Kubric: A scalable dataset generator

Supplementary Material

A. Societal Impact and Ethical Considerations

The diversity of applications that can leverage our system merits a broad discussion on both its potential societal impact and the ethical considerations one should consider when applying it. As a general purpose tool rather than a application-targeted solution, Kubric bears the potential for diverse benefits as well as the risk of harm through negligence or even malicious misuse.

Societal Impact. Synthetic dataset construction presents system engineers with the opportunity to detect and correct potentially dangerous failure modes – particularly for those applications that involve human interaction, e.g., self-driving cars and robotics – *prior to* their deployment in the wild and with real humans. This does not, however, eliminate the possibility of developing systems that could cause serious injury to humans; instead, it raises the importance of considering the possibility of such outcomes. We therefore urge Kubric users that work towards systems deployed outside the laboratory, to take *active* measures towards mitigating such risks and consider them at the forefront of the design process. Still, the potential value of leveraging synthetic dataset construction as a tool to help guard against harmful outcomes should be further explored.

Ethical Considerations. While Kubric's dataset generation relies on human-driven design automation, it sidesteps the immediate need for human-derived data. This can help to avoid ethical problems and legal obstacles to research and can also be a powerful tool for studying and mitigating undesirable societal biases. Still, any human-in-the-loop semi-automated processes are susceptible to the biases of their designers. While a more explicitly-controlled dataset design methodology allows engineers to postpone complications surrounding the (important) privacy concerns due to the treatment of data captured from the real-world, one can reasonably argue that such a benefit is offset – at least in part – by biases introduced during the dataset synthesis process. Indeed, any explicitly-constructed synthetic dataset will be vulnerable to inheriting the biases of the processes employed when constructing it – but we argue that it promotes the discussion and (hopefully) the mitigation of biases at *both* an earlier stage of the design process *and* with a greater degree of controllability. The potential downstream impacts of distributional differences between the synthetic and real-world data would then become an important additional concern, requiring explicit evaluation and potential mitigation/treatment to safeguard against real world bias.

We also note that, while Kubric, on the one hand, provides an effective way to create new datasets, helping to avoid becoming stuck on, and over-fitting to existing data, it may also, on the other hand, enable the proliferation of datasets tailored to highlight the advantages of one's method of choice. While this is true with all dataset creation, it is hoped that through experimentation and replication, as with model architectures, the field will self-select datasets that are useful, providing fair, balanced assessment of different models on tasks of common interest.

Environmental considerations. Controllable synthetic dataset construction helps to promoting a control-based scientific methodology: e.g., where confounding factors can be explicitly isolated and tested against, and by allowing smaller problems to be constructed (i.e., where only those behaviors one seeks to validate are tested against) before scaling to larger, more general settings. This strategy can help to reduce the need for repeatedly training large-scale models on huge datasets, and thus lower the overall environmental impact of the research project. However, the ability of to generate large and controllable synthetic datasets does not come without its costs; for example, our optical flow dataset required roughly 3 CPU-years of compute-time. Hence we urge researchers to be mindful of the costs of both training and generating datasets, and to avoid generating unnecessarily large datasets. As such, the design of surrogate synthesis models capable of augmenting synthetic datasets in a more energy-efficient fashion, e.g., with latent-space dynamical models, is an important line of future research.

Synopsis. Synthetic data offers the opportunity for researchers and engineers to consider and face the impact of bias on their systems, to design around detecting dangerous failure modes, and all in a privacy-preserving setting. This certainly does not preclude the presence of such issues in any resulting final system, as the manner and degree in which these opportunities are leveraged should mandate an additional level of responsibility during the design and meta-evaluation of the synthetic datasets.

B. ShapeNet pre-processing

Extensive pre-processing was performed to simplify the integration of these assets within Kubric. These conversion scripts are available in the `shapenet2kubric` subdirectory; the conversion process can be easily reproduced by the Docker container available therein. This conversion process took ≈ 16 hours on a 80 core virtual machine (Google Cloud VM `n2-highcpu-80`) and parallelization across threads executed by Python's `multiprocessing` library.

A small set of models failed to convert (e.g., they had missing textures, erroneous materials, or simply crashed the conversion process) and are listed in the conversion code.

While many of the models within the dataset produce satisfactory renderings when visualized through OpenGL, rendering quality is significantly higher when a *photorealistic* renderer is employed (i.e., Blender Cycles). We collected the community's wisdom (i.e., ShapeNet and Blender official forums) on how to tweaks models to minimize the occurrence of visual artifacts. The conversion procedure is automated via scripted Blender modifiers and involves removing doubles, disabling auto-smoothing, splitting sharp edges, and infinitesimally displacing the faces of polygonal meshes along the primitive's local normal. For the collision geometry, we first converted the assets into watertight meshes with ManifoldPlus [41], and then resorted to the VAHCD [62] implementation wrapped within PyBullet [19] to compute the convex decomposition of a 3D object, whose mass and inertia tensors were finally estimated by trimesh [22].

