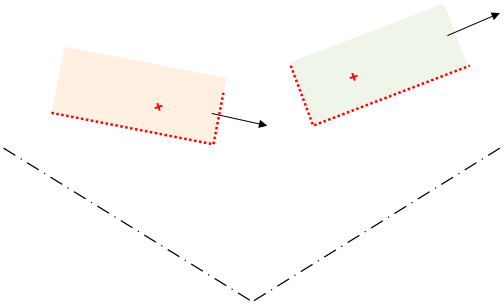
$$\mathbf{K} = \begin{bmatrix} 0 & -w_3 & w_2 \\ w_3 & 0 & -w_1 \\ -w_2 & w_1 & 0 \end{bmatrix}. \tag{5}$$

We could then show that $\mathbf{R}\mathbf{a} = \mathbf{b}$ while $\mathbf{R}\mathbf{w} = \mathbf{w}$, fulfilling our requirements for $\mathbf{R}$ being a rotation matrix around $\mathbf{w}$, aligning $\mathbf{a}$ with $\mathbf{b}$.

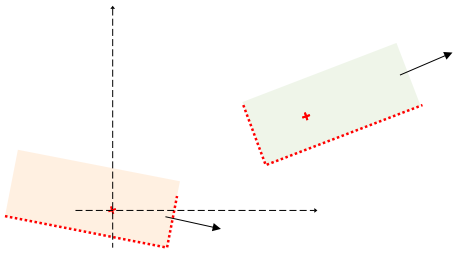
2.4.2 Alignment Algorithm

An algorithm for rough alignment of subsequent point clouds was developed during this thesis. It can be found fully explained in Figure 5.

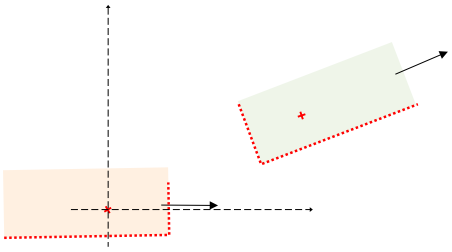
In short, each subsequent point cloud has its mass centre moved to the origin using Equation 2 and then rotated to align its movement vector, estimated during the tracking, with the x-axis using Equation 4. The point cloud is then translated again to position its axis aligned bounding box centre with the axis aligned bounding box centre of the accumulated point cloud so far. The axis aligned bounding box is the smallest rectangular cuboid encompassing all point where the sides are aligned with the x-, y- and z-axes.



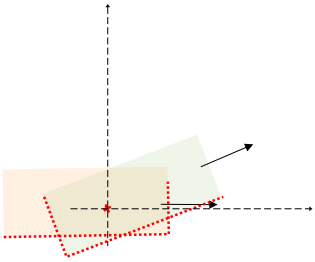
(a) The LiDAR point clouds are here denoted as red dotted lines and the movement vectors estimated during tracking are here black arrows.



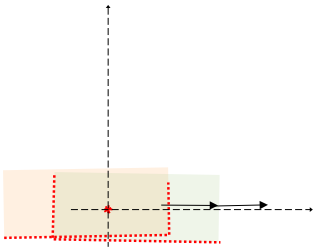
(b) Mass centre, marked as red cross, for the first point cloud is moved to the origin.



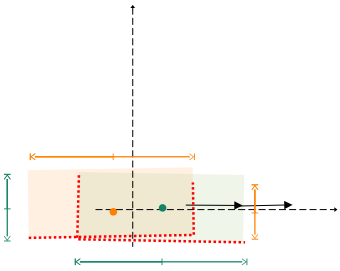
(c) The first point cloud is rotated around the origin to align its movement vector with the x-axis.



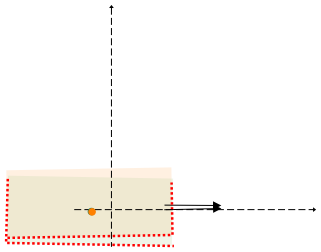
(d) Mass centre for the second point cloud is moved to the origin.



(e) The second point cloud is rotated around the origin to align it's movement vector with the x-axis.



(f) Centres of axis aligned bounding boxes, here green and orange dots, are calculated for both point clouds.



(g) Centre of the second axis aligned bounding box is moved to the centre of the first axis aligned bounding box.

Figure 5: Illustration of algorithm for initial alignment of point cloud frames.

A version of the algorithm includes a last step where one side of the axis aligned bounding box is synchronised between frames. This is useful if some frames contain almost exclusively one side. Generally in this thesis, the side being aligned is chosen as the back of the car.

2.4.3 Iterative Closest Point (ICP)

Iterative Closest Point [2], is a classic, efficient and rigorous method for registration of 3-D point clouds. As a general rule, it assumes that you have captured two versions of the exact same set of points and now want to correct any misalignment between them. It can handle missing points, but does generally preform worse in those cases. In the cases where there is very little correspondence between the point clouds, for example if two points clouds are opposing sides of the same object, it is very unlikely that ICP will end up converging to a satisfactory result.

The algorithm begins by calculating the closest point in the source point cloud to each point in the target point cloud. It then translates the target point cloud using the vector between the centre of mass of the target points and the centre of mass of the set of points in the source point cloud that they were closest too. Having done this it preforms a rotation to get each target point as close to the previously deemed closest point as possible. This procedure is then repeated until a loss criteria threshold is reached or a maximum amount of iterations has been reached, both of which can be user defined.

The algorithm works best if it is given an initial alignment transform that makes a rough attempt at aligning the point clouds. This can greatly reduce the amount of iteration needed for ICP to reach satisfaction and can avoid stagnation in local minima as the first iteration is more likely to be relatively correct. This initial transform can for example be done by feature extraction or manually finding a few point-to-point correspondences.

Variation of this algorithm include calculating a point-to-plane distance instead of a point-to-point distance.

2.5 Projection of World Coordinates Onto an Image Plane

Projection of LiDAR points to the image plane is done via the pinhole camera model. A projection matrix, $\mathbf{P}$, built from an intrinsic matrix, $\mathbf{K}$, and an extrinsic matrix, a vector $\mathbf{t}$ pointing towards the camera center, $\big[\mathbf{R}|-\mathbf{R}\cdot\mathbf{t}\big]$, is used to project points given in world coordinates onto the image plane.

$$\mathbf{P} = \mathbf{K}\cdot\big[\mathbf{R}|-\mathbf{R}\cdot\mathbf{t}\big],$$
$$\mathbf{K} = \begin{bmatrix} f_x & \frac{s}{f_x} & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix},$$

where f_x and f_y are the focal lengths corresponding to the x- and y-axis, s is the skew and (c_x, c_y) is the image centre. $\mathbf{R}$ is a 3×3 rotation matrix and $\mathbf{t}$ is a 3×1 translation vector.

A point in world coordinates, $\mathbf{x} = [x, y, z, 1]^T$, can then be projected by a simple matrix multiplication, $\mathbf{Px} = [u, v, w]^T$. The actual image point can then be extracted by dividing the vector by its third coordinate, $[\tilde{u}, \tilde{v}]^T = [\frac{u}{w}, \frac{v}{w}]^T$.

Assuming that all radial distortions from camera lenses are compensated for and a correct temporal synchronisation between the LiDAR and camera sensor, points can now be mapped to pixels and vice versa. The camera utilised contains an onboard algorithm for compensating for radial distortions and it is was therefore assumed that a point could be mapped to a pixel using only a projection matrix.

2.6 Surface Reconstruction

Surface reconstruction is often one of the first method applied to point clouds. It aims to recreate the surfaces that the points are a sample of, something that can be done in a variety of ways.

The most simple solution to the problem is taking the convex hull of the points as illustrated in Figure 6. However, this is generally not a great idea when working with objects with any level of detail. We will instead present two alternative methods for capturing more detail.

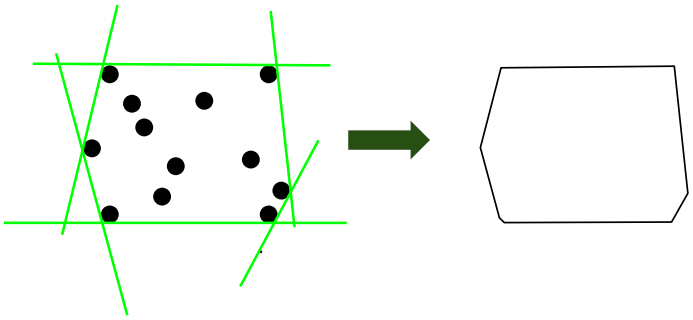


Figure 6: Illustration of constructing a convex hull of a set of points in 2 dimensions.

2.6.1 Alpha Shapes

The concept of alpha shapes [5], tackles this problem by using methods based on the concept of convex hulls.

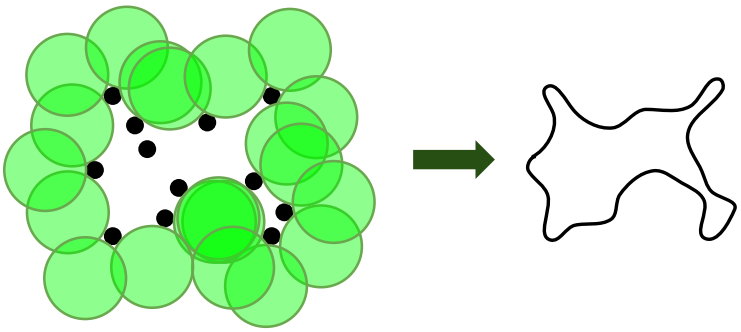


Figure 7: Illustration of the construction of an alpha shape in 2 dimensions.

The construction of an alpha shape is dependant on the user-defined α parameter. The surface surrounding the points will be shaped by allowing a sphere with a radius of $\sqrt{\alpha}$ to "eat" up anything it can reach without touching any of the points in the point cloud. This will be done both outside and, what we would consider, "inside" an object, leaving a mass which defines the final surface. This is illustrated in Figure 7 where the green circles are the spheres forming the shape.

Per definition, when $\alpha \to \infty$, the alpha shape will converge to the convex hull of any object. A small α will similarly leave only the isolated points to make up the alpha shape.

2.6.2 Poisson Surface Reconstruction

Poisson surface reconstruction [7] is a method that relies heavily on the normals of the points in the point cloud.

