A. Supplementary material

A.1. Implementation details

Dataset preparation. We infer object silhouettes using PointRend [26] with an X101-FPN backbone, using their pretrained model on COCO [30]. We set the object detection threshold to 0.9 to select only confident objects. As mentioned in sec. 3, we discard object instances that are either *(i)* too small (mask area $< 96^2$ pixels), *(ii)* touch the borders of the image (indicator of possible truncation), or *(iii)* collide with other detected objects (indicator of potential occlusion). For the object part segmentations, we use the semi-supervised object detector from [17], which can segment all 3000 classes available in Visual Genome (VG) [27] while being supervised only on mask annotations from COCO. Although this model was not conceived for object part segmentation, we find that it can be used as a cost-effective way of obtaining meaningful part segmentations without collecting extra data or using co-part segmentation models that require class-specific hyperparameter tuning, such as *SCOPS* [18]. Specifically, since VG presents a long tail of rare classes, as in [38] we found it beneficial to first pre-select a small number of representative classes that are widespread across categories (e.g. all land vehicles have wheels, all animals have legs). We set the detection threshold of this model to 0.2 and, for each image category, we only keep semantic classes that appear in at least 25% of the images, which helps eliminate spurious detections. On our data, this leads to a number of semantic classes $K \approx 10$ per image category (33 across all categories). The full list of semantic classes can be seen in Fig. 5. To deal with potentially overlapping part detections (e.g. the segmentation mask of the door of a car might overlap with a window), the output semantic maps represent probability distributions over classes, where we weight each semantic class proportionally to the object detection score. Additionally, we add an extra class for "no class" (depicted in gray in our figures).

Mesh templates and remeshing. We borrow a selection of mesh templates from [28] as well as meshes freely available on the web. In the experiments where we adopt multiple mesh templates, we only use 2–4 meshes per category. An important preliminary step of our approach, which is performed even before the pose estimation step, consists in *remeshing* these templates to align them to a common topology. This has the goal of reducing their complexity (which translates into a speed-up during optimization), removing potential invisible interiors, and enabling efficient batching by making sure that every mesh has the same number of vertices/faces. Additionally, as mentioned in sec. 3.2, remeshing is required for the semi-supervision loss term in the reconstruction model. We frame this task as an optimization problem where we deform a 32×32 UV sphere to match the mesh template. More specifically, we render each template from 64 random viewpoints at 256×256 resolution, and minimize the MSE loss between the rendered deformed sphere and the target template in pixel space ($\mathcal{L}_{\text{MSE}}$). Moreover, we regularize the mesh by adding *(i)* a smoothness loss $\mathcal{L}_{\text{flat}}$, which encourages neighboring faces to have similar normals, *(ii)* a Laplacian smoothing loss $\mathcal{L}_{\text{lap}}$ with quad connectivity (i.e. using the topology of the UV map as opposed to that of the triangle mesh), and *(iii)* an edge length loss $\mathcal{L}_{\text{len}}$ with quad connectivity, which encourages edges to have similar lengths. $\mathcal{L}_{\text{flat}}$ and $\mathcal{L}_{\text{len}}$ are defined as follows:

$$\mathcal{L}_{\text{flat}} = \frac{1}{|E|} \sum_{i,j \in E} (1 - \cos \theta_{ij})^2 \tag{5}$$

$$\mathcal{L}_{\text{len}} = \frac{1}{|UV|} \sum_{i \in U} \sum_{j \in V} \frac{\|\mathbf{v}_{i+1,j} - \mathbf{v}_{i,j}\|_1 + \|\mathbf{v}_{i,j+1} - \mathbf{v}_{i,j}\|_1}{6} \tag{6}$$

where E is the set of edges, $\cos \theta_{ij}$ is the cosine similarity between the normals of faces i and j, and $\mathbf{v}_{i,j}$ represents the 3D vertex at the coordinates i, j of the UV map.
Finally, we weight each term as follows:

$$\mathcal{L} = \mathcal{L}_{\text{MSE}} + 0.00001\, \mathcal{L}_{\text{flat}} + 0.003\, \mathcal{L}_{\text{lap}} + 0.01\, \mathcal{L}_{\text{len}} \tag{7}$$

