Bachelor's degree thesis

AI-assisted annotation of 3D sparse data

Leverage of lidar point cloud for segmentation and shape reconstruction

Louis Clergue

Supervised by:

Sanja Fidler (UofT)

In partial fulfillment of the requirements for the

Bachelor's degree in Informatics Engineering

Bachelor's degree in Mathematics

September 29, 2019

Abstract

AI-assisted annotation of 3D sparse data

by Louis Clergue

We study the usage of point cloud jointly with images to do interactive annotation of sparse 3D data. Our aim is to formulate a human-in-the-loop pipeline where we leverage deep learning models to assist and speed-up annotation. We do an overview of 3D representations and classic surface reconstruction algorithms. After, two models are presented: a point cloud (3D) centered one and an image based one. We study different kinds of sensor fusion to add point cloud information to our image based model. We see that point cloud helps improve performance of silhouette prediction but we notice that conservation of locality between 3D and 2D would achieve better results. We end presenting an overview of existing annotation tools both scholar and commercial, together with our current work in this direction.

Keywords: deep learning, computer vision, 3d, surface reconstruction, point cloud, annotation, self-driving

Preface

This report is the result of my work as a visiting research student at University of Toronto and Vector Institute in Toronto, Canada from February to July 2019. Professor Sanja Fidler from UofT has been in charge of supervising this work and Xavier Giró-i-Nieto has been tutoring it from UPC.

The efforts of this report fall under a bigger project of efficient 3d annotation which members have been of great help to the success of this work. I will mention Jun Gao for his active guidance , Amlan Kar for his ideas and pointers ; and , especially , Tianchang Shen as my coworker during most of my stay, whose contributions are tightly coupled with mine.

During this half-year I have learned to read, understand and replicate previous research works while gaining domain specific knowledge of the task. At the same time, I got hands-on experience of deep neural network implementation and its related task of data generation and preprocessing. From an engineering perspective, I got a taste of software development in the production of a web interface using our models.

It has been an overall very satisfying experience combining real academic research and collaboration with peers while also being culturally enriching thanks to living abroad in such a diverse city. I am pleased to present this work as the Bachelor's thesis of my undergraduate studies as they represent my experiments exploring sparse 3d data towards a new goal of using AI as a tool.

Contents

List of Figures

List of Acronyms

SOTA state of the art

CNN convolutional neural network

MLP multi-layered perceptron

SDF signed distance field

RNN recurrent neural network

GCN graph convolutional network

BEV bird's eye view

IoU intersection over union

IoT internet of things

Chapter 1

Introduction

1.1 Description and motivation

In the last 5 years deep learning approaches have taken over the field of computer vision leading to an explosion of performance. Such prowess has only been able to happen thanks to very large datasets like ImageNet [5] that provide enough observations to learn good models. It is known that having more data is better than increasing the complexity of a model. But collecting data is very expensive, especially if it requires manual human annotation, this is why making annotation faster and easier is a good investment.

Currently it is easy to obtain text or image data as they are the building blocks of human digital communication. This is not the case for 3d data. With the advent of 3d printers we expect that to improve with time but the results are still synthetic, like a drawing compared to an picture. Collecting real world 3d data requires very expensive setups of sensors and a good deal of engineering to clean, process and align the observations. It is inconvenient that our synthetic data is more abundant and easier to manage as we can take as much specific measurements as we want, but real data is only based on observation and is lacking in density (more about 3d data in chapter 2). Also, 3d sensors acquire huge amounts of data that would be impossible for humans to annotate individually. This leads us to create tools to improve and speed up human annotations.

Our main goal is to design and develop a pipeline that would allow a user to leverage AI for a faster and denser annotation. We aim at using sparse 3d data (point clouds) and images (2d projections of 3d data) to obtain a segmentation an a shape completion. Concretely we will

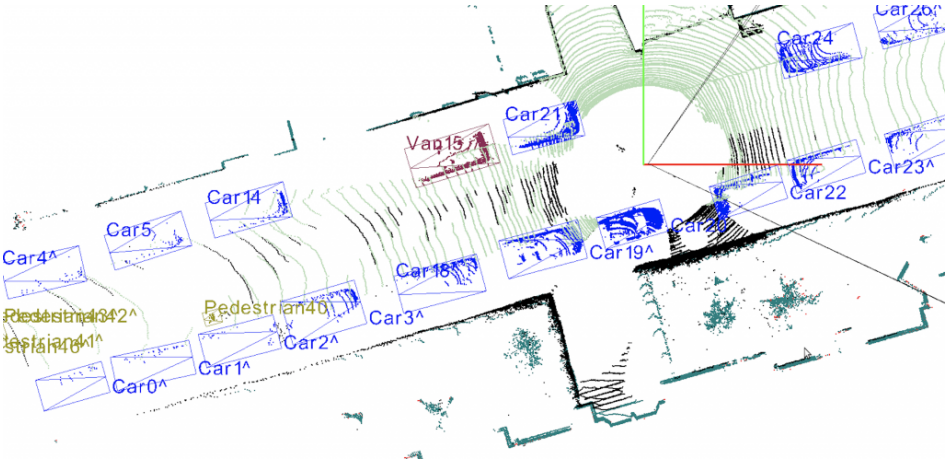


Figure 1.1: Example of a bounding box annotation of a lidar point cloud in top view. Courtesy of deepen.ai

focus on self-driving datasets with lidar (3d point clouds) and 2d images, where the results will consist of point segmentation, bounding box annotation and 3d shape estimation.

1.2 How will we address these tasks?

With the economical perspective of self-driving, most advanced 3d annotation tools are commercial products that do not publicly share their methods. Companies in this sector provide annotation tools and maybe some automated processes; just to name a few: Shapely, Playment, Supervisely. There is very little academic work that provide usable software for this task ([22], [12]). This makes our task more difficult as we can only leverage models from the literature and get inspiration from company demos.

Our approach needs to merge information from different sources: one is a sparse spatial sample (point cloud) the other is an image. So, our first step will be to study architectures that will be able to extract information from the point cloud and results from classic algorithms that try to reconstruct 3d shapes. We will show their strength and limitation but, due to their 3d nature, they do not use image information whereas learned-based models can.

Following those results we will propose two pipelines to annotate our data and define the user interaction step. Our first attempt is focused on spatial data as the point cloud as the main source of information and doesn't perform as well as expected; the second one is centered around a 2d image of the same object with extra information from the point cloud and is

able to get quite good results.

Currently 2d annotation tools are quite advanced and user friendly, but manipulating 3d data requires some adaptation (the introduction of VR could push for a more user friendly interaction like [18]). This is why we want the user step to occur in a 2d context, this is another challenge as the resulting annotation needs to be 3d.

Finally we will explain the incorporation of our models into a web application and give qualitative results of its usage. Due to the extent of this project, this tool will only provide a simple annotation user experience which can be improved upon for more advanced usages.

1.3 Overview of the next sections

In the next sections of this documents, we are going to talk about the following aspects:

- In chapter 2, we present the fundamental concepts of 3d sparse annotation and explain different data representations.
- In chapter 3, we mention previous work on the related tasks of 3d annotation: point cloud networks, sensor fusion and 3d shape completion.
- In chapter 4, we explain classic approaches to do 3d shape completion and point cloud alignment.
- In chapter 5, we describe our models for the proposed tasks and how they fit in the general pipeline.
- In chapter 6, we show experiments that have been carried out for this project: implementations and adaptations of existing models and evaluation of performance.
- In chapter 7, we discuss results and conclude.

Chapter 2

Problem formulation

2.1 Annotating

Probably the hardest thing in machine learning is obtaining data, the data in the wild is raw and non-annotated so we don't know its semantics. For example it is very easy to get millions of images on the internet, a much harder goal is to get an image and a text description. In this case the description is the **annotation** of the image and writing the descriptions for a set of images is **annotating** that dataset. As everyone can imagine this is very time consuming, therefore very expensive; this is the reason why tools that provide assistance to human annotators are so marketable.

The problem we are solving is the task of annotating sparse 3d data, this task can be broken down depending of the type of annotation. In our case, our input is a point cloud and maybe an image (with the corresponding projection). Classic annotations are point cloud instance segmentation and bounding box (which consists in position, orientation and extend of the object). A separate task is 3d reconstruction from either image or partial/uniform point cloud. In this work we focus on 3d reconstruction using a partial point cloud and one image.

We start by looking for previous works that leverage point cloud information and how this can be merged with an image of the same scene. Then, we try to adapt these models so that the user can be in-the-loop.

2.2 3D data representations

For this problem it is important to consider the format in which our 3d data is represented as it will directly affect our models. This section aims to introduce the one most commonly used in machine learning and explain their benefits and drawbacks.

2.2.1 Voxel

A voxel is the 3d version of a pixel, it's a discretization of R^3 in a grid. As pixel grids (images) store color or luminescence, a voxel grid typically stores occupancy or a distance field (representation of a surface).

$$V := \{v_{i,j,k} \mid (i,j,k) \in [0,W] \times [0,H] \times [0,D]\}$$

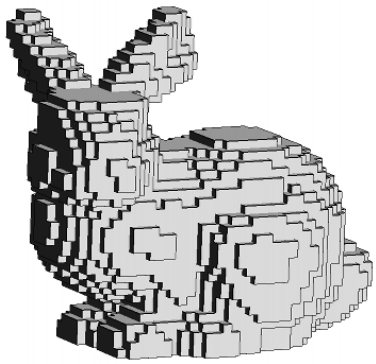


Figure 2.1: Voxel representation of the bunny.
Image from here, accessed Sep 2019

It is very easy to work with as it's possible to leverage models made for images by adding an extra dimension. The main problem is that the computational cost (time and memory) increases dramatically, and those models are therefore very limited by the resources available.

This cost is not really justified as, usually, most of the voxels represent empty space. And, the proportion of empty voxels increases greatly with the resolution (number of voxels per unit).

Octrees

The problem of uniform level of detail and wasted empty space can be reduced by introducing a more complex structure in our grid. Octrees are a space partitioning structure in which regions with more details are further subdivided. It can be thought as a compression of a voxel grid.

Octrees are hard to work with because they have a **fixed structure** which is hard to generate or to modify. And in any case require custom data structures and operations to incorporate in our models. Some works are based on custom convolution operations: [19], [24].

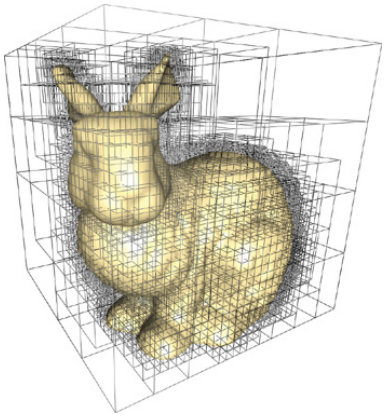


Figure 2.2: Octree partitioning around the bunny.
Image from here, accessed Sep 2019

2.2.2 Point cloud

We define a point cloud to be a collection of **unordered** points in a three dimensional euclidean space:

$$P := \{p_i = (x_i, y_i, z_i)\} \subset \mathrm{R}^3$$

This is an intuitive representation of 3d objects, we identify the object with the point cloud sampled in the volume or on the surface of the object. The major difficulty is that this is a discrete representation and there is no local topology (unordered points), the only neighborhood sense is from the embedding space with the euclidean distance. Also there is no unique representation of objects, parts can have denser resolution (which is useful for a variable amount of details).

The main difficulty with point clouds is the lack of structure as there is no sense of locality.

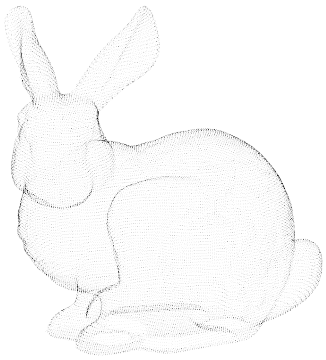


Figure 2.3: Regular point cloud of the surface of the bunny.
Image from here, accessed Sep 2019

But we solve the voxel problem of efficiency without having a fix structure, we can have more points where needed and a coarser sampling on simple regions. Also, our point cloud is defined as a set, which means that any traversal of individual points should be **order invariant**. We will present previous work that achieve this in chapter 3.

2.2.3 Polygonal mesh

The most common representation in graphics is the polygonal mesh. Which is a point cloud plus a fixed topology defining faces. The faces are polygons, typically triangles or quadrilaterals. Meshes are heavily used in graphics and are the best end product for generative models.

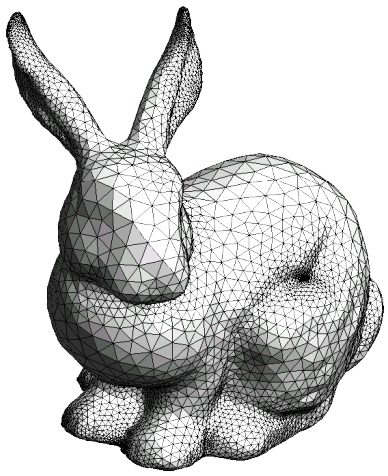


Figure 2.4: Triangular mesh of the bunny.
Image from here, accessed Sep 2019

This representation is very memory efficient and defines a surface (maybe disconnected) in space. The presence of structure gives us a strong sense of locality but also makes topological modifications really hard. Pixel2Mesh [23] is able to obtain a mesh from a single RGB image but can't introduce holes as it doesn't change topology.