The method considers the point normals to be the gradient vector field of some indicator function which is 1 inside the shape an 0 outside of it. This essentially reduces the problem to approximating the indicator function from its gradients. This problem can then be constructed as a standard Poisson differential equation and can be solved using numerical methods for differential equations.

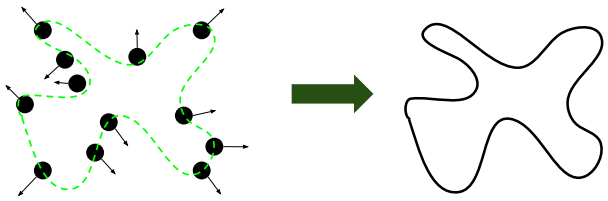


Figure 8: Illustration of Poisson surface reconstruction in 2 dimensions given a set of points with normals.

Poisson surface reconstruction will struggle with point clouds that have poorly estimated normals, something that can become a problem with messy data. As is illustrated in the Figures 6, 7 and 8, different methods for surface reconstruction can have extremely varied results depending on the set of points.

2.7 Novel View Synthesis via Neural Radiance Fields

While meshing techniques can be used to generate novel views, NeRF-based methods generate novel views of a higher quality but require multiple viewing angles. Using the methods discussed in Section 2.4 multiple view points of a rigid moving object can be extracted from an image sequence and in turn be used to train a NeRF-network.

2.7.1 NeRF

Neural Radiance Fields [9], are neural networks trained to represent a volume. The first implementation of this method accomplished this by creating a network F that used the world coordinate and viewing direction as input and returned colour and density as output:

$$F(x, y, z, \theta, \phi) = (r, \sigma), \tag{6}$$

where F is the MLP, (x, y, z) are the coordinates of the shading location, (θ, ϕ) are the angles of the viewing direction, r is the output radiance and σ is the output density from the network.

The output from this network could then be used to render an image using Equation 8.

As training data the method used a sequence of images together with the corresponding camera pose. The camera pose was computed by a general purpose SfM/MVS implementation, described in Section 2.7.3. The network used the L2-norm of the RGB difference between the rendered image and the training image as a loss function and then optimised the networks parameters via backpropagation.

To assist the network with replicating high frequency features, the positional encoding from Equation 9 was used on both the world coordinate and viewing direction.

For this first implementation of a NeRF-network, the model was trained on an NVIDIA V100 for between one to two days. Later implementations have improved upon this original model by either increasing the quality of the images generated, lowering the amount of training data needed, decreasing the amount of time needed until convergence or a combination of the three.

2.7.2 Volume Rendering

Rendering continuous volumes, for the methods [9, 11, 15] referenced in this thesis, is done by performing one ray cast per pixel. Assuming a volume of differing transparency, the colour addition from a volume segment at location $\mathbf{x}$ to the image can be seen as the probability of a ray terminating at said location

$$C(\mathbf{r}) = \int_0^\infty T(t)\sigma(\mathbf{r}(t)\mathbf{c}(\mathbf{r}(t), d)dt \text{ , where } T(t) = \exp\left(- \int_0^t \sigma(\mathbf{r}(s))ds \right), \tag{7}$$

where $\mathbf{r}$ is the ray, t is the distance along the ray, C is the colour of the ray, and σ is the density, T is the accumulated transmittance from 0 to t and $\mathbf{c}$ is the radiance. Limiting the unbounded integral above to $[0, \infty) \rightarrow [t_n, t_f]$, where t_n is the lower bound and t_f is the upper bound, and then approximating it via a Riemann sum allows it to be computed numerically as

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i\delta_i))\mathbf{c}_i \text{ , where } T_i = \exp\left(- \sum_{j=1}^{i-1} \sigma_j\delta_j \right), \tag{8}$$

where $\delta_i = t_{i+1} - t_i$ is the distance between shading locations.

2.7.3 COLMAP

COLMAP [13, 14] is an implementation of structure from motion, SfM, and multi-view stereo, MVS, algorithms able to extract 3D-data and camera matrices from image collections. The original NeRF implementation and I-NGP, as explained in Section 2.7.5, uses the SfM part of COLMAP to estimate the camera poses, while the implementation of Point-NeRF also uses COLMAP to reconstruct a point cloud.

COLMAPs SfM package starts with a correspondence search in which it tries to locate overlapping image pairs, this is accomplished by first extracting features from each image and then matching these features between images. From this, potential matching image pairs are located. These potential matches are then verified by estimating the homography for the transformation and removing matches for which no transformation that aligns enough features exists.

Given a sequence of linked image pairs, with corresponding homographies, camera poses can be extracted and used as input for NeRF or I-NGP. The same sequence of linked image pairs can also be used to compute the geometry via triangulation. Said geometry can be used by Point-NeRF to initialise the neural point cloud, described in Section 2.7.6.

2.7.4 Input Encoding

Deep neural networks are biased towards learning low frequency features. This bias is particularly visible in networks that have a low input dimensionality [12].

In order to circumvent the aforementioned bias the input is mapped from a low frequency, low dimensional domain to a high dimensional, high frequency domain by applying the following transformation elementwise:

$$\gamma(p) = \big(\sin(2^0\pi p), \cos(2^0\pi p), ..., \sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p)\big), \tag{9}$$

where p is an element of the input array, L is the dimensionality for which to increase the input dimension to and γ is the encoding function. To illustrate the issue two neural networks

(a) Unmodified input. (b) With positional encoding. (c) Ground truth.

Figure 9: Two versions of a five layer MLP where each layer contains 256 nodes each. Both networks are trained to try and recreate an image using only the pixel coordinates as variable input. Figure 9a is the result from the network that feeds the pixel coordinates directly into the network. Figure 9b is the result from the network that had its input fed through a positional encoding layer before passing it to the MLP.

were created, one baseline network and one which used positional encoding. Both networks contained five dense layers consisting of 256 nodes each and used the pixel coordinates as input. The network that used positional encoding had its input routed through a positional encoding function 9, with $L = 64$, before being passed to the first hidden layer. It is important to note that the amount of weights have increased for the network using positional encoding. There are 126 more parameters, and therefore 32 256 more weights, passed to the first hidden layer in comparison to the network without positional encoding, but even when compensating for this by adding another 256 node layer to the network, without positional encoding, the issue remains.

The baseline network was unable to recreate the image, as can be seen in Figure 9a, while the network that had its input passed through a positional encoding function managed to recreate most features, as can be seen in Figure 9b.

2.7.5 I-NGP

In July 2022 researchers at NVIDIA published a paper [11] that aimed at decreasing the amount of time needed to train a NeRF-based network, this method used the same basis as the original NeRF network with one major difference, the positional encoder was replaced with a multiresolution hash encoder.

The resulting network achieved state of the art, at the time of publishing, results or at least close to such results with as little as five minutes of training time on a high end consumer GPU (NVIDIA RTX 3090). This was achieved, in combination with an improved ray marching algorithm, by replacing the positional encoding used in the original NeRF network with a novel encoding method called multiresolution hash encoding. This encoding method, in essence, was trainable parameters and a hash-function. While this method of encoding added trainable parameters to the network, the increase in accuracy allowed for scaling down the size of the NeRF-network and thereby reducing the amount of parameters that needed to be updated per training iteration.

The method discretises the volume with a multi resolution voxel grid and uses, for the finer levels, a hash function to locate which voxel a shading location belongs to. The hash function is defined as

$$h(\mathbf{x}) = \left(\bigoplus_{i=1}^{d} x_i \pi_i \right) \bmod T, \tag{10}$$

where h is the hash encoding function, $\mathbf{x}$ is array to be encoded, π_i are unique prime numbers or one, $\bigoplus$ is the bit-wise XOR operation, T is the maximum amount of entries per level and d is the length of the input array.

2.7.6 Point-NeRF

Instead of training a network to fully represent a volume, Point-NeRF [15] stores image features in a neural point cloud and uses said feature data to render an image. The features are extracted from the training images by a pre-trained, and for the scope of this method, general 2D-downsampling network G_f. Besides the coordinates and features, all points also contain a confidence value. This confidence value represents how confident the model is that this neural point represents actual geometry. In other words, each neural point contains

$$\mathrm{np}_i = (p_i, f_i, \gamma_i), \tag{11}$$

where np_i is the neural point with index i, p_i is the neural point's location, f_i are the features and γ_i is the confidence. The location and confidence value of each neural point was computed by a 3D-CNN $G_{p,\gamma}$.

The render process is similar to the original NeRF implementation in that is uses ray marching to locate the shading locations. For each step during the ray march up to eight of the nearest neural points, the viewing direction, d, and the shading location, x are fed to the neural network *Point-NeRF*

$$(\sigma, r) = \textit{Point-NeRF}(x, d, p_1, f_1, \gamma_1, ...), \tag{12}$$

which in turn returns the density, σ, and the radiance, r for said shading location. The neural network is split into multiple smaller MLPs allowing it to handle a variable input size. Point-NeRF uses the same loss function as the original NeRF implementation to optimise the network.

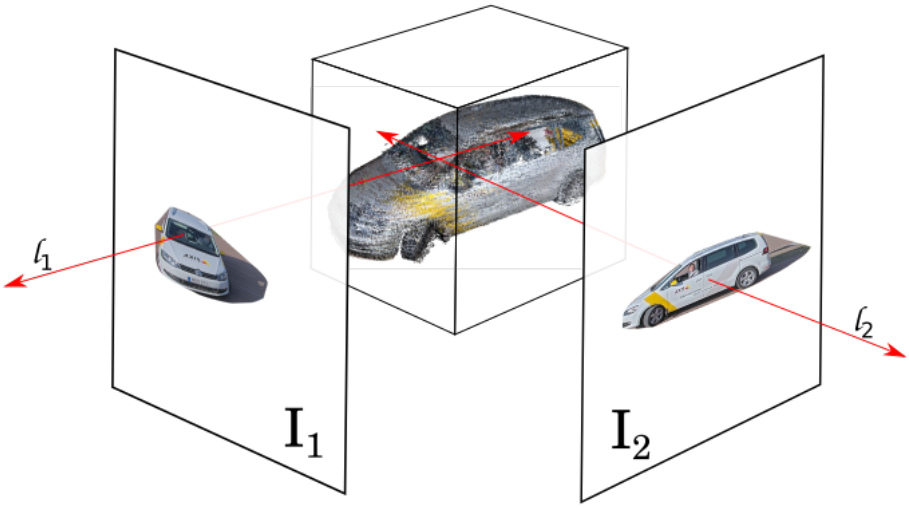


Figure 10: Illustration of the render process used in Point-NeRF. I_1 and I_2 are images from different view points of the same object. A shading location is chosen by picking a point along a line, for example l_1 or l_2 in the illustration. The closest neural points to the shading location, within a user defined radius, and the direction of the line are then used as input to the Point-NeRF network. By repeating this process for multitple shading locations on a line, a colour can be rendered via Equation 8, which in turn can be compared to the pixel colour of where the line intersects with the image. Note that the point cloud has been colourised for clarity, the neural points do not store colour but instead store extracted image features.