Additionally, in the experiments with multiple mesh templates, we add a pairwise similarity loss $\mathcal{L}_{\text{align}}$ which penalizes large variations of the vertex positions between different mesh templates (only within the same category):

$$\mathcal{L}_{\text{align}} = \frac{1}{N_t^2} \sum_{i=1}^{N_t} \sum_{j=1}^{N_t} \|\mathbf{V_i} - \mathbf{V_j}\|_2 \tag{8}$$

where $\mathbf{V_i}$ is a matrix that contains the vertex positions of the i-th mesh template (of shape $3 \times N_v$), and N_t is the number of mesh templates. This loss term is added to the total loss with weight 0.001. Note that we use a non-squared L2 penalty for this term, which encourages a sparse set of vertices to change between mesh templates.
We optimize the final loss using SGD with momentum (initial learning rate $\alpha = 0.0001$ and momentum $\beta = 0.9$). We linearly increase α to 0.0005 over the course of 500 iterations (warm-up) and then exponentially decay α with rate 0.9999. We stop when the learning rate falls below 0.0001. Additionally, we normalize the gradient before each update. Fig. 9 shows two qualitative examples of remeshing.

Figure 9. Remeshing of the mesh templates. In this figure we show two demos (one template for *car* and one for *airplane*).

Pose estimation. For the silhouette optimization step, we initialize $N_c = 40$ camera hypotheses per image by uniformly quantizing azimuth and elevation (8 quantization

levels along azimuth and 5 levels along elevation). We optimize each camera hypothesis using Adam [25] with full-matrix preconditioning, where we set $\beta_1 = 0.9$ and $\beta_2 = 0.95$. The implementation of our variant of Adam as well its theoretical justification are described in the next paragraph. We optimize each hypothesis for 100 iterations, with an initial learning rate $\alpha = 0.1$ which is decayed to 0.01 after the 80th iteration. After each iteration, we re-project quaternions onto the unit ball. As a performance optimization, silhouettes are initially rendered at 128×128 resolution, which is increased to 192×192 after the 30th iteration and 256×256 after the 60th iteration. Finally, in the settings where we prune camera hypotheses, we discard the worst 50% hypotheses as measured by the intersection-over-union (IoU) between projected and target silhouettes. This is performed twice: after the 30th and 60th iteration.

Algorithm 1 Adam with full-matrix preconditioning. Changes w.r.t. the original algorithm are highlighted.

require α (step size), $\beta_1, \beta_2, \epsilon$
initialize time step $t \leftarrow 0$
initialize parameters $\boldsymbol{\theta}_0$ (d-dimensional col. vector)
initialize first moment $\boldsymbol{m}_0 \leftarrow \boldsymbol{0}$ (d-dimensional col. vector)
initialize second moment $\boldsymbol{V}_0 \leftarrow \boldsymbol{0}$ ($d \times d$ matrix)
repeat
  $t \leftarrow t + 1$
  $\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_{t-1})$ ▷ gradient
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ ▷ first moment
  $\boldsymbol{V}_t \leftarrow \beta_2 \boldsymbol{V}_{t-1} + (1 - \beta_2)\, \boldsymbol{g}_t \boldsymbol{g}_t^T$ ▷ second moment
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ bias correction
  $\hat{\boldsymbol{V}}_t \leftarrow \boldsymbol{V}_t / (1 - \beta_2^t)$ ▷ bias correction
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha\, (\hat{\boldsymbol{V}}_t + \epsilon \mathbf{I}_d)^{-\frac{1}{2}}\, \hat{\boldsymbol{m}}_t$ ▷ update
until *stopping criterion*
return $\boldsymbol{\theta}_t$

Full-matrix preconditioning. Adam [25] is an established optimizer for training neural networks. Its use of diagonal preconditioning is an effective trick to avoid storing an $\mathcal{O}(d^2)$ matrix for the second moments (where d is the number of learnable parameters), for which a matrix square root and inverse need to be subsequently computed (an extra $\mathcal{O}(d^3)$ cost for each of the two operations). However, since our goal is to optimize camera parameters, we observe that:

1. Optimizers with diagonal preconditioning are not rotation invariant, i.e. they have some preferential directions that might bias the pose estimation result.
2. Since each camera hypothesis comprises only 8 parameters, inverting an 8×8 matrix has a negligible cost.

