Autolabeling 3D Objects with Differentiable Rendering of SDF Shape Priors (Supplementary Material)

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

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

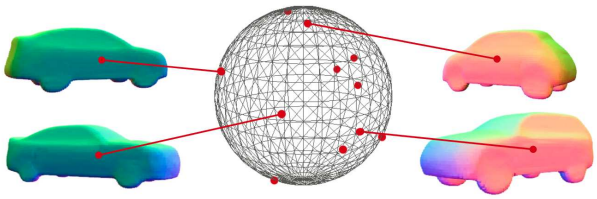

Figure 1: Visualization of the 3D latent vector space and the corresponding shapes after 0-isosurface projection.

1. DeepSDF

Our DeepSDF network is trained to cover a set of normalized predefined cars coming from the Parallel Domain (PD) dataset (visualized in Figure 2). Since we trained our latent shape vectors $\mathbf{z}$ to be 3-dimensional, we can easily visualize their positions on the surface of a 3D sphere, as shown in Figure 1. Each red point represents a specific model shape embedded in the space.

2. Data Generation

Given a set of ground truth poses with associated CAD models that we extract from our synthetic PD dataset (see Figure 4), we use our DeepSDF network and our differentiable renderer to project the models onto the screen. Instead of rendering the colors, we render the models' normalized coordinates (NOCS) represented as RGB channels (Figure 3c). Additionally, we render object normals (Figure 3b), which are subsequently used for 0-isosurface projection. Our augmentation module takes RGB crops (Figure 3a) and normal maps and applies 2D and 3D augmentations. 2D augmentations are based on the `torchvision.transforms` module operations and include random 10° rotations, horizontal flips, cropping, changes in brightness, contrast, and saturation. Moreover, normal maps provide us with local surface information that we use in conjunction with simple Phong shading. Thus, we can generate lighting based on different illumination types (namely ambient, diffusive, and specular) during training. Examples are shown in Figures 3d, 3e, 3f. Note how the bottom and top sides of the car change illumination between frames (e) and (f).

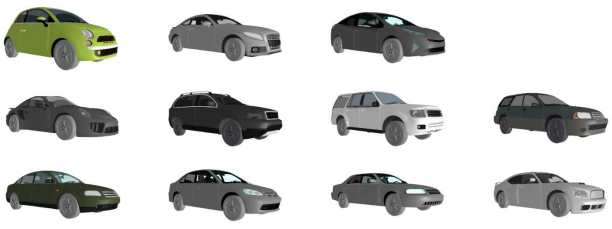

Figure 2: Cars from the PD dataset that were used to train our DeepSDF shape space.

3. Pipeline Components

The comprehensive representation of our pipeline is depicted in Figure 5. Below, we describe its main components in detail.

(a) RGB (b) Normals (c) NOCS

(d) Light 1 (e) Light 2 (f) Light 3

Figure 3: Data input modalities: (a) input RGB image, (b) rendered normal map, (c) rendered NOCS. Light module outputs: (d, e, f).

Figure 4: Some example images from the PD dataset that we used for synthetic CSS training.

3.1. CSS Net

The architecture of our CSS net is visualized in Figure 5 and we use a ResNet18 backbone architecture. The decoders use bilinear interpolation as an upsampling operation rather than deconvolution to decrease the number of parameters and the required amount of computations. Each upsampling is followed by concatenation of the output feature map with the feature map from the previous level, and one convolutional layer. Since the CSS net is trained on synthetic data, it is initialized with ImageNet weights and the first five layers are frozen in order to prevent overfitting to peculiarities of the rendered data. Five heads of the CSS net are responsible for the output of U, V, and W channels of the NOCS as well as the object's mask and its latent vector, encoding its DeepSDF shape.

3.2. Pose Estimation Block

The pose estimation block (see initial conditions estimation in (Figure 5) is based on 3D-3D correspondence estimation. The procedure is defined as follows: Our CSS net outputs normalized object coordinates (NOCS), mapping each RGB pixel to a 3D location on the object's surface. NOCS are backprojected onto the LIDAR frustum points using the provided camera parameters. Additionally, our network outputs a latent vector, which is then fed to the DeepSDF net and transformed to a surface point cloud using the 0-isosurface projection. Since our DeepSDF is trained to output normalized models placed at the origin, each point on the resulting model surface represents NOCS. At this point, we are ready to proceed with pose estimation.

NOCS are used to establish correspondences between frustum points and model points. Backprojected frustum NOCS are compared to the predicted model coordinates, and nearest neighbors for each frustum point are estimated. RANSAC is used for robust outlier rejection. At each iteration we take 4 random points (n) from the set of the correspondences and and feed them to the Procrustes algorithm, giving us initial estimates for the pose and scale of the model.

The following RANSAC parameters are used: the number of iterations k is based on a standard function of the desired probability of success p using a theoretical result:

$$k = \frac{\log(1-p)}{\log(1-w^n)}, \tag{1}$$

where w is the inlier probability and n are the independently selected data points. In our case $p = 0.9$ and $w = 0.7$.