2.2.4 Implicit representations

Implicit representations, as they name say, are implicit. This means that instead of representing the direct position and/or structure of objects they store another measure from which it's possible to infer 3d properties.

The most common one is the signed distance field (SDF). The signed distance field is the function that, evaluated on a point in space, gives the shortest signed (either inside or outside the object) to the surface. Therefore the surface is then the level set at 0 of this function hence the implicit formulation.

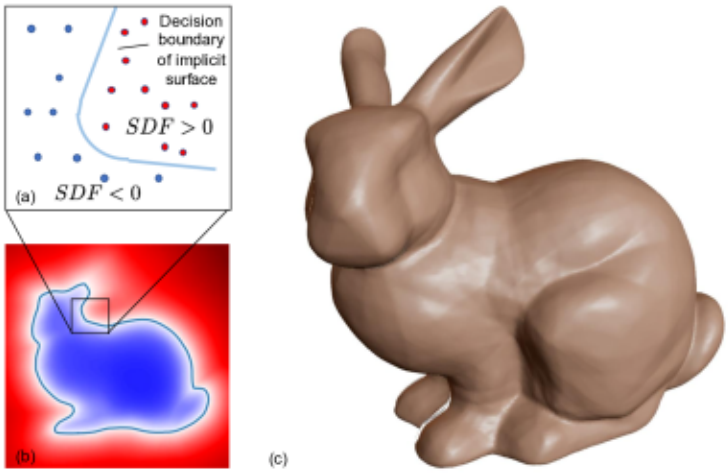


Figure 2.5: Signed distance field of the bunny.
Image from DeepSDF [13]

SDF has no fixed topology and it is easy to modify by classical mathematical operations. The difficulty lies in the implicit representation and how it is stored and manipulated. Putting a signed distance field in a voxel grid doesn't solve the problems of voxel grids. Having a model learn this function is more natural but how do we deal with it as a result ?

Chapter 3

Related work

In this section we will explain the type of models existing in the literature that work on point clouds and how is it possible to mix image (2D) and point cloud (3D) information. Then we will look at some state of the art (SOTA) for 3D reconstruction from which we take inspiration. Finally, we will show some works on assisted annotation and how we will use one of these models for our task.

3.1 Relevant models

3.1.1 Point cloud architectures

As presented in section 2.2.2 point clouds have special characteristics that can leverage custom architectures. Applying simple ideas and extending them to the properties of point clouds results effective.

The first simple idea is to apply a *fully connected layer* to our point cloud to obtain features. This is, given N points transform them into M features via a linear function. This looks very bad, for two reasons: we must fix the number of points before the layer and it is not order invariant in general. It is possible to avoid the problem of fixed input, assuming a good average size is feasible, by resampling the point cloud.

The work of **PointNet** [14] proposes an encoder for variable-size point clouds introducing two functions. The first computes a local feature for each point and the second generates a global feature from the set of local features. The global feature generation is invariant on the order of local features, enforcing that shuffling a point cloud will always result in

the same global feature. Also the local features are computed based only on the points' coordinates. Usually local features are results of a multi-layered perceptron (MLP) and the order invariant function is max-pooling. In order to give more power to the encoder they add transform layers to be more invariant to the specific MLP encoding.

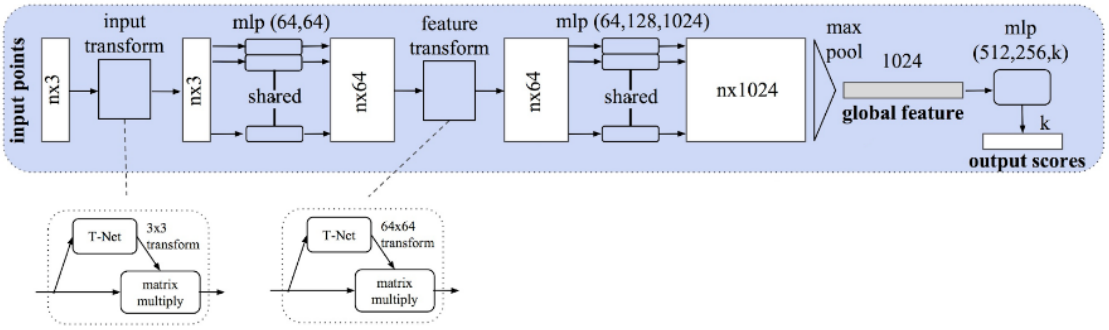


Figure 3.1: PointNet encoder.
Image from PointNet [14]

A drawback from this approach is that local features are only mixed in a global manner, further work like **PointNet++** [15] do iterative combination of local features by doing PointNet encodings of small clusters of points. This works well for segmentation of points by using local features together with the global point cloud feature.

A less prominent technique called *folding* from **FoldingNet** [26] works instead as a decoder. The main idea is that object are usually surfaces and could be obtained by deforming a 2D grid, the folding operation attempts to learn this deformation. Basically, from some resulting feature a MLP combines it with each point of a fixed 2D grid to obtain a 3D point. The resulting set of 3D points is the generated point cloud; for a more complex model, the operation can be repeated from that point cloud instead of the fixed grid.

3.1.2 Sensor fusion

The problem of sensor fusion, in this work, is to combine a 3D point cloud with a 2D image of the same scene in order to obtain more information. A point cloud has 3D information which is our objective but it's sparse and lacks features, an image only shows one view but is much denser and has color.

Assuming we know the pose and projection of the picture, we can project the points into the picture and have a corresponding pixel for each of them. Sensor fusion models use this extra

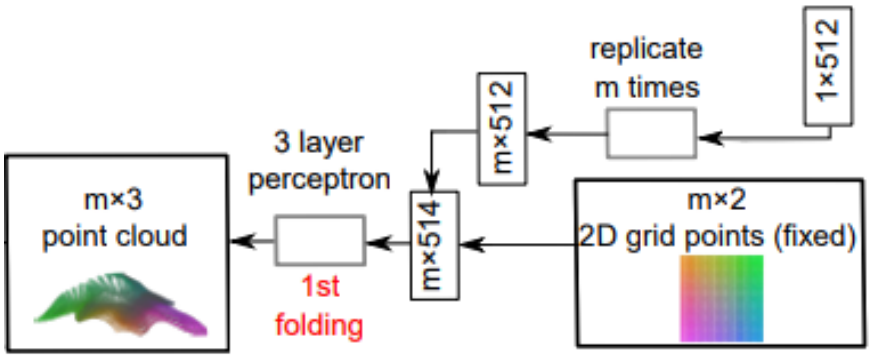


Figure 3.2: Folding operation.
Image from FoldingNet [26]

feature from the image to augment the information present in the point cloud. A very simple way is to concatenate a depth feature onto the image as an extra channel; another one is to add this feature with the 3D coordinates in a PointNet encoding (we try this approach in one of our models, see 5.3 and 6).

Deep Continuous Fusion for Multi-Sensor 3D Object Detection [9], a paper from Uber does the opposite of putting the point cloud in the image, it adds image features to the point cloud. Their model works with two streams, one based on the camera image and the other on a bird's eye view (BEV) image of the point cloud. Their fusion step is a way to augment the BEV image with features from the front image by using the 3D points as a bridge.

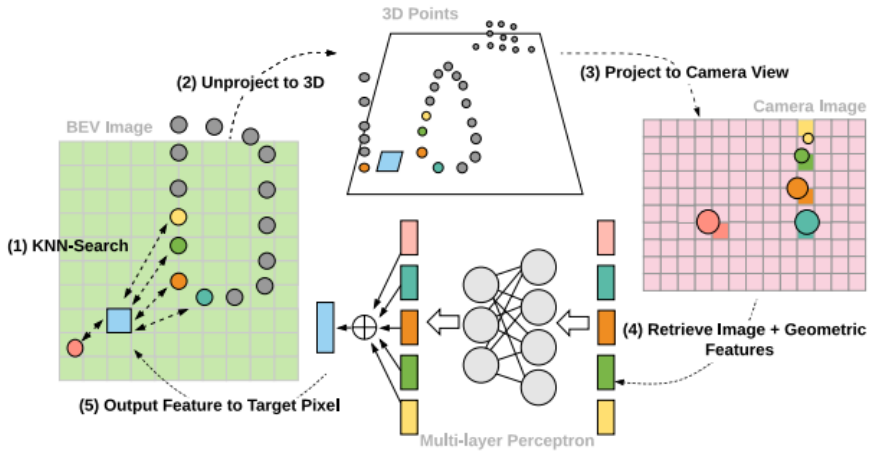


Figure 3.3: Continuous fusion layer. Steps in computing features for a target pixel in BEV
Image from [9]

In figure 3.3, the steps to compute features for a pixel is retrieve K nearest LIDAR points in

BEV (1), then transform the points from BEV to the camera plane (2) & (3), features are sampled on the image passed through the encoder (4) and a MLP does the fusion from the 3D offset of K points and their features to obtain output features on the target pixel (5).

My take of this work is that they only use 3d information to get some weight on nearby points, all the representations are 2D images which allows usage of time-proven convolutional neural network (CNN) architectures. Using the BEV makes sense because it's a view that's usually orthogonal to any camera image of the same scene; also, because they attack the detection problem, a top view is very useful and their output is not 3D data.

3.1.3 3D reconstruction

3D reconstruction is the task of obtaining a 3D representation based on incomplete input, typically an image. We will present a SOTA model (at the time of writing) that obtains a SDF from an single view: DISN [25]. This paper has the interesting formulation of computing a single SDF value of a point in space and an image, of course it is possible to get multiple points by batching the forward pass of the model.

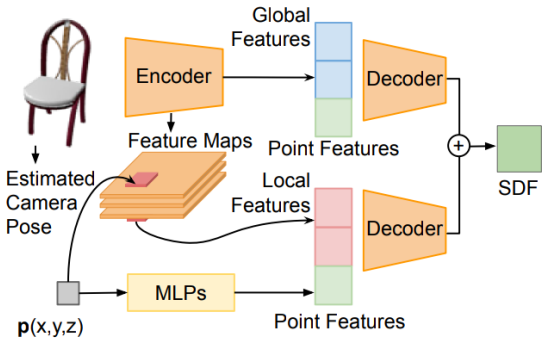


Figure 3.4: DISN general architecture. Image from DISN [25]

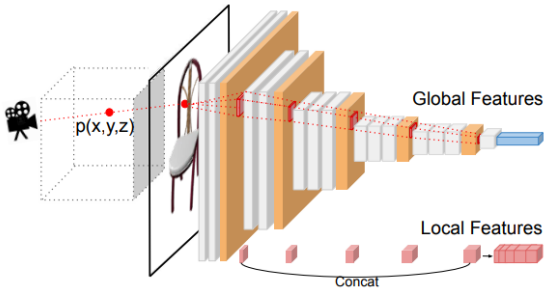


Figure 3.5: DISN local feature concatenation. Image from DISN [25]

The main pipeline, as seen in figure 3.4 is encoding the input image to some global feature, encoding the point's coordinates with an MLP (similar to PointNet) to point features and via an estimated camera pose (separate model) project the point on the image and concatenate features from activations corresponding to that pixel. The final SDF value is the sum of two components, one decoded from global features and point features, the second one from local features and points features. This combination of global information and local information is expected to give structure and detail to the result.

This formulation permits them to attain SOTA results for this task:

Figure 3.6: DISN results on online images.
Image from DISN paper [25]

3.2 Assisted annotation

In this section we will look over a couple works from our group that do assisted annotation for image segmentation: Polygon-RNN [3] (and it's extension Polygon-RNN++ [2]) and Curve-GCN [10]. They lay the path for the assisting aspect of this work.

3.2.1 Polygon-RNN

Polygon-RNN wants to speed-up segmentation annotations on images. A segmentation on an image consists of assigning each pixel a label representing its instance so that pixels corresponding to the same object have the same label. To do that they focus on segmenting only one object and they use a polygon to separate the pixels. This type of segmentation is compact (just need to store positions of the vertices of the polygon) and allows simple and fast modification from the user by dragging points.

The assistance provided by the model is an initial estimation of the contour of the object and a reprediction of points after a user modification. The architecture is an image encoder from which features are obtained, concatenated and scaled before being passed to a recurrent neural network (RNN) which predicts the location of each vertex based on the internal state (of the RNN) and the previous two vertices. When a correction is done, the following points are repredicted given the correction as ground truth.

