Master Project

Image-based Shape and Appearance Modeling

Loïc Cordey

Supervised by:
Isinsu Katircioglu
Julien Pilet
Carlos Becker

Professor:
Pascal Fua

February 18, 2022

Acknowledgement

There are many people I would like to thank for their presence and precious help in this thesis.

I would like to start by thanking Pascal Fua who gave me the opportunity to take part in this project.

A special thank goes to Julien Pilet and Carlos Becker for their encouragement and help throughout this thesis. Their curiosity and willingness to help others grow has been inspiring and has really helped shape me as an engineer. More generally, I would like to thank the whole team at Invision AI for the workplace and the atmosphere they offered, which kept me motivated throughout the project.

I would like to thank Isinsu Katircioglu for her availability despite being in the process of completing her PhD thesis. I am grateful to her for her relevant perspective in the research field, as well as for her kindness in leading me through the different stages of this thesis.

I am very grateful to my family, my friends, my girlfriend Mélissa and my church for encouraging me, listening to me and praying for me when I needed it. They provide me a place where work had no room, and where I could fulfill my relational needs.

Above all, I would like to thank God almighty for renewing my strength and intelligence throughout these six months, and my Saviour, Jesus Christ, for his sacrifice on the cross.

"May my only boast be found in the cross of our Lord Jesus Christ", Galatians 6:14 TPT

Abstract

This thesis is the result of an internship at Invision AI and was co-supervised by CVLab at EPFL. As detection of objects has already been well tackled by several computer vision methods, image-based modeling is still an underexplored topic. In this study, we introduce a neural network which interprets images processed by a vehicle tracker to model the shape and appearance of vehicles.

To achieve this goal, we rely on a neural network which follows an autoencoder architecture. To our knowledge, we are the first to train a variational decoder with 3D ground truth data, while the decoder is playing the role of a continuous signed distance and appearance function to model the vehicle. This allows us to render an object with theoretically an infinite resolution.

We introduce an unique way to take advantage of the 3D bounding box detection of an object in an image, in order to generate a meaningful input for the encoder, which is easier to interpret than a raw image. Indeed, we feed to the encoder a 3D tensor computed from multiple homographies rather than a raw images.

We evaluate our method on synthetic data and achieve satisfying results. In most cases, we are able to obtain a good reconstruction of the vehicle, which we are trying to model. The reconstructed vehicles have a realistic overall shape, and we are able to capture small details in some cases. Also, we provide suggestions to improve the accuracy of the results in a bigger application.

Finally, we pave the road for future work. We acquired real data from multiple calibrated cameras. Although we did not have the time to process it in order to test our developed algorithm, we still show some methods that we explored to extract a 3D ground truth dataset from these images.

Contents

List of Figures

Introduction

This thesis was designed to respond to a company challenge. It's the result of a 6-month internship at Invision AI, a company aiming at improving computer vision tasks with artificial intelligence. Such tasks include detection, classification, facial recognition, object tracking, analytics and anomaly detection. This Thesis was supervised by the CVLab from EPFL. Our goal in this thesis is to improve a vehicle tracker so not only will we be able to perform detection, but also modeling of the shape and appearance of a vehicle, which can allow for the classification or the creation of correspondences between views from different cameras.

1.1 Problem Definition

Here are the details of the challenges, which we want to address: We already have a vehicle tracker, which works with one or more calibrated cameras. Given a video input from one or more camera, the tracker uses a neural network to detect vehicles on every frame. For each detected vehicle, the network predicts a 3D bounding box associated to it. The tracker combines them to compute a prediction in a time coherent way. The output is a set of vehicles detected in each frame, and for each detected vehicle, a 3D bounding box is predicted together with a 3D trajectory (Fig 1.1). Our goal is to model the shape and appearance of each vehicle detected by the tracker.

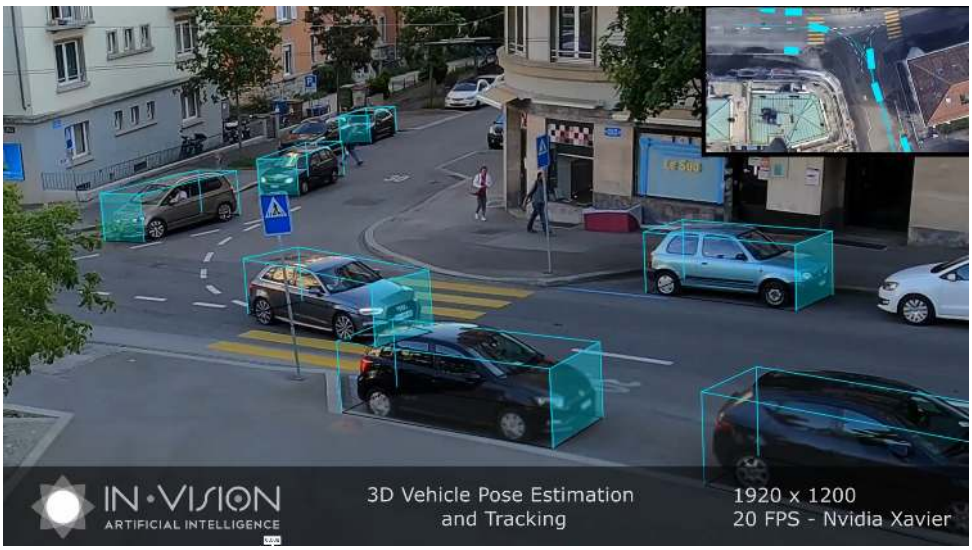


Figure 1.1: Example of the Tracker output, the camera is geo-referenced and the bounding 3D box of the detected vehicles are also geo-referenced. The tracker also detects the direction of the vehicle (front face of the bounding box is filled). The tracker also creates a bird's-eye view. Note that this result was computed using multiples frames.

1.2 Motivation

Detection and representation of 3D objects from one or more views have been addressed by many approaches and with many constraints, which we review in the section 2. While impressive results have already been achieved for vehicles and people detection, and has in fact already been well addressed by our tracker, many real world applications require an accurate shape representation, for instance, to compute the reflections of the light or waves on objects.

Also, modeling the shape and appearance would be useful to measure similarity between vehicles. As an example, it could be used to establish correspondences between the detections on multiple frames or from different camera views.

Although there have already been some explorations aimed at predicting the exact shape rather than simply predicting the bounding box, appearance modeling is still largely under-explored. Since for a specific object, like vehicles in our case, texture and shape are inherently correlated, we try to leverage this relationship to improve the shape and appearance of the prediction.

1.3 History and Actual Challenges

Standard computer vision methods, such as computing the visual hull of an object or computing a depth map from stereo images, have proven to be effective in reconstructing 3D shape from multiple views[1][2]. However, these methods rely on a multi-view input with a known silhouette at the inference time, which we do not have in our case.

To overcome this limitation, neural networks have proven to be of great help in filling the gaps in these computer vision tasks. Indeed, detection and prediction of the bounding box corresponding to an object on an input image, have already been well addressed by many object detection algorithm such as YOLO[3], SSD[4] or R-CNN[5].

Although, segmentation of 2D silhouettes from images has also been well tackled by some work such as Mask-RCNN[6], modelisation of the 3D shape of an object from one or more camera view is still an actual challenge. As obtaining ground truth 3D supervision for real datasets is challenging, this task has been tackled using only RGB images with annotated 2D silhouettes[7][8]. 3D models have been represented in several ways, the most notable being meshes[9], points clouds[10], or implicit representations such as the signed distance function (SDF)[11], which are function that return the distance to an object with respect to a specific location.

Eventually, as it is already difficult to model a 3D shape from images, modeling the appearance is even more challenging.

1.4 Our Approach

To achieve our goal in this work, we represent the 3D shape of a vehicle with a continuous implicit field. The field corresponds to a specific vehicle, which we want to represent, and is defined by the output of a neural network. The neural network predicts the SDF of the vehicle, as well as its appearance (RGB values), according to a specific 3D location in space.

As we only want to represent a specific type of object, in this case vehicles, we want to limit the number of prediction types made by our algorithm. We build our method on a network structure similar to a variational auto encoder[12], which embeds all the possible objects, that can be represented in a latent space. Then, for a specific vehicle, which will be associated to a specific code in this latent space, we will try to predict its shape and appearance. In order to train this network that will determine our latent space, we rely on 3D ground truth datasets.

Unlike the earlier approaches, which take as input a set of images, we rely on images and 3D bounding box of the vehicles. We will present a way of exploiting this additional information to enhance the

quality of an encoder prediction. Furthermore, we can refine the prediction by exploiting the latent space structure, in order to later improve the quality of the results.

1.5 Contribution

This thesis consists of two main contributions:

Simultaneous shape and appearance reconstruction with 3D data. Building a 3D ground truth dataset is a painful task, that's why most works on shape estimation are based on 2D datasets. As we were able to collect video data with numerous calibrated cameras, we took the bet, that we could build such a 3D dataset. To the best of our knowledge, we are the first to propose a shape and appearance modelisation algorithm, which learns with 3D supervision, as presented in this thesis.

Prediction from images and 3D bounding box. Furthermore, as we target a specific real world application, we present a unique way of exploiting the 3D bounding box predicted by the tracker. In order to provide an input to the encoder, which would contain more meaningful information than a simple image without any other information.

1.6 Outline

In chapter 2, we present the related work on the different technical aspects, for which we had to make choices. In chapter 3, we introduce the method, which we chose to implement. Then, we describe how we processed the data, before presenting the data evaluation in chapter 4 and 5. Eventually, we present some future work and the explorations, which we did in this direction. Finally, we expose the main conclusions of our work.

Related Work

In this chapter, we detail the different technical aspects of the work presented in this thesis. Indeed, we had to make several choices, such as: how to represent the shape of the vehicle, how to represent its appearance, and do we train our networks with 2D or 3D supervision. We also explored, if we needed to compute the silhouette at the inference time, only during the training or not at all. And finally, we tried to leverage the fact that we only detects vehicles to facilitate the prediction for a network.

We will go through the literature related to these questions, the goal being to present the pros and the cons of the possible options and what possibilities and constraints they can offer.

2.1 Shape Representation

Explicit representation. Meshes are probably the simplest way to represent a 3D object. Prior works[8][13][14][15] offered solutions to enable a differentiable rasterization with respect to some ground truth images, in order to optimize the mesh prediction by derivation and iterations. However, meshes are subject to limitations, as it is still unknown how to properly generate them dynamically, in order for them to adapt in topology. Furthermore, if we want to make a prediction using a neural network, a proper way to adapt the resolution would be needed, which is not reported in the literature. Therefore, in their study, Chen et al.[8] assumed a fixed resolution, i.e. a fixed number of vertices with predefined faces.

Note that other explicit representations exist, such as points clouds, but they are usually not well suited due to the difficulty of ensuring that they represent watertight surfaces, which are closed surfaces that doesn't contain holes and for which the inside is clearly defined. Also, as for meshes, points clouds suffer from a resolution limitation.

Implicit representation. Deep Signed distance fields, which are field where the distance to an object is defined at every point of the field, have gained much attention thanks to a study that has shown how to make the rendering process differentiable, allowing for a loss function to be computed to either train a network[16] or refine a prediction[17][18] from some input images. This has led to many recent advances, with different methods being explored to improve the quality of the results given by a neural network trained to predict a signed distance field. For example, one can improve the prediction of such a network by training from coarse to finer resolution[19], or learning the shape locally[20]. Signed distance function has the advantage of being unrestricted in its resolution and being able to represent objects of different topology while ensuring a watertight surface.

Rendering of such implicit representation can be done in various ways. For example, in the case where the implicit representation is the signed distance function corresponding to an object, one can simply generate images from different points of view with ray marching (which is an algorithm that can render images from signed distance field, more explanation in section 6.1). However, the high number of ray marching iteration required to achieve good convergence leads to a high computation cost and time, and it makes the process non-differentiable, which limits this process and make it unsuitable for

network training. To overcome this limitation, recent works[21][7] propose to perform a ray marching rendering by predicting the size of the marching step with a Long Short Term Memory network.

Other methods[21][22] based on novel view synthesis present different ways of encapsulating the object representation in a smaller dimensional space, in combination with a learning network to render an image based on this information. However, it does not ensure the consistency of the rendered views from different positions.

2.2 Appearance Representation

Representing appearance depends on the shape representation method. Although, for meshes it is standard to use a texture map or even to assign colors to vertices, representing appearance for implicit representations is still an under-explored topic.

For implicit representation, earlier approaches[22] which embed the object representation in a latent space propose to disentangle the shape from the appearance. When representing an object with a signed distance field, a first option[7] would be to predict the appearance at every location in the space with a decoder. It would be done similarly to the prediction of the distance field, but instead we would get an "appearance field". Another approach presented in the literature[21] would be to predict a feature vector for each point in the space. Thus, by performing ray marching for each pixel, we would get a feature image which could be given to a decoder to generate an image. This could improve the prediction capability of the network by using the correlation between nearby pixels, the SRN paper[21] hypothesis suggests it for the reconstruction of high frequency details.

In order to improve the appearance results, some papers propose to use a Generative Adversarial Network (GAN[23]) for the rendering of the final images. As shown in their work[8], this resulted in more realistic images and better reconstruction of high frequency details. Lastly, modeling the lightening effect can also lead to more realistic results[8].

2.3 Dimensionality reduction

PCA. Embedding the ensemble of all the possible objects which we want to represent can enhance the prediction capacity of an algorithm as the set of prediction types would be already narrowed to a specific type of object, in our cases vehicles. Indeed, a previous work[24] predicted 3D shape of vehicles, by fitting the best PCA coefficients into a pretrained latent space representing vehicles. They obtained decent results with this approach, nevertheless linear combinations of the eigen vectors can result in non-realistic vehicles. For this reason, we will prefer using neural networks, as they can offer a structured space which would embed more realistic vehicles overall.