We use a threshold of 0.2m to estimate the inliers and

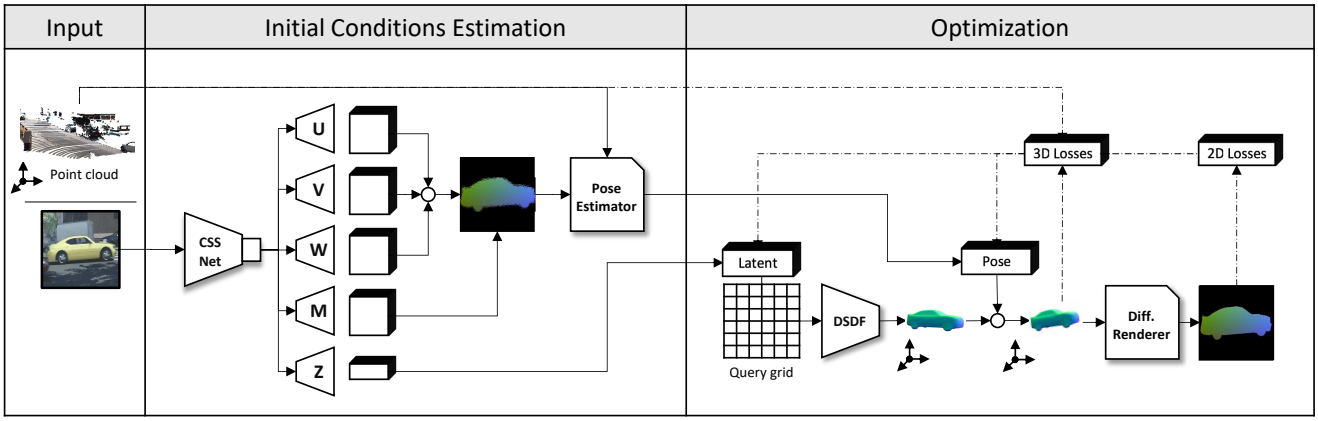

Figure 5: The core of the annotation pipeline. We fetch frames from the dataset and separately process each 2D annotation with our CSS network. The CSS net outputs 2D NOCS for each RGB input pixel, mask, and a latent shape vector. The latent vector is fed to our surface projection module to get a normalized car point cloud and the initial pose. The estimated pose transformation is then applied to the car and the 3D losses are computed. Our differentiable renderer is used to define losses in the dense 2D screen space.

choose the best fit. The final pose and scale are computed based on the inliers of the best fit.

3.3. Optimization

Given the output of the CSS network and our pose initialization, we proceed to the optimization stage (see Figure 5). By concatenating the latent vector $\mathbf{z}$ with the query 3D grid $\mathbf{x}$ we form the input for our DeepSDF network. The DeepSDF net outputs SDF values for each query point on the grid, which we use for the 0-isosurface projection, providing us with a dense surface point cloud. The resulting point cloud is then transformed using the estimated pose and scale coming from the pose estimation module. The points that should not be visible from the given camera view are filtered using simple back-face culling, since surface normals have been already computed for 0-isosurface projection. At this stage, we are ready to apply 3D losses between the resulting transformed point cloud and the input LIDAR frustum points. The surface point cloud is also used as an input to our differentiable renderer, where we render NOCS as RGB and apply 2D losses between the CSS network's NOCS prediction and the renderer's output NOCS. The latent vector and the pose are then updated and the process is repeated until termination.

3D losses allow us to get a precise pose/shape alignment with the frustum points. However, it is often the case that only few points are available resulting in poor alignment results. 2D losses, on the other hand, allow for precise alignment in the screen space over dense pixels, but are unsuitable for 3D scale and translation optimization, and heavily rely on their initial estimates. The combination of the two losses gives us the best of both worlds: dense 2D alignment and robust scale/translation estimation.

4. Surface Tangent Discs

Our surface tangent disc primitives formation requires solving a system of linear equations with a goal to compute the distance from the plane to each 2D pixel (u, v):

$$\begin{cases} u' = (u - o_u)\frac{z'}{f_u} \\ v' = (v - o_v)\frac{z'}{f_v} \\ Au' + Bv' + Cd - Au'_0 - Bv'_0 - Cd_0 = 0 \end{cases} \tag{2}$$

The first 2 are the perspective projection equations and the third is a plane equation. If we solve the above system by a simple substitution, we get the following:

$$\begin{aligned} d(\frac{A(u - o_u)}{f_u} + \frac{B(v - o_v)}{f_v} + C) \\ -Au'_0 - Bv'_0 - Cd_0 = 0 \longrightarrow \\ d = \frac{Au'_0 + Bv'_0 + Cd_0}{(\frac{A(u' - o_u)}{f_u} + \frac{B(v' - o_v)}{f_v} + C)} \\ = \frac{n \cdot p_0}{n \cdot K^{-1}(u, v, 1)^T} \end{aligned} \tag{3}$$

As a result, we can retrieve a 3D plane position (u', v', d) for each 2D point (u, v) on the screen and form primitives based on 3D distances from 3D shape points.