Using a rotation invariant optimizer such as SGD (with or without momentum) is a more principled choice as it addresses the first observation. However, based on our second observation, we take the best of both worlds and modify Adam to implement full-matrix preconditioning. This only requires a trivial modification to the original implementation, which we show in alg. 1 (changes w.r.t. the original algorithm are highlighted in green).

Semantic template inference. As mentioned in sec. 3.1, the goal of this step is to infer a 3D semantic template for each mesh template, given an initial (untextured) mesh template, the output of the silhouette optimization step, and a collection of 2D semantic maps. Recapitulating from sec. 3.1, we solve the following optimization problem:

$$\mathcal{L}_i = \|\mathcal{R}(\mathbf{V}_{\text{tpl}}, \mathbf{F}_{\text{tpl}}, \mathbf{C}_{\text{tpl}};\ \mathbf{q_i}, \mathbf{t_i}, s_i, z_{0i}) - \mathbf{C_i}\|^2 \quad (9)$$

$$\mathbf{C}^*_{\text{tpl}} = \min_{\mathbf{C}_{\text{tpl}}} \frac{1}{N_{\text{top}}} \sum_i \mathcal{L}_i \quad (10)$$

Conceptually, our goal is to learn a shared semantic template (parameterized using vertex colors) that averages all 2D semantic maps in vertex space. We propose the following closed-form solution which uses the gradients from the differentiable renderer and requires only a single pass through the dataset:

$$\mathbf{A} = \sum_i \nabla_{\mathbf{C}_{\text{tpl}}}(\mathcal{L}_i) \quad (11)$$

$$(\mathbf{C}^*_{\text{tpl}})_k = \frac{\epsilon + \mathbf{a_k}}{K\epsilon + \sum_j \mathbf{a_j}} \quad (12)$$

where $\mathbf{A}$ is an accumulator matrix that has the same shape as the $\mathbf{C}_{\text{tpl}}$ (the vertex colors), and ϵ is a small *additive smoothing* constant that leads to a uniform distribution on vertices that are never rendered (and thus have no gradient). This operation can be regarded as projecting the 2D object-part semantics onto the mesh vertices and computing a color histogram on each vertex. We show a sample illustration in Fig. 10.

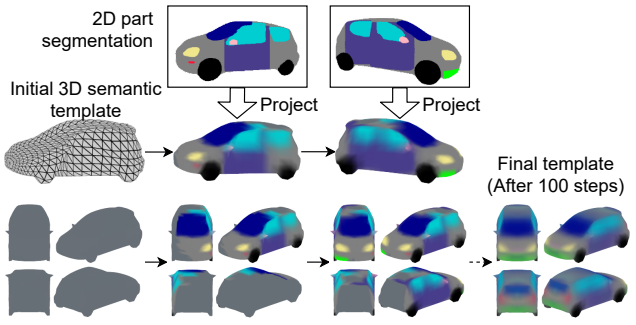

Figure 10. Semantic template inference, starting from an untextured 3D mesh template (left-to-right progression). In this figure we show a demo with two sample images, and the final result using the top 100 images as measured by the IoU.

In section sec. 3.1 we explained that we compute the semantic template using the top $N_{\text{top}} = 100$ images as mea-

sured by the IoU, among those that passed the ambiguity detection test ($v_{\text{agr}} < 0.3$). To further improve the quality of the inferred semantic templates, we found it beneficial to add an additional filter where we only select poses whose cosine distance is within 0.5 (i.e. 45 degrees) of the left/right side. Objects observed from the left/right side are intrinsically unambiguous, since there is no complementary pose that results in the same silhouette. Therefore, we favor views that are close to the left/right as opposed to the front/back or top/bottom, which are the most ambiguous views. Note that this filter is only used for the semantic template inference step.