AutoEncoders. With the rise of neural networks, AutoEncoders[25], have become an effective way to embed complex information into a reduced dimensional space. Indeed, AutoEncoders are a combination of an encoder which assigns a given input to a vector in a lower dimension, and a decoder that interprets a given code in order to recover a desired output. Moreover, Variational AutoEncoders[12] are efficient to impose a structure to the latent space. Basically, Variational AutoEncoders provide better generalization for the network, as they assign an input information into a distribution rather than a specific vector. Eventually, as shown by deepSDF[18] for instance, an auto Decoder could be used alone, without the need for the encoder, if one only wishes to create a structured latent space.

2.4 3D Supervision

Some studies encountered in the literature[18][19][20] are working with 3D supervision. However, they only try to predict the shape and not the appearance of the prediction target. To our knowledge, there are no works which uses 3D supervision to model the appearance. Indeed, this is what we will explore

in this thesis. With regard to the level of information that 3D supervision can provide, it appears that working with it would facilitate the training of a neural network, the main difficulty being the obtention of real data.

2.5 2D Supervision

Since 3D ground truth datasets are hard to obtain for real data, most of recent works have been focused on 2D supervision. Deep Network aiming at modeling an implicit surface has grown a lot recently. Great progress[26][27] has been showed in the context of reconstructing 3D textured surfaces of clothed humans from single and multi-view input, but also for other various type of 3D object with only a single-view[28]. Other method such as SDF-SRN[7] have also attempted to model appearance and can be trained only with single view of objects. Eventually, some works[8] also tried to represent objects with meshes and showed some great results using 2D supervision.

Still, even though 2D data are more accessible and easier to collect, all the works that we've seen need those 2D data to be annotated with silhouette, at least during the training time. The need of the annotated silhouettes is constraining but could possibly be tackled with known standard computer vision methods, or with neural networks such as Mask R-CNN[6].

Method

In this chapter, we describe in detail the general pipeline that we want to apply to the output of the tracker, as well as the technical choices that we've made. Our algorithm has two main parts: Similarly to a variational autoencoder architecture (Fig 3.1), we present an encoder network which takes an input generated from an image for which we know the 3D coordinate of a detected vehicle, and predicts a latent code in a latent space. We also present a decoder network which outputs a signed distance to the vehicle's surface as well as RGB values, given a specific 3D location and a latent code in a latent space corresponding to a model. However, unlike a standard variational autoencoder, we trained an encoder and decoder separately.

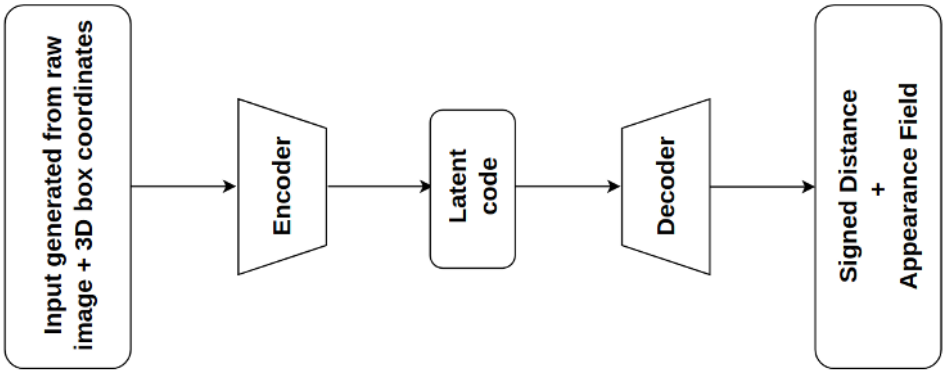


Figure 3.1: Scheme of an autoencoder architecture.

3.1 Variational Decoder

The implementation is similar to the auto-decoder presented in DeepSDF[18]. The structure of the network is a simple multi-layer perceptron. It takes as input a 3D location in space, as well as a latent code represented by a tuple of L dimensions, and outputs 4 values: the signed distance and the RGB intensity. The decoder is said to be variational, as at training time, it trains using a distribution around the input latent code rather than the exact code.

3.1.1 Implementation

Training Data We first trained the variational decoder to learn the shape and appearance of synthetic models, which we have preprocessed to extract training data. For each model, we extracted a grid of 64 x 64 x 64 locations in space. At each of these locations, we know the ground truth signed distance value relatively to the model of the vehicle, and the RGB value of the closest face.

This data is fed into the training process with a data loader in the following way: Each epoch was composed of N models x 64 x 64 x 64 sampling locations. Each training step was composed of 10'000 samples, for each sample the network predicts 4 values: the signed distance, as well as the red, blue

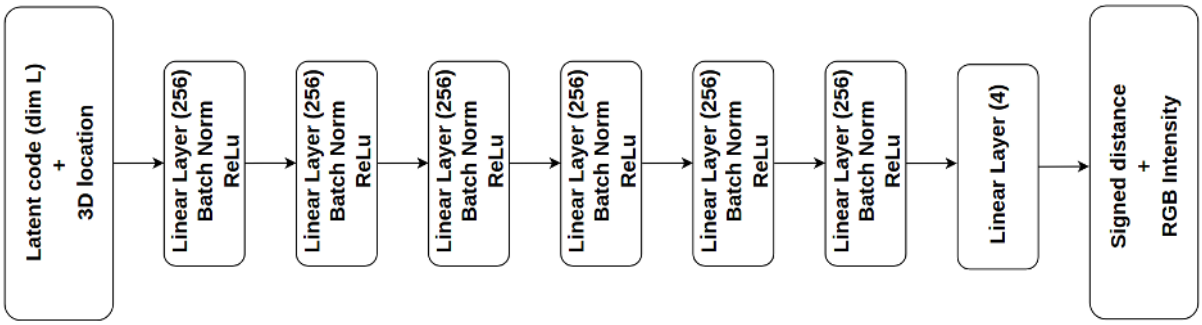


Figure 3.2: Decoder architecture, the output size of each Linear layer is specified in parentheses.

and green intensity. The batch size of 10'000 has been found empirically. Note that fewer samples per batch led to a slower convergence in time, while too many samples would raise the risk of getting stuck in local minima.

Decoder Structure. As shown on Figure 3.2, the structure of the network is an MLP. The first layer goes from 3 (3D location) + L (dimension of latent space) dimensions to 256, it is followed by 5 layers of 256 x 256 neurons. All these fully connected layers are followed by a batch normalization and a ReLu layer. There's a final fully connected layer going from 256 dimensions to 4 (SDF value, R, G, B). Finally, there's an activation layer that only applies for the RGB values, this activation layer is in the form: $f(x) = sigmoid(\lambda * x)$ with $\lambda = 3$ found empirically too. Note that lower value of λ makes the training focus more on the whole range of color intensity, whereas higher value of λ tends to minimize the importance of the RGB values close to saturation.

These parameters have been found empirically. In the design process, we tried to add/remove batch norm, change the number of fully connected layers and the number of neurons per layer, and use tanh instead of sigmoid functions. At the end, the parameters kept were chosen based on the fact that they provide a good trade-off between convergence and training time.

Latent space Construction. The decoder gives different outputs for the same location, given different codes of a latent space. In this way, this latent space is a low dimensional space where each latent code is a representation of a specific vehicle and the network behaves like a Signed Distance Function (SDF) corresponding to a specific model for a given latent code.

The goal is not only to reduce the representation of the vehicles to a lower dimensional space, but we also desire this latent space to be structured. We mean two things by that: First, we want two similar vehicles to be assigned to two codes close to each other in the latent space, so that we can measure the similarity of two vehicles by their respective proximity in the latent space. This could allow us to establish correspondence between the detections of the trackers from different points of view. Second, we want the latent space to be dense, i.e. we want that any interpolation of two codes corresponding to two different models used for training would also represent a realistic vehicle. More generally, we want that any code in our latent space represents a realistic vehicle when given to the decoder.

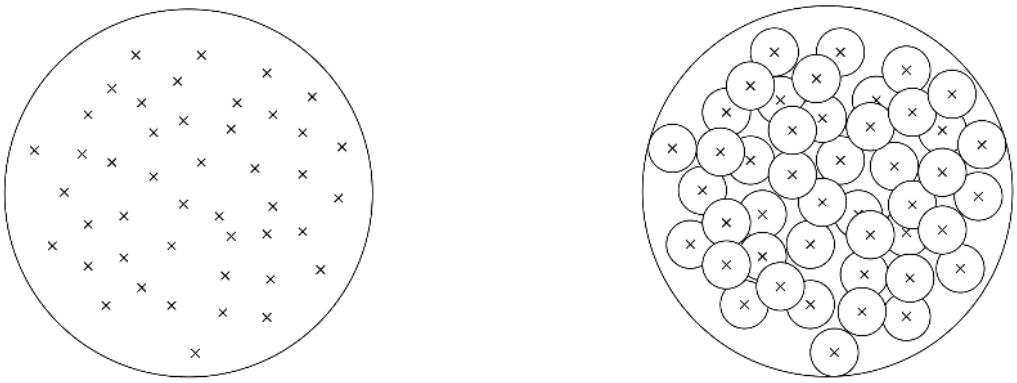
To achieve this goal, we implemented the network as a "variational decoder" similarly as a Variational autoencoder works[12]. At initialization, we assigned a random code (vector in the latent space) to every training model. While training, we train the parameters of the decoder, and we also train the codes of the models so that during training the codes of similar vehicles get closer, as it will be easier for the network to predict the good SDF and RGB values if the code of similar vehicles are close. The training is done with the standard Adam optimizer[29], and the computation of the loss is described later in this section. Furthermore, during the training time, we did not give the exact code of a model to the decoder, but rather we gave a code in the region of the model code. The codes were computed in the following way:

Let z_m be the code associated to a model m, then the code which is given as an input to the decoder is:

$$\bar{z}_m = z_m + u \cdot \sigma_m \tag{3.1}$$

with $u \sim \mathcal{N}(0, 1)$, and σ_m a standard deviation corresponding to the model m, note that σ_m is also a trainable parameter, just as z_m. This forces the network to assign a region of the latent space to a specific model, rather than just a specific location. This region's position and width were trained in parallel with the decoder parameters.

On the Figure 3.3a, we show a scheme of the latent space resulting from the training of a standard decoder. The codes of the training models are represented by an "x" and the latent space by a circle embedding all the codes. On Figure 3.3b, we show the latent space resulting from the training of a variational decoder. Rather than associating each code to a unique point in the space, the variational decoder will associate each code to a region represented by a small circle around each "x".



(a) Representation of the latent space resulting from a simple decoder

(b) Representation of the latent space resulting from a variational decoder

Figure 3.3: Latent space structure comparison, the crosses "x" represent the latent code of the models used for the training. They are embedded by a circle representing the latent space. On the second figure, the circles around the "x" represent the regions associated to a model during training, by the variational decoder.

Loss computation. As we chose to represent the vehicles with an implicit function, the implicit surface of the object is where the signed distance function evaluates to 0. Since the surface is what will be rendered, it is more important to have a better accuracy around 0, as long as the sign is correct. To achieve this, DeepSDF[18] and other studies chose to clamp the SDF values when they are higher than a certain threshold. After some experimentation, we reached better results with the following procedure: Instead of only clamping the SDF values when they are above a given threshold, we assign to them a "weights" boolean vector, which completely disregards the loss obtained when the ground truth absolute SDF value and the predicted absolute SDF value are both above a certain threshold. We define this threshold as the minimum distance between two sampling locations.

Let be a batch of N samples, with $\bar{s}_n$ the signed distance value predicted for the sample n, s_n the ground truth one, and a threshold T. Then, the weights w_n of one sample n of the vector $\mathbf{w}$ is defined as follows:

$$w_n = \begin{cases} 1, & \text{if (abs}(s_n) < \text{T or (abs}(\bar{s}_n) < \text{T or sign}(s_n) \text{ != sign}(\bar{s}_n)). \\ 0, & \text{otherwise.} \end{cases} \tag{3.2}$$

Also, we chose the Mean Square Error loss rather than the simple L1 norm, this is because we wanted to capture small details such as the shape of the rearview mirrors or the color of the headlights and the flashing lights.

Therefore, for a training step of N samples, **w** the "weights" boolean vector, $\bar{s}_n$ the signed distance value predicted for the sample n, $\bar{r}_n$, $\bar{g}_n$, $\bar{b}_n$ the predicted RGB values s_n the ground truth signed distance value, and r_n, g_n, b_n the ground truth RGB values, then the loss for shape and appearance are:

$$loss_{sdf} = \frac{1}{N} \sum^{n \in N} w_n (\bar{s}_n - s_n)^2 \tag{3.3}$$

$$loss_{rgb} = \frac{1}{N} \sum^{n \in N} w_n ((\bar{r}_n - r_n)^2 + (\bar{g}_n - g_n)^2 + (\bar{b}_n - b_n)^2) \tag{3.4}$$

As presented in the variational autoencoder work[12], we want to add some constraints when training the disposition of the latent code. We want the codes to be as close to 0 as possible, so that the training process is encouraged to arrange the codes in the space efficiently because the region associated to each code will be encouraged to overlap each other. And we want the variance, which represents the width of the region of a code used for the training, to be close to 1 for the normalization. In order to achieve this, we derived the loss from the standard Kullback-Leibler divergence[30].

It leads to the regularization loss:

$$loss_{kl} = -\frac{1}{2} \sum^{m \in M} \sum^{l \in L} (1 + 2 \log(\sigma_{m,l}) - z_{m,l}^2 - \sigma_{m,l}^2) \tag{3.5}$$

with $m \in M$ the model and $l \in L$ the dimension of the latent space.