The Point-NeRF method also contains modules for adding points, referred to as the point grower, and removing points, referred to as the point pruner. The point pruner removes points that have a confidence value below a user defined threshold. The point grower adds points to shading locations if the computed opacity is above a user defined threshold and if shading locations distance to the closest neural point is above a user defined threshold.

3 Methods

3.1 Equipment

In this project we used a rig consisting of one LiDAR sensor and one camera that are fixed relative to each other and oriented approximately in the same direction, see Figure 12.

The LiDAR utilised consists of two eyes, where each eye has a laser, a mirror and two rotary motors. By measuring the time of travel and the position of the two motors, the coordinates of a measurement are given in spherical coordinates.

Table 1: Specifications for the used LiDAR sensor.

Field of View (H / V)	120° / 26°
Minimum Measurement Range	2 m
Maximum Measurement Range	> 275 m
Frames per Second	5

The details of the LiDAR sensor are given in Table 1. The sensor outputs azimuth, elevation and range which can be converted to a orthonormal local x, y, z coordinate system.

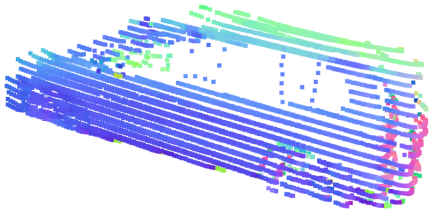


Figure 11: A LiDAR frame where a car has been doubled.

Because of how the LiDAR operates, a small time difference is introduced between each point measurement. If a moving object has a sufficient angular velocity in relation to the LiDAR, said object can result in multiple detections in the same frame, effectively duplicating the object. This can be seen on the back of the car in Figure 11. The severity of this effect depends on the angular velocity of the object, the resolution of the point cloud and the scan pattern of the LiDAR.

Table 2: Specifications for the used camera.

Field of View (H)	Unknown < 120°
Frame per Second	30
Image resolution (W × H)	2016 × 1512

The camera is a fairly standard video camera that films with the specification given in Table 2. The camera contains onboard modules for barrel distortion correction, tone mapping and data compression. This meant that we did not receive the raw data.

3.2 Datasets

3.2.1 Collection

The data for this thesis was collected by using a camera-LiDAR setup already constructed. A sketch of the camera and LiDAR location in relation to each other can be seen in Figure 12.

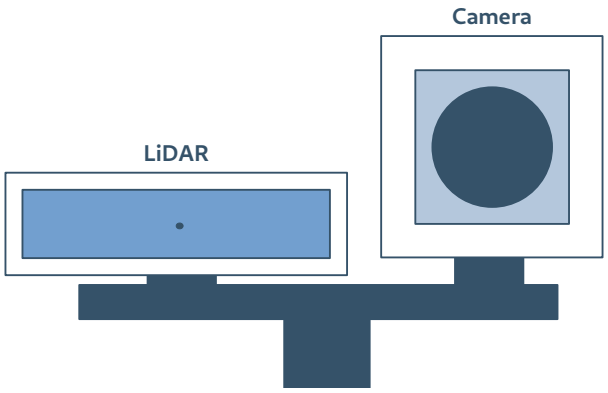


Figure 12: LiDAR-Camera pair used to record all data used in this thesis.

Two different scenes were captured with this setup. The first scene was of a smaller roundabout, see Figure 13, from which a couple of minutes of naturally occurring traffic where recorded. The second scene was more artificially constructed. The camera-LiDAR pair was placed on an empty parking lot, and a car was instructed to drive around in a circle in front of it. A couple of rotations of the car at different speeds were then recorded.

Finally this same car was recorded by moving a phone camera at an approximately equal distance from the car. This allowing the result to be compared to one achieved by more classical photogrammetry. One of the first task was using this separate sequence to train NVIDIA's I-NGP. Thus producing an optimal version of what is hoped to be achieved by the other methods.

3.2.2 Scenes

The first scene can be seen in Figure 13. Due to the middle of the roundabout being elevated, both the camera and the LiDAR had a hard time capturing the other side of the roundabout. The LiDAR-camera pair was positioned five meters above ground and about five meters away from the closest part of the road. Two minutes of the scene was recorded.

Figure 13: Snapshot of camera video and LiDAR point cloud captured for the roundabout scene.

The second scene, that will be the main subject of this thesis can be seen in Figure 14. Here the LiDAR-camera pair was positioned slightly lower; three meters above ground, and the car drove at closest two meters from the camera, slightly outside of the LiDAR's field of view. For this scene we recorded for two minutes and had the car first drive by twice at varying speeds. Then we had it make one and a half full rotations in front of the sensors. The rotation of the car happened in 115 frames which will be the ones used to aggregate the point cloud over time.

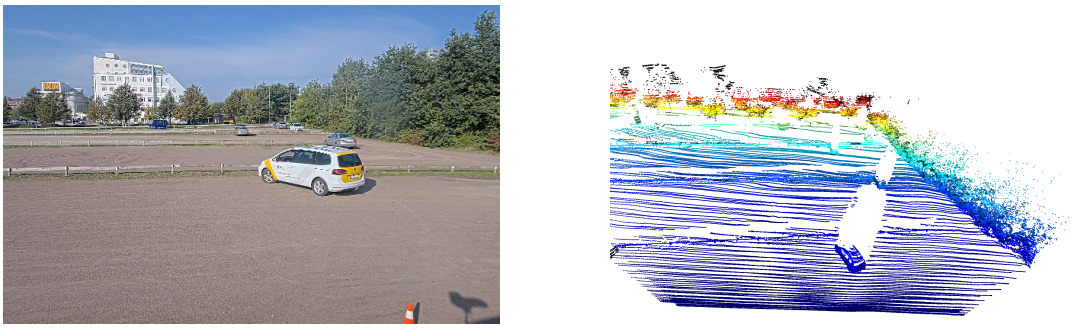


Figure 14: Snapshot of camera video and LiDAR point cloud captured for the parking lot scene. The car is here seen in the middle of the full rotation.

3.3 Processing Pipeline

In Figure 15 we can see the full processing pipeline that we have the data go through for this thesis. Each of these steps will be explained in more detail in this section.

3.3.1 Clustering and Tracking of Point Cloud Data

The very first task tackled was separating out the moving objects of a scene from the static background, and subsequently clustering and tracking these objects.

Detection of moving objects was done by computing the point cloud distance between the current frame and other frames in the point cloud sequence. For this the nearest neighbour distance was used. To account for changes in the background model, the point cloud distance was calculated against a number of frames before and after the given frame. For the first few frames only future frames were used, and similarly, for the frames at the end of the sequence only past

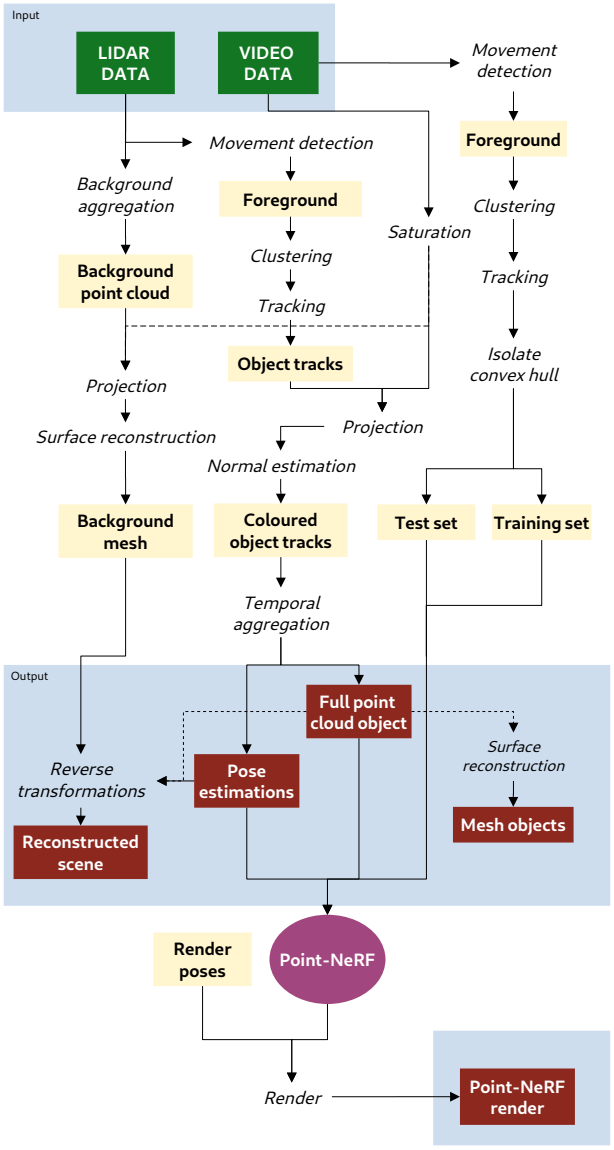


Figure 15: Illustration of processing pipeline from the LiDAR and video data input to the variety of results presented in this thesis.

frames were used. This modified distance was then thresholded with an appropriate value that was settled upon after testing.

To account for the spreading of the LiDAR detections farther away from the camera, the threshold set for determining distance from the median, and thus if an object is moving or not, was modified according to

$$D * 0.1^{z_i/z_{max}} > 0.2, \tag{13}$$

where D for the point cloud was calculated according to

$$D = \text{median}\left(\sum_{i \in A} \min_{j \in B} d(x_i, x_j)\right), \tag{14}$$

where A denotes the target point cloud, B denotes the median point cloud, x_i is a point in the point cloud, and $d(a, b)$ represents the euclidean distance between point a and point b.

