Autolabeling 3D Objects with Differentiable Rendering of SDF Shape Priors

Sergey Zakharov*
Technical University of Munich
sergey.zakharov@tum.de

Wadim Kehl*, Arjun Bhargava, Adrien Gaidon
Toyota Research Institute
firstname.lastname@tri.global

Abstract

We present an automatic annotation pipeline to recover 9D cuboids and 3D shapes from pre-trained off-the-shelf 2D detectors and sparse LIDAR data. Our autolabeling method solves an ill-posed inverse problem by considering learned shape priors and optimizing geometric and physical parameters. To address this challenging problem, we apply a novel differentiable shape renderer to signed distance fields (SDF), leveraged together with normalized object coordinate spaces (NOCS). Initially trained on synthetic data to predict shape and coordinates, our method uses these predictions for projective and geometric alignment over real samples. Moreover, we also propose a curriculum learning strategy, iteratively retraining on samples of increasing difficulty in subsequent self-improving annotation rounds. Our experiments on the KITTI3D dataset show that we can recover a substantial amount of accurate cuboids, and that these autolabels can be used to train 3D vehicle detectors with state-of-the-art results. The code is available at github.com/TRI-ML/sdflabel.

1. Introduction

Deep learning methods require large labeled datasets to achieve state-of-the-art performance. Concerning object detection for automated driving, 3D cuboids are preferred among other annotation types as they allow appropriately reasoning over all nine degrees of freedom (instance location, orientation, and metric extent). However, obtaining a sufficient amount of labels to train 3D object detectors is laborious and costly, as it mostly relies on involving a large number of human annotators. Existing approaches for scaling up annotation pipelines include the usage of better tooling, active learning, or a combination thereof [22, 16, 39, 25, 4]. Such approaches often rely on heuristics and require human effort to correct the outcomes of semi-automatic labeling, specifically for difficult edge cases.
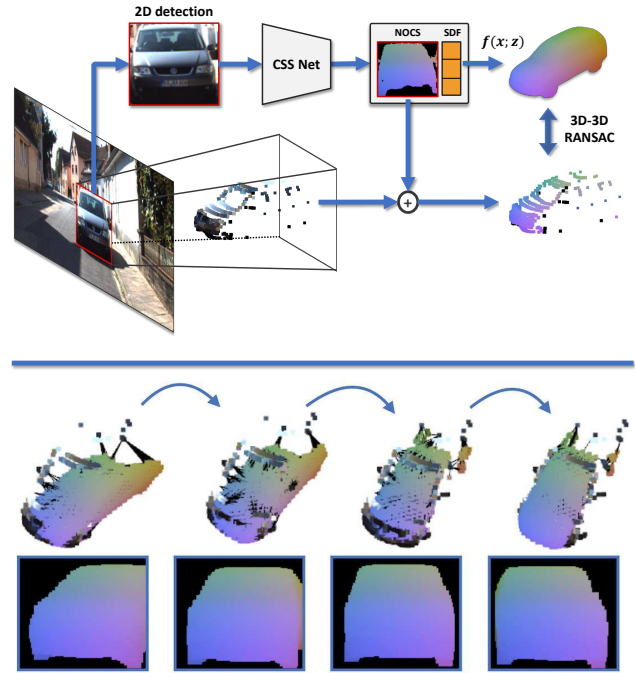

Figure 1: Our pipeline for 3D object autolabeling. Top: off-the-shelf 2D detections are fed into our Coordinate Shape Space (CSS) network to predict surface coordinates and a shape vector. We backproject the coordinates to LIDAR in the camera frustum and decode the shape vector into an object model. Then, we establish 3D-3D correspondences between the scene and model to estimate an initial affine transformation. Bottom: We iteratively refine the estimate via differentiable geometric and visual alignment.

Alternatively, we propose a novel approach relying on differentiable rendering of shape priors to recover metric scale, pose, and shape of vehicles in the wild. Our 3D autolabeling pipeline requires only 2D detections (bounding boxes or instance masks) and sparse point clouds (ubiquitous in 3D robotic contexts). Detections themselves are produced using off-the-shelf 2D detectors. We demonstrate that differentiable visual alignment, also re-

---

* First co-authorship. This work resulted from an internship at TRI.

ferred to as *"analysis-by-synthesis"* [42] or *"render-and-compare"* [20], is a powerful approach towards autolabeling for the purpose of autonomous driving.

The present study introduces three novel contributions. First, we formulate the notion of a **Coordinate Shape Space (CSS)**, which combines Normalized Object Coordinates (NOCS) [38] with the DeepSDF framework [30]. This allows to reliably set object shapes into correspondence to facilitate *deformable shape matching*. Second, we present a way to differentiate DeepSDF with respect to its surface, thereby introducing a novel **differentiable SDF renderer** for comparative scene analysis over a defined shape space. The third contribution is a **curriculum learning-based autolabeling pipeline** of driving scenes. Figure 1 presents an example optimization on the KITTI3D dataset [12].

Our pipeline starts with a CSS network, which is trained to predict 2D NOCS maps as well as shape vectors from image patches. To bootstrap an initial version, we train the network on synthetic data, for which we can easily obtain ground truth NOCS and shape vector targets, and apply augmentations to minimize the sim2real domain gap. Our autolabeling loop includes the following steps: 1) leveraging 2D detections to localize instances; 2) running the CSS network on an extracted patch; 3) reprojecting NOCS into the scene using LIDAR, 4) decoding an object model from the shape space; 5) computing an approximate pose using 3D-3D correspondences; and 6) running projective and geometric alignment for refinement. After processing all images, we collect the recovered *autolabels* and retrain our CSS prediction network to gradually expand it into the new domain. Then, we repeat this process to achieve better CSS predictions and, consequently, better autolabels. To avoid drifting due to noisy autolabels, we employ a curriculum that is focused on easy samples first and increases the difficulty in each loop.