The main problem of *Polygon-RNN* is given by the sequential nature of RNN, there is a first

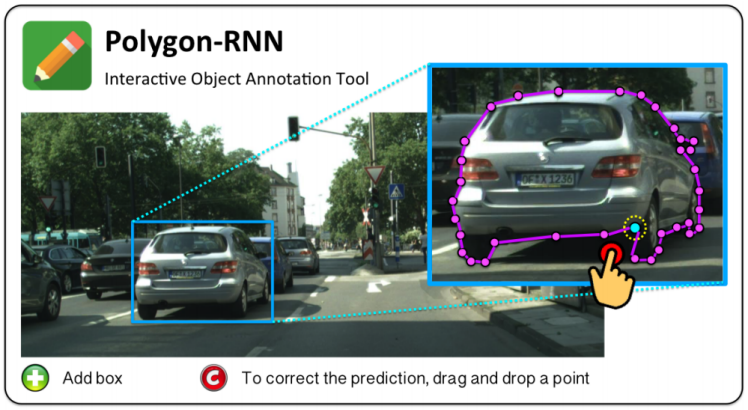


Figure 3.7: Polygon-RNN segmentation in image.
Image from Polygon-RNN paper [3]

and last point. Of course the loss function is order invariant but it's not a strong restriction on the model prediction; secondly, user changes are only propagated forward (in the order of the vertices) which is non-intuitive.

Polygon-RNN++ is a follow-up paper improving the performance of the previous model by a newer encoder, a better training procedure and a bigger output resolution but doesn't solve those inherited issues.

3.2.2 Curve-GCN

In order to solve this order problem in the user interaction, another work from the same group tries another formulation of the task. Instead of using a RNN which forces a sequential prediction of vertices, it leverages a graph convolutional network (GCN) to predict all vertices at the same time and correct neighboring vertices correction (in both directions!). This model also makes uses of splines (in addition to polygons) to have a smoother contour.

Figure 3.8 shows the main pipeline which is very similar to Polygon-RNN, an CNN image encoder gets a feature map. Then, starting from a fixed circle, each vertex samples a feature from its pixel in the feature map and then some GCN layers predict a movement of vertices, those sample feature on the new position and repeat. User interaction is done with a different model that takes the vertex displacement of the user editing and propagates this to nearby vertices.

My personal explanation of this model is that it learns to create a feature map on the image that permits the GCN layers to select a contour that represents an object. Further down in

this project, this is the model we will base our user interaction on, with GCN layers that connect related points together so that editing operations are propagated smartly.

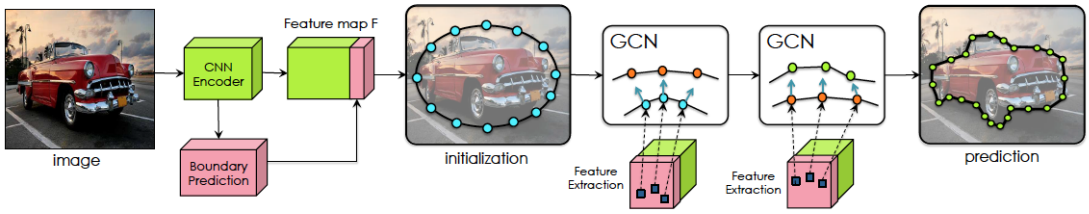


Figure 3.8: Curve-GCN pipeline.
Image from Curve-GCN paper [10]

Chapter 4

Classic approaches

3D data has been around for a long time in computer science, and some classic algorithms exist to solve part of our problems. They are relevant to this work as they provide efficient techniques than can be embedded in our models, ideas to design our architectures and lower bounds to the quality of the results.

In this chapter we'll present reconstruction algorithms to obtain 3D meshes from point clouds, an alignment algorithm of two point clouds and the space carving paper [8].

4.1 Reconstruction from point cloud

In this section the task of 3D surface reconstruction from point clouds consists of generating a triangular mesh given a point cloud, the algorithms presented here have no "learning" capabilities and are only relying on geometrical and distribution features. For each algorithm, we will explain the general idea behind it, show some results on our test case and discuss their performance.

We manually test those algorithms on a lidar sampled point cloud (figure 4.1) of a triangular mesh of a car (figure 4.2). We separate them in groups based on similarity of approaches, this division is not standard but personally defined for explanatory purposes.

Those results are obtained using the tool Meshlab on our custom lidar sampling method on an open source 3d model.

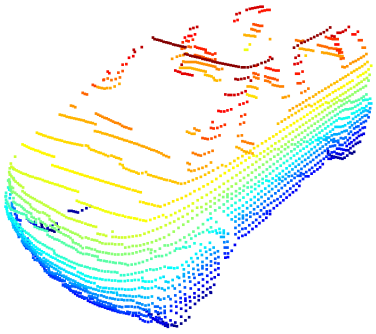


Figure 4.1: Lidar sampled point cloud of a car, input to classic algorithms

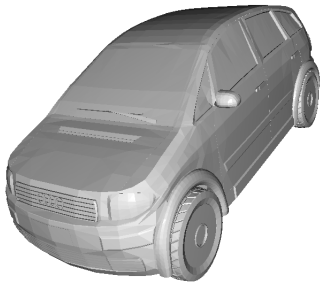
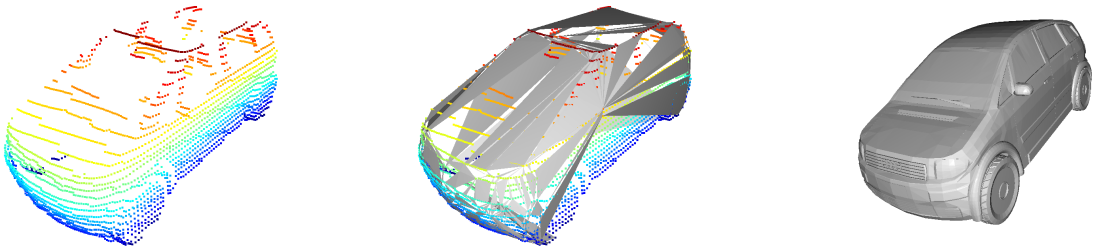


Figure 4.2: Triangular mesh of the car, ground truth result of completion

4.1.1 Delaunay, Convex hull and Alpha shapes

These three methods have in common that they rely on triangulation and they can be thought as increasing in generality.

First we define the **convex hull** of a set of points as the smallest convex polyhedron that contains all the points of the set. In figure 4.3 we display the triangular faces of such a convex hull for our input lidar; we can see a protrusion on the side mirror that is connected to the wheels, this is due to the convexity of the polyhedron.



Input lidar Convex hull Ground truth

Figure 4.3: Convex hull result on lidar sample

Delaunay triangulation is technically a tetrahedralization but triangulation is commonly used as in the 2D case. A triangulation in 3D is a partition of the volume occupied by the convex hull of the points in tetrahedras using all vertices in the point set such that the sphere englobing each tetrahedra contains no points. Figure 4.4 shows those two last concepts in 2d together with the voronoi diagram (which we do not explain in this writing)

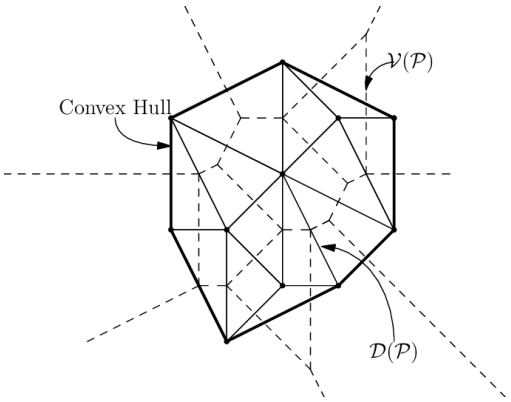
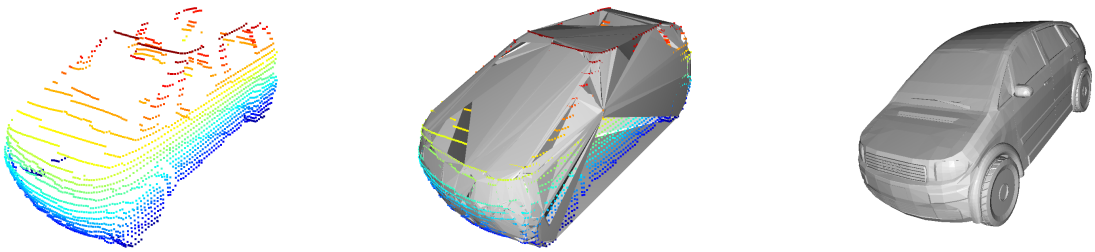


Figure 4.4: Relation between convex hull, Delaunay triangulation and Voronoi diagram. Image from [16]

Delaunay triangulation is not that useful for us because it doesn't generate a simple triangular mesh, as visible in figure 4.5. It is relevant though to explain convex hulls and its generalization, alpha shapes.



Input lidar Delaunay triangulation Ground truth

Figure 4.5: Delaunay triangulation result on lidar sample

Alpha shapes are a generalization of the convex hull, their are a parametric family of sets of edges dependent on the real positive or infinite parameter α. If we define the generalized disk of radius α as:

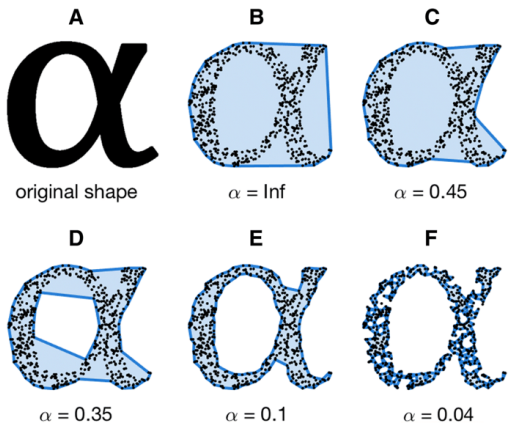
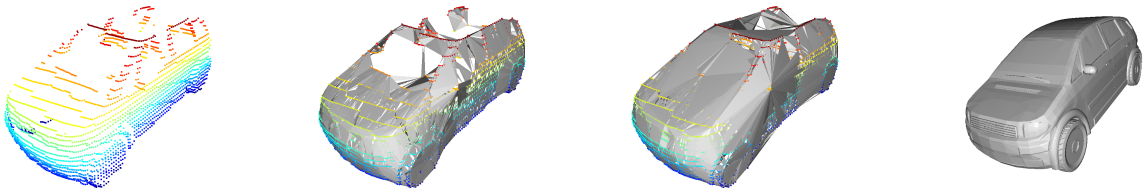


Figure 4.6: Various alpha shapes at different values of α.
Image from [6]



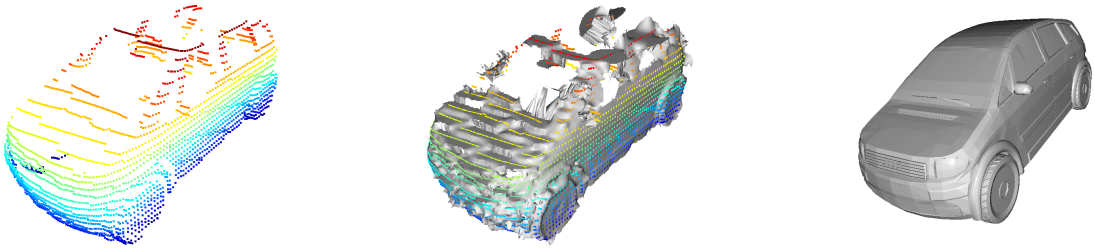
Input lidar Alpha shape $\alpha = 5$ Alpha shape $\alpha = 10$ Ground truth

Figure 4.7: Alpha shape results on lidar sample

$$\text{Generalized disk}(\alpha) = \begin{cases} \text{closed half plane} & \alpha = \infty \\ \text{closed disc of radius } \alpha & \alpha \neq \infty \end{cases}$$

then the α-shape is the set of edges such that there is a generalized disk of radius α that contains the entire point set and the edge's endpoints lie on its boundary. It is closely related to the Delaunay triangulation in the sense that if we only take edges and triangles with radii of at most α we get a shape very similar to the corresponding α-shape. Also the convex hull is the α-shape with $\alpha = \infty$. Look at figure 4.6 for a better understanding of this parameter.