C. Further datasets and challenges

C.1. Robust NeRF

Neural Radiance Fields [66] or NeRF, trains a representation of a static 3D scene via volume rendering by minimizing a photometric reconstruction loss of the form $\mathcal{L}(\theta) = \mathbb{E}_r \|I(r) - I_\theta(r)\|_2^2$, where r are rays corresponding to pixels of a multi-camera system. The nature of this loss implies that when the scene is *not perfectly static* across views, the recovered representation is corrupted; see Figure 11 (center). This challenge demonstrates that further research is still needed to fully address this problem; see Table 7. In the "teleport" challenge, while most of the scene remains rigid, we add impostor non-static object (i.e. the monkey head) randomly within the scene bounds, while in the "jitter" challenge the impostor position jitters around a fixed position. In other words, the two datasets evaluate the sensitivity of unstructured (teleport) vs. structured (jitter) outliers in the training process. Figure 10 showcases some of the training frames for each challenge. As shown in Table 7, while unstructured outliers are *to some degree* addressed by NeRF-L1 (-2.4 dB), structured outliers are significantly more challenging to overcome.

C.2. Multi-view object matting

Salient Object Detection (SOD) aims to segment out the most salient object in an image from the background. Classical methods involve active-contour [47, 69] or graph-cut [20, 104] techniques, but there also exist techniques with human-in-the-loop [72, 81, 100], and more recently deep learning variants [30, 73, 74, 105, 117]. With human feedback, interactive methods are typically robust, but also costly. Automatic segmentation, be it with traditional methods or deep networks, are less performant. Here we propose a new mode of operation in SOD, a significantly harder task (see Figure 12), yet with sufficient information for a human to solve the problem without ambiguity. We compare several single view state-of-the-art SOD algorithms on this dataset, and propose two datasets with increased complexity. Instead of one image, we assume access to *multiple*

	static	teleport	jitter
NeRF-L2	43.5 dB	25.5 dB	27.4 dB
NeRF-L1	42.8 dB	40.4 dB	37.1 dB

Table 7. **Robust NeRF –** Performance in PSNR↑ of classical NeRF-L2 [66]) and its L1-robust version [113]. Note the performance is evaluated on test views (i.e. novel view synthesis) and *without* any impostors present.

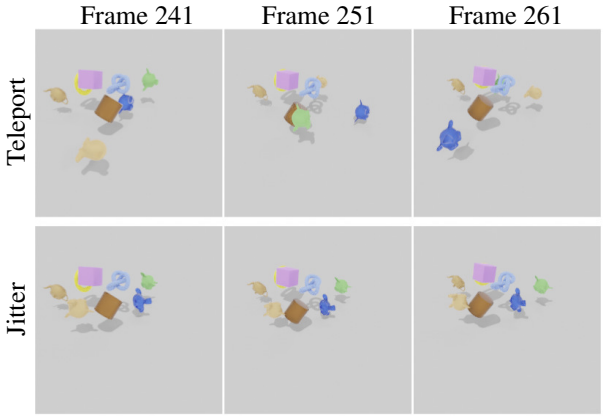


Figure 10. **Robust NeRF (training data)** – a dataset that violates the rigidity assumptions typically assumed by NeRF training workloads. Here, we qualitatively visualize the "teleport" and "jitter" versions of the training dataset.

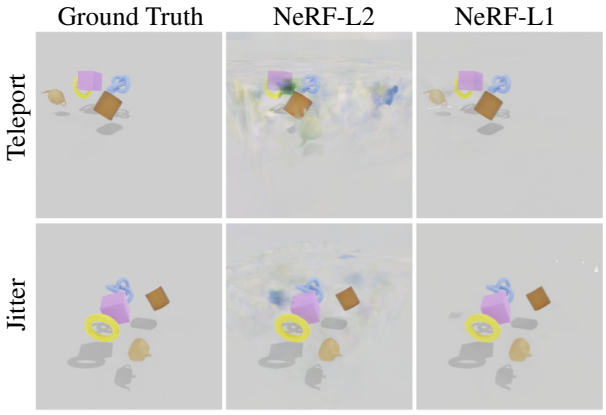


Figure 11. **Robust NeRF (training outcome)** – Visualizing the rendering of NerF-L2 vs Nerf-L1 for models trained on Teleport and Jitter training sets. During test the ground truth does not have dynamic objects. Typical NeRF-L2 models render a shadow in place of transient objects where as the NeRF-L1 model can successfully remove the floaters.

Figure 12. **Salient Object Detection** – Images, ground truth mask, as well as predictions from U^2-Net (pretrained and fine-tuned) over an example scene in the easy and hard datasets.

	easy			hard		
	maxF$_\beta$	MAE	S$_m$	maxF$_\beta$	MAE	S$_m$
Pre-trained						
SINet [30]	0.494	0.097	0.597	0.401	0.093	0.568
EGNet [117]	0.652	0.090	0.753	0.462	0.133	0.641
BASNet [74]	0.822	0.034	0.878	0.581	0.086	0.730
CPD [105]	0.811	0.029	0.872	0.594	0.068	0.742
U^2-Net [73]	0.825	0.032	0.882	0.594	0.083	0.743
Fine-tuned						
CPD [105]	0.958	0.007	0.968	0.901	0.015	0.925
U^2-Net [73]	0.975	0.006	0.977	0.909	0.015	0.931
Trained from scratch						
CPD [105]	0.957	0.008	0.967	0.906	0.015	0.928
U^2-Net [73]	0.967	0.009	0.971	0.890	0.021	0.916