Generative model. We train the single-category reconstruction networks (setting **A**) for 130k iterations, with a batch size of 32, and on a single GPU. The multi-category model (setting **B**) is trained for 1000 epochs, with a total batch size of 128 across 4 GPUs, using synchronized batch normalization. In both settings, we use Adam [25] (the original one, not our variant with full-matrix preconditioning) with an initial learning rate of 0.0001 which is halved at 1/4, 1/2, 3/4 of the training schedule. For the GAN, we use the same hyperparameters as [39], except in the multi-category model (setting **B**), which is trained with a batch size of 64 instead of the default 32. Furthermore, in setting **B**, and for both models (reconstruction and GAN), we equalize classes during mini-batch sampling. This is motivated by the large variability in the amount of training images, as explained in sec. 4.1, and as can also be seen in Table 3. Finally, as in [39, 20, 10, 29], we force generated meshes to be left/right symmetric.

Semantic mesh generation. In the setting where we generate a 3D mesh from a semantic layout in UV space, we modify the generator architecture of [39]. Specifically, we replace the input linear layer (the one that projects the latent code $\mathbf{z}$ onto the first 8×8 convolutional feature map) with four convolutional layers. These progressively downsample the semantic layout from 128×128 down to 8×8 (i.e. each layer has stride 2). The first layer takes as input a *one-hot* semantic map (with K semantic channels) and yields 64 output channels (128, 256, 512 in the following layers). In these 4 layers, we use Leaky ReLU activations (slope 0.2), spectral normalization, but no batch normalization. We leave the rest of the network unchanged. In this model, we also found it necessary to fine-tune the batch normalization statistics prior to evaluation, which we do by running a forward pass over the entire dataset on the *running average* model. As for the discriminator, we simply resize the semantic map as required and concatenate it to the input.

A.2. Additional results

Pose estimation. In Fig. 11, we provide more insight into the *geodesic distance* metric, which measures the cosine

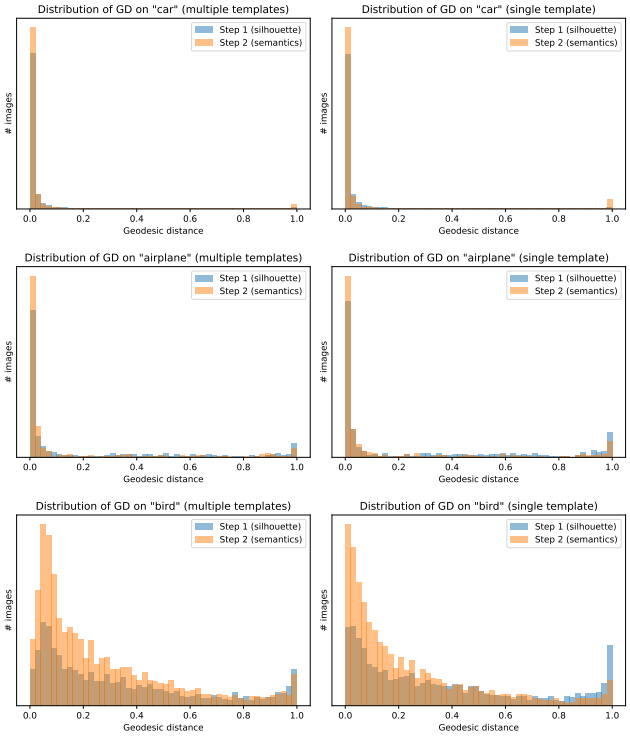

Figure 11. Distribution of pose estimation errors on *car*, *airplane*, and *bird*. We compare settings where we use multiple mesh templates (left) and a single template (right).

distance between the rotations predicted by our approach (sec. 3.1) and SfM rotations. In particular, as opposed to the results presented in Table 1 (which shows only the average), here we show the full distribution of errors. A distance of 0 means that the two rotations match exactly, whereas a distance of 1 (maximum value) means that the rotations are rotated by 180 degrees from one another. On the analyzed classes (*car*, *airplane*, and *bird*, for which we have SfM poses), we can generally observe a bimodal distribution: a majority of images where pose estimation is correct, i.e. the GD is close to zero, and a small cluster of images where the GD is close to one. This is often the case for ambiguities: for instance, in cars we sometimes observe a front/back confusion. As expected, exploiting semantics (step 2) mitigates this issue and increases the amount of available images (this is particularly visible on *bird*). We also note that, for rigid objects such as *car* and *airplane*, the distribution is more peaky, whereas for *bird* the tail of errors is longer, most likely because pose estimation is more ill-defined for articulated objects.