This leads us to three losses: one loss for the SDF error and one for the RGB error, which are computed using the mean square error of the meaningful samples (samples for which the weights vector are not 0), and one regularization loss. These three losses are summed and weighted by a coefficient to obtain the total loss:

$$loss_{total} = \lambda_{sdf} \cdot loss_{sdf} + \lambda_{rgb} \cdot loss_{rgb} + \lambda_{kl} \cdot loss_{kl} \tag{3.6}$$

Where $\lambda_{sdf} = 1$, $\lambda_{rgb} = 0.01$ and $\lambda_{kl} = 0.001$ were found empirically.

3.2 Encoder

The encoder architecture follows the one from a convolutional neural network. A 3D input tensor given as input passes through convolutional, MaxPool and ReLu layer to obtain a feature vector, which is then regressed to a latent code by passing through an MLP.

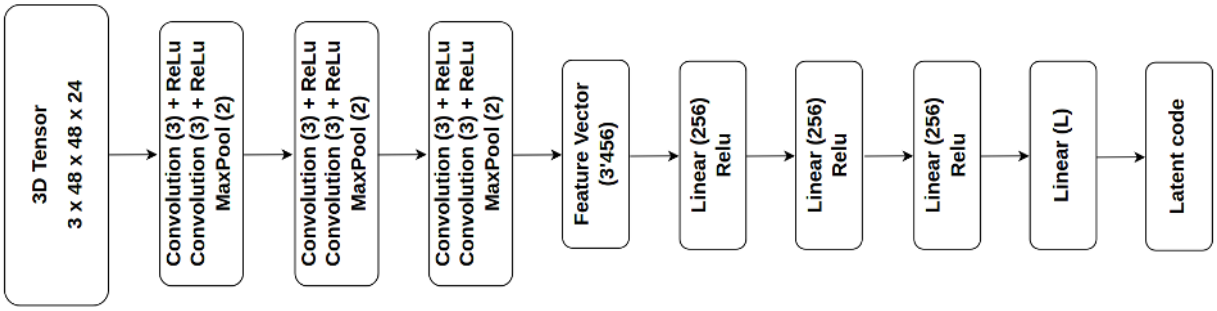


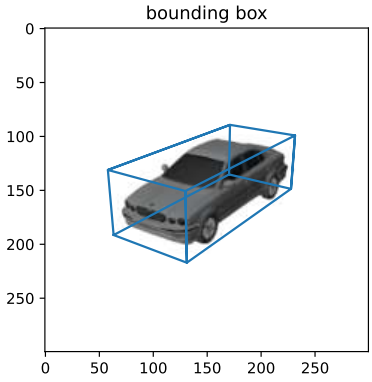
Figure 3.4: Encoder architecture, the kernel size of 3D Convolution and 3D MaxPooling as well as the output size of Linear layers are indicated in parentheses.

3.2.1 Implementation

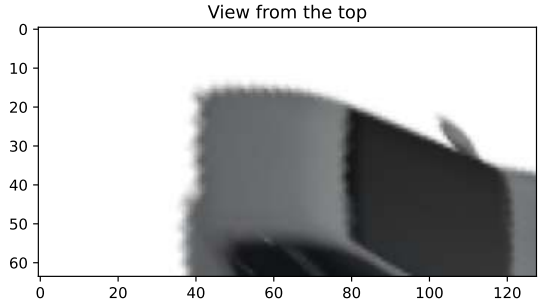
Training Data The challenging part of the encoder is to find an efficient way to use the output of the tracker, which is an image from a calibrated camera with a prediction of the 3D bounding box of the detected vehicle. We want to give an input for the encoder which will be easier to interpret than the raw image, to then predict the latent code corresponding to the observed vehicle. Also, the ground truth code is known from the training of the decoder, which is explained before. To design and train such a network, we tried to convert the output of the tracker into an encoder input in three different ways.

The first naive way was simply to give the raw image and the 2d coordinates of the bounding box in the image. This solution does not leverage well the information at our disposal, its main purpose was to be used as a baseline to evaluate the other ones.

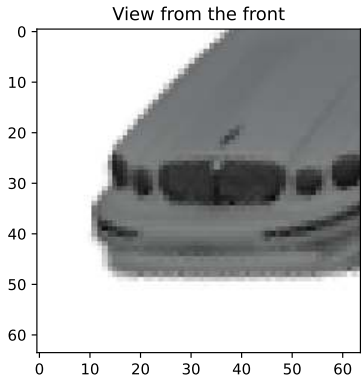
For the next method, we aimed at giving more relevant images as the input of the encoder than the raw ones, which are taken from an arbitrary point of view. To that end, we computed five homographies (geometric transformation that projects a surface plane into another point of view) from the raw image, using the bounding box coordinates (3.5a). With this approach, we obtain 5 views which are then given to the network, one from the front, the back, the left, the right, and the top. An example of the result of such homographies is shown on Figure 3.5.



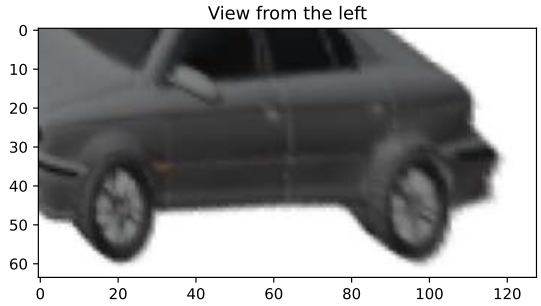
(a) Example of frame with the bounding box.



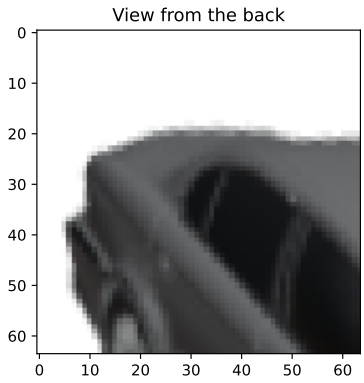
(b) Top view.



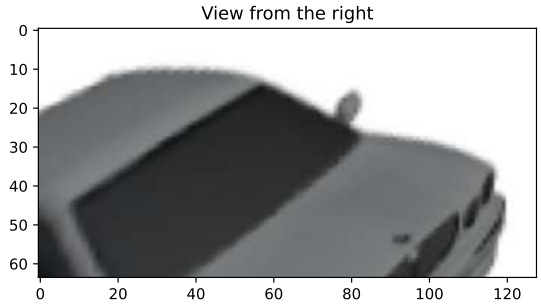
(c) Front view.



(d) Left view.



(e) Back view.



(f) Right view.

Figure 3.5: Example of the second way to transform the input for the encoder. From the raw images and the bounding box, we extract 5 homographies to simulate a view from each side. Of course, when the angle of a homography becomes large, the result does not make sense for a human. As an example, the back (3.5e) and right views (3.5f) are occluded. Still, we make the assumption that it will facilitate the training for the encoder as we do it consistently.

Finally, here is the solution we chose. We kept the idea of the second one where we computed homographies in order to give more meaningful views, while at the same time taking advantage of having the 3D bounding box positions. However, unlike the second solution, we only computed views from the front. Furthermore, we computed several ones at different depths to linearly interpolate along the length of the vehicle. In the Figure 3.6, two examples of slices used to compute the homographies are shown in red, only 2 of them are shown, even though 48 of them are used to obtain the final grid given as input to the encoder.

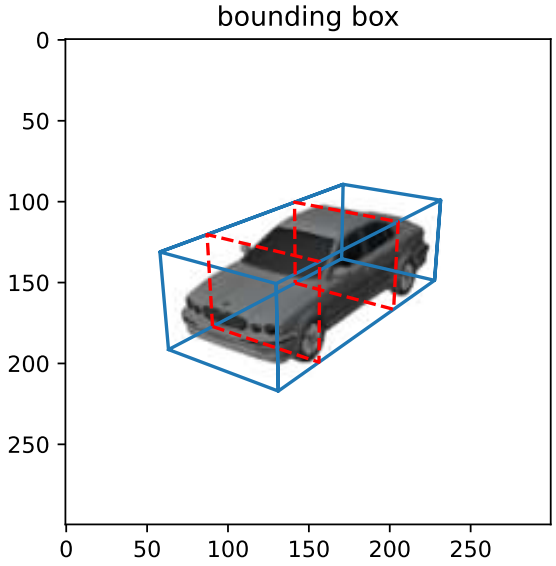
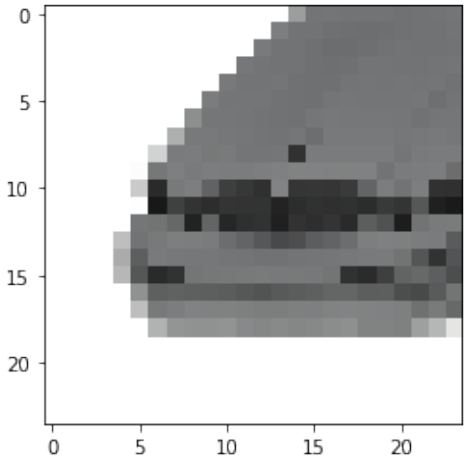
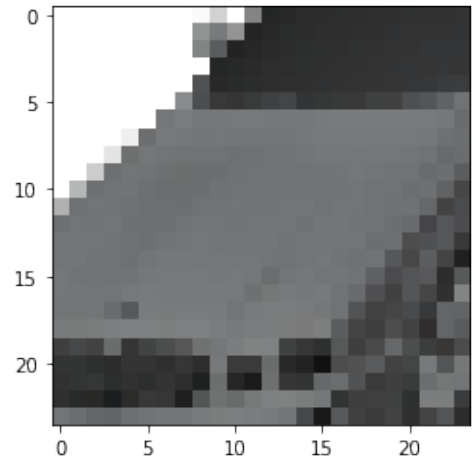


Figure 3.6: Example of slices used to compute the homographies are shown in red.

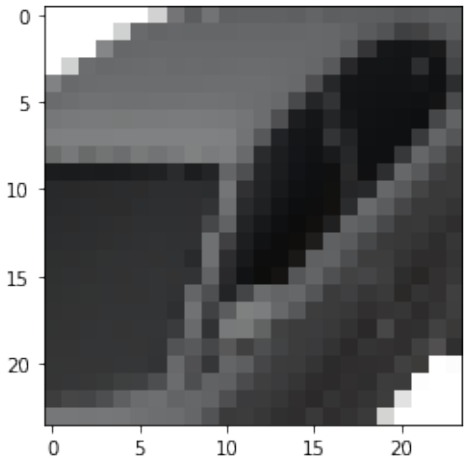
In this way, the computed homographies are correlated with one another and instead of giving separated views, we can assemble a 3D grid from the images resulting from the homographies and apply a 3D convolution on them to take advantage of the correlation between them. Due to the high computational cost of a 3D convolution, we constructed a 3D grid with a reasonable resolution. Therefore, for each images with the 3D bounding box output of the tracker, we constructed a grid of 3 RGB channels x 48 slices x 24 x 24 images that we give as input to the encoder. In Figure 3.7, one can observe a sample of the images that are assembled to obtain the 3D grid, the first (3.7a) and last (3.7d) figures are the images at the edges of the grid. the Figures 3.7b and 3.7c are the results of the homographies computed from the two red slices shown in Figure3.6.



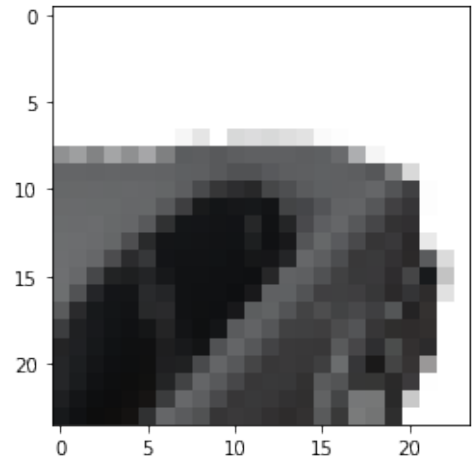
(a) Front edge of the 3D grid.



(b) Homography corresponding to the 1st red slice on Figure 3.5a.



(c) Homography corresponding to the 2nd red slice on Figure 3.5a.



(d) Back edge of the 3D grid.

Figure 3.7: Samples of some slices assembled to build the 3D grid.

Encoder Structure As shown on Figure 3.4, the network architecture is a standard convolutional neural network, with the particularity that we apply 3D convolutions. The feature extraction is computed by passing sequentially through three convolutional blocks. Each block is composed by a sequence of 3D Convolution - ReLu - 3D Convolution - ReLu - 3D MaxPool. The convolution parameters were set to a kernel size of three and a padding of one in every direction. For the MaxPooling layers, the kernel size as well as the stride were set to two. Therefore, these features extraction layers take as input a 3 x 48 x 24 x 24 3D grid, and output a 64 x 6 x 3 x 3 features map which is then flatten to a 1D vector with a size of 3'456. This vector is then passed to a classification block, composed of four fully connected layers, with the first three of them followed by a ReLu layer.

All convolutional and fully connected layers have 64 channels, except for the entry of the classification block for which the first fully connected layer has an input size of 3'456, and the last fully connected layer has an output size equal to the dimension of the latent space L. In general, increasing the number of layers and channels could improve the results. In our case we kept them reasonably low as we worked with a dataset of limited size, but it might be necessary to design a network with more parameters for more complex and bigger datasets.

As for the decoder, the choices made in the architecture design were decided empirically. Note that we also tried to use batch norm layers after the convolutional and/or after the fully connected layers.

Also, dropout layers could improve the network generalization, but this hasn't been explored during this project and could be interesting for future works.

As a reminder, the goal of this encoder is to predict the latent code associated to the model of a vehicle on an image given the corresponding 3D bounding box. The ground truth code is known from the results of the variational decoder training. The computed loss used for the training is simply the Mean Square Error between the ground truth code z_m and the predicted code $\bar{z}_m$.

$$loos = \frac{1}{M} \sum^{m \in M} (\bar{z}_m - z_m)^2 \tag{3.7}$$

3.2.2 Variational AutoEncoder

Note that the most standard way of using both an encoder and a variational decoder is to train them together from end to end. This is what we usually refer to as a variational AutoEncoder [12].