In summary, **our main contributions** are as follows: (i) a novel, fully-differentiable renderer for signed distance fields that can traverse smooth shape spaces; (ii) a mixed synthetic/real curriculum framework that learns to predict shape and object coordinates on image patches; and (iii) a multi-modal optimization pipeline combining differentiable alignment over vision and geometry. We evaluate our approach on the KITTI3D dataset [12] and show that our method can be used to accurately recover metric cuboids with structural, differentiable priors. Furthermore, we demonstrate that such cuboids can be leveraged to train efficient 3D object detectors.

2. Related Work

In recent years, assisted labeling has gained growing attention as the increasing amount of data hinders the ability to label it manually. In [41], the authors utilize a 2D detector to seed 2D box annotations further refined by humans and report an increase of 60% in the overall labeling speed. The authors in [2] train a recurrent CNN to predict polygons on an image to accelerate semantic segmentation tasks. A follow-up work [25] further improves the system by predicting all polygon vertices simultaneously and facilitates real-time interaction. In [22], the authors propose a 3D labeling interface that enables the users selecting spatial seeds to infer segmentation, 3D centroid, orientation, and extent using pretrained networks. In [10], 2D labels are used to seed a LIDAR-based detector combined with human annotation based on uncertainty. All mentioned works are active learning frameworks in which a human is assisted by predictive models. Instead, we aim to investigate how well an *automatic* pipeline with geometric verification can perform in this context.

Independently from our research, several recent works focused on differentiable rendering have been published. The works [27, 15, 4] discuss different ways to produce gradients for rasterization of triangle-based meshes. The study described in [26] represents a soft rasterizer approach in which each pixel is softly assigned to each triangle in the mesh, followed by a softmax to emulate z-buffering. In contrast, the authors in [23] propose a path tracing approach towards differentiable rendering of triangle meshes.

Concerning the more practically relevant research works, in [20, 29] the authors employ a learned shape space from PCA or CAEs to predict the shape of cars. Nonetheless, their shape space has been either not differentiable in an end-to-end manner, or they use finite differences of local samples to approximate the gradient aiming to avoid a complex back-propagation through the rasterizer. Unlike these works, our differentiable rendering approach utilizes a shape space derived from DeepSDF [30], which enables backpropagation onto a smooth shape manifold. Therefore, our method avoids the typical topology problems inherent to the related mesh-based approaches.

DensePose [13] introduces a framework to densely map canonical 2D coordinates to human bodies. The subsequent works [18, 44, 45] discuss how to employ such coordinates with differentiable rendering in the wild to recover the pose, shape, texture and even future motion for different entities from localized crops. The study in [19] presents impressive results using a similar approach trained with a cycle consistency and differentiable rendering. These methods allow for projective scene analysis up to scale and cannot be immediately used for 3D automotive purposes.

Dense coordinates have recently been extended to 3D space. The authors in [43, 31, 24, 17, 33] apply such representations for monocular pose estimation of known CAD models. In [38], the authors learn to predict normalized coordinates (NOCS) for object categories in RGB data and recover the metric scale and pose using an additional depth map via 3D-3D correspondences.
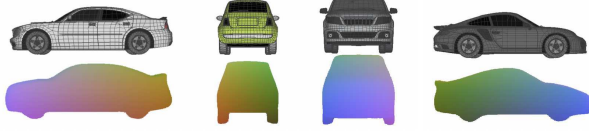
Figure 2: CSS representation. Top: Car models from the PD dataset [1]. Bottom: The same cars in the CSS representation: decoded shape vector **z** colored with NOCS.

In [8], the authors use stereo depth and run a detector to initialize the instances which are further optimized for pose and shape via SDF priors. In a follow-up work [9], the authors extend the framework with temporal priors to simultaneously recover smooth shapes and pose trajectories. Finally, [37] explores 3D object completion using a shape space and LIDAR as weak supervision with a probabilistic formulation. This work assumes correct localization and is focused solely on the reconstruction quality.

3. Methodology

We first discuss our shape space construction and the coupling into the CSS representation. Afterwards, we introduce our differentiable rendering approach tailored towards implicit surface representations. Eventually, our autolabeling pipeline is described in more detail.

3.1. Coordinate Shape Space

We employ DeepSDF [30] to embed watertight car models into a joint and compact shape space representation within a single neural network. The idea is to transform input models into signed distance fields in which each value corresponds to a distance to the closest surface, with positive and negative values representing exterior/interior area. Eventually, DeepSDF forms a shape space of implicit surfaces with a decoder f that can be queried at spatially-continuous 3D locations $\mathbf{x} = \{x_1, ..., x_N\}$ using the provided latent code $\mathbf{z}$ to retrieve SDF values $\mathbf{s} = \{s_1, ..., s_N\}$ as follows:

$$f(\mathbf{x}; \mathbf{z}) = \mathbf{s}. \tag{1}$$

To facilitate approximate *deformable shape matching*, we combine the shape space with NOCS [38] to form the Coordinate Shape Space (CSS). To this end, we resize our models to unit diameter and interpret the 3D coordinates of the 0-level set as dense surface descriptions.