Table 8. **Salient Object Detection** – Quantitative results as evaluated by max F-measure with $\beta^2 = 0.3$ [1], Mean Absolute Error [10] and Structure measure [29].

images (taken from different angles) of the same salient object. With multiple views of the same object, we theorize that automatic SOD would be more robust as the 3D structure implied from multiple images provides information that could help disambiguate boundaries of the target object. For the *easy* challenge, scenes only contain *one* salient object within the scene, while in the *hard* challenge we additionally insert clutter. All target objects are randomly selected from SHAPENETCORE V2 the background from POLYHAVEN HDRIs [115]. In the case of the *hard* challenge, clutter objects are also sampled randomly from SHAPENETCORE V2. We render 10 images for each scene,

	Lambertian		Specular	
	PSNR	SSIM	PSNR	SSIM
LFN [86]	29.71	0.883	26.77	0.816
PixelNeRF [114]	**32.11**	**0.922**	**29.96**	**0.885**

Table 9. **Complex BRDFs –** Comparison of novel view synthesis results for specular vs. diffuse materials.

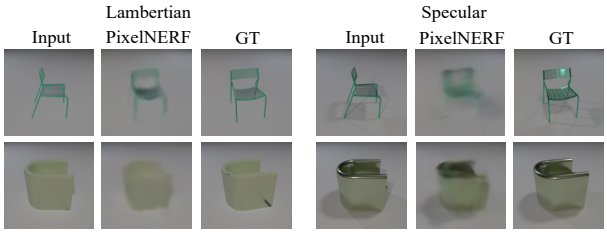


Figure 13. **Complex BRDFs –** Existing approaches struggle to model ShapeNet shapes rendered with specular materials.

and export images and segmentation masks with Kubric. The training/test sets contains $1000/200$ scenes respectively for both *easy* and *hard*.

To the best of our knowledge, multi-view SOD baselines do not exist. We therefore evaluate recent SOTA single-view SOD models as baselines. We first evaluate the pretrained models on the *easy* dataset, as summarized in Table 8. Some pretrained models (e.g. U^2-Net) performs decently. Next, we fine-tune the best performing pretrained models (U^2-Net and CPD) on the *easy* dataset. Despite U^2-Net and CPD's strong performance, however, multivew enabled SOD models should be stronger as the failure cases of single view SOD models (e.g. see Figure 12) indicate the lack of 3D understanding is often the culprit. Last but not least, we train U^2-Net and CPD on the *easy* dataset from scratch for additional baseline results; see Table 8. We repeat the experiments on the *hard* dataset. In the presence of clutter, the task becomes significantly harder. Clutter is often mistaken to be the salient object especially when the objects are in close proximity. Again, the lack of 3D understanding is an important factor for the relative poor performance of the models. Models that work with multiple views, we hypothesize, will significantly improve upon the baselines.

C.3. Complex BRDFs

Consider the core vision problem of reconstructing a 3D scene from few observations [68, 86, 87, 114]. Current datasets [48, 114] mostly feature Lambertian scenes, i.e., scenes that consist of mostly diffuse surfaces, with few specular highlights. In this case, the only relevant scene parameters are the 3D geometry, as well as the *diffuse* surface color. When scene surfaces are highly reflective, the number of scene properties required for accurate novel view synthesis grows significantly. Instead of just 3D shape and appearance, the model needs to address 3D geometry, the BRDF

of every surface point, as well as a full characterization of the light incident onto the scene. To this end, we render out a highly specular version of the ShapeNet dataset as a challenge for few-shot novel view synthesis algorithms. We follow Kato et al. [48] and render objects across 13 classes from the same 24 views. To each object, we randomly assign an RGB color. We place three light sources at randomized positions on the upper hemisphere. In this challenge, we fix the material properties of each object to the properties of the specular CLEVR [44], and ray-trace each scene with 12 ray bounces. We benchmark two recently proposed models on this dataset: Light Field Networks [86], which parameterizes a scene via its 360-degree light field, and PixelNeRF [114], a conditional 3D-structured neural scene representation. In order for these models to successfully train and perform at test-time, they need to both model the view-dependent forward model correctly, and correctly infer the position of the light sources. We find that both models perform substantially worse in Table 9 and specular compared to Lambertian shapes. In Figure 13, we illustrate how existing approaches struggle to represent inherent specularities in shapes.

C.4. Single View Reconstruction

Reconstructing an explicit 3D representation of an object (e.g. polygonal mesh) exclusively from 2D image supervision is challenging due to the ill-posed nature of the problem. Given input 2D images and their associated 3D viewpoint parameters (i.e., comprising the azimuth, distance, and elevation of the camera looking at the object), current methods (e.g. SOFTRAS [58]) combine an encoder to first extract latent features from images followed by a decoder to extract 3D vertices and face connectivity from the encoded feature vectors. Next, a differentiable renderer can project the 3D faces according to the viewpoint and all while respecting view-dependent occlusion constraints. To train such a rasterization-based differentiable rendering model, a loss function can be formulated from the difference between this projected (i.e., rendered) output and an image of the silhouette of the object against ground-truth viewpoints. One common loss function is based on a soft IoU loss between the projected and ground-truth images. Notably, this entire optimization no longer relies on any direct 3D supervision on the explicit 3D object parameterization, only viewpoint labels are needed and these can be readily determined from sourcing camera responsible for producing the 2D supervision images.