The choice of training them separately was taken because in the first place we did not need the encoder. Indeed, we can use only the variational decoder to create a latent space. Then, initialize a random latent code and render an image from it, compute a loss with a ground truth image containing a vehicle and try to find the best code corresponding to the vehicle by optimization using this loss.

The Encoder was introduced later, as it helps to have a crude estimation rather than a random code at the beginning. Even though, it is still possible to optimize the code given by the encoder in the same way as we would do with a random initialization.

Also, it was easier to implement them separately, as it is more controlled and easier to debug. Indeed, we tried to combine them afterwards, but the results were not satisfying as we did not manage to make it converge, although we could have put more effort into it.

Data Processing

The training of our network requires training data. We need the ground truth signed distance value sampled in the space area of the vehicle with a sufficient resolution. For each model, we also need images from random points of view, for which we know the position of the 3D bounding box of the vehicle, as well as the intrinsic and extrinsic calibration of the camera. In this chapter, we will detail how we managed to acquire and process the data. Note that all the Figures are meshes computed with the marching cube algorithm[31] that we rendered in MeshLab, which is a software that we use to visualize meshes.

4.1 Synthetic Data

The major part of this work was done using synthetic data, this is mainly due to the fact that acquiring 3D ground truth supervision is not an easy task, but also because it is easier to work with synthetic data for which we have a more controlled environment. The data are meshes downloaded from the Shapenet dataset[32], in which we only kept the models of type vehicle. Among these models, we sorted them to only keep the types of vehicle that we could supposedly encounter on the road and which we would like to detect. As an example, we removed the vehicles with a shape that is not representative of what we want to detect, and we removed as the well the ones with a non-uniform color distribution. Some examples are shown on the Figure 4.1.

(a) While the shape is standard, we want our network to learn that usually the color of a vehicle is uniform, and not multicolor like this vehicle.

(b) Since the shape of this vehicle is relatively non-standard, we don't want the decoder to learn this type of shape.

Figure 4.1: Examples of models that weren't kept because we do not want our network to learn the shape and appearance of these vehicles.

4.1.1 SDF Extraction

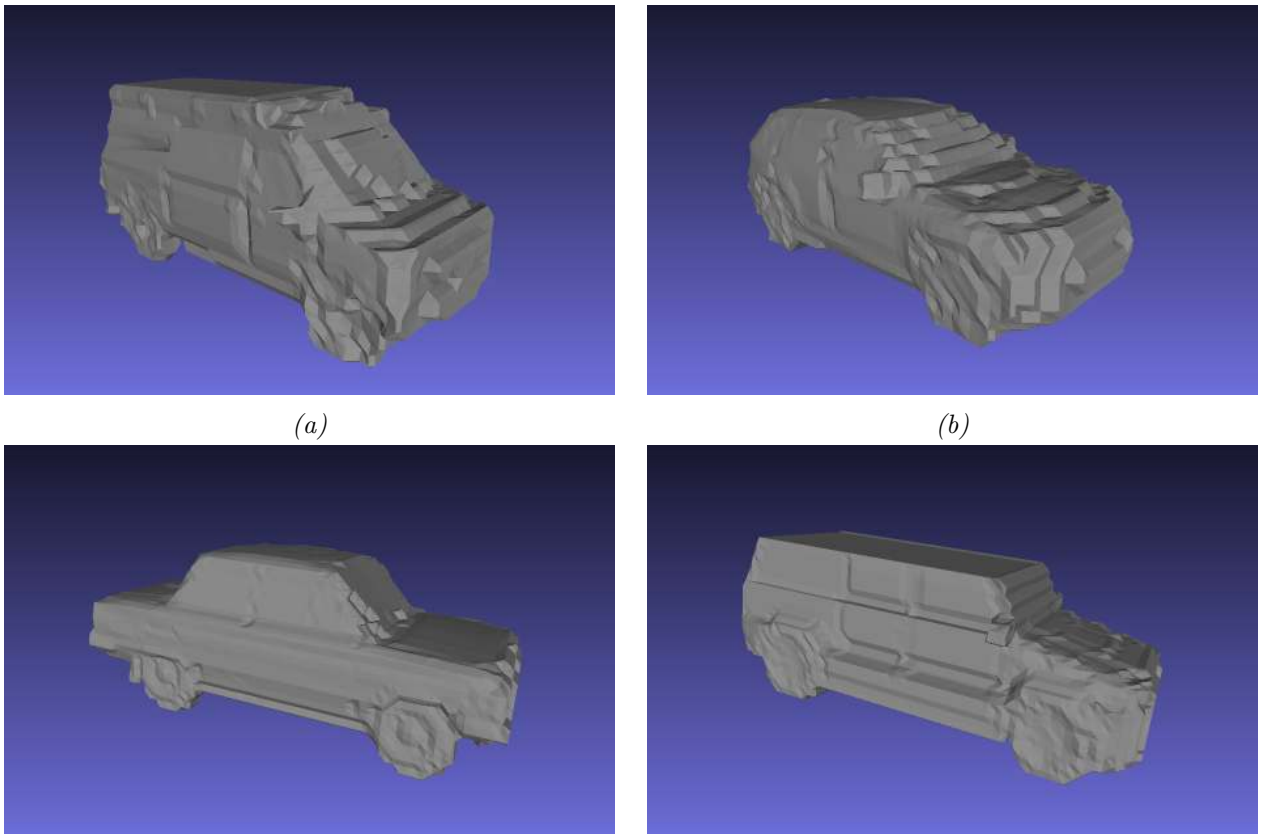
The Training of the decoder needs SDF ground truth. For this, we need to be able to extract the SDF values from the meshes obtained in the Shapenet dataset. We used the mesh voxelization project found

on GitHub (https://github.com/davidstutz/mesh-voxelization)[1] [33][34]. This folder contains multiples scripts, but only three of them are mainly used in this thesis:

- *scale_off.py*: Scale the raw meshes to lie in [0, H] x [0, W] x [0, D] corresponding to the chosen resolution H x D x W. Input format should be an ".off" file, the output is an ".off" file too. This script also accepts a "padding" parameter that we use to introduce empty space around the mesh. This is necessary to have positive values at the edges, when we render the meshes. Otherwise, we would not see the surface, as it is corresponding to the space where the SDF crosses 0.
- *voxelize (bin)*: Voxelize the scaled meshes into SDFs. The output file, will be a HDF5 file containing either an N x height x width x depth tensor, where N is the number of files, which is discarded if only a single file is processed and height, width and depth are inputs parameters. It should have a ".h5" extension. The script is also able to output occupancy grid instead of SDFs, but this feature won't be used in this thesis.
- *marching_cubes.py*: Convert the SDFs tensor obtain from the voxelization to meshes with an ".off" file extension, for the visualization.

These scripts are useful but they need an ".off" file as an input, however the Shapenet dataset contains meshes with ".obj" files. Therefore, we used an "OBJ Loader" found on https://github.com/Bly7/OBJ-Loader which we adapted to convert the ".obj" to ".off" files.

The full pipeline is constituted of the following steps: converting the files containing the meshes from ".obj" to ".off", scaling them, extracting the SDF tensor, and finally reconstructing the meshes from these SDF tensors. Figure 4.2 depicts some examples of the results that we obtained.

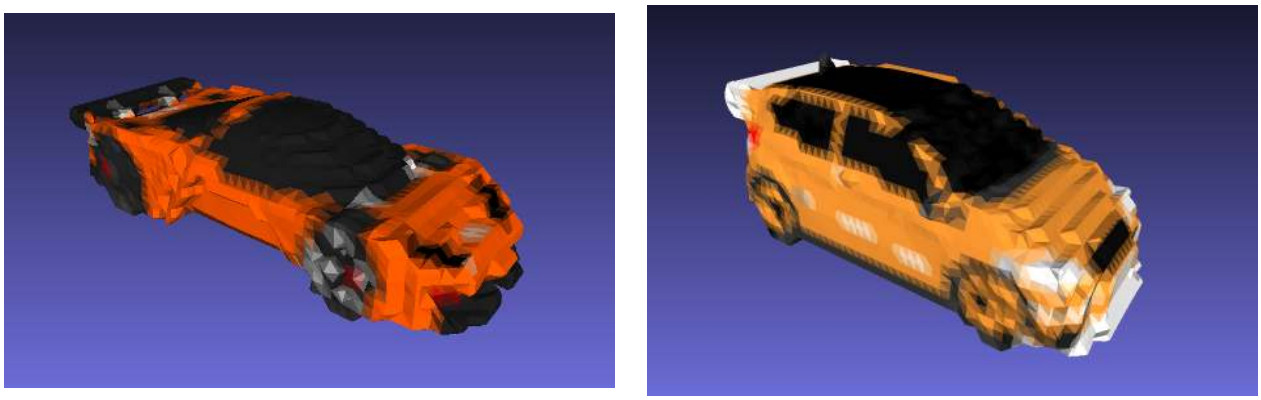


(a) *(b)* *(c)* *(d)*

Figure 4.2: Example of shape reconstruction after the processing of the data using the voxelization script.

[1]We used the branch from Carlos Becker which include some bug fixes: (https://github.com/cbecker/mesh-voxelization/tree/cb/temp-fixes)

Also, the scripts does only extract the shape via the SDF, thus we have to adapt every script to also keep the color of the meshes, when converting them from ".obj" to ".off". Then we have to scale them, to extract the colors of the closest faces in addition to the SDF at every sampling locations, and to reconstruct a colored mesh from the tensor obtain by the voxelization. The Figure 4.3 shows a subset of the reconstructions we obtained after extracting the color.

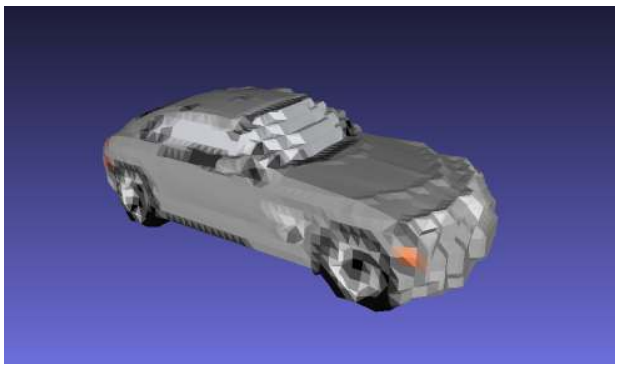


(a) *(b)*

Figure 4.3: Examples of two colorful reconstruction.

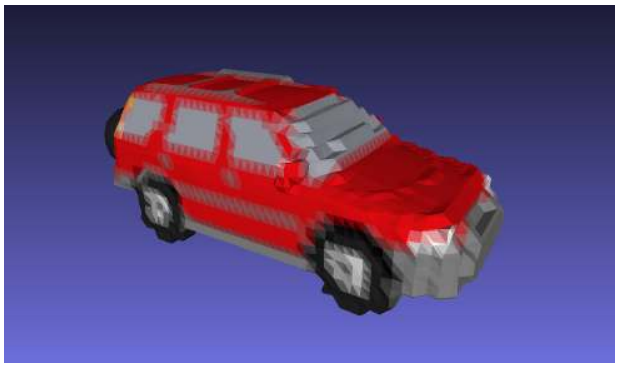
Ideally, we would like to use random locations in space for every sample during the training of the network. However, the task of extracting the ground truth SDF and RGB values with the scripts is computationally expensive. So we decided to preprocess each mesh at a resolution of 64 x 64 x 64, in order to obtain a satisfying resolution while keeping it computationally reasonable. Note that due to this sampling in the data pre-processing, the decoder will be trained to learn an approximation of the original mesh. The Figure 4.4 shows the difference between the original mesh and its reconstruction after the sampling.

(a) Original mesh of the vehicle.



(b) Reconstruction after the data processing.

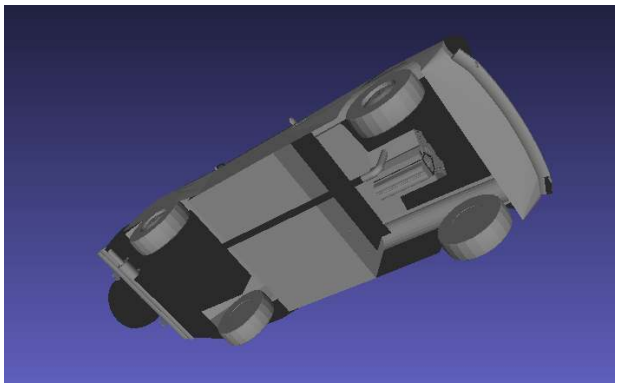
(c) Original mesh of the vehicle.



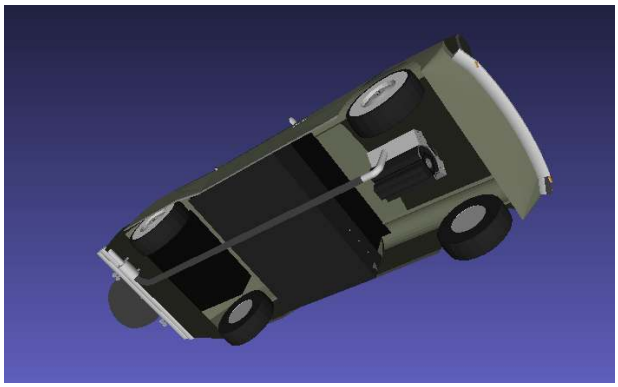
(d) Reconstruction after the data processing.

Figure 4.4: Due to the sampling in the processing of the data, the tensor used for the training is an approximation of the original mesh. On the left, one can see the original mesh downloaded from shapenet, and on the right, we observe the reconstruction with the tensor obtained after the data processing.