For clustering the moving segments of the point DBSCAN was used, as outlined in Section 2.2. The parameter values were adjusted for each scene according to how far away from the objects the LiDAR was positioned.

Finally the clusters were tracked throughout the entire scene using the tracking algorithm outlined in Section 2.3. During the tracking an approximate movement vector for each cluster in each frame was calculated by taking the vector between the centre of the previous cluster in the track and the centre of next cluster in the track. If the target cluster was last or first in a track the movement vector was estimated to be the distance between its centre and that of the previous or next frame.

Completed tracks that were above a certain length and above a certain amount of points were kept, and the others considered noise or too lacking information and thus discarded.

3.3.2 Camera Projection

As the setup of the LiDAR sensor and the camera was one that had already been used by the company before, a very approximate projection matrix was already known. This matrix was tested by continuously projecting down the point cloud clusters and making adjustments manually. A tool for manually adjusting the values of the projection matrix with immediate feedback of the projection of the point cloud onto the image was written, which was then used to fine tune the projection matrix. This matrix was for later use split into its extrinsic and intrinsic components via PQ-factorisation.

For both scenes a synchronisation between the LiDAR frames and the camera frames had to be found. This was done manually by selecting a LiDAR frame and finding the camera frame for which the projected point cloud had the best fit. As there were six camera frames for each LiDAR frame, the direction of this operation ensured a better fit. An equation for synchronising the sequences could then be settled on, assuming that both sensors are consistent in their frame rate. Finally, before projecting the point cloud onto the image and fetching a colour value for each point, the images were saturated.

All RBG values can be converted into other colour spaces. By changing the basis from the orthogonal RGB space to a cylindrical coordinate system one gets the HSV (Hue, Saturation, Value) colour space, where hue is determined by the angle, saturation is determined by the radial distance and value corresponds to the regular z-coordinate. To saturate the images, this conversion was done and then the saturation value was multiplied with a factor 1.5. Lastly the values were converted back into normal RGB values. The reason this saturation was done was so that the colours would be more visible on the points considering that the resolution of the points is less than that of the pixels. Each point in each LiDAR frame could then be given a colour value.

3.3.3 Aggregation of Point Clouds

With the tracks of point clouds extracted and coloured the next task was to construct them as if they were fragments of the same detection. This proved a rather difficult task.

Before any transformations were done the normals to the points in the point clouds were estimated as outlined in Section 2.1, and then oriented towards the camera for each frame. The first and last frame of the track was also not used as these had poor movement vector estimations.

The unmodified point cloud segments were translated to have their centre of mass in the origin. An Iterative Closest Point algorithm, as outlined in Section 2.4.3, was then applied directly on these point cloud. This was attempted both with and without discarding frames with a poor fitness score.

To get a better initial transformation for the ICP algorithm, the next idea was to rotate the point clouds after translating their centres of mass to the origin. This was done by aligning the movement vectors that were calculated during the tracking, using the rotation matrix presented in Equation (4). For added precision these vectors were first smoothed over time and projected onto the ground plane.

Through trial and error an algorithm for doing the initial alignment of the point clouds was settled upon. This algorithm is outlined in Section 2.4.2. The parameters for the ICP algorithm were also tweaked until satisfaction was reached.

3.3.4 Background Model Computation

To separate out a model for the background all of the coloured LiDAR frames were first added together. By then removing statistical outliers in this aggregated point cloud we will have removed all of the moving parts of the scene. This as the background areas will have a much higher point density when added together. This point cloud was then downsampled to a suitable voxel size to minimise the amount of points in it. A range of value for the outlier removal and the voxel size were tested before ideal values were settled on for each scene.

The background point cloud was then coloured by computing a median image of 100 video frames. This image was saturated and the points were projected onto it as described in Section 3.3.2.

3.3.5 Surface Reconstruction

Before moving on to the Neural Radiance Fields a couple of methods for surface reconstruction were tested for both the aggregated moving objects and the background.

For the background surface reconstruction, both Poisson surface reconstruction as described in Section 2.6.2 and alpha shape reconstruction as seen in Section 2.6.1, were applied to the background of the roundabout scene. The values of the parameters were varied and visually evaluated to find a good fit. The chosen method was then also applied to the background of the parking lot scene.

For alpha-shapes, the value of the α parameter had to be chosen. When it comes to Poisson surface reconstruction, implementations of the algorithm have a different parameters that can

be tweaked to reached the desired result. In the graphics library used here the most important such parameter was *depth*, which defines the depth of the octree data structure that is used to reconstruct the surface. A higher depth will result in a more structured reconstruction while also increasing the computation time markedly.

When attempting surface reconstruction of objects, two cars were used. One being our main car object from the parking lot scene and the other one being a car from the roundabout scene where in 55 frames, three of the sides were relatively well captured. This car did however suffer from frame doubling, as illustrated in Figure 11, and the colouring was not well applied and was thus disregarded.

For these cars both alpha shapes and Poisson surface reconstruction was applied with varying parameters values and the level of detailing and clarity was compared by simple observation.

3.3.6 Image Processing

In order to train a NeRF-network the object's video sequences needed to be extracted. This was accomplished by first converting all images to grey scale, applying a Gaussian blurring kernel to all video frames in a sequence and then constructing a background estimation such that

$$I_B = \text{med}\{I_{\text{seq}}(x, y)\}, \quad x \in [0, W), y \in [0, H), \tag{15}$$

were I_B denotes the computed background candidate, med{...} is the median operation, I_{seq} is a sequence of images, (x, y) is the image coordinates and (W, H) is the image size. By taking the median value of each pixel in an image sequence, a background estimation could be constructed. By then computing the grey scale colour distance between the computed background estimation, I_B, and the current frame, I_n, and then applying a threshold, T_h,

$$|I_n - I_b| > T_h, \tag{16}$$

a binary image representing potential movement in the image could be constructed. For every image the coordinates of pixels that fulfilled the threshold in Equation 16 where then extracted and DBSCAN was applied per image. Small clusters where discarded and the remaining clusters where compared with the clusters in the neighbouring image. If a cluster in a neighbouring image was close enough, they where deemed to be of the same object track. After repeating this process for the whole image sequence, object tracks that where to short where discarded.

Lastly the convex hull of each frame in the object track was computed, which was in turn used to extract the relevant part of the image for that object.

To have something to compare our Point-NeRF results with, we used this extracted video sequence as a classic sequence of different views of an object and fed it to I-NGP. We also extracted just the point cloud that was produced by the COLMAP step of the pipeline.

3.3.7 Point-NeRF

In the paper on Point-NeRF [15] it is mentioned that using already computed COLMAP data is a valid option. The implementation by Xu et al. allows for bypassing the usage of the 3D-CNN $G_{p,\gamma}$ and instead manually initialising the locations and confidence values of the neural point cloud. We therefore initialised the neural point clouds locations and confidence values by extracting the locations of the aggregated point cloud from Section 4.2 and setting the initial

confidence value for each point to $\gamma_i = 0.3$.

The camera pose in relation to the point cloud is the same transformation as were applied to each point cloud frame to align it with the others. To this transformation the extrinsic part of the projection matrix was then added.

Before the method could run, a variety of parameters had to be changed to accommodate for the variable distance to the object and the larger scene of interest. The data set consisting of the clustered and extracted frames of the video sequence were then split into a training (80%) and a testing (20%) set. This division was done by random. The network was trained on this data for a variety of number of iterations.

Finally rendering poses were calculated by creating a circle around the objects at different angles depending on how novel of a view was desired. These camera poses were given to the network that then rendered the object.

3.4 Evaluation

Since the goal of this thesis is to improve the visualisation of point clouds, the different methods are only judged by simple observation by the thesis authors. The evaluation image sequence, the parking lot scene, was taken in a controlled environment where a vehicle was driven around in a circle. The methods that are compared were a projection of the colourised point cloud, a kernel smeared version of the projected colourised point cloud, and a Point-NeRF render. I-NGP was not included in the final evaluation set since we were unable to train a model that managed to correctly represent the geometry of the test vehicle.

The kernel smeared version was created by adding nine images, of the original colourised point cloud projection, with a one pixel offset. After normalising the image the process was repeated once more.

4 Results

The initial training and rendering of the separate optimal video sequence can be seen in Figure 16. Notable for these images is that the ground has been included as well due to the car standing still and thus no movement detection being done.

Figure 16: Result of running NVIDIAs I-NGP on a phone camera video sequence on a stationary vehicle.

For the clustering the parameters for DBSCAN had to be set appropriately. In Table 3 the chosen parameters can be seen for the different scenes as well as the approximate least distance to the clusters in the scene.

Table 3: Optimal parameter values for DBSCAN for the different scenes.

Scene	ε	*min_points*	Approx. min dist. to objects
Parking lot	0.9	100	2 m
Roundabout	1.5	40	6 m

4.1 Projection

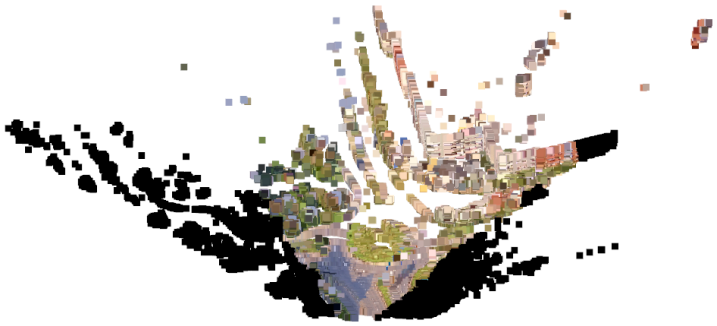


Figure 17: Illustration in the difference of Field of View for the LiDAR and the camera. The black points have been seen by the LiDAR but end up outside of the video frame after projection.

After adjusting the projection matrix it became clear that the cameras previously unknown FOV was well below that of the LiDAR. This can be seen in Figure 17, where the black points are outside of the cameras view. Henceforth the back points will thus be discarded from the data set and not used in point cloud aggregation.

4.2 Point Cloud Aggregation over Time

For the point cloud aggregation over time the results were many and varied. In Figure 18 a single frame cluster where colour has been added from projection can be seen. This is one of the many frames that were then fused together. Note that the point size has been increased slightly in this result compared to the later ones. Note also the reflection in the point that stems from the estimation of the normals that is done in each frame.