We train a SOFTRAS [58] model on the *entire* SHAPENET-COREV2 dataset instead of the commonly-used subset of SHAPENET that only has 13-category, that's typical for published work in this area [48, 58, 109]. The full SHAPENET-COREV2 consists of 55 categories with a total of approx. $51,300$ object models. We leverage Kubric's ability to automatically process these object models and project each into 24 random viewpoints, all while maintaining consistent meta information (camera pose and object category) that allows us to train SOFTRAS efficiently. We trained SoftRas on two experimental setups: "in-distribution", for which we follow the training regimen of [58], train on 80% of each category, and test and report performance on the remaining 20% of each category, and "out-of-distribution" where we train on all categories except 4 classes that we leave out for testing. They are *train, tower, washer and vessel*. Our results for "in-distribution", summarized in Figures 14 and 15, illustrate that we perform best on pillows and bowls (IoU 0.75 and 0.72), and worst on microphones and earphones (IoU 0.34). For "out-of-distribution" the results on the test classes are close to those reported in Figure 14, suggesting that SoftRas can generalize to new classes, but its limitations are on reconstructing images from classes with complex shapes like *headphones*. We observe that this processed dataset allows us to train a SOFTRAS capable of reconstructing a *wider range of objects* than in the original work of Liu et al. [58] but the performance for some classes are poor, which will hopefully inspire further research into more powerful and 2D-to-3D reconstruction methods.

C.5. Video Based Reconstruction

As discussed in Section C.4, the ill-posed nature of single-shot 3D reconstruction makes it an extremely challenging task. To better supervise surface reconstruction, the extensive availability of video data provides an attractive alternative to images. Multi-frame consistency of video sequences imposes additional constraints. However, since most 3D scenes are not static and many interesting real-world objects are not rigid, it brings up a new challenge for video-based surface reconstruction methods to be robust to deformations.

LASR [111] presents a pipeline that jointly recovers object surface mesh, articulation, and camera parameters from a monocular video without using category-specific shape templates. The method first uses off-the-shelf networks to generate a mask (silhouette) of the main object and optical flow for each frame. Then, by leveraging SOFTRAS [58], LASR jointly optimizes the object's rest shape, articulation, skinning weights, and camera parameters by minimizing the difference between the input and re-rendered color image, silhouette, and optical flow for each frame.

In this challenge, we first leverage Kubric to generate videos of both rigid (ShapeNet assets) and non-rigid [4] objects to evaluate the general performance of LASR. As shown in Figure 16, LASR fits the mesh well with input views but fails to extrapolate to unseen views. As the optical flow loss has been demonstrated to be critical and the ground truth flow can never be obtained from real data, we also evaluate how much LASR relies on the accuracy of the flow estimation.

[4] Human rigs imported from https://quaternius.com.

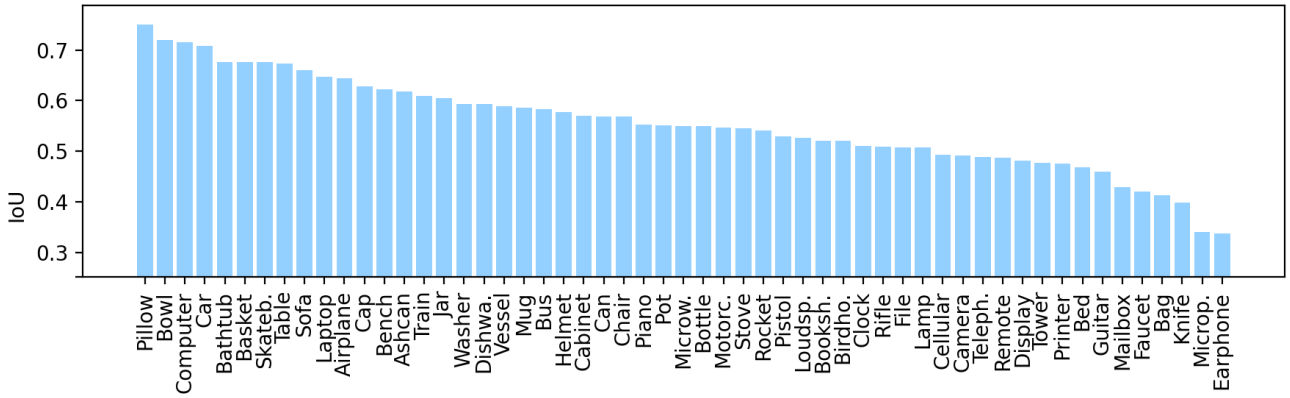


Figure 14. **Single View Reconstruction** – IoU results of training SoftRas on all ShapeNet categories. Pillows and bowls have the highest IoU and the model struggle's most with microphones and earphones.