Eventually, some models had to be removed manually from the training set because their mesh was not watertight, e.g. Figure 4.5, making it impossible for our script to predict the SDF, as it would be positive everywhere. This constraint makes it impossible for us to compare our results properly with others papers, as we can only use a subset of the dataset.



(a) Empty mesh, colorless visualization.



(b) Empty mesh, colorful visualization.

Figure 4.5: Example of an empty mesh model. This model cannot be used, as the SDF value in the vehicle would be positive, because the mesh is not watertight.

All in all, we end up with a dataset of 706 models, which are represented by tensors of size 64 x 64 x 64 x 4. The SDF values are normalized in order for the distance between two sampling locations to be 1, while the RGB values are normalized between 0 and 1.

4.1.2 Random View Generation

To train the encoder we need a way to simulate some tracker outputs, i.e. calibrated views from cameras with different angles and knowing the 3D bounding box associated to the observed model. Fortunately, Blender (free and open-source 3D computer graphics that provides a python API), is very appropriate for this task, as it allows working with ".obj" files and rendered views automatically with its API.

Blender has an API that we can use to automatically:

- Place a mesh in the world from an ".obj" file.
- Render a view of the mesh from a given point of view.
- Save the intrinsics and extrinsics corresponding to the rendered image.
- Save the position of the object relatively to the world.
- Get and save the 3D coordinate of the bounding box containing the mesh.
- Paste the rendered view on a white background and save the image.
- Change the orientation of the vehicle.
- Repeat the process until we have enough images of a model and start again with a new one.

The images were rendered with a resolution of 300 x 300, it is not necessary to have a higher one as we will feed lower resolution inputs to the encoder. To have different points of view, we changed randomly the angle between each render. Then, we assigned a random angle between 0° and 360° for the yaw angle, and an horizontal angle in between 15° and 75° to obtain a realistic view.

However, we did not change the distance from the camera, as it shouldn't affect the input of the encoder, considering the way we compute the 3D grid using homography. The constant camera distance only limits the resolution, but this is not a problem as long as the vehicle is not too far away from the camera.

Also note that all the rendered images have a white background, which is far from the real conditions. We leave the more general and challenging case of a more generic background to future works.

In the end, we are able to generate as many images as we want, with 3D bounding box, intrinsics, extrinsics and relative position of the object in the world. As for the decoder training, it was computationally more efficient to generate the images before the training. Even though we might use the same images several times, this is not a problem as long as we have enough of them, for the network to perform a good generalization. To ensure this, we used 300 different images for each model during the training. As an example, here is a subset of the images we generated in the figure 4.6.

Figure 4.6: Subset of generated images used for the training of the encoder, the intrinsics, extrinsics and the relative position of the vehicles in the worlds are saved in a separate file.

4.2 Real Data

During my internship at Invision we did some data acquisition hoping that we would have enough time to process it to extract a 3D dataset of real data from the images we acquired. Unfortunately, retrieving the 3D model of a vehicle is a non-trivial task, and we didn't manage to do it in time. Still, in this section we will present the data we acquired, the results of the vehicle tracker applied to it, and some approaches to reconstruct the 3D models of the vehicles from the real data.

4.2.1 Data Acquisition

We acquired 2 sequences of respectively 42 and 24 minutes each, with 8 and 9 different cameras (one camera was not recording in the first sequence). We used "GoPros" to record the footage, because they record the GPS timestamp information, which make it easier to synchronize them afterwards. On Figure 4.7, one can see a sample of the footage which we acquired.

(a) View from the 1st GoPro. *(b) View from the 2nd GoPro.* *(c) View from the 3rd GoPro.*

(d) View from the 4th GoPro. *(e) View from the 5th GoPro.* *(f) View from the 6th GoPro.*

(g) View from the 7th GoPro. *(h) View from the 8th GoPro.* *(i) View from the 9th GoPro.*

Figure 4.7: Here is a sample of the footage which we recorded. We can see the views from 9 different GoPros. These views are synchronized thanks to their GPS. Unfortunately, it was turned off for the last one, that is why the 9th view is not synchronized.

4.2.2 Tracker Evaluation

Using the vehicle tracker of Invision, Figure 4.8 show an example of the tracker output for real data. Note that the result of the tracker is enhanced when we give it multiple calibrated camera views, as it makes the results coherent with all the input views that we give.

Figure 4.8: Example of the Tracker output on the data we acquired. A network output bounding box predictions (in red) for each image. The predictions from this frame and the previous ones are then merged to give the final prediction (in green) which is coherent through all the images. Also, we plot the trajectory based on this new prediction and the previous ones. Note that the tracker also predict people, as we can see a red box around the guy in the bottom left image. There are no green box, as it was not detected on enough frames.

Evaluation

In this chapter, we evaluate the encoder and the decoder presented earlier in chapter 3, on the synthetic dataset. The decoder was trained using 706 models. The encoder was trained using 686 models for the training and 20 models for the validation.

5.1 Decoder

For the decoder, we are interested in the evaluation of two main features: the structure of the latent space, resulting after the training, and the quality of the shape and appearance reconstruction of the training models.

5.1.1 Latent Space Structure

We desired two main outcomes with the latent space structures: The first one being that visually similar vehicles end up with associated codes with a small difference. The second being that we want our latent space to be dense, in the sense that not only the codes of the training models will be associated to a real vehicle, but more generally, we want that a random code in the latent space will be associated to a "realistic" vehicles when given to the decoder.

Visually similar vehicles end up with similar codes. As a reminder, at the start of the training, we initialized each training model with a random code. Then during the training, the optimizer trains the decoder parameters but also the code position in the latent space of each training model. Thus, we assume that it will be easier for the network to place the code of similar vehicles in similar regions in the latent space.

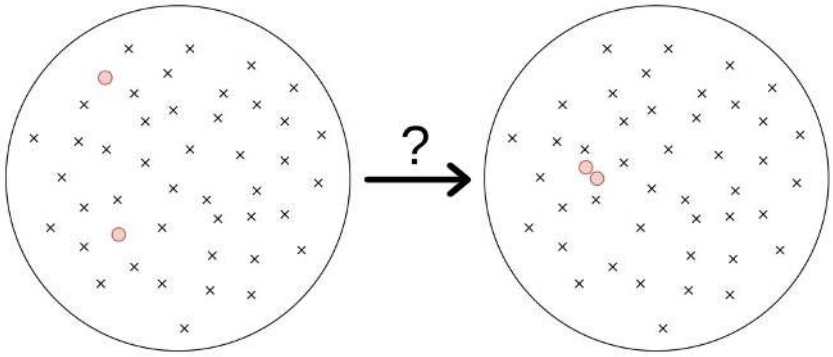


Figure 5.1: We introduced some models twice in the training dataset (represented by two red "o"), which are initialized with a random code. Then, their codes moved during the training, and we are interested in knowing if the two code of the duplicated models converge.

It is not easy to quantify how visually similar two vehicles are. So, we introduced some models twice in the training dataset, and we compared the distances between the code of one model and its duplicate with respect to the distance of the code between two random models. On figure 5.2, we show that identical models would result with almost identical codes at the end of the training, as the distance gets close to 0.

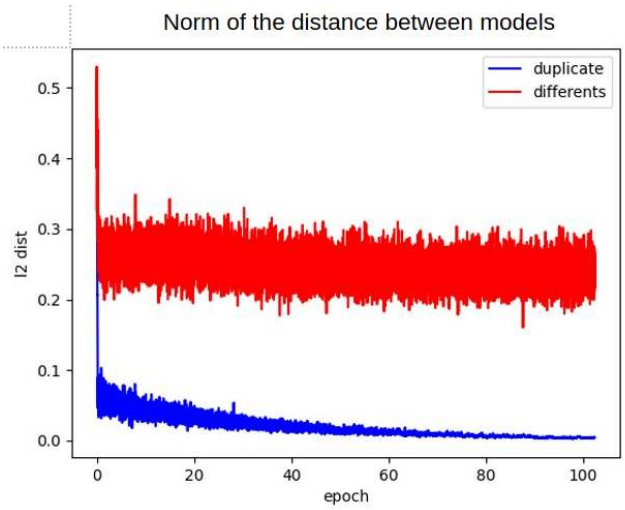
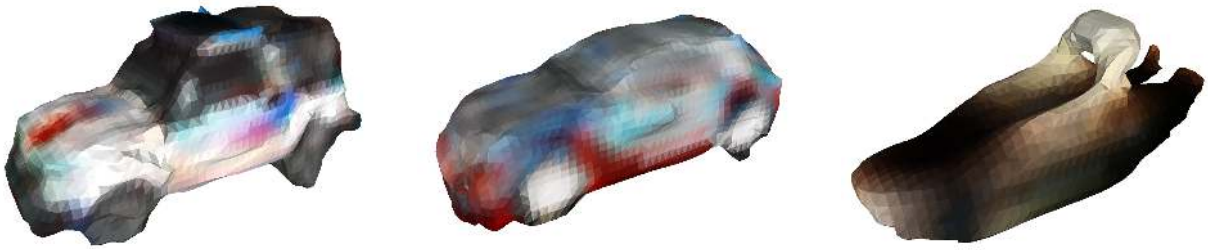


Figure 5.2: Here, we plot the distance between two random codes in red, and the mean distance between one code and its duplicate model in blue. As a result, we see that the codes of identical models converge during training.

Latent Space Density. The reason we introduced the variational decoder is to fill the latent space regions between the codes of the models used for the training. Since it is not easy to design a metric that evaluate how realistic is a rendered vehicle, we will only provide qualitative results for this part (Note that GAN based methods use inception score and inception distance to check the quality of the results[35], however they are not well suited in our case). To evaluate this phenomenon, we use the decoder to render some code taken randomly in the latent space within a certain distance from the origin. On the figure 5.3, one can see the rendering of some random codes in the latent space generated after a standard training for the decoder. In opposition, on figure 5.4, one can see the resulting rendering from the training of the variational decoder.



(a) *(b)* *(c)*

Figure 5.3: Examples of rendering from random codes after the training of a standard decoder.

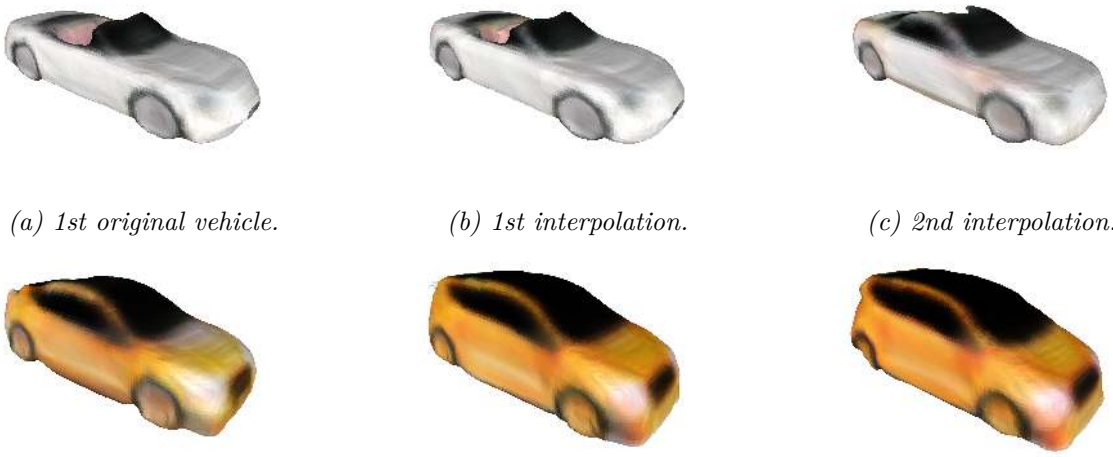
These few examples show how we can obtain very unrealistic vehicles. This is because in opposition to the variational decoder, we do not apply any conditions on the rendering of these codes. One main problem is that the appearance is not uniform at all, and even worse, on the figure 5.3c we see that sometimes even the shape can be really unrealistic.

We see that in this case the results are more realistic, they all have the shape of a vehicle, and even the color is uniform in most cases. In general, the network is able to well separate the wheels and the windows of the vehicles.

(a) *(b)* *(c)*

Figure 5.4: Examples of the rendering from random codes after the training of a variational decoder.

Eventually, in the figure 5.5, we interpolate between the code of two models used for the training. We see that the interpolations are similar to the original models. The transition is smooth until the code becomes closer to the other model, with a very sharp transition. Finally, there are not any unrealistic vehicles in between.



(a) 1st original vehicle. *(b) 1st interpolation.* *(c) 2nd interpolation.*

(d) 3rd interpolation. *(e) 4th interpolation.* *(f) 2nd original vehicle.*

Figure 5.5: The first and last images are rendered using the codes from models in the training datasets. The others images are rendered using a code obtained from a linear interpolation between the first and last vehicles.

Overall, the variational decoder greatly improves the structure of our latent space, as we intended. Indeed, the space is dense with realistic vehicles.

5.1.2 3D Model Reconstruction

Metric used. First, we introduce a metric to quantify the quality of the decoder reconstruction. As meshes and points clouds representation are more standard than signed distance fields, we chose a metric which applies to a mesh. Therefore, each time we evaluate the accuracy of a decoder prediction we will recover a mesh from the predicted SDF. The main drawback is that we do not fully take advantage of the fact that the SDF is continuous because we will need to sample it to compute a mesh.

To estimate the differences between two sets of points, there are already a good number of existing metrics such as the Earth Mover's Distance, the Chamfer Distance or the Hausdorff distance. We chose to use the Chamfer Distance[36] to quantify the quality of the shape reconstruction, mainly because it was already implemented in the "PyTorch 3D" library, which is a library for deep learning with 3D data.

The Chamfer Distance is a metric that evaluate the similarity between two sets of points. It is defined as the mean of the closest distance from a point in one set to the closest point in the other set.