Figure 18: A single coloured LiDAR frame of a car.

In Figure 19 the result of the pre-ICP algorithm can be seen. Here each cluster frame has been rotated and translated to match up toward each other as well as possible. The frames' rotation has not aligned them perfectly due to variations in the estimated movement vectors. The translation is also somewhat skewed on frames where a smaller part of the car is visible to the sensor.

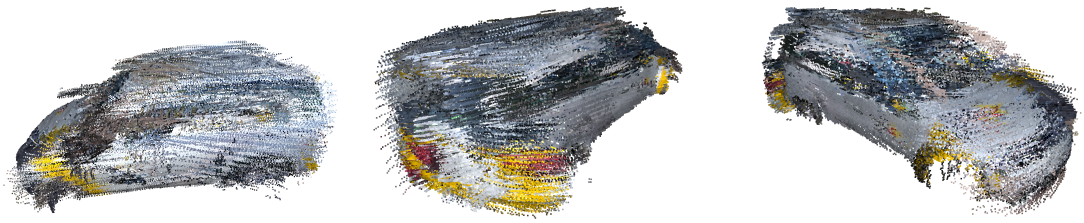


Figure 19: Point cloud aggregation through only translation and rotation, no ICP algorithm applied.

After having the ICP algorithm iteratively register the frames of point clouds after they have been pre-aligned as in Figure 19, the car looks as in Figure 20. Note that the colours are not consistent across all surfaces and that the colours are not always in the correct place. This can particularly be seen on the muddled details on the back of the car, and the yellow right front wheel respectively. Note also the details that can be seen inside of the car. Upon closer

inspection both the interior seating and the driver can be discerned fairly well. An attempt to illustrate this can be found in Figure 21.

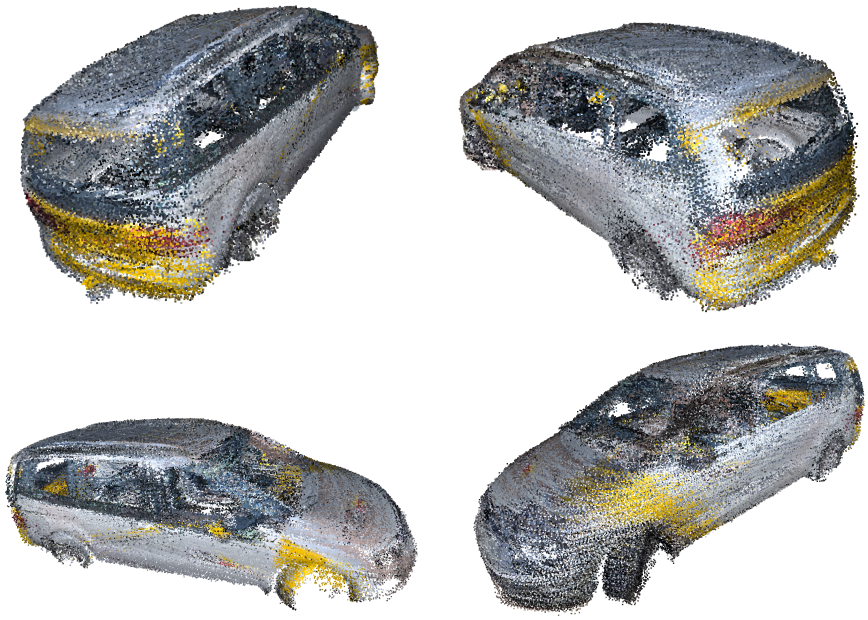
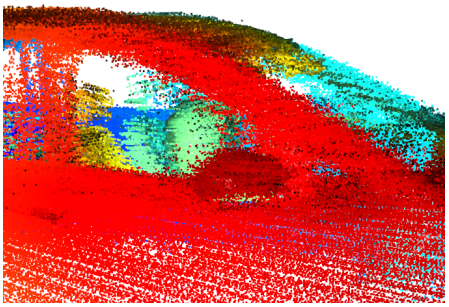
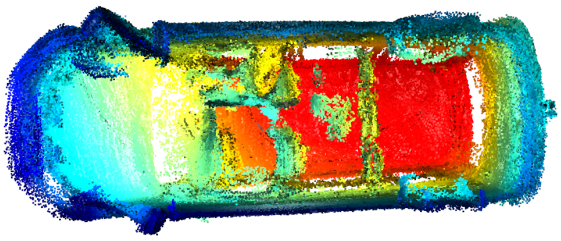


Figure 20: Point cloud aggregation with both initial frame alignment and ICP applied to car.



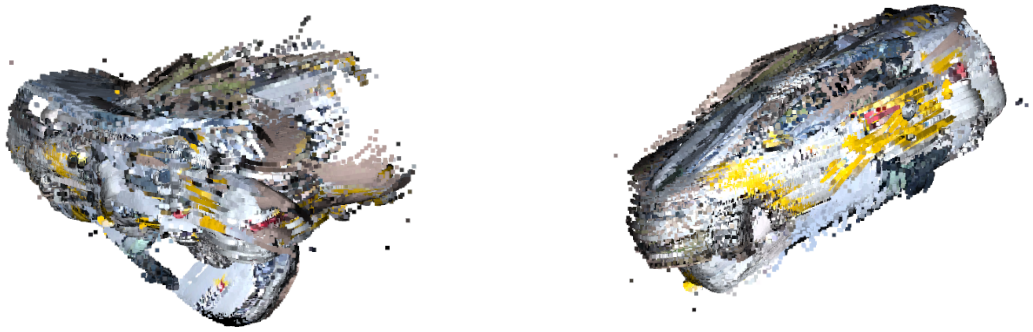
(a) A closeup of the view through the passenger side window of the aggregated point cloud car.



(b) The aggregated point cloud seen upside-down.

Figure 21: Other views of the aggregated point cloud car from Figure 20 coloured by z-coordinate value for increased clarity and contrast.

In contrast, Figure 22 illustrates the result of only applying the ICP algorithm. In Figure 22b no frames have been removed, no matter the fit and in Figure 22b the same pruning conditions applied in later rounds resulted in 23 out of the 113 frames being removed.



(a) ICP without discarding frames with a low fitness score.

(b) ICP and discarding frames with a low fitness score. Here 23 out of 113 frames were skipped.

Figure 22: Point cloud aggregation using ICP without doing an initial alignment of frames.

We found that the ideal parameters for the ICP algorithm was using a loss criteria threshold of 0.5 and additionally setting the maximum amount of iterations to 200. Moreover, the point-to-plane distance was used.

The same procedure as in Figure 20, can in Figure 23 be seen on other object, this time in the other scene. These vehicles have only been seen from one or two sides and are thus missing their other half. Note especially the seating in the back of the yellow bus.

Figure 23: Point cloud aggregation with both initial frame alignment and ICP applied to other object in the roundabout scene with ground truth images for comparison.

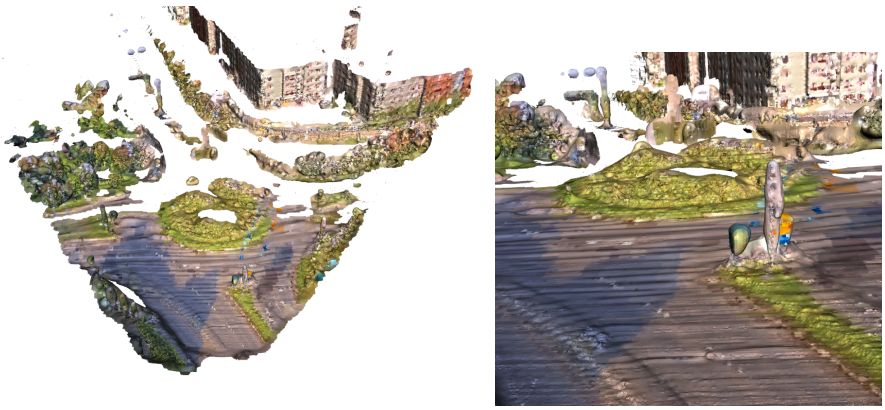
4.3 Surface Reconstruction

4.3.1 Background

For the removal of the statistical outliers in the creation of the background point cloud, the 20 nearest neighbours of each point was considered when calculating the average distance to other

points, and the standard deviation threshold for the distance was set to 0.1. Then for the voxel downsampling of the point cloud a voxel size of 0.1 was used.

After having processed the roundabout scene, the mesh surface reconstruction of the background in the scene can be found in Figure 24, 25 and 26.



(a) (b)

Figure 24: Surface reconstruction of the background in the roundabout scene using Poisson surface reconstruction with depth = 14.

In Figure 24 Poisson surface reconstruction has been performed with a relatively high value on the depth parameter. In Figure 24b we can, to the right of the picture, see how a lamppost and signs have been reconstructed. This can be contrasted by Figure 25 where the same lamppost can be seen in Figure 25b, this time much more shapeless. Here the same method has been applied but with the lower value on the depth parameter, resulting in a much more bubbly appearance.

(a) (b)

Figure 25: Surface reconstruction of the background in the roundabout scene using Poisson surface reconstruction with depth = 9.

The alpha shape method was also applied to this background. In Figure 26 the value of $\alpha = 4$

has been used. The lamppost can be seen in the bottom right corner as a mound. The holes in the road represent where most likely the distance between the LiDAR lines has become larger than the set α value.

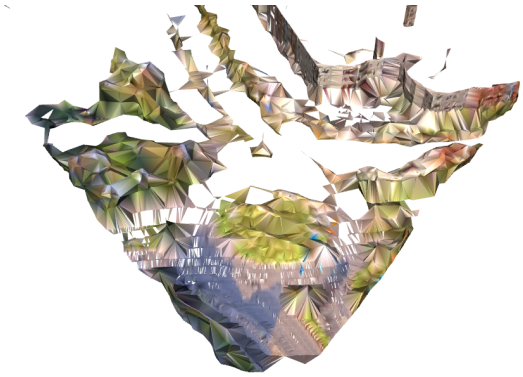


Figure 26: Surface reconstruction of the background in the roundabout scene using Alpha shapes with $\alpha = 4$.

As the result in Figure 24 was deemed the most detailed, this method was applied to the other scene. The result of this can be seen in Figure 27. In Figure 27b a closeup of a parked car in the background can been seen as well as the small fence separating the parking lot rows. The varying amount of details in these scenes come with varying computational costs, which can be found in detail in Table 4.