This algorithm will be very relevant in some of our models to obtain a decent "filling" of a point cloud. Check results on figure 4.7



Input lidar Marching cubes Ground truth

Figure 4.8: Marching cube results on lidar sample

4.1.2 Marching cubes and Ball pivoting

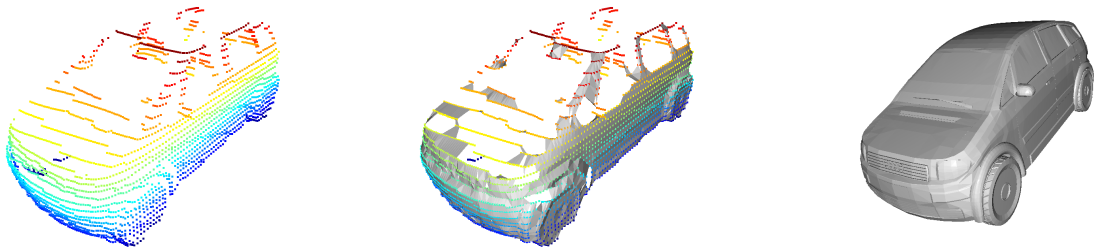
These two algorithms consist of "scanning" the space with some object. In the case of **Marching cubes** (a very famous algorithm in graphics) [11] this is done by sliding a cube of fixed size and generating triangular faces depending on which vertices of the cube are inside or outside the object. This can be done manually for the few unique cases plus rotations and symmetries. For the **Ball pivoting** algorithm, we start with a seed triangle and a sphere of a fixed size touching the three points; the sphere will pivot around an edge until it touches another point which is then included creating a new triangular face. This process is then repeated for all possible edges and seed triangles.

Both results can be seen on figures 4.8 and 4.9. We can see that both fail to create a full surface but rolling ball gives a smoother result when marching cubes has the problem of very obvious stairs-like result. On the other side, marching cubes is faster.

4.1.3 Poisson

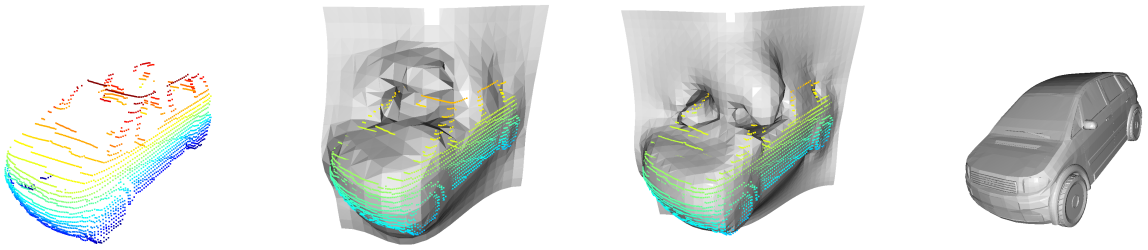
Finally **Poisson** surface reconstruction works on oriented point clouds (point cloud where each point has a normal defined). It is solving for an indicator function, whose gradient matches the input normals; after solving the PDE the iso surface can be extracted into a triangular mesh. Multiple outputs can be obtained depending on the PDE solver and the surface extraction, we show a coarse and more detailed result in figure 4.10

Finally it is possible to compare results in figure 4.11.



Input lidar Rolling ball Ground truth

Figure 4.9: Rolling ball results on lidar sample



Input lidar Poisson (coarse) Poisson (fine) Ground truth

Figure 4.10: Poisson results on lidar sample

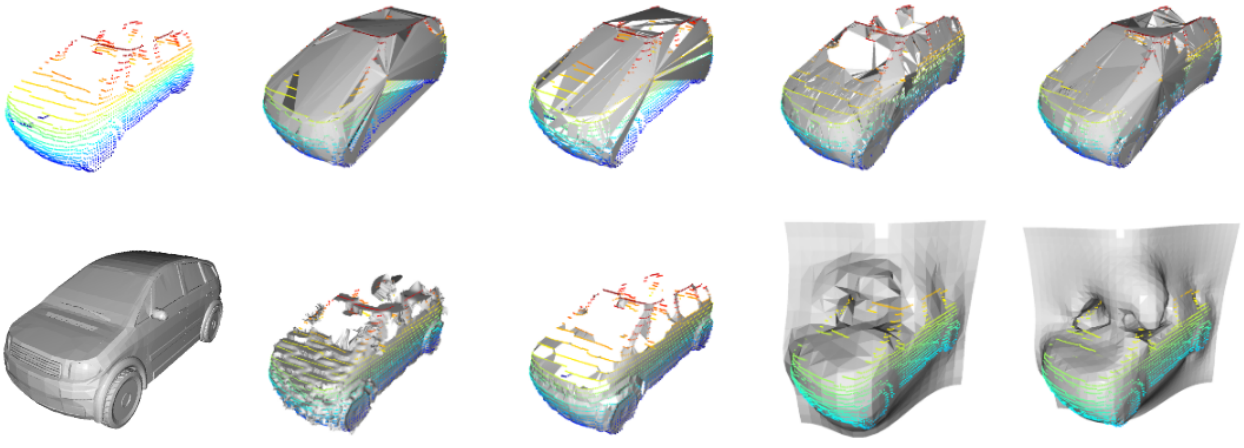


Figure 4.11: Comparison table (left-right, top-bottom): input, delaunay, convex hull, alpha-5, alpha-10, ground truth, marching cubes, ball pivoting, poisson coarse, poisson fine

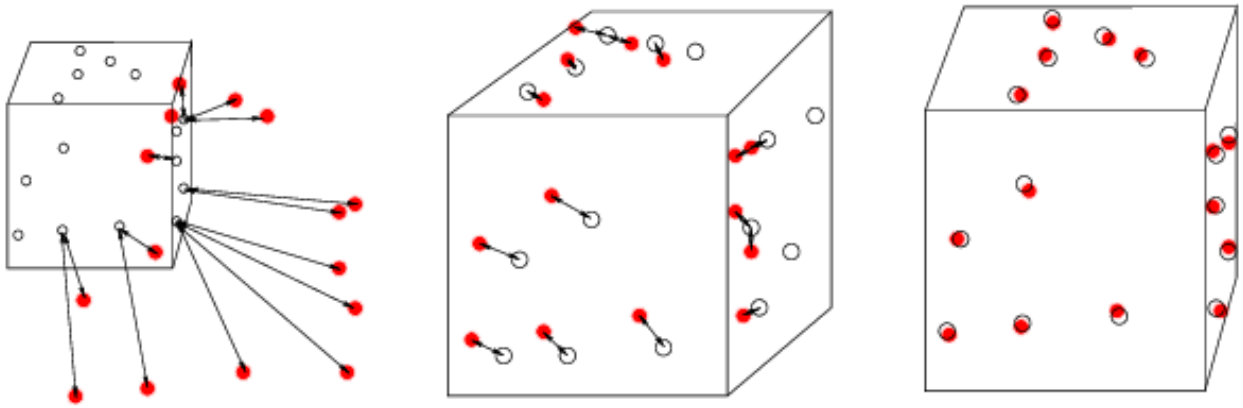


Figure 4.12: A few iterations of the ICP algorithm. Image from [21]

4.2 Point cloud alignment

In this section we will briefly explain an algorithm that minimizes the difference between two point cloud by trying to match them. It is called **Iterative closest point** (ICP). The algorithm is very simple and consists of matching each point in the source point cloud to the closest in the reference one, then compute the rotation and translation that better maps this matching. Apply this rotation on the source point cloud and repeat from the beginning until convergence (figure 4.12 shows three iterations of it). It is very simple but works well when there's a dense number of points and starting positions are close enough. We think that it could be used as a refinement step to merge two point clouds of the same instance but different views (lidar sources).

4.3 Space carving

Space carving is a method introduced in [8] for 3D reconstruction from multiple images. We start with an full occupancy voxel grid, for each voxel in the grid we project it into the multiple views and discard the voxel if the views are not consistent. The paper further improves the efficiency of the method by doing plane swipes and other tricks. For this work we only care about the idea that a full voxel grid can be "carved" by having a consistency check on multiple views.

Our first main idea comes from this paper where we get the inspiration of using 2D silhouettes of objects and then carving the result. We select top, right and front silhouettes and use them as filters for a carving operation:

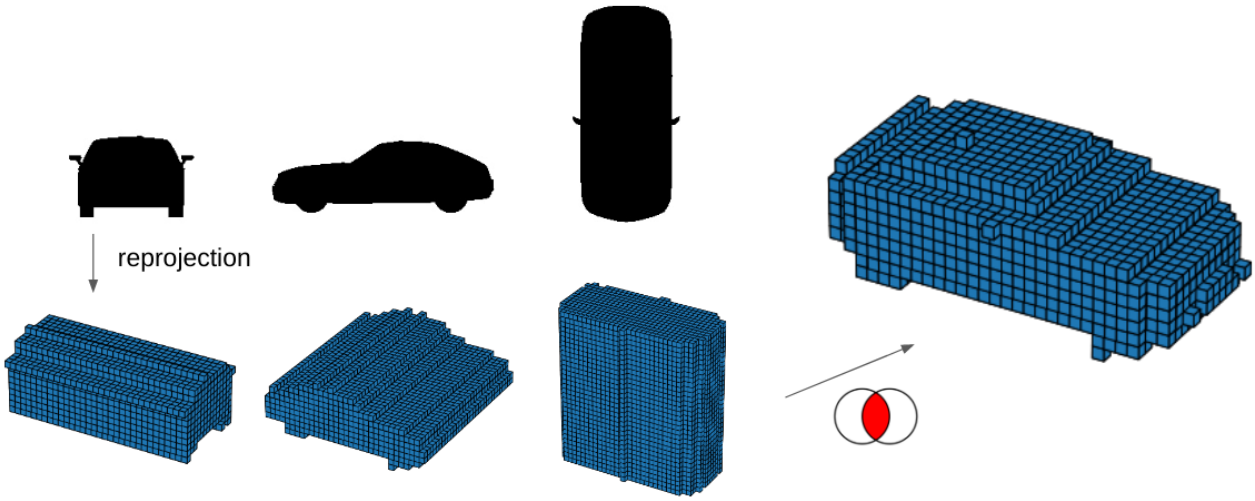


Figure 4.13: Carving pipeline from top,right,front silhouettes

An advantage of this approach is getting a 3D representation directly from 2D segmentations in multiple views. If we can get a rough automatic voxel this could be an simple editing mechanism based on user 2D annotations.

Chapter 5

Models and architectures

In this chapter we provide an overview of the different models we implemented and experimented on, of the design decisions behind them and their intended purpose. This chapter is tightly coupled with the next one, chapter 6.

Our general objective is to find a good way to incorporate point clouds in an interactive model to obtain a good 3d shape completion. The general pipeline, as shown in figure 5.1, is to start with a point cloud and an image, obtain an initial 3D reconstruction and some views to display to the user, who will do as many corrections as wanted in 2D to obtain a better 3D reconstruction.

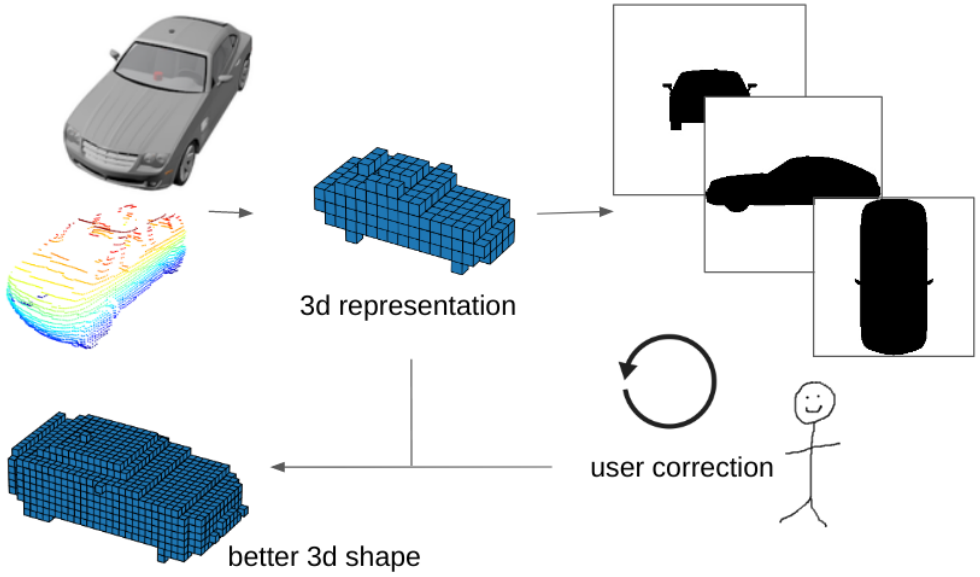


Figure 5.1: General interactive pipeline

Figure 5.2: Input/Output of the Point2Sil model

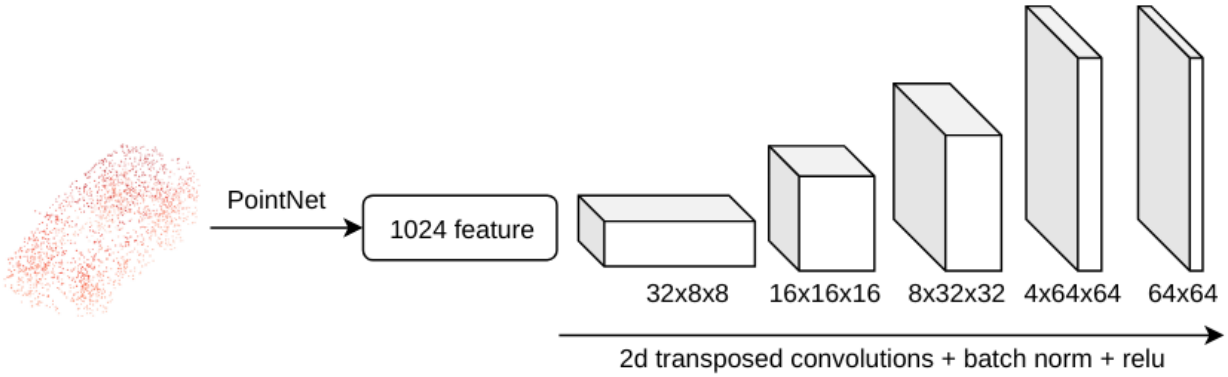


Figure 5.3: Point2Sil architecture

5.1 Point2Sil

We started by something very simple to validate the capacity of point clouds. So we designed a very light model to predict a fixed view silhouette from a uniform point cloud (figure 5.2). Precisely this is mapping 2048 points to a 64x64 image.