Qualitative results. We show extra qualitative results in Fig. 14. In particular, we render each generated mesh from two random viewpoints and showcase the associated texture and wireframe mesh. Additionally, in Fig. 12 we show the most common failure cases across categories. We can

Figure 12. Failure cases for a variety of categories.

identify some general patterns: for instance, in vehicles we sometimes observe incoherent textures (this is particularly visible in *truck* due to the small size of this dataset). On animals, as mentioned, we observe occasional failures to model facial details, merged/distorted legs, and more rarely, mesh distortions. To some extent, these issues can be mitigated by sampling from the generator using a lower truncation threshold (we use $\sigma = 1.0$ in our experiments), at the expense of sample diversity.

Semantic templates. Fig. 13 shows the full set of learned semantic templates for every category. Most results are coherent, although we observe a small number of failure cases, e.g. in *truck* one or two templates are mostly empty and are thus ineffective for properly resolving ambiguities. This generally happens when the templates have too few images assigned to them and explains why the multi-template setting does not consistently outperform the single-template setting.

Demo video. The supplementary material includes a video where we show additional qualitative results. First, we showcase samples generated by our models in setting **A** and explore the latent space of the generator. Second, we analyze the latent space of the model trained to generate multiple classes (setting **B**), and discover interpretable directions in the latent space, which can be used to control shared aspects between classes (e.g. lighting, shadows). We also interpolate between different classes while keeping the latent space fixed, and highlight that style is preserved during interpolation. Finally, we showcase a setting where we generate a mesh from a hand-drawn semantic layout in UV space, similar to Fig. 8.

Class	Synsets	Raw images	Valid instances
Motorbike	n03790512, n03791053, n04466871	4037	1351
Bus	n04146614, n02924116	2641	1190
Truck	n03345487, n03417042, n03796401	3187	1245
Car	n02814533, n02958343, n03498781, n03770085, n03770679, n03930630, n04037443, n04166281, n04285965	12819	4992
Airplane	n02690373, n02691156, n03335030, n04012084	5208	2540
Sheep	n10588074, n02411705, n02413050, n02412210	4682	864
Elephant	n02504013, n02504458	3927	1434
Zebra	n02391049, n02391234, n02391373, n02391508	5536	1753
Horse	n02381460, n02374451	2589	664
Cow	n01887787, n02402425	2949	861
Bear	n02132136, n02133161, n02131653, n02134084	6745	2688
Giraffe	n02439033	1256	349

Table 3. Synsets and summary statistics for our ImageNet data. For each category, we report the number of raw images in the dataset, and the number of extracted object instances that have passed our quality checks (size, truncation, occlusion).

A.3. Dataset information

For our experiments on ImageNet, we adopt the *synsets* specified in Table 3. Since some of our required synsets are not available in the more popular ImageNet1k, we draw all of our data from the larger ImageNet22k set.

A.4. Negative results

To guide potential future work in this area, we provide a list of ideas that we explored but did not work out.

Silhouette optimization. For the silhouette optimization step with multiple templates, before reaching our current formulation, we explored a range of alternatives. In particular, we tried to smoothly interpolate between multiple meshes by optimizing a set of interpolation weights along with the camera parameters. This yielded inconsistent results across categories, which convinced us to work with a "discrete" approach as opposed to a smooth one. We then tried a reinforcement learning approach inspired by multi-armed bandits: we initialized each camera hypothesis with a random mesh template, and used a UCB (upper confidence bound) selection algorithm to select the optimal mesh template during optimization. This led to slightly worse results than interpolation. Finally, we reached our current formulation, where we simply replicate each camera hypothesis and optimize the different mesh templates separately. We adopt pruning to make up for the increase in computation time.