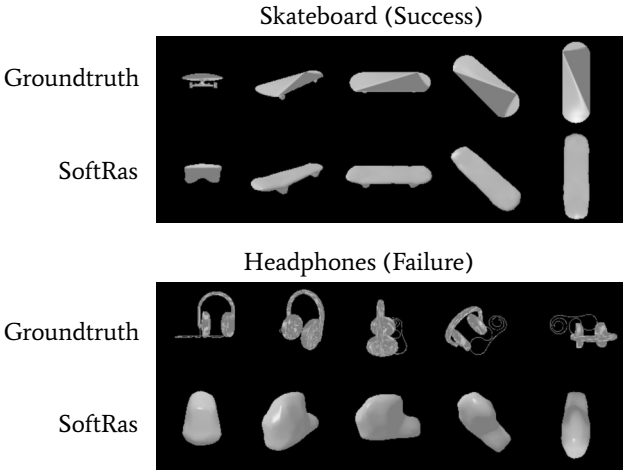


Figure 15. **Single View Reconstruction** – Qualitative results of SoftRas trained on the full ShapeNetCore V2 dataset of around 51,300 objects.

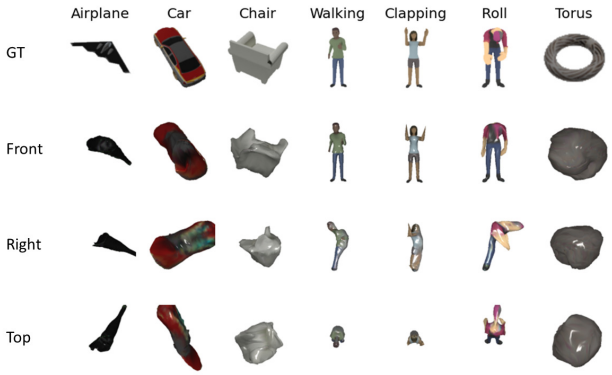


Figure 16. **Video Based Reconstruction** – Example results of 3D mesh reconstruction using LASR [111]. For each object class, we challenge LASR with a Kubric-generated 30-frame video consisting of the color image, object silhouette, and optical flow for each frame. We show the input (training) view of the object in the first row (GT). Others show the test views of the reconstructed object rendered from multiple viewing positions. Note that only the second row (Front) shares the same camera view as the input.

We separately train LASR using either the estimated optical flow or the ground truth provided by Kubric and compare the results of reconstruction; see Table 10. As expected, training with the ground truth optical flow improves performance, although this improvement is marginal. Another fundamental limitation of LASR, as with many other mesh-based differentiable renderers, is that it assumes a *fixed* mesh topology and thus cannot handle topological changes. In Figure 16, we show an example where LASR fails to reconstruct a non zero genus shape (i.e. torus), highlighting the need for additional research towards more robust approaches.

Input	airplane	car	chair	walking	clapping	roll
True flow	**0.62**	**0.22**	**1.00**	3.81	**1.30**	**1.75**
Estimated flow	1.52	0.26	1.08	**3.19**	1.34	1.85

Table 10. **Video Based Reconstruction** – Mesh reconstruction error measured by Chamfer Distance for each object with either the ground truth optical flow provided by Kubric or the estimated flow as training input.

C.6. Point Tracking

The concept of optical flow can be easily extended to longer-term tracking, following the approach proposed in Tracking Any Point [3]. That is, given a video and a target point on a surface in the scene (specified in 2D), the goal is to output the 2D locations where that point appears in other video frames. This kind of motion inference has many applications, from 3D surface reconstruction, to inference of physical properties like center of gravity and elasticity, to memory of objects across long episodes for an interactive

method	AJ	$< \delta^x_{avg}$	OA	Jac. δ^0	Jac. δ^1	Jac. δ^2	Jac. δ^3	Jac. δ^4	$< \delta^0$	$< \delta^1$	$< \delta^2$	$< \delta^3$	$< \delta^4$
Naïve	28.6	44.7	82.1	9.7	16.2	25.8	38.5	52.9	18.2	28.6	42.6	59.0	74.8
Contrastive	49.5	68.7	80.5	17.2	35.5	53.9	67.1	73.6	30.8	55.1	75.0	88.0	94.2

Table 11. **Point Tracking** – Performance of our contrastive point tracking baseline. Jac. δ^x is the Jaccard metric measuring both occlusion estimation and point accuracy, with a threshold of δ^x; AJ is the Average Jaccard across x between 0 and 4. $< \delta^x$ is the fraction of points not occluded in the ground truth for which the prediction is less than δ^x, and $< \delta^x_{avg}$ is the average across x between 0 and 4. Occlusion Accuracy is denoted with OA. We set $\delta = 2$.

method	AJ	$< \delta^x_{avg}$	OA	Jac. δ^0	Jac. δ^1	Jac. δ^2	Jac. δ^3	Jac. δ^4	$< \delta^0$	$< \delta^1$	$< \delta^2$	$< \delta^3$	$< \delta^4$
Naïve	28.6	44.7	82.1	9.7	16.2	25.8	38.5	52.9	18.2	28.6	42.6	59.0	74.8
Contrastive	45.2	63.6	80.2	12.6	28.9	48.8	64.1	71.8	23.7	47.0	69.9	85.1	92.4

Table 12. **Point Tracking** – Performance for on vertically flipped videos from the evaluation dataset. We see a roughly 4% loss in performance on the average Jaccard score, suggesting that our method somewhat overfits to the scene layout. However, the performance is far from collapsing.