The model is based on the PointNet encoder presented in section 3.1.1 and a custom deconvolutional decoder. After obtaining relatively good results (check section 6.1) we know it is possible to encode/decode 2D shape information from a 3D point cloud. The next step is then available: the initial 3D reconstruction.

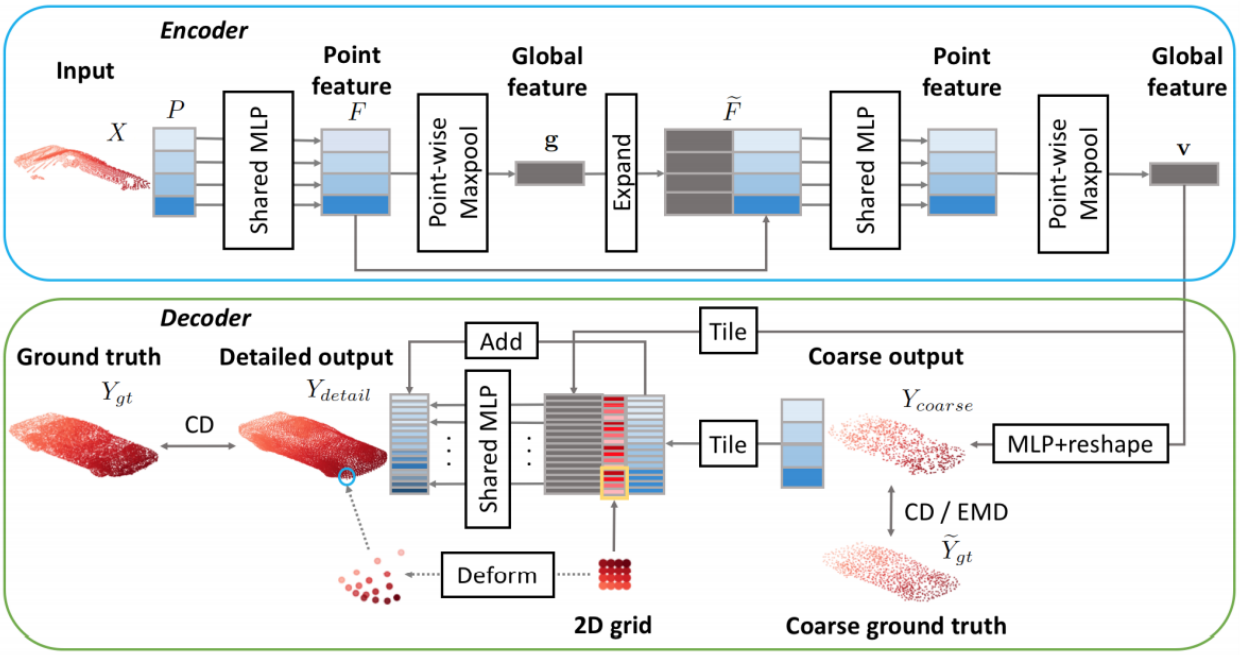


Figure 5.4: PCN architecture. From PCN paper [27]

5.2 Point Completion Network

Point Completion Network (PCN) [27] is a model that, given a partial point cloud, generates a dense completion. Our idea is to densify our input point cloud and use a surface reconstruction algorithm from section 4.1.

Figure 5.4 shows its encoder-decoder architecture. The encoder is two PointNet steps without input or feature transforms: a shared MLP to get a feature for each point in the input, a maxpool merges it into a global feature; this is repeated but the input is the global feature concatenated with each point feature instead of the coordinates.

The decoder is more interesting as it incorporates various techniques introduced in section 3.1.1. First a coarse point cloud is directly generated via a MLP then a residual folding operation is performed on the coarse point cloud to obtain a dense one. Here the folding operation can be thought as learning a local surface (deformation from a flat 2d grid) at every point of the coarse output.

Figure 5.5: Examples of PCN results on cars. From PCN paper [27]

Figure 5.6: Nested shape layers of Matryoshka Network. From Matryoshka [17]

5.3 Matryoshka Network

The previous model (PCN) was centered around point clouds, **Matryoshka Network** [17] works from an image perspective. Basically it's an image encoder-decoder (figure 5.8), taking a view of an object it decodes a voxel grid. The idea is that instead of directly generating the grid it predicts a fixed number of what the paper calls "shape layers". A shape layer is basically a cube depth map (6 orthogonal depth maps, figure 5.7). The special contribution is having nested shape layers so they predict voxel grid surfaces layer by layer (like an onion), see figure 5.6. This technique allows them to improve metrics on 3D reconstruction mainly because the dataset used has models with structure inside the main shell (like seats for cars).

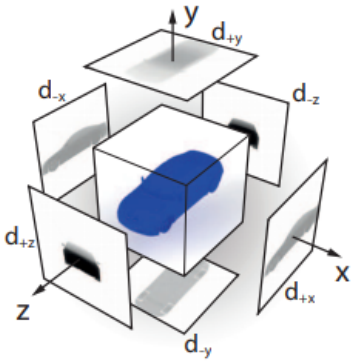


Figure 5.7: Depth layers.
Image from Matryoshka [17]

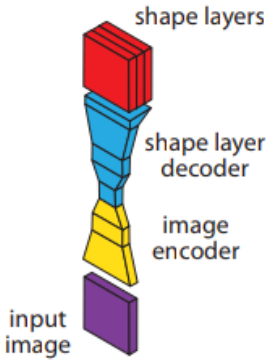


Figure 5.8: Matryoshka encoder-decoder architecture.
Image from Matryoshka [17]

5.4 Interactivity

As shown in figure 5.1, we want some kind of interaction with the user to refine our 3D completion. We settle for an image based approach as our results with PCN had some issues. For our architecture, we modify the decoder of Matryoshka to predict a fixed set of features and we use that as a feature map for the Curve-GCN. Curve-GCN does the initial prediction of three silhouette (top, front and side) based on those features. We then obtain a polygon around the silhouettes with an alpha shape and use Curve-GCN and Matryoshka's features to do interactive user modification from that initial proposal.

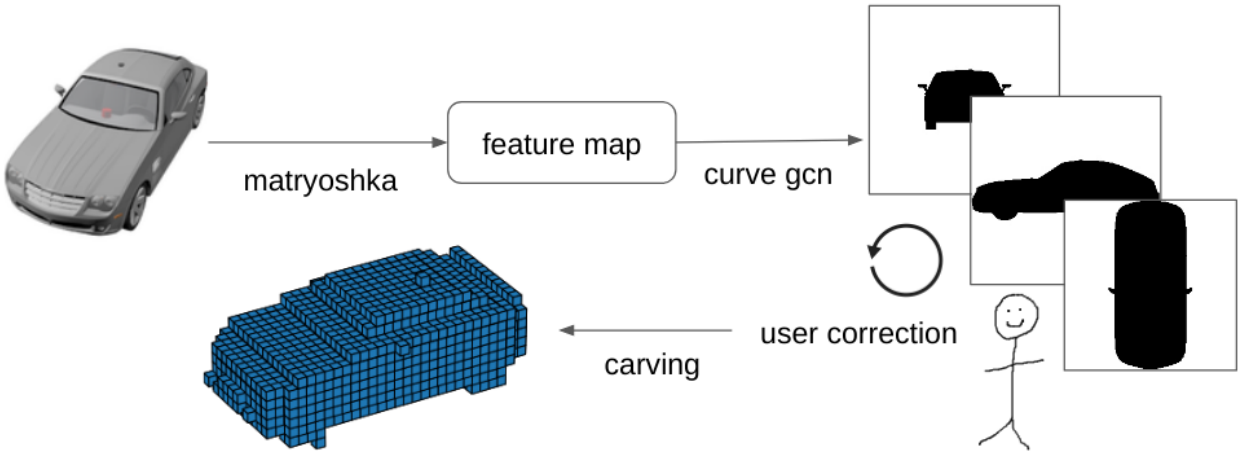


Figure 5.9: Interaction pipeline with matryoshka network

Chapter 6

Experiments

In this chapter we add implementation details on the theoretical background of last chapter, the data acquisition and processing, and most importantly, the experiments and their discussion. Each section is a progression from the previous one as it applies the conclusions reached.

In general we use one dataset of 3D shapes, ShapeNet [4], where we work with the car category. We also check our generalization with lidar samples from KITTI [7].

From ShapeNet we developed some data generation scripts: renders using Blender, uniform point clouds sampling from triangular meshes, lidar like sampling from triangular meshes and depth capture from a monocular camera.

These experiments were carried out using the resources of Vector Institute, Toronto; all our models are run on NVIDIA Titan Xp (12GB).

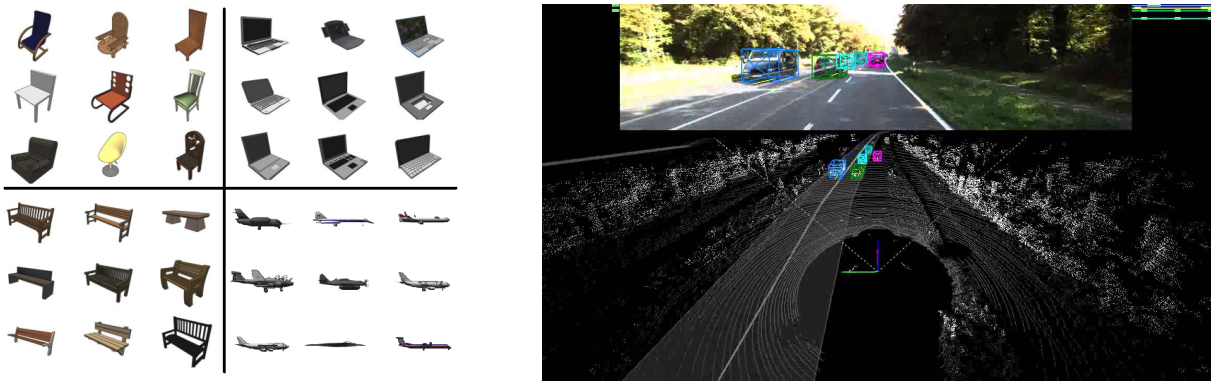


Figure 6.1: ShapeNet and KITTI lidar dataset

6.1 Point2Sil

For our silhouette prediction from uniformly sampled point cloud we generated ground truth silhouettes rendering car meshes on a fixed side view and 2048 uniformly sampled (in surface area) point cloud. We then train our model (section 5.1) as a pixel-wise classification problem.

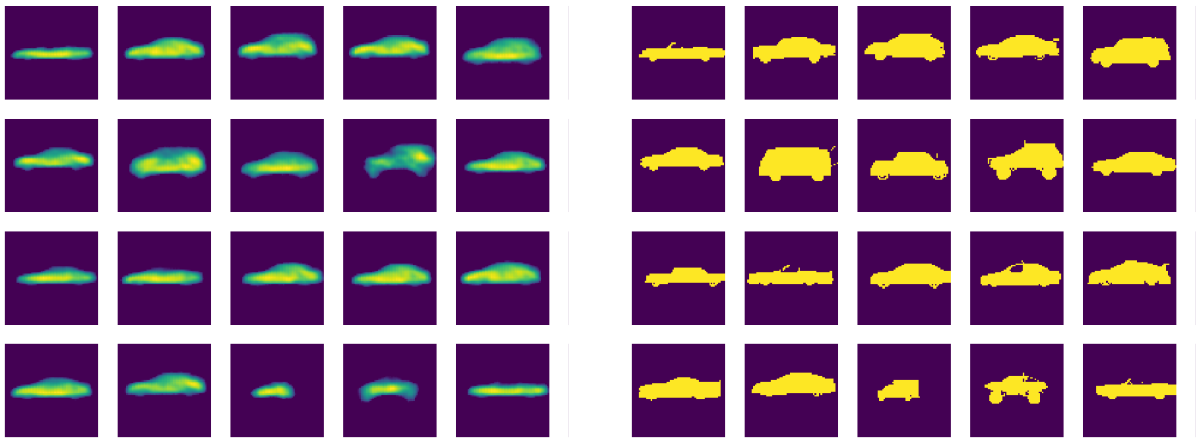


Figure 6.2: Validation results from Point2Sil experiment (prediction left, gt right)

Results can be seen on figure 6.2. The intensity can be interpreted as the probability of presence per pixel. They look quite good and achieve above 95% accuracy. But, how good is this ? We need some baseline to compare ourselves to.