Let be two meshes $\mathcal{M}$ and $\mathcal{N}$ composed of M and N vertices respectively, then the chamfer distance used to measure the shape accuracy is:

$$CF_{shape}(\mathcal{M},\mathcal{N}) = \frac{1}{M}\sum^{m\in\mathcal{M}}\{dist(m,n)|\ n = argmin_{n\in\mathcal{N}}(dist(m,n)\} \tag{5.1}$$

The total shape error is therefore:

$$CF_{shape} = \frac{M\cdot CF_{shape}(\mathcal{M},\mathcal{N}) + N\cdot CF_{shape}(\mathcal{N},\mathcal{M})}{M+N} \tag{5.2}$$

To quantify the appearance error, we use a similar process. The only difference is that instead of measuring the mean of the distance to the closest points, we will compute the mean of the RGB differences with the closest vertex.

Again, let be two meshes $\mathcal{M}$ and $\mathcal{N}$ composed of M and N vertices respectively, and m_r, m_g and m_b the RGB values of a vertex m, then the chamfer distance used to measure the appearance accuracy is:

$$CF_{appearance}(\mathcal{M},\mathcal{N}) = \frac{1}{M}\sum^{m\in\mathcal{M}}\{\frac{|m_r-n_r|+|m_g-n_g|+|m_b-n_b|}{3}|\ n = argmin_{n\in\mathcal{N}}(dist(m,n)\} \tag{5.3}$$

The total appearance error is therefore:

$$CF_{appearance} = \frac{M\cdot CF_{appearance}(\mathcal{M},\mathcal{N}) + N\cdot CF_{appearance}(\mathcal{N},\mathcal{M})}{M+N} \tag{5.4}$$

Shape Reconstruction. We want to assess the reconstruction quality of the models used in the training dataset. We will do this qualitatively and quantitatively with the chamfer distance, which we presented earlier. As a reminder, the data used as targets for the network are samplings of the mesh downloaded from the Shapenet dataset. Therefore, the target data already contain some inaccuracies due to the loss of resolution.

Here are some qualitative results, where we compare the decoder reconstruction with the reconstruction from the training data.

(a) Target reconstruction. (b) Decoder reconstruction. (c) Target reconstruction. (d) Decoder reconstruction.

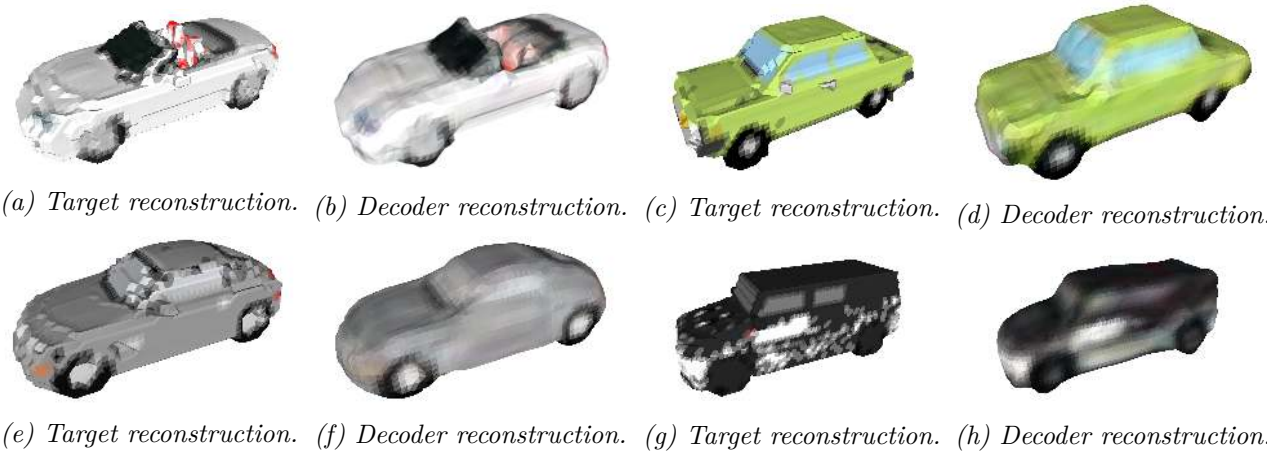
(e) Target reconstruction. (f) Decoder reconstruction. (g) Target reconstruction. (h) Decoder reconstruction.

Figure 5.6: Comparison between shape reconstruction from training data and decoder's predictions.

The figure 5.6 shows that overall the decoder is able to understand the general shape of the vehicles, even when they have no roof for example. However, the network struggles to capture high frequency details. This can be observed as the decoder reconstruction is much smoother, which is good, but it is missing small details such as the rearview mirrors, or the car spoilers.

In order to have a quantitative evaluation, we compute the Chamfer distance for all the training models. As a reminder, the distance between two sampling locations is 1. Overall, we achieve an average Chamfer distance of 0.8. In other words, the vertices of the meshes computed using the marching cube algorithm are placed with an average error of less than 1 "interpolation cube".

Appearance Reconstruction. Again, we want to evaluate the predictions of the decoder, both qualitatively and quantitatively, but this time we are doing it to assess the quality of the appearance. In the figure 5.7, we present qualitative results.



(a) Target reconstruction. *(b) Decoder reconstruction.* *(c) Target reconstruction.* *(d) Decoder reconstruction.*

(e) Target reconstruction. *(f) Decoder reconstruction.* *(g) Target reconstruction.* *(h) Decoder reconstruction.*

Figure 5.7: Comparison between the shape and the reconstruction appearance from the training data and the decoder predictions.

The decoder is able to recover the general color of the vehicles in most cases. It can also recover the medium details, such as the separation between the windows, and the separation between the wheels and the tires. Again, it is mostly struggling with the high frequency details such as the indicators and the headlights. The reconstruction in the last subfigure 5.7h is not acceptable, but this is mainly due to the fact that the original vehicle has a non-uniform appearance, and therefore the sampling done in the data processing part gives a bad approximation of it. This highlights more a data processing problem rather than an issue with the decoder, even if it still shows the decoder difficulty to recover high frequency details.

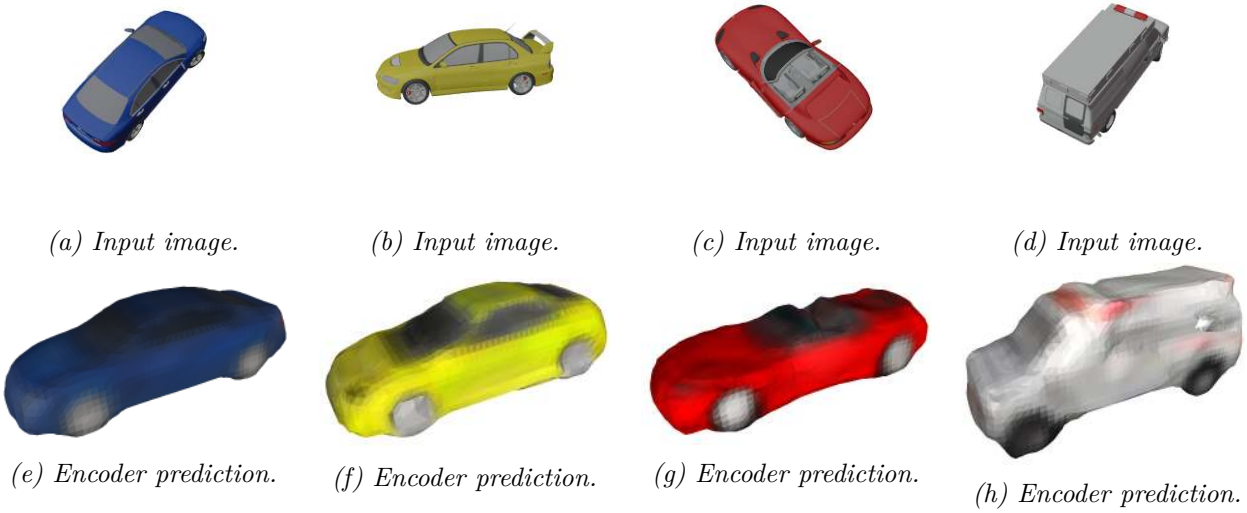
Eventually, we evaluated the appearance error using the metric presented in section 5.1.2. On average, each RGB value of each vertex predicted by the decoder is predicted with an error of 33 (with each pixel values belonging in the range 0-255).

5.2 Encoder

To evaluate the encoder predictions, we qualitatively compare the decoder reconstruction using the targeted code obtained. This is done while training the decoder and the decoder reconstruction using the code predicted by the encoder.

In the figure 5.8, we present some good predictions made by the encoder. The input images come from a validation dataset, thus the encoder has never seen these vehicles.

Despite the fact that the Encoder has never seen any of these vehicles, it manages to predict a latent code which corresponds to a vehicle that looks similar.

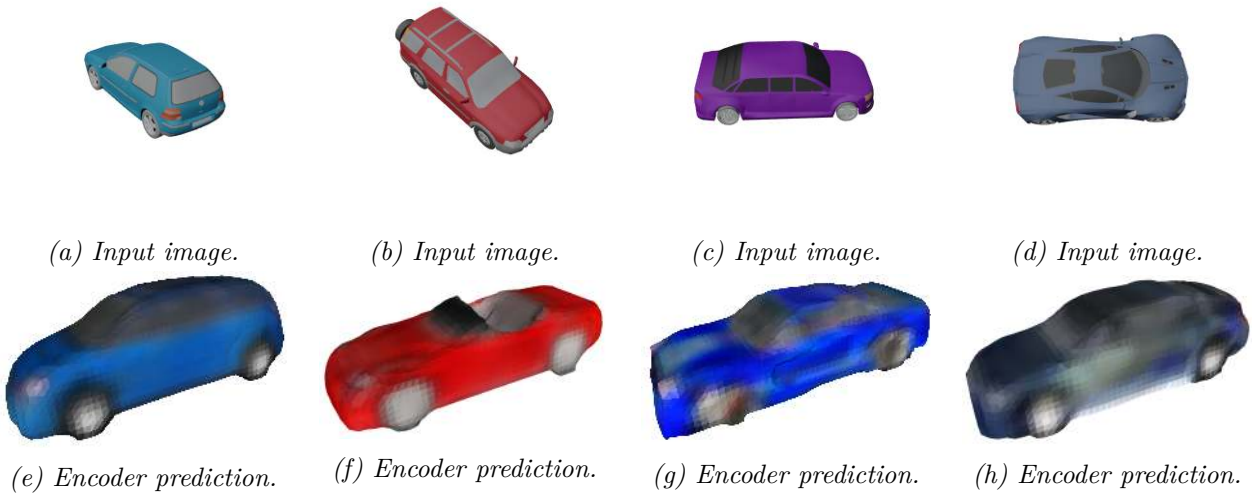


(a) Input image. *(b) Input image.* *(c) Input image.* *(d) Input image.*

(e) Encoder prediction. *(f) Encoder prediction.* *(g) Encoder prediction.* *(h) Encoder prediction.*

Figure 5.8: Example of accurate predictions by the encoder. The top row shows the images that were used with their associated 3D bounding box to generate the input of the encoder. The bottom row shows the rendering of the decoder using the latent code predicted by the encoder.

Still, it makes some mistakes, here are a sample of them.



(a) Input image. *(b) Input image.* *(c) Input image.* *(d) Input image.*

(e) Encoder prediction. *(f) Encoder prediction.* *(g) Encoder prediction.* *(h) Encoder prediction.*

Figure 5.9: Example of inaccurate predictions by the encoder.

Several mistakes can be observed in the Figure 5.9. Sometimes the predicted vehicle has a different shape (5.9f), sometimes a different color (5.9e). In general, as we only have 686 models in the training set, most of the errors come from the small size of the dataset. For example, it is probable that none of the vehicles in the training dataset is similar to the 3rd (5.9g) and 4th (5.9h) vehicle.

Eventually, the reconstruction from the encoder prediction achieves an average Chamfer distance of 2.2 with the original vehicle, and 42 using the adapted metric to measure the color error.

Exploration and Future work

In this thesis, we propose an encoder which predicts a code in a latent space, and a decoder which associates this code to an implicit representation of the vehicle. However, this prediction is not always accurate and could be refined in some cases to obtain a better prediction. Also, the training of these networks relies on 3D ground truth datasets, which are hard to obtain with real data.

In this chapter, we will first investigate how we can render a prediction using ray marching and compare it to an input image, in order to refine it. Then, we will explore how we could train the decoder while only requiring the 3D ground truth for the shape, knowing that we could learn the appearance only from images. Eventually, we show some exploratory work on building a 3D dataset solely from images, with some examples from the real footage that we recorded.

6.1 Rendering from SDF

Until now, all the rendering shown have been done through the computation of a mesh, that we visualized using "MeshLab". While this is handy to visualize results, this leaves us we two major issues.

Sampling. As the vehicles are represented with a continuous signed distance field, it is suboptimal to compute a mesh which is obtained by sampling the SDF at different locations. The visualized mesh is therefore an approximation and is not as smooth as the implicit representation of the SDF.

Differentiability. Ideally, after getting a prediction from the encoder when we try to predict the shape and appearance of a vehicle, we would like to refine the predicted code by comparing the input image with the reconstruction from the decoder. However, we need the rendering reconstruction process from the decoder to be differentiable, and it is not the case because the marching cube algorithm is not differentiable.

To tackle these two issues, we implement a ray marching renderer[37]. Here is its operation: Assuming that we want to render an image from a given point of view, for each pixel we compute a ray going from the center of the virtual camera to the position of the pixel we want to render, in the image plane. Then we march along this ray, the size of each step is given by the value of the signed distance field. Eventually, if the signed distance field is accurate, and if the ray is intersecting the object, we will make smaller and smaller steps and the signed distance field measured between each step will converge to zero. If the ray is not intersecting the shape, it will diverge to infinity. Therefore, by doing this for every pixel on the image, we will know for which pixel we should render the vehicle, and for which one we would have to render the background. The appearance of the pixels which render the vehicle is given by evaluating the decoder appearance on the last position of the ray marching.