Re-optimizing poses multiple times. In our current formulation, after the semantic template inference step, we use the semantic templates to resolve ambiguities, but there is no further optimization involved. Naturally, we explored the idea of repeating the silhouette optimization step using semantic information. However, we were unable to get this step to work reliably, even after attempting with multiple

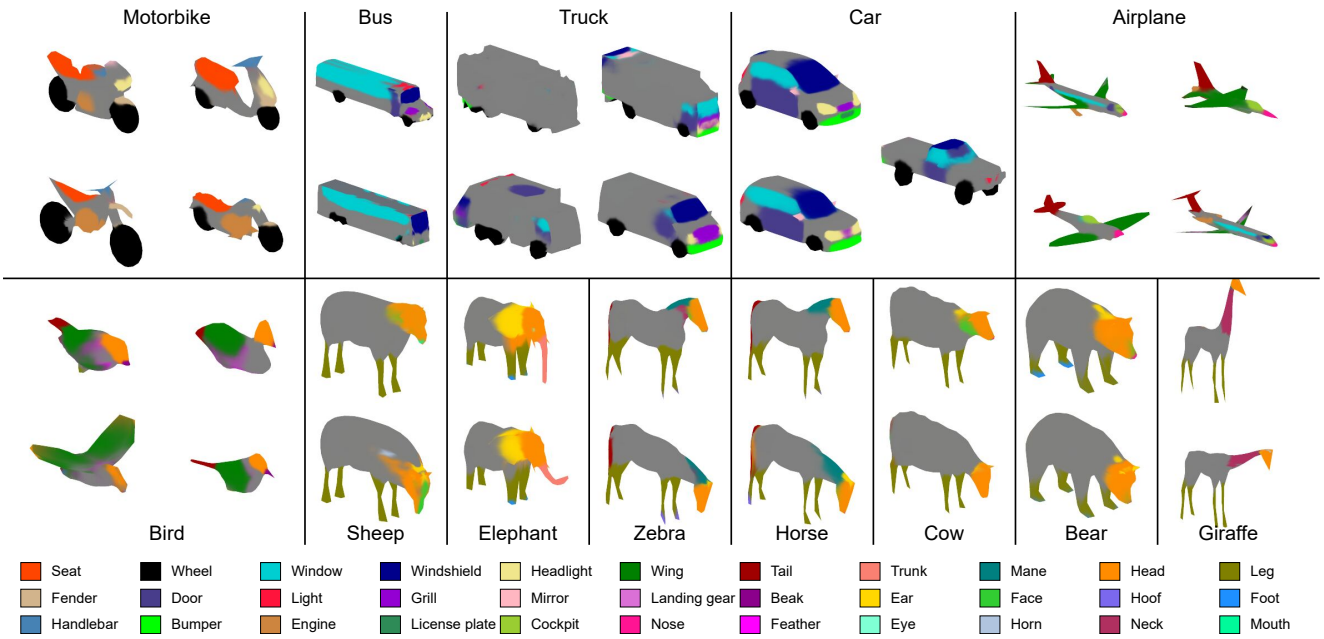

Figure 13. Visualization of *all* the learned 3D semantic templates (2–4 per category). While most results are as expected, the figure highlights some failure cases, e.g. in *truck* some templates have very few images assigned to them, which leads to incoherent semantics.

renderers (we tried both with DIB-R [4] and SoftRas [32]). We generally observed that the color gradients are too uninformative for optimizing camera poses, even after trying to balance the different components of the gradient (silhouette and color). We believe this is a fundamental issue related to the non-convexity of the loss landspace, which future work needs to address. We also tried to smooth out the rendered images prior to computing the MSE loss, without success.

Remeshing. Since target 3D vertices are known in this step, we initially tried to use a 3D chamfer loss to match the mesh template. This, however, led to artifacts and merged legs in animals, and was too sensitive to initialization. We found it more reliable to use a differentiable render with silhouette-based optimization.

Figure 14. Additional qualitative results. We show three examples per category. Each example is rendered from two random views, and the corresponding texture/wireframe mesh is also shown.