The first baseline is a **null model** consisting of a constant, we predict the mean silhouette of our training set (figure 6.3). The second baseline is projecting our points in the silhouette plane, do the **convex hull** of the points and use the filled and rasterized image as the prediction (figure 6.4).

Figure 6.3: Mean silhouette baseline for Point2Sil

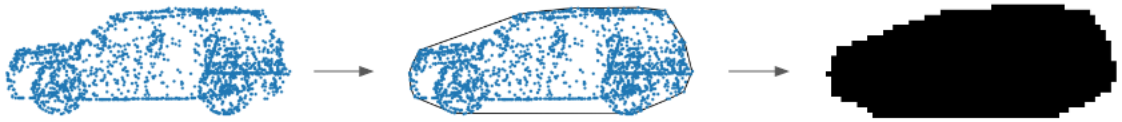


Figure 6.4: Convex hull baseline for Point2Sil

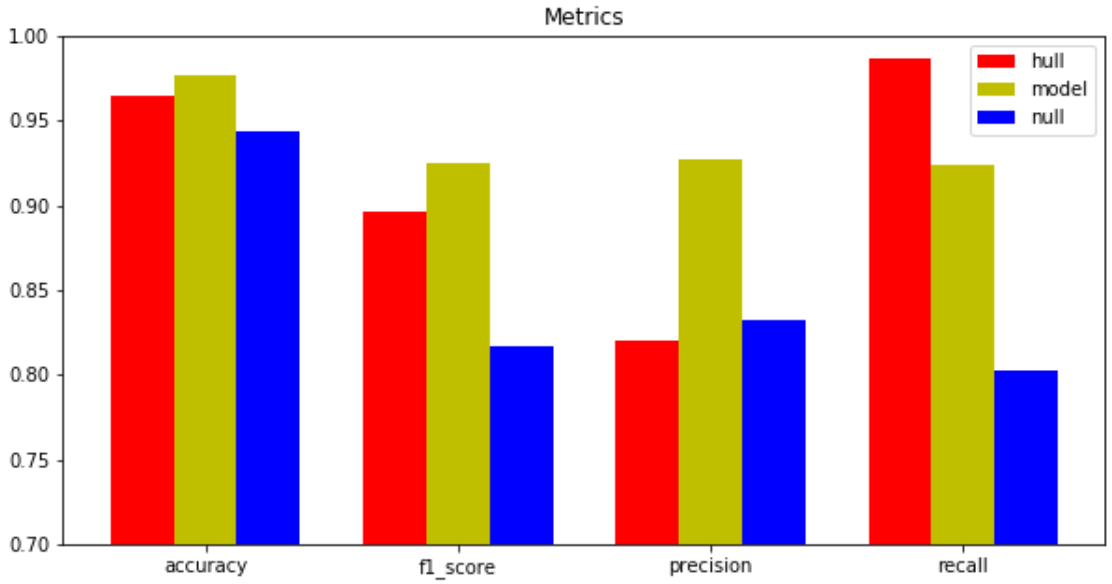


Figure 6.5: Metrics for Point2Sil with baselines

As we can see in figure 6.5 it is very easy to achieve high accuracy in this task because there is a lot of empty space that is consistent across all silhouettes (cars are longer than tall). Also cars have very similar silhouettes (good performance of mean model). If we look at other metrics the result is quite different. For example at precision our model beats the baselines by 10%, almost perfect (rasterization create small errors) recall is achieved by convex hull due to its nature. Our model has a good balance of precision and recall which shows as a good f1 score. Also, as expected, convex hull is better than the null model.

6.2 Point Completion Network

For the PCN, data preparation was easier (point cloud sampling) but we implemented the model from scratch due to the usage of old out-of-date software in the published source. An important thing to note is that point cloud regression has specific metrics, the most important two are:

Chamfer distance

Measures the distance by matching each point of one set to the closest in the other one. It has two terms to make the distance symmetric (similar to precision and recall):

$$d_{CD}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|_2^2 + \sum_{x \in S_2} \min_{y \in S_1} \|x - y\|_2^2$$

Earth Mover's Distance

Measures the distance by matching points by an optimal assignment, where optimal here means by total distance "moved":

$$d_{EMD}(S_1, S_2) = \min_{\phi: S_1 \to S_2} \sum_{x \in S_1} \|x - \phi(x)\|_2 \quad \text{where } \phi : S_1 \to S_2 \text{ is a bijection}$$

EMD requires the sets to have the same cardinality, CD can always be used. In our case we only use CD because we couldn't find a working GPU-accelerated PyTorch-compatible implementation of EMD.

When trying to replicate results we found out that the model was generating almost a constant point cloud ($1e^{-5}$ CD) and we couldn't obtain the same quality as in the original paper (figure 6.6). After better examination, we noticed that the reported results on KITTI (real data) had their bounding box aligned so the model was dependent of having a very specific coordinate basis centered and oriented properly; impossible to get on raw real data.

Also, incorporating the image to this architecture was challenging as we had no insights on how to ensure the consistency of reconstructed point cloud with the input image. For this motive we move to an image centered architecture.

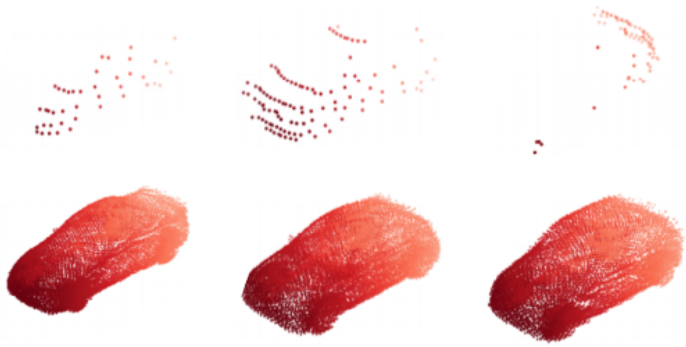


Figure 6.6: PCN results, showing the similarity between predictions

6.3 Matryoshka Network

For Matryoshka, we first train a model to predict three channels that we interpret as binary masks for silhouettes from a single image. We use the same loss function as in the original paper [17], a modified L1 loss which penalizes background pixels only if there are within a

margin of one unit to being considered foreground. Similar to the complementary of a hinge loss.

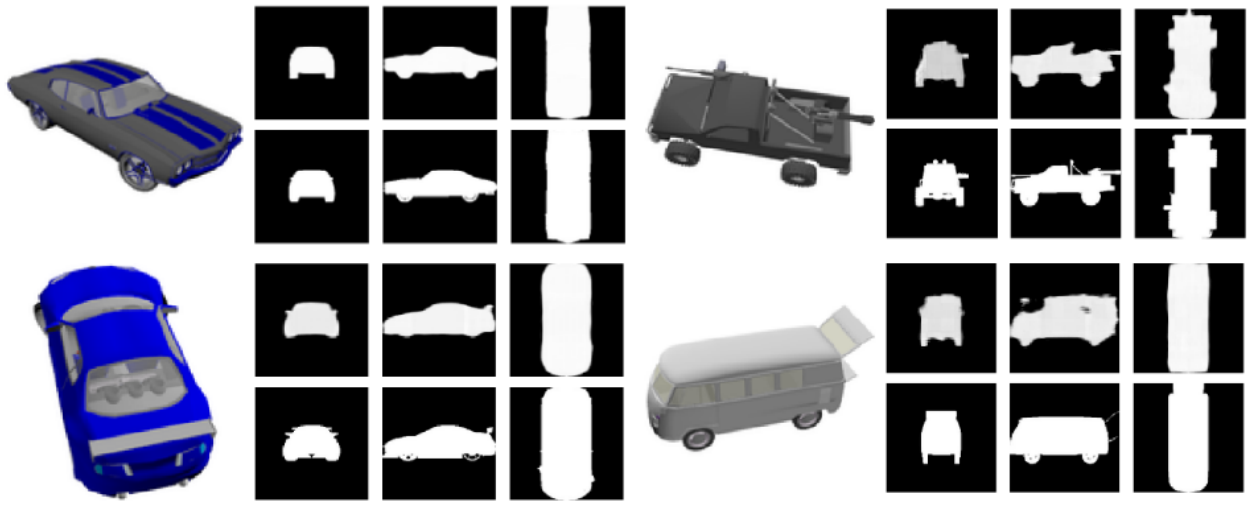


Figure 6.7: A few results on three silhouette prediction with Matryoshka Network. Top row is good and bad example from training set, bottom row is good and bad example from testing set. For each example, the top row of three views is prediction and the bottom row is ground truth

Matryoshka does a really good job at predicting consistent silhouettes even from strange cars (figure 6.7). The metric we use to evaluate the quality of our prediction is intersection over union (IoU) which is the area of overlap between our prediction and the reality over the area of union of these two. An IoU of 1 means perfect match, 0 is no intersection. Our first model achieves a mean of 84% IoU.

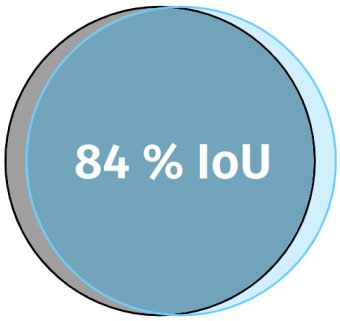


Figure 6.8: Visualization of 84% IoU

6.3.1 Sensor fusion

Now that our image model works we want to increase the performance by adding information from the point cloud in our model. As Matryoshka is an encoder-decoder architecture we can concatenate extra channels just before the decoder so that image encoding is mixed with our custom features. We study the performance of different scenarios:

Only image

No features from point cloud added.

Image + local features

We add local image features corresponding to the point cloud. They are collected by projecting each point onto the image plane and grabbing activations on a few layers of the encoder.

Image + coordinates

Add spatial coordinates of the point cloud (x, y, z).

Image + local features + coordinates

Adding both coordinates and local image features

These features are concatenated together and passed through a PointNet like encoding before being added to the global image feature of the encoder (figure 6.9).

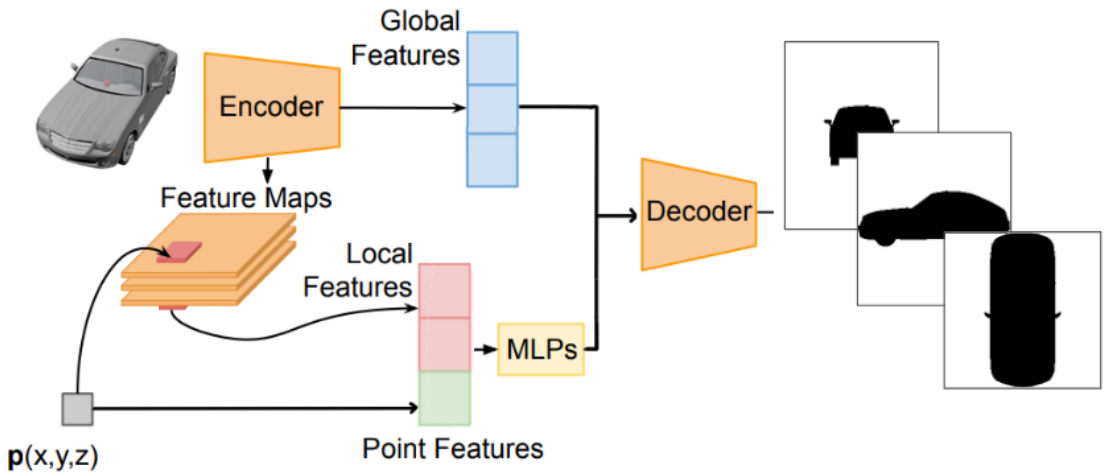


Figure 6.9: Pipeline for the injection of point cloud features

One might think that the option with all the different sources of information will be the best, this is not taking into account the capacity of our decoder and the relevance of the new data.

If we add too much noise the training will be harder and might not converge properly.