(a) (b)

Figure 27: Surface reconstruction of the background in the parking lot scene using Poisson surface reconstruction with depth = 14.

Table 4: Computational time for the different methods of surface reconstruction across the two scenes.

	Roundabout (134 939 points)	Parking lot (59 217 points)
Poisson, depth = 14	37 s	23 s
Poisson, depth = 9	3.0 s	3.8 s
Alpha, alpha = 4	2.4 s	0.84 s

Lastly we turn our attention to Figure 28. Here the point cloud from Figure 20 has been placed into the scene again for easy comparison with Figure 14.

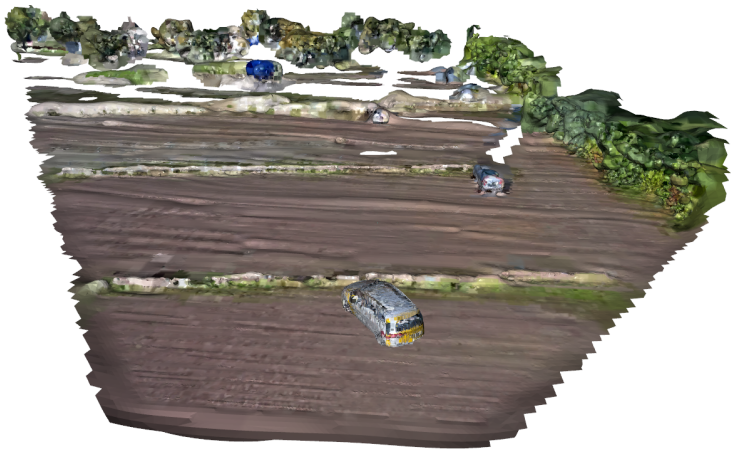
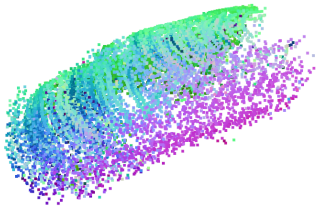


Figure 28: Fully reconstructed parking lot scene from Figure 14 with the background mesh from Figure 27, and car point cloud from Figure 20 repositioned into its original trajectory in the scene.

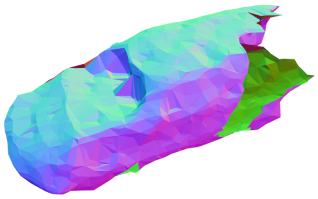
4.3.2 Objects

For surface reconstruction of objects we tried the methods on two different car point clouds, one more dense and one more sparse. In Figure 29 the results for the sparse point cloud are presented. In Figure 29a the point cloud itself can be found. This is a car that has driven toward the sensors in the roundabout, and thus all but the back side has been captured. However, the amount of points is much lower than that of the main car object (see Figure 20). Figure 29b and 29c present the surface reconstruction for this sparse car using the alpha shape and the Poisson method respectively. Parameters have been set to what was deemed to be the best value for this specific object.

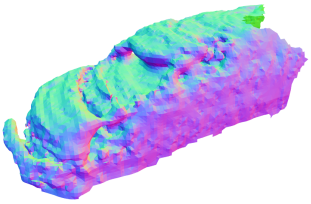
An interesting detail to note is the green patch at the back of the car in Figure 29b. The alpha shape reconstruction has turned that section of normals inward despite the normals of the point cloud pointing outward. This has not, however occurred in the Poisson surface reconstruction if Figure 29c.



(a) Aggregated point cloud for a car in the roundabout scene. 6 out of 55 frames in the track have been skipped in registration process.



(b) Alpha shape of the aggregated point cloud. Outliers have been removed and $\alpha = 0.2$ in the reconstruction.



(c) Poisson surface reconstruction of the aggregated point cloud. The depth parameter has been set to 7.

Figure 29: Surface reconstruction on a more sparse point cloud. Colouring have been set to represent normal direction.

The same test was done on the main car object and the results can be found in Figure 30. Due to the difference in density, the parameter values for the different methods have been changed to give a better representation.

(a) Aggregated point cloud as seen in Figure 20.

(b) Alpha shape of the aggregated point cloud. The α parameter has been set to 0.05.

(c) Poisson surface reconstruction of the aggregated point cloud. The depth parameter has been set to 8.

Figure 30: Surface reconstruction on the main car object point cloud.

4.4 Video Sequence Extraction

Figure 31: Three of the extracted video frames for the main car object.

The processing, movement detection, clustering, tracking and convex hull extraction, of the video data resulted in frames as shown in Figure 31. Note that the shadows of the object have ended up in the final cropped image as well.

When feeding only this isolated video sequence to COLMAP it produced the result seen in Figure 32. Here it has estimated camera poses and points for each frame, but not quite succeeded in putting it all together.

Figure 32: Point cloud produced by COLMAP using only the information from the image sequence and estimating camera poses via photogrammetry.

Going through the full process of giving this same video frame sequence to NVIDIAs Instant NeRF resulted in Figure 33. This can be contrasted against the use of I-NGP on the more optimal video sequence, the result of which could be seen in Figure 16.

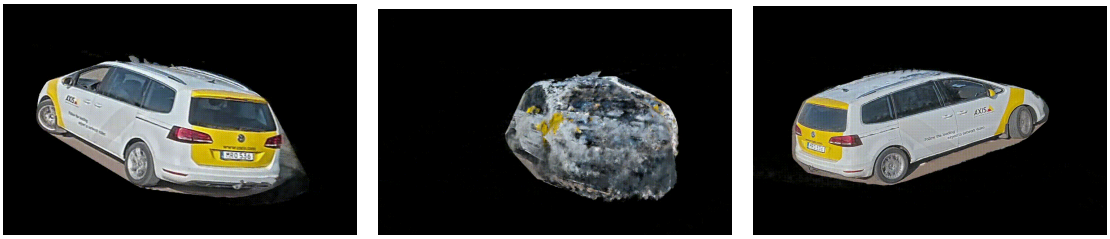


Figure 33: Result of running NVIDIAs Instant NeRF on the clustered and extracted main camera video sequence. Car seen from the right side, from the front and from the left side in that order.

It becomes apparent that the issues in the COLMAP point cloud caused some major disturbances in representing the front of the car. Moreover, since this was done on a sequence with the object isolated there is minimal to no ground present in this render.

4.5 Point-NeRF

From the transformations that aligned the frames making up the car in Figure 20 the relative camera poses look as in Figure 34. Since a couple of frames were discarded in the point cloud aggregation, the corresponding camera poses will be discarded here also. They have been marked in pink in the figure.

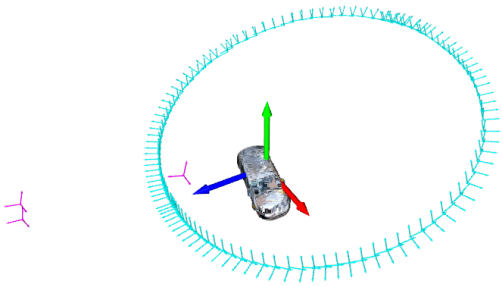


Figure 34: Camera poses calculated from point cloud registration. The pink poses are the ones that were skipped in the registration process and thus discarded. The turquoise are the ones included.

The non-discarded camera poses could then be fed to the point-NeRF network together with the point cloud in Figure 20. The network was trained for a varying number of iterations with tests being done every 10 000 iterations. It was judged that in this case the ideal number of iteration was around 10 000 iterations. The rendered car then looks as seen in Figure 35. The network was trained on images with a white background, and the rendering have also been done with a white background.

Figure 35: Volumetric render of Point-NeRF network trained for 10 000 iterations on an aggregated point cloud without colouring, together with the camera poses and the extracted image sequence.

In order to test how well Point-NeRF can generate very novel views, the network was tasked with generating wholly unseen views. These can be seen in Figure 36. The corresponding camera poses we can for certain say that the network has not trained on.

Figure 36: Render of the same network as in Figure 35 from camera positions further away from the ones in the training data set.

Figure 37 clearly illustrates the issue with the projection matrix. Here the networked has trained for a large number of iterations and thus both tried to point-grow to fill in the part of the image that does not overlap with the point cloud, as well as having trained parts of the point cloud on the background of the image. These issues can be seen on the roof of the car and on the back of the car respectively.

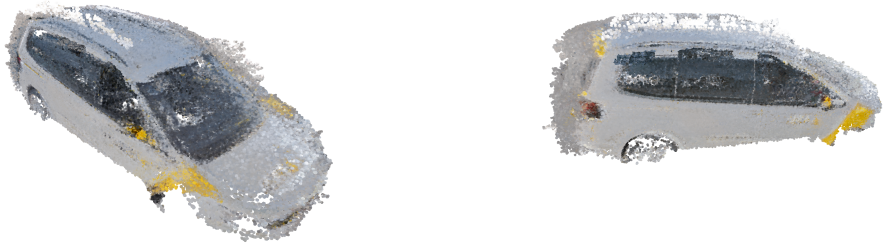


Figure 37: Point-NeRF rendering of a network that has trained for 710 000 iterations.

The sequence of both LiDAR- and camera-data was then cut in about four to simulate a more realistic scenario. The poses used can be seen in Figure 38 along with the aggregated point cloud which clearly can be seen missing a side.

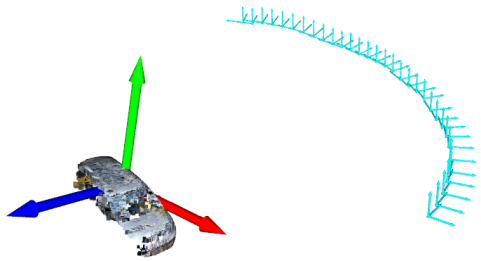


Figure 38: Camera poses for a quarter of a rotation calculated from point cloud registration.

After calculating these poses the network was trained on this reduced point cloud with the corresponding camera poses. The result of which, after the same amount of iteration as was deemed ideal before, can be seen in Figure 39. The network has been trained on the side of the car where the radial distortion was the least distorting.

Figure 39: Point-NeRF rendering of half of a sequence of point cloud and video frames after 10 000 iterations.

The last step in the application of point-NeRF was to use another object. The chosen object was the truck that can be seen in Figure 23. Poses were calculated in the same way as for the other rounds of using Point-NeRF. The poses in question are found in Figure 40.