To train f, we use a synthetic dataset provided by Parallel Domain [1], which comprises car CAD models, as well as rendered traffic scenes with ground truth labels. Other synthetic datasets (*e.g.*, CARLA [7] & VKITTI [11]) could be used here as well. We trained on a subset of 11 models and with a latent code dimensionality of 3. We follow the original DeepSDF training, but project the latent vector
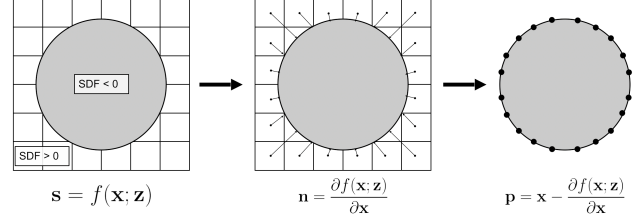

Figure 3: Surface projection. DeepSDF outputs the signed values **s** for query locations **x**. Normals **n** can be computed analytically by a single backward pass. Given the signed values and normals, we project the query locations onto the object surface points **p**. Only exterior points are visualized.

onto the unit sphere after each iteration. In Figure 2, we depict example models with their CSS representations.

3.2. Differentiable SDF Rendering

An essential component of our autolabeling pipeline is the possibility to optimize objects with respect to the pose, scale, and shape. To this end, we propose, to the best of our knowledge, the first differentiable renderer for signed distance fields. Our renderer avoids mesh-related problems such as connectivity or intersection, but necessitates implementing a different approach for sampling the representation. Rendering implicit surfaces is done either with ray-tracing [6] or variants of Marching Cubes [28]. Here, we present an alternative that lends itself to backpropagation.

Projection of 0-Isosurface Given query points x_i and associated signed distance values s_i, we need a differentiable way to access the implicit surface encoded by **z**. Simply selecting query points based on their distance values do not form a derivative with respect to the latent vector **z**. Moreover, the regularly-sampled locations can be estimated only approximately on the surface. However, we utilize that deriving SDFs with respect to their location yields a normal at this point, practically computed in a backward pass:

$$n_i = \frac{\partial f(x_i; \mathbf{z})}{\partial x_i}. \tag{2}$$

As normals outline the direction to the closest surface and signed distance values provide the exact distance, we project the query location onto a 3D surface position p_i:

$$p_i = x_i - \frac{\partial f(x_i; \mathbf{z})}{\partial x_i} f(x_i; \mathbf{z}). \tag{3}$$

To obtain clean surface projections we disregard all points x_i outside a narrow band ($|s_i| > 0.03$) of the surface. A schematic explanation can be found in Figure 3. With this formulation, we can define derivatives at p_i with respect to the scale, pose, or latent code.
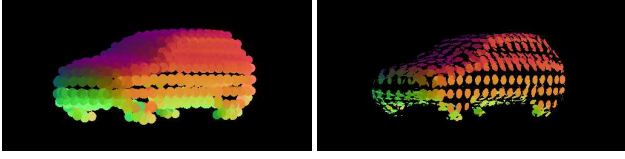
Figure 4: Oriented tangent discs (right) represent the surface geometry more accurately than billboard ones. We reduced spatial sampling and diameters for better emphasis.

Surface Tangent Discs In the computer graphics domain, the concept of surface elements (surfels) [34] is a well-established alternative to connected triangular primitives. Our differentiable SDF representation yields oriented points and can be immediately used to render surface discs. To obtain a water-tight surface, we determine disk diameters large enough to close holes. In Figure 4 we outline the difference between (oriented) surface tangent discs and billboard ones pointing straight at the camera.

We construct the surface discs with the following steps:

1. Given the normal of a projected point $n_i = \frac{\partial f(p_i;\mathbf{z})}{\partial p_i}$, we estimate the 3D coordinates of the resulting tangent plane visible in the screen. The distance d of the plane to each 2D pixel (u, v) can be computed by solving a system of linear equations for the plane and camera projection, resulting in the following solution (we refer to the supplement for details):

$$d = \frac{n_i \cdot p_i}{n_i \cdot K^{-1} \cdot (u, v, 1)^T}, \qquad (4)$$

where K^{-1} is the inverse camera matrix, followed by backprojection to get the final 3D plane coordinate:

$$P = K^{-1} \cdot (u \cdot d, v \cdot d, d)^T. \qquad (5)$$

2. Estimate the distance between the plane vertex and surface point and clamp if it is larger than a disc diameter:

$$M = max(diam - ||p_i - P||_2, 0) \qquad (6)$$

To ensure water-tightness we compute the diameter from the query location density: $diam = \min_{i \neq j} ||x_i - x_j||_2 \sqrt{3}$. Executing the above steps for each pixel yields a depth map $\mathbf{D}_i$ and a tangential distance mask $\mathbf{M}_i$ at point p_i.

Rendering Function To generate a final rendering we need a function to compose layers of 2D-projected discs onto the image plane. Similarly to [26], we combine colors from different point primitives based on their depth values. The closer the primitive is to the camera, the stronger its contribution. We use softmax to ensure that all primitive contributions sum up to 1 at each pixel. More specifically, the rendering function is:
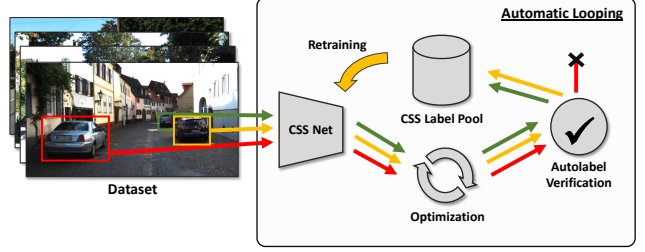

Figure 5: Automatic annotation pipeline. We fetch frames from the dataset and separately process each 2D detection using our CSS network and differentiable optimization procedure. Afterwards, we perform a verification to discard incorrect autolabels before saving them into our CSS label pool. Once all frames are processed, we retrain our CSS network and begin the next loop over the dataset.

$$\mathcal{I} = \sum_i NOCS(p_i) * w_i, \qquad (7)$$

where $\mathcal{I}$ is the resulting image, $NOCS$ returns coordinate coloring, and w_i are the weighting masks that define the contribution of each disc:

$$w_i = \frac{exp(-\tilde{\boldsymbol{D}}_i \boldsymbol{\sigma}) \boldsymbol{M}_i}{\sum_j exp(-\tilde{\boldsymbol{D}}_j \boldsymbol{\sigma}) \boldsymbol{M}_j}, \qquad (8)$$

where $\tilde{D}$ is the normalized depth, and $\boldsymbol{\sigma}$ is a transparency constant with $\boldsymbol{\sigma} \to \infty$ being completely opaque as only the closest primitive is rendered. This formulation enables gradient flow from pixels to surface points and allows image-based optimization.