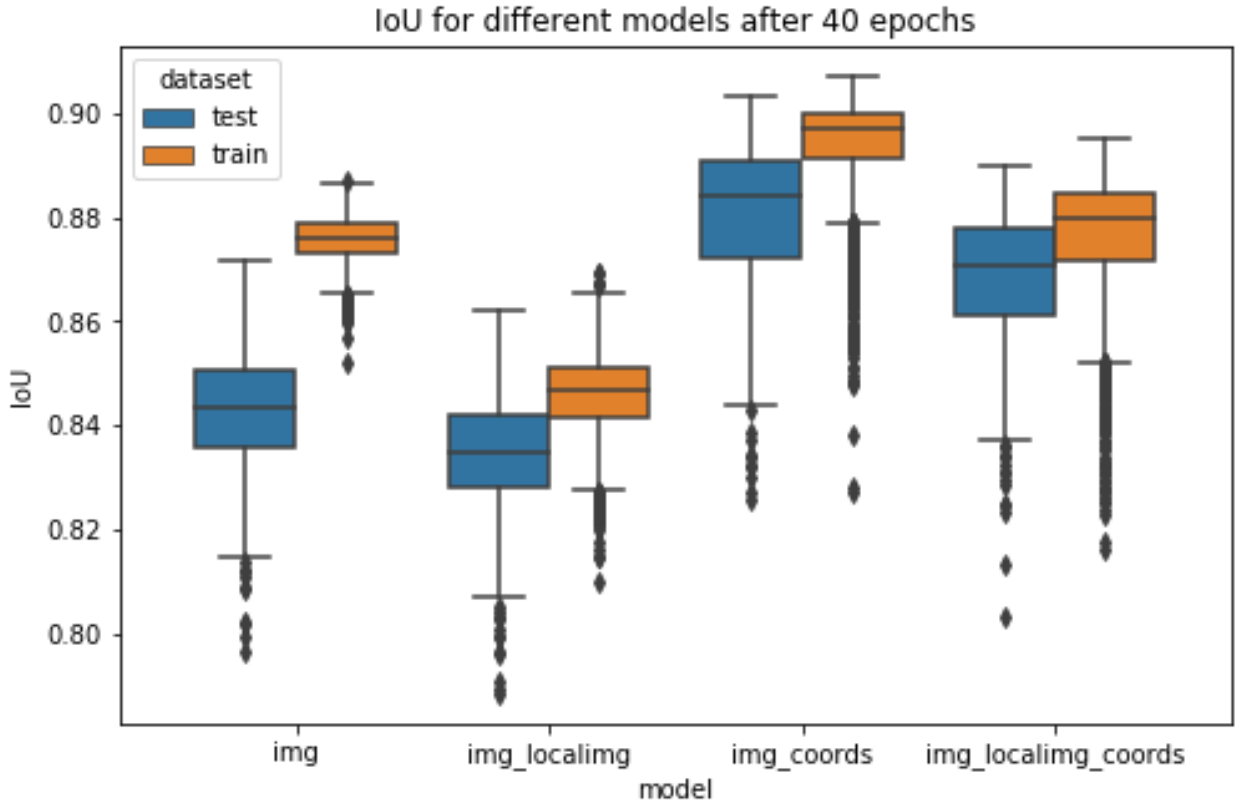


Figure 6.10: Boxplot of IoU for the different scenarios for sensor fusion

The results of that big experiment can be seen in figure 6.10. The first thing we notice is that training IoU is higher than testing IoU but not by a big margin which means that we are not overfitting too much. We can also observe that adding coordinates increases performance by +2-3% with respect to only image (two on the right vs two on the left). Surprisingly adding local image features just makes it harder to converge (visible from image and from image+coordinates), we think this is because the encoder already computes good image feature and adding a mix of sampled activations to that only confuses the decoder. Also image features have very strong locality, meaning they represent small details, mixing that with a PointNet encoder might not be a good idea.

Finally we conclude that image encoder with coordinates in a PointNet encoder is the best option. We can't forget that the performance of all models is quite good, the image features alone do a great job probably because the car class is easy to learn visually.

A few key points about that experiment is the usage of a canonical pose for the point cloud,

it simplifies the learning as there is no shift or rotation across the data. Another downside is the usage of uniformly sampled point cloud; computing the same metrics after finetuning our previous models with lidar sampled point cloud, we lose a consistent 1%.

In parallel of my work on point clouds features, Tianchang Chen modified Curve-GCN to have graph connections between separate GCN layers on each silhouette. This allowed the corrections to be propagated from an image to another because close edit points were connected in the graph. When we trained our interactive pipeline with Curve-GCN at the end with user correction we obtained a consistent 85% 3D IoU.

6.4 Interactive tool

The main objective of the project is to develop models for assisted annotation. After getting good results on development environments, the next step was to create a web application where users can load and interact with point clouds. The app should provide normal annotation tools like cuboid creation and label assignment. In the long run we aim at incorporating our models inside the tool so that the 3D reconstruction is shown in real-time to the user.

Obviously this is a lot of work, so we focused on getting basic cuboid annotation with visualization of point cloud in 3D with front, side, top 2D views. The application is developed with React and the graphics library Three.js. The resulting main window can be seen in figure 6.11.

6.4.1 Existing solutions

From an engineering perspective we studied what existing products offered, and what were the requirements of the software solution. That allows us to have a roadmap for future development.

LATTE

Linked to the paper [22]. This tool provides top view rectangle annotation and a nice one click bounding box feature by clustering and optimizing a rectangle fit. They leverage MaskRCNN [1] to colorize the points from the front image. Bounding box annotations are propagated forward in time using a simple Kalman Filter. See figure 6.12.

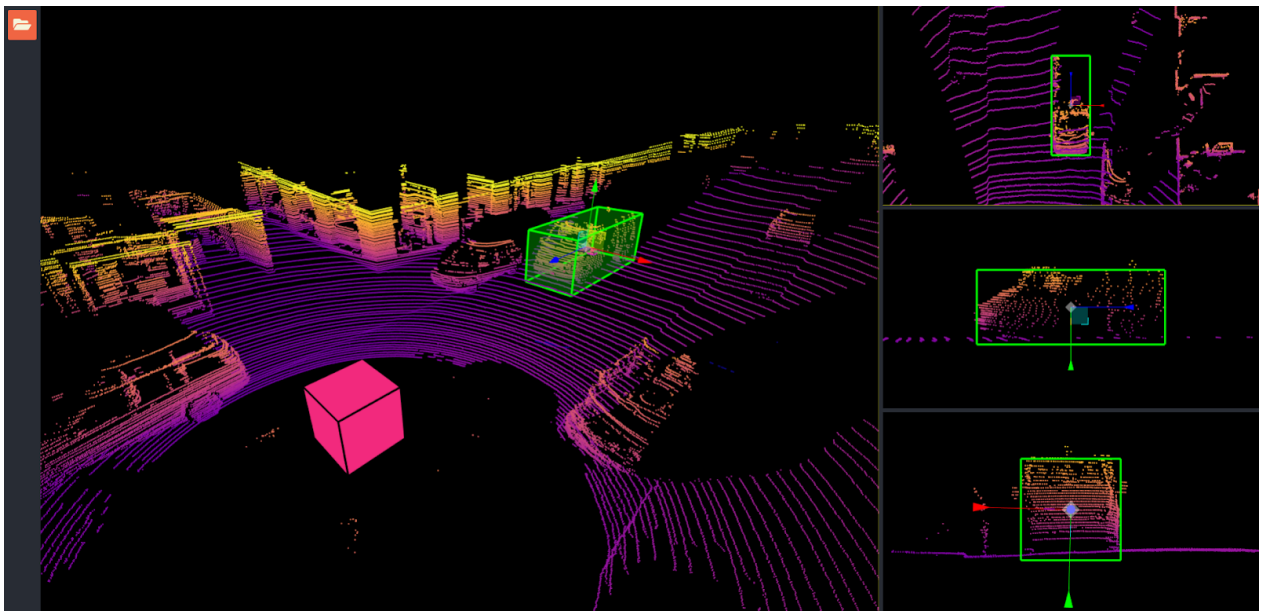


Figure 6.11: Screenshot of the main window of our point cloud tool

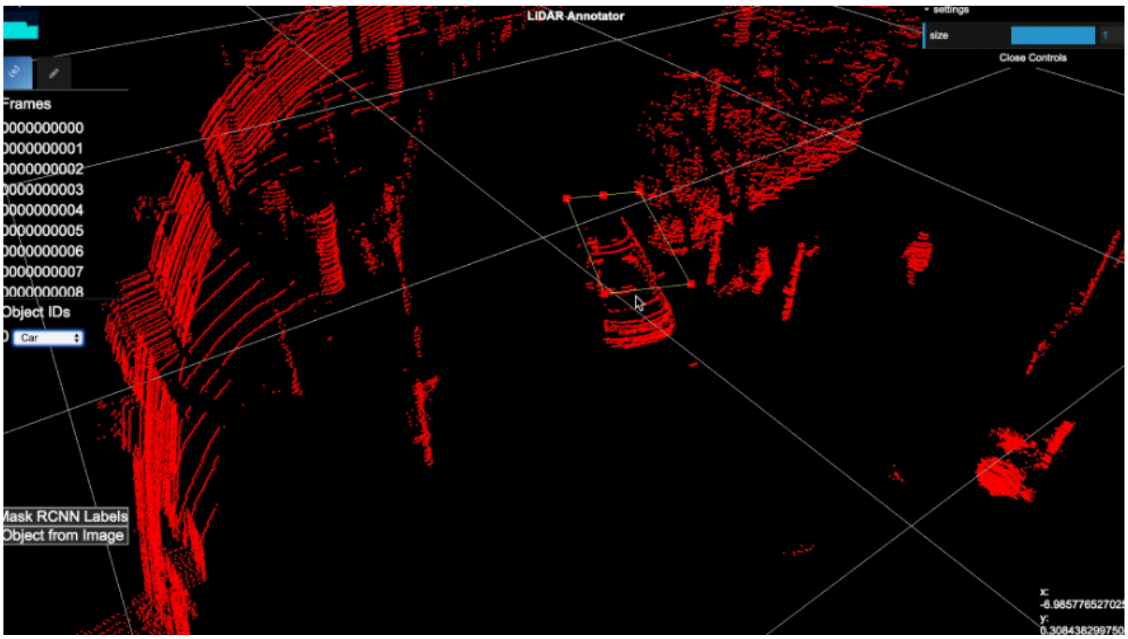


Figure 6.12: Screenshot of LATTE tool. As in their reposity in Sep. 2019

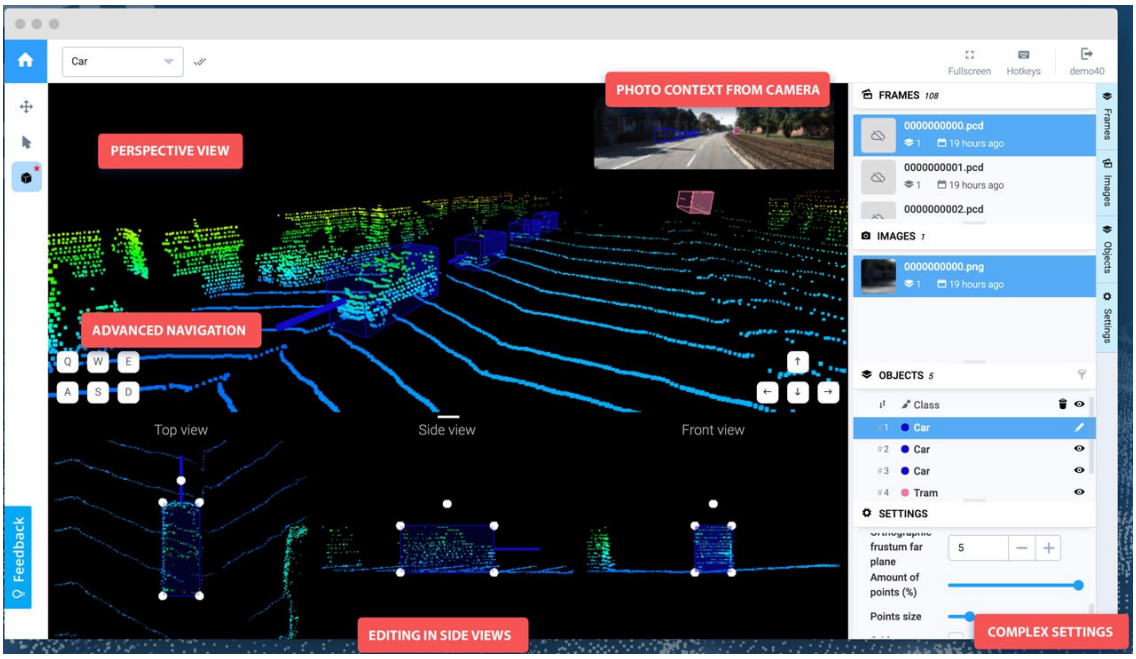


Figure 6.13: Screenshot of Supervisely's tool.
Image from here, accessed Sep 2019

Supervisely

This company has a nice tool that they released for open usage at the time of this writing. They have a bounding box edition in three views while displaying annotation in front image. It doesn't have advanced features but it's the most complete user experience handling upload of point clouds, multiple frames, and labels. See figure 6.13.