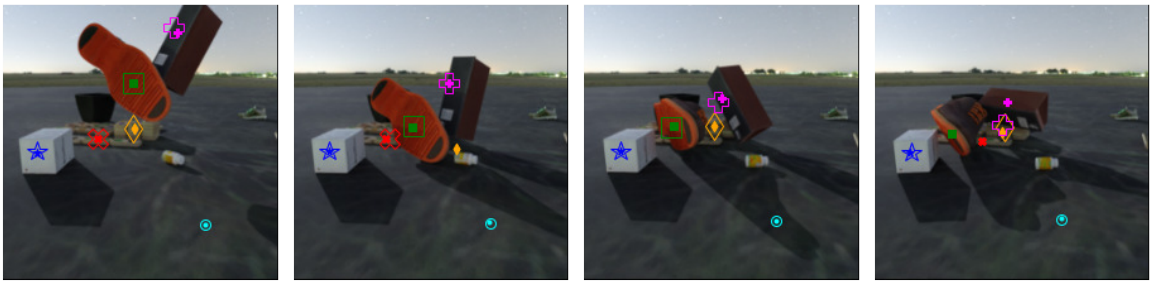
agent. Optical flow is insufficient in many of these domains because it cannot deal with occlusion, and small errors on frame pairs can lead to drift, resulting in large errors over time. It is straightforward to obtain long-term tracks from Kubric by identifying a point on a 3D object, and then projecting that point throughout the scene. Large-scale training datasets for this problem are very difficult for humans to annotate, so synthetic data can serve a critical role in achieving good performance.

Given a video, the annotations consist of a set of trajectories, i.e. a set of 2D points $(x_{i,t}, y_{i,t}) \in \mathcal{R}^2$ where i indexes the different tracked points, and t indexes time. The ground truth also includes $v_{i,t} \in \{0, 1\}$, which indicates whether a point i is visible in frame t is visible ($v_{i,t} = 1$) or not ($v_{i,t} = 0$). The tracking algorithm receives one visible point (x^*_i, y^*_i, t^*_i) for each trajectory, and must output an estimate $(\hat{x}_{i,t}, \hat{y}_{i,t})$ for the other frames, as well as a visibility prediction $\hat{v}_{i,t} \in \{0, 1\}$.

Metrics. We use three metrics proposed for Tracking Any Point [3]. The first ignores the output positions and evaluates occlusion estimation alone, via a simple classification accuracy which gives equal weight to each point on each frame. The second metric evaluates only tracking accuracy. Frames marked as occluded in the ground truth are ignored. For the rest, we report a PCK-style [2] accuracy across several different thresholds. That is, for a given threshold α, which is a distance in pixels, we consider a point correct if $\sqrt{(x_{i,t} - \hat{x}_{i,t})^2 + (\hat{y}_{i,t} - y_{i,t})^2} < \alpha$. Our final metric combines both classification accuracy and detection accuracy, and is inspired by the Jaccard-style metrics from the object tracking literature [60]. Let TP_α be the set of true positives: that is, all visible ground-truth points $(x_{i,t}, y_{i,t})$ for which $(\hat{x}_{i,t}, \hat{y}_{i,t})$ is predicted as unoccluded, and the spatial prediction is within a distance of α. Let FN_α be false negatives: visible ground truth points which are predicted to be occluded, or for which the predicted spatial position is farther than α from the ground truth. Let FP_α be false positives: points $(\hat{x}_{i,t}, \hat{y}_{i,t})$ which are predicted to be visible, where the ground truth is farther than distance α or where the ground truth is marked as occluded. The Jaccard metric is then $|TP_\alpha|/(|TP_\alpha| + |FN_\alpha| + |FP_\alpha|)$. In practice, we compute these metrics across 5 different thresholds of the form $\alpha = \delta^x$ pixels, for $x \in \{0, 1, 2, 3, 4\}$ and $\delta = 2$. We compute an average across thresholds to get an overall metric.

Baselines. We next define a baseline method that can begin to solve the point tracking problem. One of the closest problems in the literature is segment tracking, using datasets like DAVIS [70]. Therefore, our baseline is inspired by VFS [108], a state-of-the-art method for DAVIS. VFS has two key components: first, a self-supervised training phase where the aim is to learn a good similarity metric between points across images, and second, a test-time tracking algorithm based on earlier work [101] which can associate points in unlabeled frames with the points in labeled frames. Our model, however, is modified to deal with points rather than segments, and to leverage the labeled training data we have available. For pre-training, we adopt a contrastive approach [39]. We use a standard ResNet-50 [38] as a backbone, up to the final convolution, and with stride $= 1$ for the final two blocks, which gives us a feature grid F_i (which is $L2$-normalized over channel axis) for each frame, at stride 8. Given a query point, (x^*, y^*, t^*) (note: we drop the i index for clarity as we consider a single point), we first extract a feature f^* for that point, via bilinear interpolation at position $(x^*/8, y^*/8)$ from the feature grid for frame t^*. We then compute the following contrastive loss function:

$$\sum_{f_j \in F_i} \gamma_j \log \left(\frac{\exp(f_i^* \cdot f_j)/\tau}{\sum_k \exp(f_i^* \cdot f_k/\tau)} \right) \quad (1)$$



frame 3 frame 7 frame 11 frame 15

Figure 17. **Point Tracking** – Visualization of points tracked using our contrastive learning algorithm. We show each of 6 tracked points with a different symbol and color, where the smaller, filled symbol is the prediction, while the larger, unfilled symbol is the ground truth. If the ground truth is occluded or the point is predicted to be occluded, the corresponding symbol is not shown. We show only frames 3, 7, 11, and 15 out of a full 24-frame sequence for brevity. Note that the query frame is not necessarily at the beginning. The query was in frame 3 for the cyan circle, the green square, and the red X; frame 7 for the blue star and the magenta plus, and frame 11 for the orange diamond.

where j and k index over the spatio-temporal dimensions of F. The temperature hyperparameter τ is set to 0.1. γ_j is the source of supervision: its value is high if f_j is in correspondence with f^*, and it is 0 if they aren't. Note that if there is exactly one other point considered to be "in correspondence" with f^*, then the sum over k has a single term, and we are left with a standard contrastive loss. However, in our case, we have multiple potential correspondences, and this loss encourages all of them to have roughly equally high dot products.