3.3. 3D Autolabeling Pipeline

The general idea of our autolabeling approach is to exploit weak labels and strong differentiable priors to recover labels of higher complexity. While this idea is generic, we focus specifically on cuboid autolabeling of driving scenes.

We present a schematic overview in Figure 5 where we run multiple loops of the annotation pipeline. In the first loop, the CSS label pool solely consists of synthetic labels and the trained CSS network is therefore not well-adapted to real imagery. The results are noisy NOCS predictions which are reliable only for *well-behaved* object instances in the scene. Therefore, we define a curriculum in which we first focus on *easy* annotations and increase the difficulty over more loops. We define the difficulty of a label by measuring the pixel sizes, the amount of intersection with other 2D labels, and whether the label touches the border of an image (often indicating object truncation). We also establish thresholds for these criteria to define a curriculum of increasing difficulty.
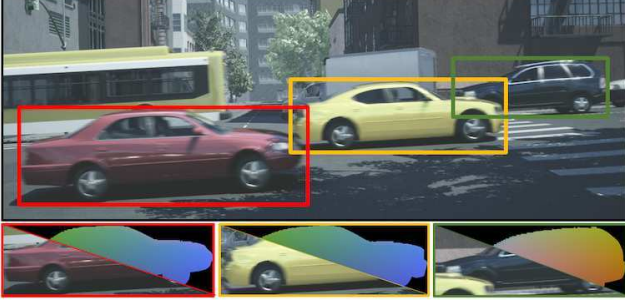
Figure 6: Synthetic PD dataset. Top: Random RGB frame. Bottom: Patches used for CSS training.

CSS Network The network is derived from ResNet18 and adopts an encoder-decoder structure, processing 128×128 input patches to output a NOCS map of the same size and a 3D shape vector. Before executing the first annotation loop, our CSS network must learn to infer 2D NOCS maps and shape vectors from patches. As mentioned, we bootstrap such a mapping from a synthetic dataset. In total, we extract around 8k patches and, having access to CAD models, also create the necessary regression targets. We demonstrate some frames and training data in Figure 6 and provide the additional information in the supplement.

3.3.1 Initialization and Optimization

Here, we describe the process represented in Figure 1 in more detail. For a given patch we infer the 2D NOCS map $\mathcal{M}$ and shape vector z. We decode z into an SDF and retrieve the 3D surface points $\mathbf{p} = \{p_1, ..., p_n\}$ of the object model (as described in Section 3.2) in its local frame, for which we compute the NOCS coordinates $\mathbf{p}^c = \{p_1^c, ..., p_n^c\}$. We also project the 3D LIDAR points $\mathbf{l} = \{l_1, ..., l_k\}$ contained inside the camera frustum onto the patch and collect the corresponding NOCS coordinates $\mathbf{l}^c$. To estimate an initial pose and scale, we establish 3D-3D correspondences between $\mathbf{p}$ and $\mathbf{l}$. For each p_i, we find its nearest neighbor based on NOCS distances:

$$j^* = \arg\min_j ||p_i^c - l_j^c||_2 \qquad (9)$$

and keep it if $||p_i^c - l_{j^*}^c|| < 0.2$. Finally, we run Procrustes [35] with RANSAC to estimate pose (R, t) and scale s.

On this basis, we apply our differentiable optimization over complimentary 2D and 3D evidence. While the projective 2D information provides strong cues about the orientation and shape, 3D points allow reasoning over the scale and translation. At every iteration we decode the current shape vector estimate $\hat{z}$, extract surface points p_i, and transform them based on the current estimates of the pose and scale:

$$\hat{p}_i = (\hat{R} \cdot \hat{s}) \cdot p_i + \hat{t}. \qquad (10)$$

Given these surface model points in the scene frame, we compute the individual losses as follows.

2D loss: We employ our differentiable SDF renderer to produce a rendering $\mathcal{R}$ for which we seek maximum alignment with NOCS map $\mathcal{M}$. Since our predicted $\mathcal{M}$ can be noisy (especially in the first loop), minimizing dissimilarity $min||\mathcal{M} - \mathcal{R}||$ can yield suboptimal solutions. Instead, for each rendered spatial pixel r_i in $\mathcal{R}$ we determine the closest NOCS space neighbor in $\mathcal{M}$, named m_j within a small radius θ, and set them in correspondence if their NOCS distance is below a predefined threshold. The loss is then defined as the mean distance over all such correspondences C_{2D} in the NOCS space:

$$loss_{2D} = \frac{1}{|C_{2D}|} \sum_{(i,j) \in C_{2D}} ||\mathcal{R}(r_i) - \mathcal{M}(m_i)||_2. \qquad (11)$$

3D loss: For each $\hat{p}_i$, we determine the nearest neighbor from l and keep it if it is closer than 0.25m. As the initializations are usually good, we avoid outliers in the optimization with such a tight threshold. The loss is then calculated as the mean distance over all correspondences C_{3D}:

$$loss_{3D} = \frac{1}{|C_{3D}|} \sum_{(i,j) \in C_{3D}} ||\hat{p}_i - l_j||_2. \qquad (12)$$

Altogether, the final criterion is the sum of both losses:

$$loss = loss_{2D} + loss_{3D}. \qquad (13)$$

As the loss terms have similar magnitudes we did not consider a need for any additional balancing.