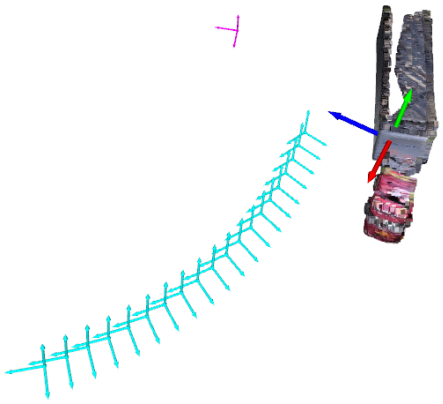


Figure 40: Camera poses calculated from point cloud registration. The pink poses are the ones that were skipped in the registration process and thus discarded. The turquoise are the ones included.

While a roundabout is round, all vehicles passing through it do not travel in a perfect circle. This means that the camera poses have a different pattern for the roundabout scene, as seen in Figure 40, in comparison to the parking lot test scene, as seen in Figure 34. They are also fewer view points for the roundabout scene than for the parking lot scene as the cars were driving by faster. Several of these poses also position the truck at the edge of the video images.

Using these poses Point-NeRF was trained on the image sequence extracted from the roundabout video. After training for 10 000 iterations the result was as shown in Figure 41.

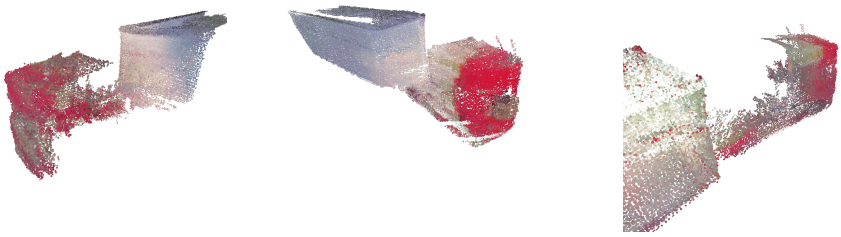


Figure 41: Point-NeRF rendering of the truck after 10 000 iterations.

4.6 Evaluation

The results of the evaluation of the different methods can be seen in Figure 42. Here a viewing angle from the test set has been taken. Since the test set images are close to the training set we can see that the overtraining issues are not apparent in Figure 42e.

(a) Ground truth.

(b) Projection of the colourised point cloud.

(c) Smeared version of the projected colourised point cloud.

(d) Point-NeRF render after 10 000 iterations.

(e) Point-NeRF render after 700 000 iterations.

Figure 42: Comparison of results.

The ground truth image from the test set is from when the car was in the middle of the video frame. Any offset issues are thus minimal here.

5 Discussion

5.1 The Results

Tracking

As a general rule, the methods used up until the possession of the final tracks in both video and point cloud data, are methods that for these purposes have been deemed good enough. There are however many better and more accurate methods for extracting this data.

One place where the space for improvement in the algorithms was most apparent was in the tracking algorithm. As our tracking algorithm only considers one frame at a time, the risk of tracks being split in several sections due to one or two poorly clustered frames is high. Adding a simple mechanic for matching up tracks that are deemed to be parts of the same track would improve the accuracy of the tracking.

Moreover tracking in two dimensions is much harder than in three. When clusters overlap it quickly becomes very hard to follow one continuous track. For the purposes of this thesis we generally had to splice together pieces of tracks that we manually found belonged together.

Point Cloud Aggregation

Over all, we are impressed by the quality that could be achieved by only doing point cloud aggregation of objects over time. As long as the individual detections themselves are of a high enough quality, the final result captures the shape and details of the object well. In this theis we have mostly presented the best results however, and it is important for the reader to note that there were several results where the detections were done from far away, or somewhat obstructed where the aggregated point cloud was hardly legible.

There were several tricky aspects of the point cloud aggregation process. One of them was knowing how much of the track to include, and subsequently, on which frame to start.

As a general rule in most scenes, the first and last few frames of a track does not include the entire object. This as it takes a few frames for it to enter and exit the scene fully. If we include these half-frames in our aggregation they are way more likely to end up in an incorrect position in the registration process. Since the full registration is done iteratively based on the first frame there is also a sizeable risk of not getting any result at all if the first frame is only a partial view of the object. Even though it means reducing the amount of information given we thus found it crucial to trim off a few frames at the start of every track. It is also possible to trim a few frames at the end, but we found that the way the registration works takes care of that by discarding any ill-fitting frames at the end without any further consequences.

The first and last frame in each track were however excluded each time as to be able to preform an approximation of the movement vectors.

Another issue of some complexity was choosing the parameter values for the ICP algorithm. Both finding general values that could act as a 'good enough' fit for all objects, but then also finding ideal values for each object. The ICP algorithm can be very fickle and depend heavily on the initial alignment. We also found that when letting the algorithm run for many iterations there was a chance of it drifting far from the initial alignment, something that was usually not

desired.

When it comes to applying colour to the aggregated point cloud there were two ways this could have been done. Either colour each frame and then put them together, or put all the frames together and then colour the aggregated point cloud from as few angles as possible. The advantages of the first method is that any offsets in the projection will affect the final result less as other angles can provide a more accurate colour. On the other hand, the second method would allow for a lot more detail to be captured while running the risk that the specific angle chosen has some major offset. As we throughout this thesis struggled with an incorrect projection matrix we decided to only use the first method for applying colour.

Something to note is that the point cloud aggregation method does rely on the cars moving, and more specifically, moving forwards. If a car were to stop, or even start reversing, the movement vectors would not help in the initial alignment of the frames anymore. Getting a result like the one in Figure 22 then becomes a risk. This could of course be compensated for by discarding a movement vector if it is too small and do something like account for a turning radius if the movement vectors were to change direction too suddenly.

Surface Reconstruction Methods

In their thesis [1], Afsén and Boye Frick come to the conclusion that Poisson surface reconstruction is the more appropriate method for background reconstruction. This lines up with our results. The background of a LiDAR scene has a tendency to have holes of very varying sizes, making methods like alpha shapes less appropriate. Moreover, details that remain in the background scene after the removal of statistical outliers are details that we want to have remain in the final reconstruction as they have proved themselves to be statistically significant. For example, lamp poles. Poisson Surface Reconstruction is much better at capturing this.

When it comes to aggregated moving object however, we find that the same does not apply. Due to the nature of the point cloud aggregation, it's not always certain that the final point cloud perfectly represents the object. If we turn our attention toward Figure 29 we can see this. The point cloud is from the roundabout scene and thus has the scan pattern doubling issue making it not completely uniform no matter how well the aggregation is done. The points have also been observed from a further distance and from less frames than our main car object. All of this results in detailing that does not generalise well. A rougher reconstruction can preserve the shape while not reflecting all the faulty detailing. Therefore, in this particular case, using alpha shapes to preform the reconstruction gives a more readable result than Poisson surface reconstruction.

However, if we look at Figure 30 we can see that in the case of a more dense and detailed point cloud we would make the judgement that the Poisson surface reconstruction is the more appropriate method. This as it does a better job of capturing individual details like the front tire and even the general shape of the driver.

Point-NeRF

As can be seen in Figure 42 a well trained Point-NeRF network is capable of generating a good representation of the object. In comparison to the other methods, listed in 4.6, it manages to generate the image that is closest to the ground truth.

In Figure 41 we can see the result of a less ideal output. There are three main issues here that are causing the distortion of colours. Firstly, the truck drives very near the camera and for a large portion was seen in a corner of the video frame. This caused some major radial distortion in the projection which we believe caused the beige parts of the drivers hut. Secondly, the projection could not be perfect since the truck point cloud is longer than the ground truth due to the frame doubling in the direction of movement. Lastly, there were simply a much smaller number of video frames to use as a training set and they were at very varying distances to the object.

In general we do not find that the usage of Point-NeRF was worth the added computation time in regards to the results it produced. To improve the visualisation of point clouds we believe that a simpler model, or just optimising the RGB values of a voxel subsampled aggregated point cloud, would suffice.

5.2 Complications

Half a Car

Due to the nature of cars and their tendency to not make full rotations in one spot, one very common occurrence when scanning cars with a stationary LiDAR sensor, is to only get half of a car. A typical case of this is illustrated in Figure 39, but also in Figure 23 where it is harder to see the missing side. Afsén and Boye Frick go more in depth on this issue in their thesis [1]. It is however something we in this thesis actively chose to not address. This mainly due to the fact that this would mean generating data in some way or another. In this thesis we have tried to not work generatively, but instead use as much of the data that has already been given to us.

One method we briefly considered exploring was mirroring of the point cloud. Since cars are generally relatively symmetrical this felt like it would produce fairly realistic results despite its non-generative nature. However, finding symmetry axes and registering mirror images together proved a more strenuous task than expected and was thus abandoned. It is, however, not unlikely to think that a robust method and some patience easily could accomplish the desired results.

Other methods that one could consider using would be for example the one presented by Mittal, Okorn, Jangid, and Held in 2021 [10]. As cars is one of the main objects of interest in the LiDAR community due to it's usage in self-driving car technology there are many good papers and trained networks that we believe could produce realistic point clouds as well.

All in all, this is not something that we in this thesis consider a major issue. Even when only seeing half of a car in a scene we found that it is generally clear what the object is in context. Especially if the point cloud has been appropriately coloured and aggregated to contain a larger number of points.

DBSCAN

As established in section 2.2 there are more appropriate clustering methods than DBSCAN, that were not used in this thesis. Using a method better made to deal with density variations would likely have given a better result of the clustering. On the other hand, most of the improvement would probably have been at the back of the scene, where the level of detail in the detections is so low that it most likely would not have been worth it.

A larger issue concerning DBSCAN has been when it has been used for clustering images. The clustering can then vary a lot in time depending on how many clusters there are in the image and how close to the camera they are. If there was suddenly one object very close to the camera, taking up most of the space and blocking the view of other objects, clustering a single frame could take up toward half a minute. This could be solved by adjusting ε and $min\_samples$, but then the values would not be appropriate for frames later or earlier in the sequence when that one object is not taking up all of the space. We thus believe that it would have been a good idea to pick another clustering algorithm for this purpose.