We compute γ_j via bilinear interpolation. Say that the feature f_j is on frame t at position $\hat{x}, \hat{y}$ within the convolutional feature grid, and the ground truth position is at $(x_t/8, y_t/8)$ (in the coordinate frame of the feature grid). If $(x_t/8, y_t/8)$ lies within the grid cell which has one of its corners at $\tilde{x_j}, \tilde{y_j}$, then we set $\gamma_j = (1 - |\tilde{x_k} - x_t/8|) * (1 - |\tilde{y_k} - y_t/8|)$. Otherwise, it is 0. γ_j is also set to 0 if the ground truth is marked as occluded for frame t.

At test time, given a query point (x^*, y^*, t^*), we begin by computing a correspondence to every frame. For a single frame at time t, this is done via a dot product between f^* (again extracted via bilinear interpolation) and each feature in frame t, followed by a softmax S across space, which gives us a heatmap of likely locations, i.e., S_{xy} is the probability that x, y corresponds to (x^*, y^*, t^*). We then compute $\tilde{S}$ by finding the argmax of S and zeroing out any grid cells further than 40 pixels away (5 grid units) from it, to suppress potential multimodal correspondences. Then we compute the weighted average of the potential locations $[\hat{x}, \hat{y}] = \sum_{x,y}[x, y] * \tilde{S}_{xy} / \sum_{x,y} \tilde{S}_{xy}$.

In order to classify whether the point is occluded, we use cycle consistency [99, 101]. That is, we extract a new feature $\hat{f}$ from point $[\hat{x}, \hat{y}]$ in frame t, and reverse the process, computing a softmax over locations in frame t^* and converting it into a point correspondence. If the estimated point is further than 48 pixels of its starting location, we consider it occluded.

We evaluate this procedure on MOVi-E at a resolution of 256×256. For each evaluation video, we sample 256 query points randomly across all frames. We attempt to sample an equal number of points from each object as well as the background, but cap the number of samples per object at a maximum of 0.2% of the visible pixels. We use standard data augmentations, including a random crop of the image as small as 30% of the image area and an aspect ratio between 2:1 and 1:2, and the same color augmentations and pixel noise that was used for our optical flow experiments.

Results are shown in Table 11. For comparison, we also include a naïve baseline which assumes no motion and no occlusion, which is the best that can be done without reference to the pixels. We see that the contrastive approach is fairly good at coarse tracking, reducing the error rate on the largest threshold from 25.2% down to just 5.8%, a reduction of more than 6 times relative to the naïve baseline. However, for more precise tracking, the reduction in error is not nearly as great. On the other hand, accuracy at detecting occlusions is quite poor for this method; for the threshold we used for cycle consistency (48), the accuracy is actually worse than chance. However, points for which the cycle consistency check failed are actually unlikely to be within any distance threshold; therefore, we find that removing these points from the output improves the average Jaccard metric.

Because the network is trained and evaluated on data from the same distribution, there is a possibility that the algorithm is memorizing some aspects of the training data, such as the common trajectories followed by objects. To evaluate out-of-distribution transfer, we also applied our algorithm to vertically-flipped videos, and show the results in Table 12. This harms performance by about 4%, suggesting that the network is memorizing trajectories to a small extent.

Finally, Figure 17 shows a qualitative example of our point tracking algorithm on a kubric validation video. We

		KLEVR	ToyBox5	ToyBox13
2D	DeepLab [15]	97.1%	81.6%	63.1%
	NeSF [97]	92.7%	81.9%	56.5%
3D	SparseConvNet [35]	99.7%	93.4%	83.2%
	NeSF [97]	97.8%	88.7%	60.1%

Table 13. **Scene Semantic Segmentation** – We compare mean intersection-over-union in 2D image segmentation (top) and 3D point cloud segmentation (bottom) on the three datasets.

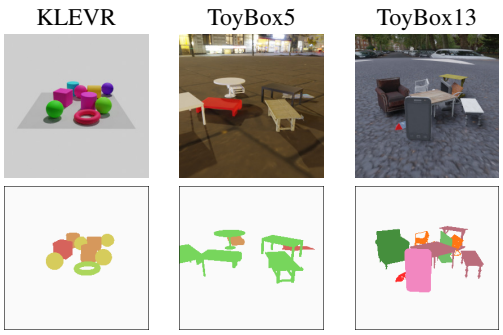


Figure 18. **Scene Semantic Segmentation** – Example RGB and segmentation renders from the datasets.

see that for easy points with relatively little motion, like the blue star or cyan circle, the algorithm is quite accurate, even when there is relatively little texture. Tracking can also be good for the points that have reasonably distinctive texture, like the green square. Occlusions are also sometimes detected correctly: for the red X, the algorithm correctly determines the occlusion on frame 11, although it prematurely finds the point again on frame 15. However, there are also obvious failures: for instance, the algorithm loses the magenta plus, likely because the object has rotated so much that the appearance has changed substantially. This suggests that it may be useful to employ global reasoning about the orientation of objects rather than relying on appearance alone. We also see a very large error for the diamond on frame 7, when the point is occluded; the algorithm instead places the point between the occluder objects, possibly because the feature descriptors are relying too heavily on context. This result suggests that simultaneously capturing global object motion while remaining robust to occlusion will be a substantial challenge in this dataset moving forward.