3.3.2 Verification and CSS Retraining

Our optimization framework will inevitably output incorrect results at times, so we need to ensure that the influence of badly-inferred autolabels is minimized. To this end, we enforce geometric and projective verification aiming to remove incorrect autolabels with the largest impact. To achieve this, we measure the amount of LIDAR points that are in a narrow band (0.2m) around the surface of an autolabel and reject it if less than 60% are outside this band. Furthermore, we define a projective constraint where autolabels should be rejected if the rendered mask's IoU with the provided 2D label is below 70%.

All autolabels that remain after the verification stage are gathered and added to the CSS label pool. After the first loop, we obtain a mixture of synthetic and real samples that are then used to retrain and improve the robustness of our CSS network. Over multiple self-improving loops, we keep growing and retraining, which results in better CSS predictions, better initializations, and more accurate autolabels.
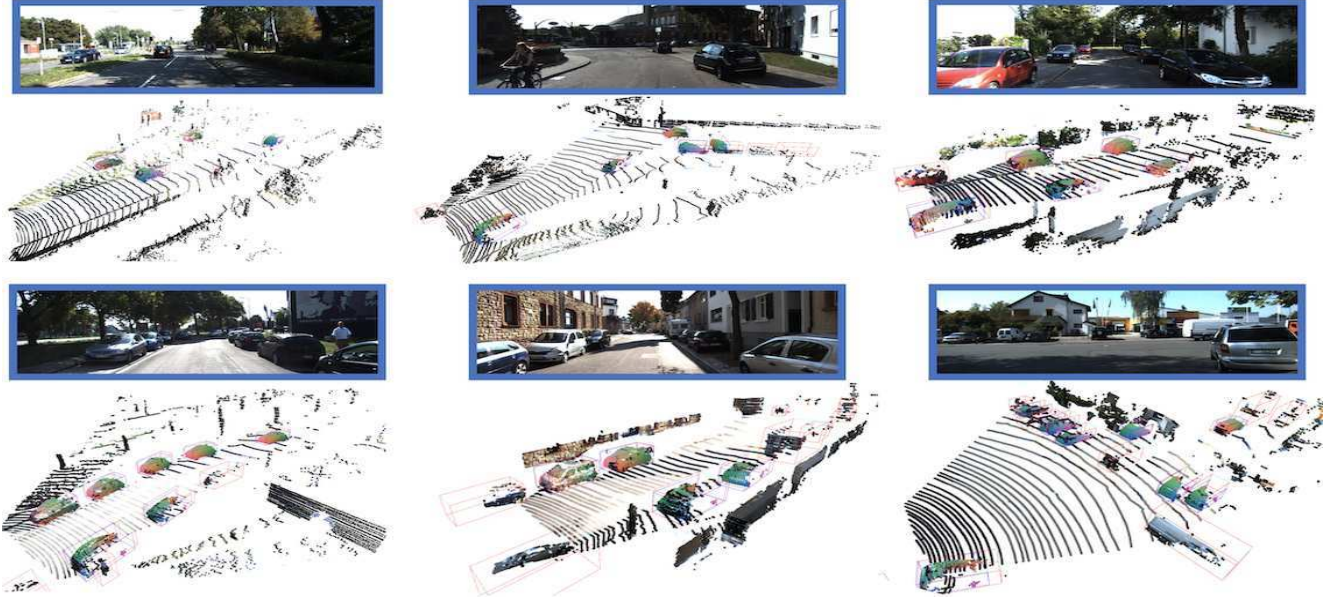
Figure 7: Qualitative results of our labeling pipeline. We mark the ground truth cuboids in red and our predictions in blue. We can achieve rather tight fits that lead to cuboids of slightly different sizes compared with the ground truth.

4. Experimental Evaluation

We evaluate our approach on the well-established KITTI3D dataset [12], including 7481 frames with accompanied cuboid labels for the "Car" category, which we focus on. We consider the most-widely used 3D metrics for driving datasets: BEV IoU and 3D IoU from KITTI3D as well as the distance-thresholded metric (NS) from NuScenes [3], which decouples location from scale and orientation. All three metrics are utilized to evaluate Average Precision for matches at certain cutoffs, and we threshold BEV and 3D IoUs at 0.5 whereas NS is computed at distance cutoffs of 0.5m and 1m.

The KITTI3D metrics are often evaluated at a strict threshold of 0.7. After thorough inspection, we observed that it is difficult to infer the correct cuboid size from tight surface fits. The KITTI3D cuboids have a varying amount of spatial padding and 3D detection methods learn these offsets. Therefore, we opted to relax the threshold aiming to facilitate a fairer comparison with respect to the estimated tight cuboids. Figure 7 represents several examples of appropriate autolabel estimates among which some do not pass the 0.7 3D IoU criterion.

Implementation Details We use PyTorch [32] to implement the whole pipeline. For each 2D-labeled instance we run 50 iterations and use the ADAM optimizer for the pose variables with a learning rate of 0.03 whereas SGD is applied to the scale and shape with smaller learning rates (0.01 and 0.0005) and no momentum to avoid observed over-

shooting behavior. It takes approximately 6 seconds to autolabel a single instance on a Titan V GPU, and one autolabeling loop takes 1-2 hours to complete for all frames when parallelizing on 2 GPUs.