The LiDAR Scan Pattern Issue

One of the main flaws of the project was discovered after already having recorded the majority of the data. Due to the LiDARs scanning pattern there was an ever present issue in ever LiDAR frame: a doubling of the points along the direction of movement. We have not delved into how the sensor works in details but from what we have gathered the most likely explanation is that the LiDAR scans the same spot twice or more times, and in the time between scans the object has had the time to move a little. This was not an issue when the objects moved very slowly or when they were very far away, but since most object in our second scene were of the opposite category this ended up being a major hurdle.

The problem with this arose both in the point cloud aggregation and in the camera projection. In the case of the aggregation it became near impossible to register point clouds perfectly if their speed varied during the recording, this due to the distance between the doubling of the frames then being different. Moreover it was in general harder to align objects when every front plane and every back plane existed twice.

This also lead to the completed aggregated point cloud object being simply too long, which created problems in the camera projection. No matter what was done (except for scaling, something that we have attempted to avoid like the plague) either the front or the back of the point cloud would extend beyond the bounds of the object in the image.

If these experiments were to be reproduced, one would ideally choose a scan pattern for the LiDAR that eliminated this issue.

Shadows

When processing and performing motion detection on 2D images one common occurrence is having shadows of moving objects also being classified as moving. This of course is natural and expected, but does not lend itself to be synchronised with motion detection of 3D point clouds where shadows are very much not included. While we attempted to compensate for this, the camera seams to apply some form of tonemapping automatically. Which in turn meant that shadows where harder to detect.

The inclusion of shadows in the segmented object mainly caused an issue when the cropped image sequence is given as input to the Point-NeRF network. Due to the networks point grower it will over time slowly attempt to construct something out of the shadow as well, which results in something that looks like the point cloud is bleeding out in the edges if let train for too long. This can be seen in Figure 37.

Blackbox Camera and Projection Offset

The camera compensates for barrel distortion before saving the data as a video. What manner of operations that are applied to the image are unknown, meaning that any issues derived from this automatic distortion correction are hard to compensate for. When projecting the LiDAR data onto the image plane we noticed an offset between the image and the projected points that got worse along the edges of the image. We suspect that this is because of issues with the automatic compensation for radial distortions and seeing as we didn't have access to the data before the automatic undistortion, we where unable to compensate for this.

Issues With the Point-NeRF Model

The Point-NeRF model lacking any form of automatic camera pose correction meant that the model was very sensitive to errors in the computed camera poses. This meant that, because of the issues with projection offsets, the feature extractor network extracted incorrect features which in turn led to blurry images and that the generated images had a slight offset. In Figure 39 we can see that the details are generally better re-represented than in Figure 35, which is reasonable since the camera poses for the half car where generated from images further from the image border.

Another issue with Point-NeRF is that there is a white bias independent to the chosen background colour. This is a code issue in the dataset code, located in the *read_meta* function in *nerf_synth360_ft_dataset.py*, where pixels with a fully opaque value are hard coded to get the colour white. This meant that shading locations around neural points that where bad at representing the actual geometry where biased to return the colour white during the render step. We attempted to correct by adding a random colour to fully/partially opaque pixels in order to force the network to learn that these points should return a lower opacity, but this didn't happen. Instead the network got a bias to the colour grey which is simply the same issue.

5.3 Future Advancements

We strongly believe that there are several ways to continue upon this work in the future, and would have done so ourselves if time had allowed.

Perhaps the most obvious advancement would be to continue working on the camera calibration. To establish a system where both camera parameters and settings are fully known and thus can be exploited. First doing calibration on the camera and the LiDAR would greatly improve all of the parts in this thesis that have been less than ideal. We believe that if one were to implement this technique into a real system, calibration would be the absolutely most crucial step.

Point-NeRF

In the process of using the gathered data to train a Point-NeRF network there is a lot of space for improvement. As it stands right now, rendering even one view given the trained network takes up to a minute, making it a rather tedious task. It has also not been possible for us to place the final result back into the scene with for example, a meshed background. Having the network have the ability to produce some kind of final point cloud or even a mesh, would result in a more cohesive final result. The risk could then be that the resulting mesh may lose reflections and angle-dependant colouring, but the alternative would be trying to insert the background model into the network renderer, which would cause more of a headache for everyone.

In this thesis the point grower and pruner where, sparingly, used. But they where not systematically evaluated, adjusting the way and at what rate these work would perhaps result in a better final render. Given that the point cloud we want to use is in general more dense than the sparse point cloud COLMAP produces it is not unreasonable to think we would need less point growing. However, if, as in our case, the camera and the LiDAR are not perfectly calibrated one may want the point grower to work a lot to correct the offset.

Either way there are plenty of possibilities in things to tweak and try. We would also have loved to try applying other networks and comparing the results. For example, there was an idea of modifying the Points2NeRF method as presented by Zimny, Trzciński and Spurek in 2023 [17]. As this network is trained on point clouds, we feel it would not be a large alteration to also have it test and render point clouds instead of a typical NeRF volumetric render.

Multiple Sensors

To address the issue of the typical result being half a car we would suggest the idea of using the same setup as used in this thesis, but in two locations at the same time. If there could be one LiDAR sensor on each side of a road for example, one could capture the entirety of the structure of the car by having it drive by just once. If the sensors were well calibrated we don't see that there should be an issue in applying the same procedure to this data as has been used in this thesis. We are aware that at the time LiDAR sensors are very expensive, and that the ideal scenario includes only one, but we definitely believe that as LiDAR technology takes off and becomes more commonly used this will no longer be the case. In this scenario, using two LiDAR sensors does not seem unrealistic.

If this would not be possible, we also believe that just having several cameras in the environment would improve the visualisation. Since reconstructing a surface using only LiDAR data is generally possible even where the point cloud has holes, the final mesh could be more precisely coloured by having cameras capture different angles that may not be seen otherwise. This is also another way of addressing the differing fields of view and thus making the most out of all of the data captured by the LiDAR. Lastly, it would provide more data for the training of a potential network, and the network would avoid over-fitting itself to a specific sequence.

Live Application

Moreover, one development that would need to be done to have this technology be of use in real scenarios would be to develop a way of performing this pipeline semi-live. This is something that we early on in the process considered, but fairly soon gave up on. The movement detection could be done by comparing the frame with the last few frames if at the start there is a slight delay before tracking starts. Clustering should, in general, not take too much time, and if it does, one would just have to adjust the frequency of the LiDAR to have a frame rate that it can keep up with. As previously discussed, a better clustering algorithm may also help with this. The way the point cloud aggregation is done also makes it very much possible to have it happening live, as frames are iteratively layered onto the previous frames. However, this step also can get computationally heavy when ICP has to run a larger number of iterations.

Applying the pipeline up until the point cloud aggregation should work in a semi-live environment. Together with first calculating a background model and creating a mesh for it this could greatly improve the visualisation of point clouds in a system without having to do heavy post-processing.

Shading

In the purely visual realm, One thing that definitely could be worked on or added would be some sort of shading on the whole scene. The NeRF networks does do this in their angle dependant colouring, but the background mesh has no shading and the objects when places back into the scene have no shadow. This is in no way a strictly necessary improvement, but it would give a more realistic view of the reconstructed scene.

6 Conclusion

In conclusion we believe that performing temporal aggregation on rigid body point clouds will be a useful tool in the future. When taking care to optimise each step in the data processing, most importantly the camera calibration, we believe that you can achieve readable and clear results with only a stationary camera-LiDAR pair.

If the choice to preform surface reconstruction on these point clouds is made, we believe that Poisson surface reconstruction is better for denser point clouds, while alpha shapes can handle more sparse ones. However we have seen that the most benefit was gotten from performing surface reconstruction on the background, and that meshing the objects was less necessary for overall clarity.

When it concerns the NeRF networks we are very positive toward the level of detail achieved by the networks even on data with calibration issues as ours. Resolving these could result in some truly impressive renders. We are however less certain that this is something that will be applicable in many situations. We found that the level of detail might not justify the much longer processing time, especially if the data set sequences are on the shorter side.

References

[1] Afsén, Philip and Boye Frick, Kasper. Visualization of tagged items in a point cloud. In *LUP Student papers, LU-CS-EX*, 2023. http://lup.lub.lu.se/student-papers/record/9128176 [Acccessed: 16 January 2024].

[2] P.J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[3] Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. Density-based clustering based on hierarchical density estimates. In Jian Pei, Vincent S. Tseng, Longbing Cao, Hiroshi Motoda, and Guandong Xu, editors, *Advances in Knowledge Discovery and Data Mining*, pages 160–172, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[4] Doyle, Seamus and Nilsson, Gustav. Improving a Background Model for Tracking and Classification of Objects in LiDAR 3D Point Clouds. In *LUP Student papers, Master's Theses in Mathematical Sciences*, 2022. http://lup.lub.lu.se/student-papers/record/9090366 [Acccessed: 16 January 2024].

[5] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29(4):551–559, 1983.

[6] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

[7] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson Surface Reconstruction. In Alla Sheffer and Konrad Polthier, editors, *Symposium on Geometry Processing*. The Eurographics Association, 2006.

[8] Lind, Hjalmar and Holtz Bernståle, Robin. Segmentation, Classification and Tracking of Objects in LiDAR Point Cloud Data Using Deep Learning. In *LUP Student papers, Master's Theses in Mathematical Sciences*, 2022. http://lup.lub.lu.se/student-papers/record/9072762 [Acccessed: 16 January 2024].

[9] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[10] Himangi Mittal, Brian Okorn, Arpit Jangid, and David Held. Self-supervised point cloud completion via inpainting. *arXiv preprint arXiv:2111.10701 **[cs.CV]***, 2021.

[11] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022.

[12] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred A. Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *ICML*, 2019.

[13] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-Motion Revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[14] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise View Selection for Unstructured Multi-View Stereo. In *European Conference on Computer Vision (ECCV)*, 2016.

[15] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-nerf: Point-based neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5438–5448, 2022.

[16] Tianwei Yin, Xingyi Zhou, and Philipp Krähenbühl. Center-based 3d object detection and tracking. *CoRR*, abs/2006.11275, 2020.

[17] D. Zimny, T. Trzciński, and P. Spurek. Points2nerf: Generating neural radiance fields from 3d point cloud. *arXiv preprint arXiv:2206.01290* ***[cs.CV]***, 2023.

Master's Theses in Mathematical Sciences 2024:E6
ISSN 1404-6342

LUTFMA-3525-2024

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/