C.7. Scene Semantic Segmentation

As another use case, we consider the task of comparing 2D and 3D semantic segmentation models. As these methods operate on fundamentally different substrates, it is challenging to quantify the effectiveness of one method versus another. To this end, we construct three synthetic datasets where 2D image and 3D point cloud are in direct correspondence: KLEVR, ToyBox5 and ToyBox13. Each of the ToyBox datasets each consists of 525 scenes. Each scene contains 4-12 upright ShapeNet objects on a flat surface with one of 382 randomly chosen HDRI backdrops. The datasets differ only in the set of ShapeNet objects employed. In ToyBox5, we use the top 5 most common object categories; in ToyBox13, the 13 most common object categories. For each scene, we select 300 camera poses and render 3 images per pose: an RGB map, a segmentation map, and a depth map. With knowledge of camera parameters, we are able to construct a camera ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ corresponding to each pixel in the dataset. When combined with depth and segmentation maps, we obtain a labeled 3D point cloud where each 3D point corresponds to one camera pixel. We define a scene's point cloud to be the union of all 3D points constructed from all camera poses. The KLEVR dataset is constructed identically to the ToyBox datasets except with a fixed, neutral-grey backdrop and 5 platonic object shapes.

Experiments. We demonstrate two representative baselines for 2D image and 3D point cloud segmentation: DeepLab [15] and SparseConvNet [35], respectively. In addition, we compare these methods with NeSF [97], a method for dense 2D and 3D scene segmentation from posed RGB images. We train all methods with semantic supervision derived from 9 cameras per scene from 500 scenes and hold out 4 cameras per scene from the remaining 25 scenes for evaluation. In 2D, we see that NeSF and DeepLab perform similarly, with DeepLab outperforming NeSF by 0.3% to 6.6% (Table 13). In qualitative results, we find that DeepLab's predictions more tightly outline objects at the expense of multiview consistency. In 3D, we see that SparseConvNet achieves between 1.9% and 23.1% higher mean intersection-over-union than NeSF with larger margins as dataset complexity increases. We attribute this to the SparseConvNet's access to ground truth 3D geometry and supervision in the form of a dense point cloud. This results in an exceedingly dense and accurate representation of 3D geometry. NeSF, on the other hand, must infer 3D geometry and semantics from posed 2D images alone. Further results and comparison to NeSF are presented in [97].

C.8. Conditional Novel View Synthesis

Neural scene representations such as [66] have led to state-of-the-art results in novel-view synthesis tasks on real-world datasets [63], however, a new model must be trained on each new scene, which does not allow the model to learn a *prior* over the dataset. Prior works commonly use the ShapeNet dataset [14] to evaluate how well a method can generalize to novel scenes [86, 114]. However, ShapeNet consists of *canonically* oriented objects with thousands of same-class examples for some categories (*e.g.*, airplanes) rendered with flat shading. We introduce a new large-scale dataset using Kubric to generate photo-realistic scenes with

	PSNR	SSIM	LPIPS
LFN [86]	14.77	0.328	0.582
PixelNeRF [114]	21.97	0.689	0.332
SRT [82]	23.41	0.697	0.369

Table 14. **Conditional Novel View Synthesis** – Quantitative evaluation for novel view synthesis.

Figure 19. **Conditional Novel View Synthesis** – The scenes are very difficult as they contain a large number of objects in random poses, contain realistic backgrounds, and are rendered with ray-tracing. LFN fails to capture these datasets in a global latent, severely under-fitting. PixelNeRF shows better quality renders, which however degrade for out-of-training distribution target views(*e.g.*, bottom row).

groups of ShapeNet objects. Each of the 1M scenes use one of 382 randomly chosen background maps. We render ten 128×128 random views for each scene and use five as *conditioning views*, and the other five as *target views*. The task is to reconstruct the target views *given* the conditioning views. We compare the following recent methods: PixelNeRF [114], which projects points into the conditioning views to interpolate encoded image features, LFN [86], which condenses the scene into a single latent code, and decodes it into an implicit scene representation using a hypernetwork that produces the weights of the scene-specific MLP, and SRT [82], which learns a scalable set-latent scene representation through a transformer encoder-decoder architecture. Fig. 19 compares these methods on our new challenge. While all methods above produce fairly high-quality results on ShapeNet (see [82]), they scale vastly differently to our photorealistic, complex dataset: while PixelNeRF has apparent difficulties with views that are far away from the conditioning views, LFN does not scale at all to this complexity, and SRT's reconstructions suffer from blurriness. Further details on this challenge and the dataset release are presented in [82].