Data Augmentation Our synthetic bootstrapping requires many kinds of augmentation to allow for initial domain transfer. We utilize an extensive set of functions, including random rotations up to $10°$, horizontal flips, and random cropping. This allows not only to considerably expand the data size, but also enables explicitly covering cases of partial occlusion and truncation. Additionally, we vary the brightness, contrast, and saturation of input patches. Moreover, as we have access to corresponding CAD models, we render surface normals and use Phong shading to generate geometrically-sensible relighting.

4.1. Correctness of autolabels

The most important quantitative criterion is the actual correctness of the estimated cuboids. Although our method is fully automatic, we have access to KITTI3D 2D ground truth boxes and therefore evaluate two scenarios. Firstly, we obtain 2D boxes from KITTI3D for autolabeling and then, utilize their predefined criteria to determine whether an annotation is considered easy or moderate. Secondly, we employ the detectron2 implementation [40] of Mask-RCNN [14] using a ResNeXt101 backbone trained on COCO to evaluate the applicability of off-the-shelf object detectors for full automation. In the detection scenario, we run separate experiments for boxes and masks and apply following

Loop	Diff.	KITTI GT				RCNN				MASK-RCNN			
		BEV@0.5	3D@0.5	NS@0.5	NS@1.0	BEV@0.5	3D@0.5	NS@0.5	NS@1.0	BEV@0.5	3D@0.5	NS@0.5	NS@1.0
1	E	78.09	63.53	85.59	95.58	78.45	63.71	85.85	95.62	78.46	63.69	86.27	95.76
2	E	77.84	62.25	82.40	90.84	80.57	60.11	86.05	94.62	80.70	63.96	86.52	94.31
	M	59.75	42.23	60.27	77.91	61.17	42.37	64.11	85.85	63.36	44.79	64.44	85.24

Table 1: Cuboid autolabel quality when inputting into the CSS network (a) 2D ground truth boxes, (b) RCNN detections, and (c) Mask-RCNN detections. We run two self-improving loops to slowly incorporate more labels into the pool.

difficulty criteria similar to KITTI3D: easy if label height > 40px and not touching other labels or image borders; moderate if height > 25px and not having an IoU > 0.30 with any other label.

We present the obtained results in Table 1. As expected, the first loop with a purely synthetically-trained inference on easy real samples does not yield a considerable difference between the three scenarios. All of them are impacted by noisy CSS predictions and start from the same RANSAC initialization, although the detection-derived labels are tighter and slightly less influenced by CSS background noise. Overall, each scenario achieves a **BEV AP close to 80%** and a 3D AP of approximately 60% over all easy samples. We execute a second loop over the dataset using the retrained CSS network and observe that the results for the easy samples stabilized. Additionally, we note that we can recover approximately 60% BEV AP over all considered moderate samples. Noteworthy, the estimated NS scores are quite high, indicating that most autolabels (more than 90%) are within one meter of the real location.

However, we observe a drop of around 20 points across all metrics for the harder samples. As our method is reliant on 3D-3D RANSAC, it requires a minimum set of inliers to
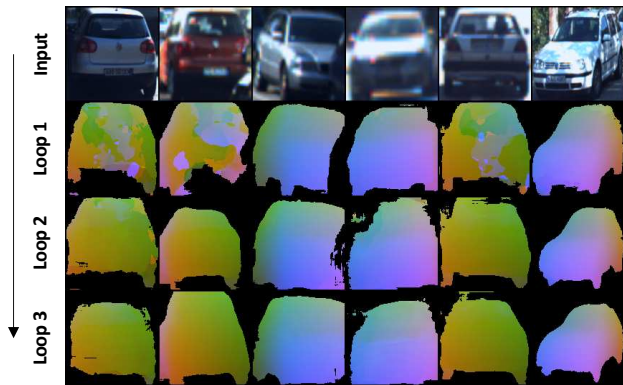

Figure 8: NOCS prediction quality of our network over consecutive loops for the same patch. Initially, the predictions are rather noisy because of the synthetic domain gap. Within each subsequent autolabeling loop the predictions become more accurate overall.

achieve proper solutions. We often observe that this made correspondence finding difficult for occasional samples due to occlusion and distance, thus impacting recall.

Figure 8 shows the increasing quality of the estimated NOCS predictions over multiple loops. Overall, we observe a rather fast diffusion into the target domain and that executing two loops is sufficient to stabilize the results.

4.2. Ablation

We aim to investigate how much the initial estimates from 3D-3D RANSAC benefit from our optimization. To this end, we consider the easy ground truth boxes from KITTI3D and utilize the synthetic CSS network to analyze the first annotation loop with the worst initialization. As represented in Table 3, the RANSAC baseline provides rather good localization which is best captured by the NS metrics (81.36% and 95.45%). Nonetheless, the pose-optimized autolabels yield a significant jump in 3D IoU (41.85% vs. 63.42%), suggesting that we recover substantially better rotations, given that the NS scores are similar. When ablating over the other variables we observe rather mixed results in which certain metrics increase or decrease.

When ablating over the losses, we note a drastic drop in the 3D metrics when optimizing only in 2D. Intuitively, our differentiable renderer aligns the data rather appropriately in the image space; however, both scale and translation are freely drifting. Therefore, optimizing the 3D loss results in strong spatial alignment. Nonetheless, optimizing the sum of both losses trades BEV AP (80.61 to 78.09) for 3D AP (60.92 to 63.53).

4.3. 3D Object Detection

Since autolabels are usually not the final goal but rather a means to an end, we investigate the applicability of our labels to the task of 3D object detection. We evaluate the quality of our labels concerning both a traditional LIDAR-based detection setting and a purely monocular setting, based on several recent works that achieved high quality results on the KITTI dataset [29, 36, 21].