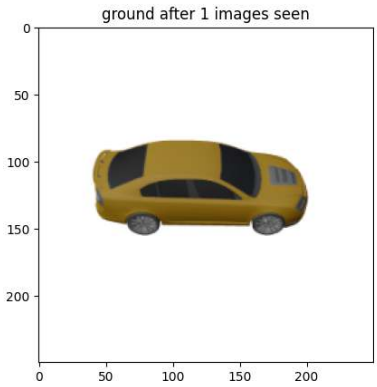
Since the vehicle is represented by a continuous signed distance field, the resolution of the image is only limited by the computation resources, leading to a much smoother rendering of the car. Also, note that our decoder tends to overestimate the SDF when it is far away. To prevent any error related to this inaccuracy, we bound the marching step with a maximum value. Figure 6.1b and 6.1d depict examples of such rendering

6.1.1 Training Decoder appearance through ray marching

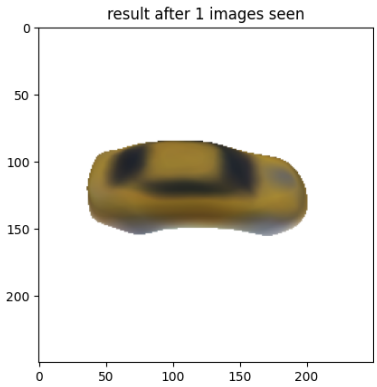
The process of ray marching is not easily differentiable in practice due to the relatively high number of marching steps. Nevertheless, it is easy to differentiate the appearance of the rendered pixel with an image, for example the image used to give the input to the encoder.

Therefore, we split our initial decoder in two separate ones: the first decoder predicts the shape and is still trained using 3D ground truth data. However, once this decoder is trained, we train a second decoder which predicts the appearance using ground truth images only. This is done with the computation of a loss between an image and the rendering of the vehicle from the same point of view, with the pre-trained decoder for the shape.

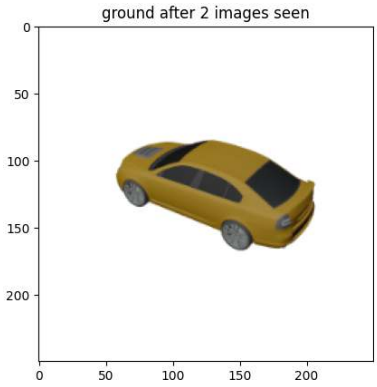
After training this network for only one vehicle, in the figure 6.1, we present renderings using ray marching, next to the ground truth image which they should match.



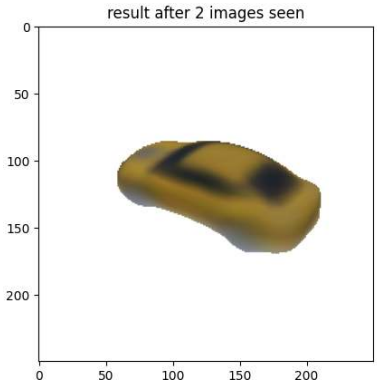
(a) Target Image.



(b) Rendered Image.



(c) Target Image.



(d) Target Image.

Figure 6.1: Comparison between target image and rendered image after training the decoder for the appearance.

In the figure 6.2, we show some other points of view using the old method, i.e. computing a mesh that we visualize after in "MeshLab".

While we are still able to recover the general appearance of the car, we struggle to capture the details convincingly. The transitions around the windows and the wheels are less sharp, and the transitions between the front and side windows are hardly noticeable. Also, the decoder was trained to only learn the appearance of this vehicle. These results are disappointing considering the fact that we train it with only one vehicle, thus it might indicate a bug that we were not able to catch. Still, this method in the given state leads to less accurate results than with the training on the 3D ground truth data and also for the appearance.

(a) *(b)*

Figure 6.2: Render from MeshLab after using the marching cube algorithm on the results shown in Figure 6.1

Finally, computing the loss between a rendering from the prediction and an image can help to refine a prediction, or even initialize a random code and refine it, which would allow us to find the code representing a vehicle without the need of the encoder.

6.1.2 Training Decoder shape through ray marching

In this section, we introduce an approximation to differentiate the intersection of a line of sight with the SDF, with respect to latent code and network parameters of the signed distance function given by the decoder. For this purpose, let us introduce some definition:

Let be:

- A pixel with coordinate $\mathbf{u}$ in an image we want to render.
- A neural network $f_\theta(\mathbf{x}, \mathbf{z})$ with parameters θ which plays the role of a function that returns the signed distance value, given a position in space $\mathbf{x}$ and a latent code $\mathbf{z}$ corresponding to the object.
- A neural network $g_\phi(\mathbf{x}, \mathbf{z})$ with parameters ϕ which plays the role of a function that returns the RGB values, given a position in space $\mathbf{x}$ and a latent code $\mathbf{z}$ corresponding to the object.
- A unit vector $\vec{d}(\mathbf{u})$ along the line of sight of a pixel $\mathbf{u}$
- The center of the camera projection $\mathbf{c}$
- A ray marching function $\rho(f_\theta, \mathbf{u}, \mathbf{z})$ which return the 3D position $\mathbf{x}$ which is the final position obtained through ray marching for a given pixel $\mathbf{u}$ to render, and with the marching steps computed with the neural network f, using parameters θ and for an object represented with the latent code $\mathbf{z}$

Then we can express:

$$\rho(f_\theta, \mathbf{u}, \mathbf{z}) = \mathbf{c} + \vec{d}(\mathbf{u}) \cdot argmin_\lambda[f_\theta(\mathbf{c} + \lambda \cdot \vec{d}(\mathbf{u}), \mathbf{z}) = 0] \tag{6.1}$$

We would like to differentiate how the final position obtained by the ρ function changes when the parameters θ of the signed distance function f change, i.e. we want $\frac{\delta\rho(f_\theta, \mathbf{u}, \mathbf{z})}{\delta\theta}$.

However, this is not directly differentiable mathematically, as we cannot solve for lambda easily, therefore we introduce a first order Taylor approximation leading to:

$$\rho(f_\theta, \mathbf{u}, \mathbf{z}) \tilde{=} \mathbf{c} + \vec{d}(\mathbf{u}) \cdot argmin_\lambda [f_\theta(\mathbf{c} + \lambda_0 \cdot \vec{d}(\mathbf{u}), \mathbf{z}) + (\lambda - \lambda_0) \cdot [\frac{\delta f_\theta(\mathbf{c} + \lambda \cdot \vec{d}(\mathbf{u}), \mathbf{z})}{\delta \lambda}]_{\lambda_0} = 0] \tag{6.2}$$

With λ_0 the marching distance obtained when computing the ray marching.

Thus, we can now easily solve for λ:

$$\lambda = \lambda_0 \cdot \frac{[\frac{\delta f_\theta(\mathbf{c}+\lambda \cdot \vec{d}(\mathbf{u}), \mathbf{z})}{\delta \lambda}]_{\lambda_0} - f_\theta(\mathbf{c} + \lambda_0 \cdot \vec{d}(\mathbf{u}), \mathbf{z})}{[\frac{\delta f_\theta(\mathbf{c}+\lambda \cdot \vec{d}(\mathbf{u}), \mathbf{z})}{\delta \lambda}]_{\lambda_0}} \tag{6.3}$$

Leading to:

$$\rho(f_\theta, \mathbf{u}, \mathbf{z}) = \mathbf{c} + \vec{d}(\mathbf{u}) \cdot \lambda_0 \cdot \frac{[\frac{\delta f_\theta(\mathbf{c}+\lambda \cdot \vec{d}(\mathbf{u}), \mathbf{z})}{\delta \lambda}]_{\lambda_0} - f_\theta(\mathbf{c} + \lambda_0 \cdot \vec{d}(\mathbf{u}), \mathbf{z})}{[\frac{\delta f_\theta(\mathbf{c}+\lambda \cdot \vec{d}(\mathbf{u}), \mathbf{z})}{\delta \lambda}]_{\lambda_0}} \tag{6.4}$$

Which is now differentiable by θ.

This explains that intuitively:

The final position $\rho(f_\theta, \mathbf{u}, \mathbf{z})$ corresponds to the intersection between the surface of the object and the line define by the vector $\vec{d}$ passing by the center of the camera projection.

Also, the surface of the object changes in the direction of the gradient of the neural network f at position $\mathbf{x}$: $\frac{\delta f_\theta(\rho(f_\theta, \mathbf{u}, \mathbf{z}), \mathbf{z})}{\delta \mathbf{x}}$.

Therefore, we can project this gradient vector on the vector $\vec{d}$ and ponder the result by the differentiation $\frac{\delta f_\theta(\rho(f_\theta, \mathbf{u}, \mathbf{z}), \mathbf{z})}{\delta \theta}$ that we can approximate to $\frac{\delta f_\theta(\mathbf{c}+\lambda_0 \cdot \vec{d}(\mathbf{u}), \mathbf{z})}{\delta \theta}$, which expresses how much the signed distance value of given by the neural network f changes at a position $\mathbf{x}$ when the parameters θ change.

This leads to:

$$\frac{\delta \rho(f_\theta, \mathbf{u}, \mathbf{z})}{\delta \theta} \tilde{=} (\frac{\delta f_\theta(\rho(f_\theta, \mathbf{u}, \mathbf{z}), \mathbf{z})}{\delta \mathbf{x}} \cdot \vec{d}) \cdot \vec{d} \cdot \frac{\delta f_\theta(\mathbf{c} + \lambda_0 \cdot \vec{d}(\mathbf{u}), \mathbf{z})}{\delta \theta} \tag{6.5}$$

Allowing us to compute this differentiation $\frac{\delta \rho(f_\theta, \mathbf{u}, \mathbf{z})}{\delta \theta}$ can be useful for several applications. For example, assuming that we have a ground truth and a render image, for each pixel we can compute a loss to train the shape of a decoder produced by a neural network f. As a reminder, we assume that the color of the pixels are rendered with a network g with parameters ϕ.

This leads to:

$$\frac{\delta g_\phi(\rho(f_\theta, \mathbf{u}, \mathbf{z}), \mathbf{z})}{\delta \theta} = \frac{\delta g_\phi(\rho(f_\theta, \mathbf{u}, \mathbf{z}), \mathbf{z})}{\delta \rho(f_\theta, \mathbf{u}, \mathbf{z})} \cdot \frac{\delta \rho(f_\theta, \mathbf{u}, \mathbf{z})}{\delta \theta} \tag{6.6}$$

All in all, we get:

$$\frac{\delta g_\phi(\rho(f_\theta, \mathbf{u}, \mathbf{z}), \mathbf{z})}{\delta \theta} \tilde{=} \frac{\delta g_\phi(\rho(f_\theta, \mathbf{u}, \mathbf{z}), \mathbf{z})}{\delta \rho(f_\theta, \mathbf{u}, \mathbf{z})} \cdot (\frac{\delta f_\theta(\rho(f_\theta, \mathbf{u}, \mathbf{z}), \mathbf{z})}{\delta \mathbf{x}} \cdot \vec{d}) \cdot \vec{d} \cdot \frac{\delta f_\theta(\mathbf{c} + \lambda_0 \cdot \vec{d}(\mathbf{u}), \mathbf{z})}{\delta \theta} \tag{6.7}$$

6.2 3D ground truth reconstruction

In this section, we will focus on how we can reconstruct a 3D ground truth dataset using images only. Extracting a 3D model only from images is a difficult task and was tackled by many approaches. Some of them use more standard computer vision approaches. As an example, visual hull is a standard solution but it requires silhouettes, which are hard to obtain with the needed precision. Segmenting the image by computing a depth map from stereo paired images is also one of the most standard ways to go. More recently, prior work[38] used a deep learning based approach.

All things considered, there are a lot of ways to approach this problem, but none of them are trivial while still providing convincing results. Thus, we didn't have time to provide a good solution to this problem. If one is interested in the state of the art of this task, one can look at the Middlebury evaluation site (https://vision.middlebury.edu/) which is a repository for computer vision evaluations and datasets on which the community can propose and evaluate different methods.

6.2.1 Extracting 3D model from silhouettes

While we hoped that extracting the silhouettes of real data would be manageable, here are our tries to recover the 3D model of a synthetic vehicle from raw images using the associated silhouette.

SDF estimation. We do not want to solely recover the surface, but we also want to compute the SDF at every sampling locations in space. We proceed similarly to the computation of a visual hull. We will look at the 2D silhouette on the image and compute a 2D SDF image from it, which we will project in the 3D space. Note that the distance on the 2D SDF image should be pondered with the distance from the camera, when projected in the 3D space. This results in a 3D SDF which is underestimated everywhere. The next step is to compute as many underestimations of the 3D SDF as we possibly can by using different images from other points of view. Eventually, for every sampling locations, we keep the maximum distance of all the underestimations. On figure 6.3, one can observe the reconstruction obtained using multiples views.