Playment

Playment provides the best set of general annotation tools and not only for lidar scans. For our task they have bounding box in three views, brush and polygon segmentation tools, visualization of annotation in front image and bounding box frame interpolation. See figure 6.14

Scale.ai

Scale is the most secretive company, we only got a glimpse of their tool in a demo video. It looks like they have the simple set of features: bounding box edition in three views and point cloud aggregation across frames. See figure 6.15

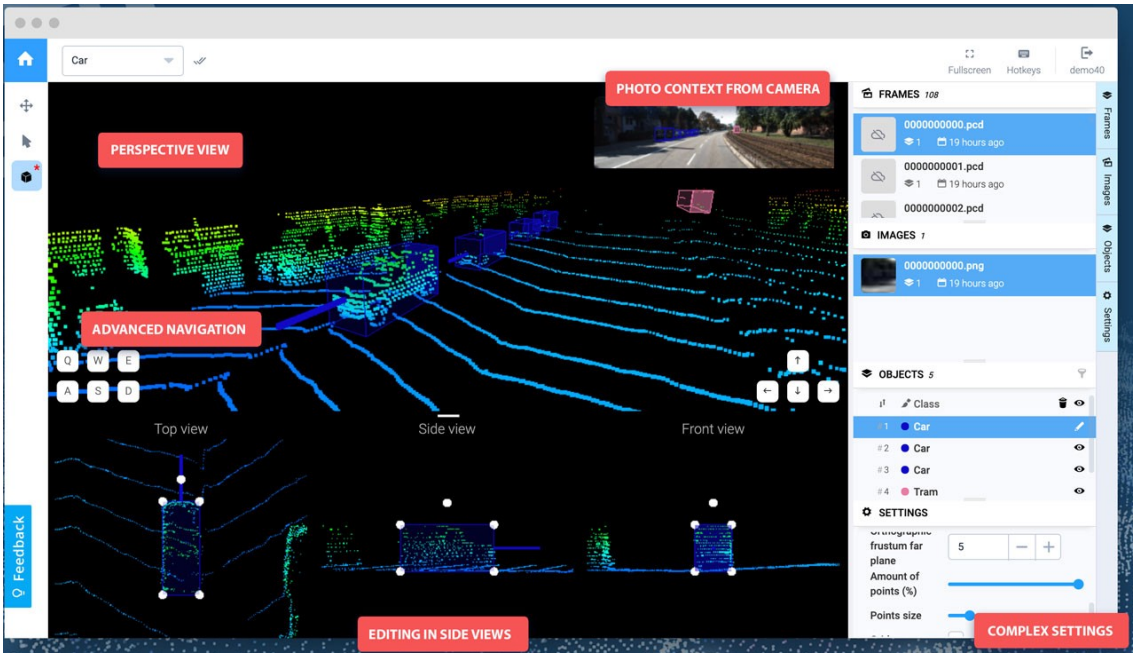


Figure 6.14: Screenshot of Playment's tool.
Image from here, accessed Sep 2019

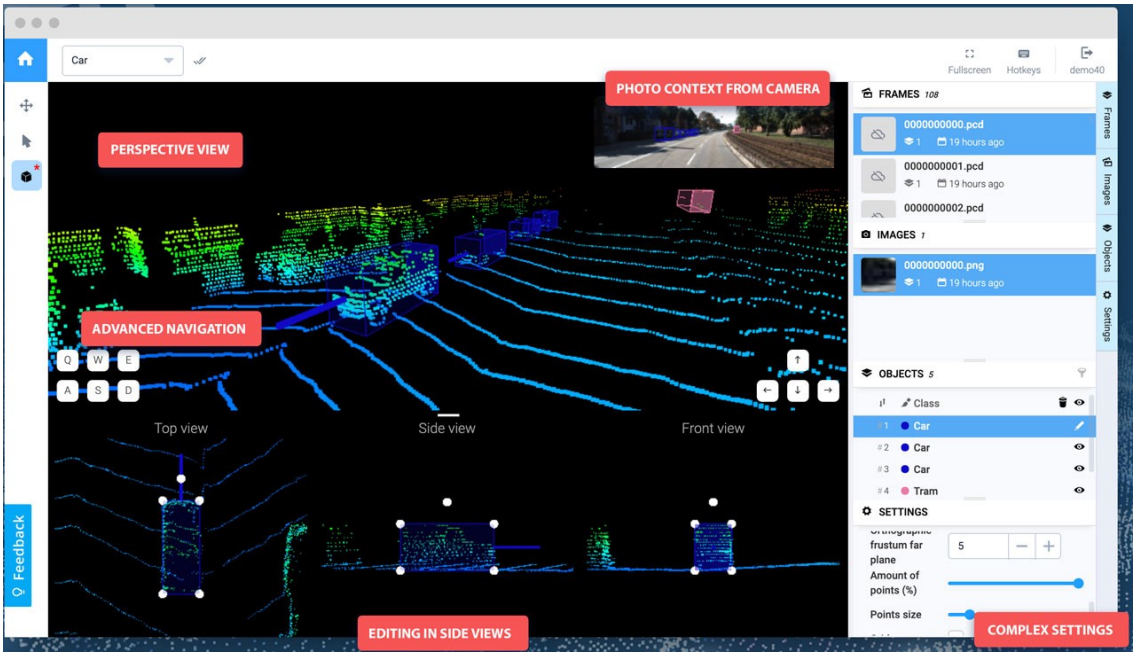


Figure 6.15: Screenshot of Scale.ai's tool.
Image from here, accessed Sep 2019

We think that the most important set of features those tools should have is bounding box annotation with 3 orthogonal views, displaying the resulting box in the front view will help the user see obvious mistakes and imagine occlusions. After this, if we are dealing with multiple frames I would add some kind of interpolation between frames maybe with a Kalman Filter to predict positions. If we manage to label moving objects consistently across frames, we could aggregate points in a unique frame; our 3D surface reconstruction would be improved by this extra information. The extra step would be adding a one-click segmentation like LATTE to get a good initial guess for some objects, having an automatic bounding box from a point cloud segmentation is also a plus.

After this we can think of augmenting some steps with our models: adding 3D reconstruction visualization based on the points lying inside the box. Show colors from an image classification in LATTE's way. Do a refinement step of interpolated bounding boxes based on the 3D model obtained previously. And these are only few ideas that could improve the user's experience and the speed of annotation.

Chapter 7

Conclusions and future work

The goal of this work was to study sparse 3D data with the objective of providing AI assistance to human operators. With the raise in a need for more interactive products like home voice assistants, internet of things (IoT) devices or self-driving cars, there is a stronger focus put on AI assisting user instead of replacing them. We consider Curve-GCN to be one of few models that attacks this problem for geometric purposes, also we realised along this work that most models are based on images and there is still a lot of improvement to be done in other kind of data. We fit in that space, studying how point cloud data can be used either alone or with image in a deep learning model.

Our results concern the usage of point cloud. We found out that point clouds helps but using direct coordinates is very dependent on the origin and axis chosen. For this reason, transformations that try to make models invariant on movements in space are very much relevant (e.g. transform block in PointNet). There is a need for good proven architectures for point cloud, like CNNs for images; and it feels like this is where research in this field is going.

For the sensor fusion, it is easier to extend image models with point features than the other way around, mainly because image models are much more mature. For a good fusion model, we think they should be 3D based as we usually have multiple 2D views for a single 3D scene. In a perfect world we would want to preserve locality between 3D and 2D, adding global point features before the decoder compresses that information too much and helps very slightly. We propose, for a future work, adding point features later in the decoder by projecting the points in the different silhouettes and learning a feature fusion function.

Also, we think that 3D reconstruction networks are not reasoning in 3D but instead do feature detection or classification with the input and then the decoder does retrieval of a parametrized shape, see [20] for a detailed observation of this phenomena. This may be because we're trying to learn a completion plus some geometric operations like projections. Future work like GenRe [28] that separate 2D and 3D steps by deterministic changes of representation are probably the way to go.

It is good to remember that classic (old) methods are useful, either to build intuition, simplify a step in a model or have as a baseline. We are sure there is recent work in the graphics field that could be leveraged for better representational capabilities in our models.

I would continue this work by changing our encoder to something with better generalization (e.g. GenRe) and focus on multiple 2D views aggregation into the point cloud model. I also think it would be easier to separate the task of getting a somewhat decent shape from very sparse data like lidar and the objective of precise reconstruction from denser inputs. The second one can be easily applied to indoor scenes (denser point cloud, more/infinite categories) where robotics and augmented reality could use a surface estimation. Also, adding the models to a web application to test the usability and interaction is important, and should be done since the very beginning of the experimentation.

This thesis has suffered lots of changes based on immediate results and learning steps in the field, it looks more like a lab report of experimentation than a scholar work of problem-method-result. Personally it felt a lot more like real world research where you stumble facing the unknown and develop an understanding as you go through.

I learned a lot during those 6 months, going from theoretical understanding of geometry to practical development of deep learning models. Reading papers can feel daunting at the beginning but after a few dozens, you reach a general understanding of the field; I cannot stress more the importance of having someone with more experience and good availability to guide you through the sea of information.

I have been very happy with the dynamics of the group, especially at the end of my stay, I wish we reached that state sooner. I put all my trust without worry in the remaining members of the team to skyrocket this project in the next years. I will be very happy to read your future papers!

Bibliography

[1] W. Abdulla. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. here, 2017.

[2] D. Acuna, H. Ling, A. Kar, and S. Fidler. Efficient interactive annotation of segmentation datasets with polygon-rnn++. 2018.

[3] L. Castrejón, K. Kundu, R. Urtasun, and S. Fidler. Annotating object instances with a polygon-rnn. In *CVPR*, 2017.

[4] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[6] J. Gardiner, J. Behnsen, and C. Brassey. Alpha shapes: Determining 3d shape complexity across morphologically diverse structures, 10 2018.

[7] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[8] K. N. Kutulakos and S. M. Seitz. A theory of shape by space carving. *International Journal of Computer Vision*, 38(3):199–218, Jul 2000. ISSN 1573-1405. doi: 10.1023/A:1008191222954. URL here.

[9] M. Liang, B. Yang, S. Wang, and R. Urtasun. Deep continuous fusion for multi-sensor 3d object detection. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XVI*, pages 663–678, 2018. doi: 10.1007/978-3-030-01270-0_39. URL here.

[10] H. Ling, J. Gao, A. Kar, W. Chen, and S. Fidler. Fast interactive object annotation with curve-gcn. In *CVPR*, 2019.

[11] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pages 163–169, New York, NY, USA, 1987. ACM. ISBN 0-89791-227-6. doi: 10.1145/37401.37422. URL here.

[12] D. Mandrioli. Semantic segmentation editor. here, 2018.

[13] J. J. Park, P. Florence, J. Straub, R. A. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. *CoRR*, abs/1901.05103, 2019. URL here.

[14] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation, 2016. URL here. cite arxiv:1612.00593.

[15] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space, 2017. URL here. cite arxiv:1706.02413.

[16] F. Razafindrazaka. Delaunay triangulation algorithm and application to terrain generation. 04 2009.

[17] S. R. Richter and S. Roth. Matryoshka networks: Predicting 3d geometry via nested shape layers. *CoRR*, abs/1804.10975, 2018. URL here.

[18] J. D. Stets, Y. Sun, W. Corning, and S. Greenwald. Visualization and labeling of point clouds in virtual reality. *CoRR*, abs/1804.04111, 2018. URL here.

[19] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. *CoRR*, abs/1703.09438, 2017.

[20] M. Tatarchenko*, S. R. Richter*, R. Ranftl, Z. Li, V. Koltun, and T. Brox. What do single-view 3d reconstruction networks learn? 2019.

[21] R. van Liere and R. J. An experimental comparison of three optical trackers for model based pose determination in virtual reality. 06 2004. doi: 10.2312/EGVE/EGVE04/025-034.

[22] B. Wang, V. Wu, B. Wu, and K. Keutzer. Latte: Accelerating lidar point cloud

annotation via sensor fusion, one-click annotation, and tracking. *arXiv preprint arXiv:1904.09085*, 2019.

[23] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y. Jiang. Pixel2mesh: Generating 3d mesh models from single RGB images. *CoRR*, abs/1804.01654, 2018. URL here.

[24] P. Wang, Y. Liu, Y. Guo, C. Sun, and X. Tong. O-CNN: octree-based convolutional neural networks for 3d shape analysis. *CoRR*, abs/1712.01537, 2017. URL here.

[25] W. Wang, Q. Xu, D. Ceylan, R. Mech, and U. Neumann. Disn: Deep implicit surface network for high-quality single-view 3d reconstruction. In *NeurIPS*, 2019.

[26] Y. Yang, C. Feng, Y. Shen, and D. Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation, 2017. URL here. cite arxiv:1712.07262Comment: Accepted as a spotlight paper in CVPR'18.

[27] W. Yuan, T. Khot, D. Held, C. Mertz, and M. Hebert. PCN: point completion network. *CoRR*, abs/1808.00671, 2018. URL here.

[28] X. Zhang, Z. Zhang, C. Zhang, J. B. Tenenbaum, W. T. Freeman, and J. Wu. Learning to Reconstruct Shapes from Unseen Classes. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.