We implemented a version of the current state-of-the-art monocular detector MonoDIS [36] and ensured that we can reproduce the reported results. Additionally, we utilize the official implementation of PointPillars [21], a state-of-the-

Method	2D AP @ 0.5/0.7		3D AP @ 0.5/0.7		BEV AP @ 0.5/0.7	
	Easy	Moderate	Easy	Moderate	Easy	Moderate
PointPillars [21] (Original Labels)	- / -	- / -	94.8 / 81.1	92.4 / 68.2	95.1 / 92.1	95.1 / 84.7
PointPillars [21] (Autolabels)	- / -	- / -	90.7 / 22.4	71.1 / 13.3	94.9 / 81.0	88.5 / 59.8
MonoDIS [36] (Original Labels)	96.1 / 95.5	92.6 / 86.5	45.7 / 11.0	32.9 / 7.1	52.4 / 17.7	37.2 / 11.9
MonoDIS [36] (Autolabels)	96.7 / 85.8	86.2 / 67.6	32.9 / 1.23	22.1 / 0.54	51.1 / 15.7	34.5 / 10.52

Table 2: The performance comparison of the 3D object detectors trained on the true KITTI labels vs. our autolabels. Concerning the BEV metric, **the detectors trained on autolabels alone achieve the results equal to the current state of the art**. In the case of the 3D AP metric, the competitive results are achieved in both considered variants at the IoU 0.5 threshold.

art LIDAR-only detector. To train MonoDIS, we follow the training schedule proposed in [36]. Concerning PointPillars training, we accelerate the training by means of 8 V100 GPUs and a batch size of 16. Accordingly, we scale the learning rate by a factor of 8. We evaluate the obtained results on the MV3D train/val split [5]. While training on autolabels, we do not change any hyperparameters defined in the baseline protocols.

We present the comparison results in Table 2 and depict several qualitative detections obtained from autolabel-trained detectors in Figure 9.

Remarkably, concerning the BEV metric, both detectors trained *on autolabels alone* achieve competitive performance compared with detectors trained on true KITTI labels at both the 0.5 and 0.7 IoU thresholds. This indicates
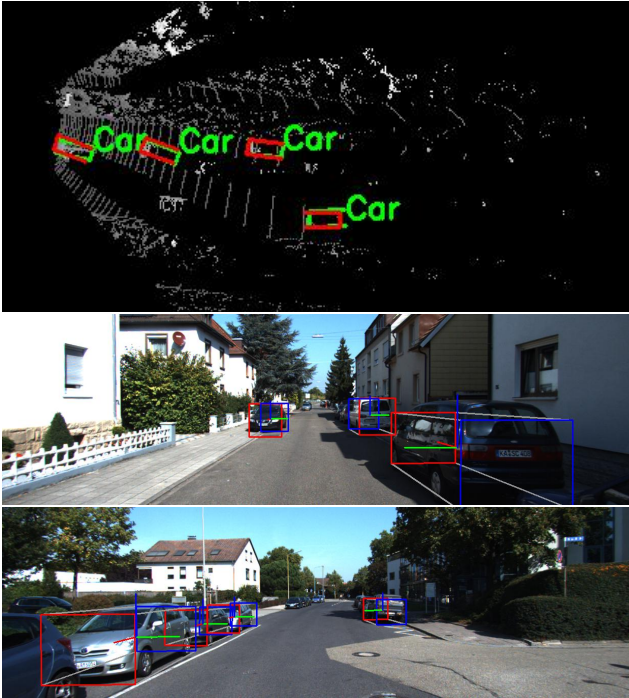

Figure 9: Detections from the autolabel-trained detectors. We draw local 3D frames to identify correct orientation.

Config	BEV@0.5	3D@0.5	NS@0.5	NS@1.0
RANSAC	77.00	41.85	81.36	95.45
(R, t)	77.19	63.42	86.20	95.53
$(R, t), s$	77.23	62.92	86.01	95.32
$(R, t), s, z$	78.09	63.53	85.59	95.58
2D loss	18.08	11.09	18.35	46.19
3D loss	80.61	60.92	85.63	95.49

Table 3: Ablation study over each optimization variable and each separate loss.

that our autolabeling pipeline is capable of highly accurate localization of cuboids.

Considering the 3D AP metric, the obtained results are in line with the conclusions represented in Table 1. At the more tolerant IoU 0.5 threshold, our autolabel trained detectors performed within $70 - 90\%$ of the true labels. Occasionally missing detections and poor shape estimates do not deteriorate the overall performance.

At the IoU 0.7 threshold, the detector performance worsens. We observe that this is not caused by poor predictions, but by the fact that KITTI labels are often inflated with respect to the estimated cuboids. Therefore, at the strictest thresholded 3D IoU, we observe a corresponding drop in precision for detectors trained on our "tight" autolabels.

5. Conclusion

We present a novel view on parametric 3D instance recovery in the wild based on a self-improving autolabeling pipeline, purely bootstrapped from synthetic data and off-the-shelf detectors. Fundamental to our approach is the combination of dense surface coordinates with a shape space, and our contribution towards differentiable rendering of SDFs. We show that our approach can recover a substantial amount of cuboid labels with high precision, and that these labels can be used to train 3D object detectors with results close to the state of the art. Future work will be focused on investigating additional categories for parametric reconstruction, such as pedestrians or road surfaces.