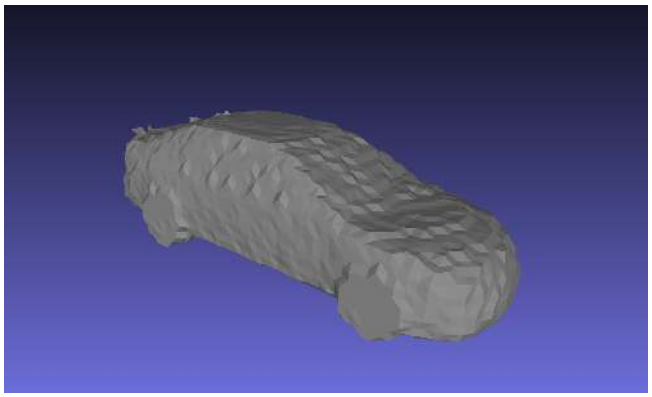
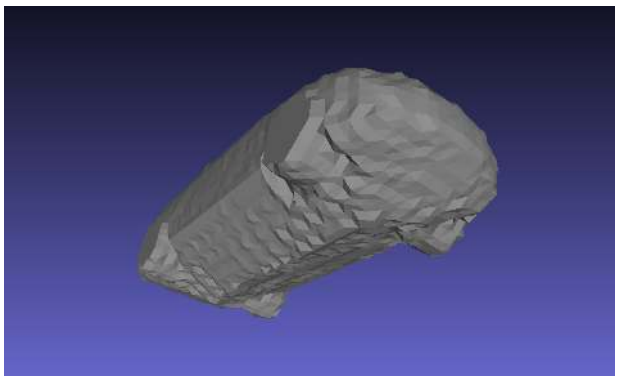


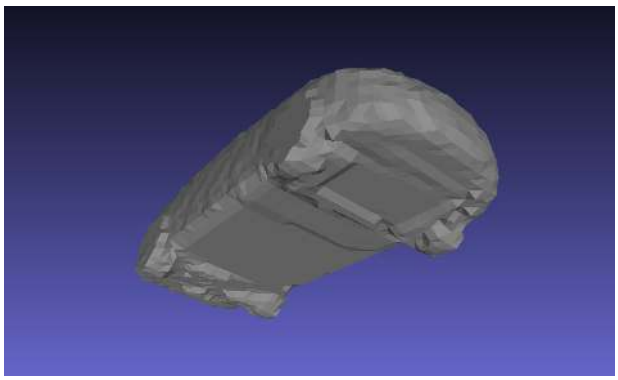
Figure 6.3: Shape reconstruction of a synthetic model using 20 images with the associated silhouette.

This process results in a good estimation which converges to the perfect one, if we have an infinite number of views, and if the shape has no concavities. However, it is in practice not feasible to have views from everywhere, especially from the bottom. Next, On the figure 6.4, we show what happens to the shape estimation, if we do not have views from under the vehicle.

This process worked well on synthetics data as we have the exact silhouette. However, we suppose that it is not well suited for real data, as it is very sensitive to errors in the silhouette estimation and in the 3D bounding box prediction.



(a) Shape reconstruction viewed from the bottom. The images were taken with from a view angle in between 15° and 75° above the horizontal.



(b) Shape reconstruction viewed from the bottom. The images were taken with from a view angle in between -15° and 75° above the horizontal.

Figure 6.4: Shape reconstruction from images. One can see that if we do not have enough views, we will overestimate the shape, because we cannot predict the shape of the occluded areas.

Appearance estimation.

Now that we have the shape, we also need to extract the appearance. The way we estimate it is by using the fact that when we extract the SDF at every location, we keep the one where the SDF is maximum, assuming that the shape is convex. The kept value is estimated using a view for which the vector going from the sampling location to the camera center is normal to the vector going from the sampling location to the closet point of the surface of the vehicle. Moreover, the smallest value is most of the time computed using the view for which the angle between these two last vectors is the closest to 0 (This is not true every time, but it is a good approximation in practice). On the next Figure 6.5, one can observe the result using this approximation.

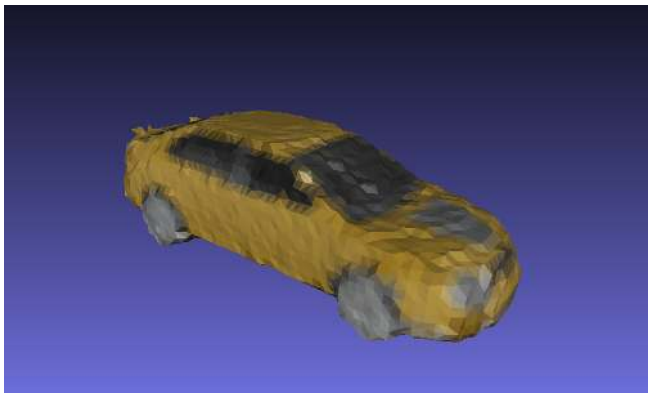


Figure 6.5: Shape and appearance reconstruction of synthetic model using 20 images with the associated silhouette.

6.2.2 Estimating silhouettes from real images.

As we show in the section 6.2.1, one way to recover the 3D model using a camera view uses the silhouette. We will now show an attempt to recover the silhouette using background subtraction. Here is a sample of the footage used for the testing of the background subtraction method.

We used the background subtractor Mixture of Gaussian function from OpenCV, resulting in a raw mask with some noise. We then applied an opening to remove the background noise, and then a closing to fill in the silhouette of the vehicle. In the Figure 6.7 we show the masks at the different steps.

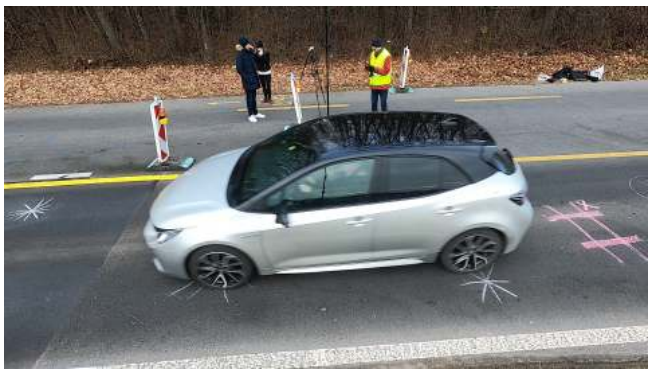


Figure 6.6: Frame sample.

(a) Raw mask, computed using the Mixture of Gaussian foreground segmentation function.

(b) Mask after performing an opening and the raw mask.

(c) Final mask obtained after the opening and then the closing.

Figure 6.7: Here are the masks and the steps to extract the silhouette of the vehicle, using a background subtraction method. We applied an opening to remove the background noise, and then a closing to close the hole in the silhouette.

We are able to get a rough estimation of the silhouette of the vehicle. Still, there are some inaccuracies, the general contour is not smooth, and the result can be erroneous at times. Also, it might be hard to differentiate the vehicle from other objects in movement in the image. As an example, here (Figure 6.7) we detect the person moving in the background, but more generally there can be other vehicles in the field of view. This could be tackled in most cases by using the bounding box predicted by the tracker, but for cases where there are occlusions, it would be very hard to differentiate the different vehicles. On the figure 6.8 one can see the superposition between the raw image, the raw mask and the silhouette prediction.

Figure 6.8: Superposition of the raw mask (red) and the final estimation of the silhouette (blue) on the raw image.

While giving a good rough estimation of the silhouette, this method is very noisy and unreliable. Thus, we will need to find a better solution. A future plausible solution would be to predict the silhouette via a neural network such as a Mask R-CNN[6].

Conclusion

In this thesis we introduced a neural network which interprets images processed by a tracker to model the shape and appearance of vehicles. This was done in the context of a real world application at Invision AI.

We implemented a variational decoder that allows us to construct a latent space which embeds the representation of the set of objects that we are likely to encounter in a detection task. We showed that we are capable of rendering them with a theoretically infinite resolution, thanks to our decoder which predicts a continuous signed distance and appearance functions.

We presented a unique way to generate an input from an image with the 3D coordinates of an object, which is easier to interpret for an autoencoder, which aims at modeling it. We experimented a few approaches to achieve it, and we are satisfied with our final methodology.

Synthetic data were processed to test our method. We had to sort and filter manually the models of interest from the shapenet dataset. We developed a pipeline to convert and adapt them to a specific type of file, before extracting tensors containing the signed distance values related to an object.

We evaluated our method and achieved satisfying results considering the relatively small size of the data. In most cases, the encoder is capable of predicting a code which produces a representation of the vehicle, which is similar to the one we are trying to model. Still, the network struggles to capture high frequency details. This could also probably be improved by processing the data with a higher resolution in future work.

We acquired real data in order to perform tests for a real world application. Unfortunately we run out of time to build a 3D ground truth dataset ourselves. Still, we show that the acquisition of images from calibrated cameras is manageable, as we did it with 9 views. We also demonstrate that it is easily scalable. Then, we presented some results and reflections on them, which support the hypothesis that it is feasible to reconstruct 3D models from real data, in the case where we would possess a high number of views, from which we could extract an estimation of the 3D bounding box of the vehicle with our tracker.

Appendix

All the code used for this project can be found on: https://github.com/lcordey/codePDM and https://github.com/lcordey/mesh-voxelization

Bibliography

[1] A. Laurentini, "The visual hull concept for silhouette-based image understanding," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 2, pp. 150–162, 1994.

[2] T. Kanade, A. Yoshida, K. Oda, H. Kano, and M. Tanaka, "A stereo machine for video-rate dense depth mapping and its new applications," in *Proceedings CVPR IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 196–202, 1996.

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016.

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," *Lecture Notes in Computer Science*, p. 21–37, 2016.

[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2014.

[6] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," 2018.

[7] C. Lin, C. Wang, and S. Lucey, "SDF-SRN: learning signed distance 3d object reconstruction from static images," *CoRR*, vol. abs/2010.10505, 2020.

[8] W. Chen, J. Gao, H. Ling, E. J. Smith, J. Lehtinen, A. Jacobson, and S. Fidler, "Learning to predict 3d objects with an interpolation-based differentiable renderer," *CoRR*, vol. abs/1908.01210, 2019.

[9] I. Guskov, K. Vidimče, W. Sweldens, and P. Schröder, "Normal meshes," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 95–102, 2000.

[10] L. Linsen, *Point cloud representation*. Univ., Fak. für Informatik, Bibliothek Technical Report, Faculty of Computer . . . , 2001.

[11] S. Osher and R. Fedkiw, "Signed distance functions," in *Level set methods and dynamic implicit surfaces*, pp. 17–22, Springer, 2003.

[12] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2014.

[13] H. Kato, Y. Ushiku, and T. Harada, "Neural 3d mesh renderer," *CoRR*, vol. abs/1711.07566, 2017.

[14] B. Nicolet, A. Jacobson, and W. Jakob, "Large steps in inverse rendering of geometry," *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, vol. 40, Dec. 2021.

[15] J. Hasselgren, J. Munkberg, J. Lehtinen, M. Aittala, and S. Laine, "Appearance-driven automatic 3d model simplification," in *Eurographics Symposium on Rendering*, 2021.

[16] Y. Jiang, D. Ji, Z. Han, and M. Zwicker, "Sdfdiff: Differentiable rendering of signed distance fields for 3d shape optimization," *CoRR*, vol. abs/1912.07109, 2019.

[17] E. Remelli, A. Lukoianov, S. R. Richter, B. Guillard, T. Bagautdinov, P. Baque, and P. Fua, "Meshsdf: Differentiable iso-surface extraction," 2020.

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

[19] Y. Duan, H. Zhu, H. Wang, L. Yi, R. Nevatia, and L. J. Guibas, "Curriculum deepsdf," 2020.

[20] R. Chabra, J. E. Lenssen, E. Ilg, T. Schmidt, J. Straub, S. Lovegrove, and R. Newcombe, "Deep local shapes: Learning local sdf priors for detailed 3d reconstruction," 2020.

[21] V. Sitzmann, M. Zollhöfer, and G. Wetzstein, "Scene representation networks: Continuous 3d-structure-aware neural scene representations," *CoRR*, vol. abs/1906.01618, 2019.

[22] H. Rhodin, V. Constantin, I. Katircioglu, M. Salzmann, and P. Fua, "Neural scene decomposition for multi-person motion capture," *CoRR*, vol. abs/1903.05684, 2019.

[23] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[24] R. Wang, N. Yang, J. Stückler, and D. Cremers, "Directshape: Photometric alignment of shape priors for visual vehicle pose and shape estimation," *CoRR*, vol. abs/1904.10097, 2019.

[25] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," *CoRR*, vol. abs/2003.05991, 2020.

[26] S. Saito, Z. Huang, R. Natsume, S. Morishima, A. Kanazawa, and H. Li, "Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization," 2019.

[27] S. Saito, T. Simon, J. Saragih, and H. Joo, "Pifuhd: Multi-level pixel-aligned implicit function for high-resolution 3d human digitization," 2020.

[28] Q. Xu, W. Wang, D. Ceylan, R. Mech, and U. Neumann, "Disn: Deep implicit surface network for high-quality single-view 3d reconstruction," 2021.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[30] J. R. Hershey and P. A. Olsen, "Approximating the kullback leibler divergence between gaussian mixture models," in *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, vol. 4, pp. IV–317, IEEE, 2007.

[31] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *ACM siggraph computer graphics*, vol. 21, no. 4, pp. 163–169, 1987.

[32] 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," 2015.

[33] D. Stutz and A. Geiger, "Learning 3d shape completion from laser scan data with weak supervision," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE Computer Society, 2018.

[34] D. Stutz, "Learning shape completion from bounding boxes with cad shape priors." http://davidstutz.de/, September 2017.

[35] K. Shmelkov, C. Schmid, and K. Alahari, "How good is my gan?," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 213–229, 2018.

[36] H. G. Barrow, J. M. Tenenbaum, R. C. Bolles, and H. C. Wolf, "Parametric correspondence and chamfer matching: Two new techniques for image matching," tech. rep., SRI INTERNATIONAL MENLO PARK CA ARTIFICIAL INTELLIGENCE CENTER, 1977.

[37] L. J. Tomczak, "Gpu ray marching of distance fields," *Technical University of Denmark*, vol. 8, 2012.

[38] A. Düzçeker, S. Galliani, C. Vogel, P. Speciale, M. Dusmanu, and M. Pollefeys, "Deepvideomvs: Multi-view stereo on video with recurrent spatio-temporal fusion," 2021.