References

[1] Parallel domain: Data generation for autonomy. https://www.paralleldomain.com/. 3

[2] David Acuna, Huan Ling, Amlan Kar, and Sanja Fidler. Efficient interactive annotation of segmentation datasets with polygon-rnn++. In *CVPR*, 2018. 2

[3] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multi-modal dataset for autonomous driving. *arXiv*, 2019. 6

[4] Wenzheng Chen, Jun Gao, Huan Ling, Edward J. Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. In *NeurIPS*, 2019. 1, 2

[5] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *VPR*, 2017. 8

[6] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH*, 1996. 3

[7] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *CoRL*, 2017. 3

[8] Francis Engelmann, Jörg Stückler, and Bastian Leibe. Joint object pose estimation and shape reconstruction in urban street scenes using 3D shape priors. In *GCPR*, 2016. 3

[9] Francis Engelmann, Jörg Stückler, and Bastian Leibe. Samp: Shape and motion priors for 4d vehicle reconstruction. In *WACV*, 2017. 3

[10] Di Feng, Xiao Wei, Lars Rosenbaum, Atsuto Maki, and Klaus Dietmayer. Deep active learning for efficient training of a lidar 3d object detector. In *IV*, 2019. 2

[11] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual worlds as proxy for multi-object tracking analysis. In *CVPR*, 2016. 3

[12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, 2012. 2, 6

[13] Rıza Alp Güler, Natalia Neverova, and Iasonas Kokkinos. Densepose: Dense human pose estimation in the wild. In *CVPR*, 2018. 2

[14] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. In *ICCV*, 2017. 6

[15] Yoshitaka Ushiku Hiroharu Kato and Tatsuya Harada. Neural 3d mesh renderer. In *CVPR*, 2018. 2

[16] Calvin Huang. Adding a dimension: Annotating 3d objects with 2d data. 2018. https://scale.com/blog/3d-cuboids-annotations. 1

[17] Omid Hosseini Jafari, Siva Karthik Mustikovela, Karl Pertsch, Eric Brachmann, and Carsten Rother. ipose: instance-aware 6d pose estimation of partly occluded objects. In *ACCV*, 2018. 2

[18] Angjoo Kanazawa, Shubham Tulsiani, Alexei A. Efros, and Jitendra Malik. Learning category-specific mesh reconstruction from image collections. In *ECCV*, 2018. 2

[19] Nilesh Kulkarni, Abhinav Gupta, and Shubham Tulsiani. Canonical surface mapping via geometric cycle consistency. In *ICCV*, 2019. 2

[20] Abhijit Kundu, Yin Li, and James M. Rehg. 3d-rcnn: Instance-level 3d object reconstruction via render-and-compare. In *CVPR*, 2018. 2

[21] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *CVPR*, 2019. 7, 8

[22] Jungwook Lee, Sean Walsh, Ali Harakeh, and Steven Waslander. Leveraging pre-trained 3d object detection models for fast ground truth generation. In *ITSC*, 2018. 1, 2

[23] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. In *SIGGRAPH Asia*, 2018. 2

[24] Zhigang Li, Gu Wang, and Xiangyang Ji. Cdpn: Coordinates-based disentangled pose network for real-time rgb-based 6-dof object pose estimation. In *CVPR*, 2019. 2

[25] Huan Ling, Jun Gao, Amlan Kar, Wenzheng Chen, and Sanja Fidler. Fast interactive object annotation with curve-gcn. In *CVPR*, 2019. 1, 2

[26] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *ICCV*, 2019. 2, 4

[27] Matthew M. Loper and Michael J. Black. OpenDR: An approximate differentiable renderer. In *ECCV*, 2014. 2

[28] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *SIGGRAPH*, 1987. 3

[29] Fabian Manhardt, Wadim Kehl, and Adrien Gaidon. Roi-10d: Monocular lifting of 2d detection to 6d pose and metric shape. In *CVPR*, 2019. 2, 7

[30] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *CVPR*, 2019. 2, 3

[31] Kiru Park, Timothy Patten, and Markus Vincze. Pix2pose: Pixel-wise coordinate regression of objects for 6d pose estimation. In *CVPR*, 2019. 2

[32] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017. 6

[33] Sida Peng, Yuan Liu, Qixing Huang, Xiaowei Zhou, and Hujun Bao. Pvnet: Pixel-wise voting network for 6dof pose estimation. In *CVPR*, 2019. 2

[34] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In *SIGGRAPH*, 2000. 4

[35] P.H. Schönemann. A generalized solution of the orthogonal procrustes problem. In *Psychometrika*, 1966. 5

[36] Andrea Simonelli, Samuel Rota Bulo, Lorenzo Porzi, Manuel Lopez-Antequera, and Peter Kontschieder. Disentangling monocular 3d object detection. In *ICCV*, 2019. 7, 8

[37] David Stutz and Andreas Geiger. Learning 3d shape completion under weak supervision. *IJCV*, 2018. 3

[38] He Wang, Srinath Sridhar, Jingwei Huang, Julien Valentin, Shuran Song, and Leonidas J. Guibas. Normalized object coordinate space for category-level 6d object pose and size estimation. In *CVPR*, 2019. 2, 3

[39] Zian Wang, Huan Ling, David Acuna, Amlan Kar, and Sanja Fidler. Object instance annotation with deep extreme level set evolution. In *CVPR*, 2019. 1

[40] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019. 6

[41] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving video database with scalable annotation tooling. In *arXiv*, 2018. 2

[42] Alan Yuille and Daniel Kersten. Vision as bayesian inference: analysis by synthesis? *Trends in cognitive sciences*, 10(7):301–308, 2006. 2

[43] Sergey Zakharov, Ivan Shugurov, and Slobodan Ilic. Dpod: Dense 6d pose object detector in rgb images. In *ICCV*, 2019. 2

[44] Jason Y. Zhang, Panna Felsen, Angjoo Kanazawa, and Jitendra Malik. Predicting 3d human dynamics from video. In *ICCV*, 2019. 2

[45] Silvia Zuffi, Angjoo Kanazawa, Tanya Berger-Wolf, and Michael J. Black. Three-d safari: Learning to estimate zebra pose, shape, and texture from images "in the wild". In *ICCV*, 2019. 2