

SIMstack: A Generative Shape and Instance Model for Unordered Object Stacks

Supplementary Materials

Zoe Landgraf¹, Raluca Scona¹, Tristan Laidlow¹, Stephen James¹
Stefan Leutenegger² and Andrew J. Davison¹ *^{†‡}

Abstract

In this **supplementary document**, we provide the following additional content: (1) complementary graphs for our single-view evaluation and additional qualitative results comparing SIMstack to our baselines, (2) a more detailed description of our multi-view optimisation pipeline, (3) a detailed description of our network architectures and (4) some additional qualitative examples showing results on non-convex object scenes and a latent code analysis.

1. Single-view evaluation

In addition to our results reported in the main paper, we show the corresponding graphs displaying results per surface visibility ratio for predicted voxel occupancy and stability under physics simulation (Figures 1 and 2). We also provide an additional evaluation of stability; in the case of bad predictions (unstable decompositions), rolling objects as well as intersecting objects, pushed apart by contact force can cause large object centre displacements. To provide a more intuitive notion of stability, we estimate the percentage of stable piles according to a stability threshold, which we determine through observation. To allow some shuffling of objects, but exclude falling objects, we set the following threshold: we define a stack to be stable if none of its composing objects' centres move by more than 20cm and all objects' orientation change stays within 30°. We plot the percentage of stable stacks by surface visibility on our test datasets in Figure 3.

We provide additional qualitative results on our Superquadric (SQ) shape test dataset and our YCB object test set, comparing SIMstack to our baselines (see Figure 4).

*Research presented in this paper has been supported by Dyson Technology Ltd.

^{†1}Zoe Landgraf, Raluca Scona, Tristan Laidlow, Stephen James and Andrew J. Davison are with the Dyson Robotics Laboratory, Department of Computing, Imperial College London, UK. zoe.landgraf15@imperial.ac.uk

^{‡2}Stefan Leutenegger is with the Smart Robotics Lab, Department of Computing, Imperial College London, UK.

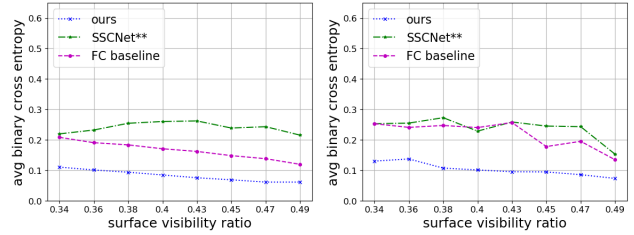


Figure 1: We compare our C-VAE against *SSCNet*** and our fully convolutional (FC) baseline for predicted (expected) voxel occupancy. **Left**: test dataset of SuperQuadric shapes **Right**: our YCB object test dataset.

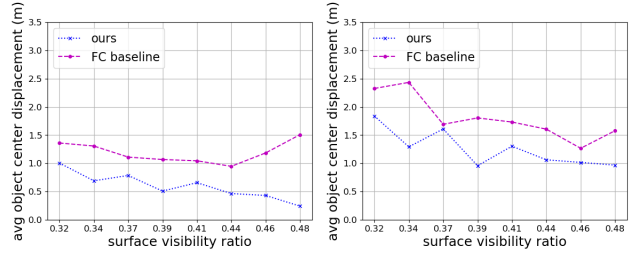


Figure 2: We compare SIMstack and our fully convolutional (FC) baseline in terms of stability under physics simulation (10000 steps) by computing the average object displacement per object stack (m). **Left**: test dataset of SuperQuadric shapes **Right**: our YCB object test dataset.

2. Multi-view optimisation

As shown in the main paper, SIMstack can integrate additional views to improve reconstruction and instance segmentation using *multi-view conditioning* and *multi-view optimisation*. In this section we provide more details on our multi-view optimisation pipeline.

Differentiable Depth Renderer Our differentiable depth renderer is based on SDF ray casting. Similarly to [1], we use sphere tracing to render depth. While stepping along each ray, we compute the exact SDF value using trilinear interpolation. Jiang *et al.* [1] only backpropagate the

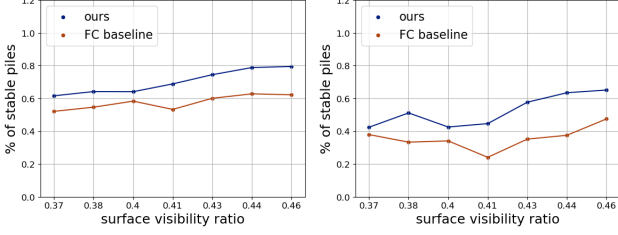


Figure 3: Percentage of stable piles generated according to our stability threshold (all objects center displacement stays within 20cm and all objects orientation change stays within 30°). **Left:** test dataset of SuperQuadric shapes **Right:** our YCB object test dataset.

gradients into the immediate neighbourhood of each ray-surface intersection, which is correct when using a depth-image based loss. Since we use a more precise SDF based loss (Equation 1), we need gradients in the entire field of view of the camera. Although there has been recent work on fully differentiable sphere tracing [2], we choose a simpler approximation, sampling SDF values at regular intervals along every ray outside the surface to backpropagate gradients within the entire camera frustum. We estimate our gradients using the binary loss described by Equation (2). Parallelised, our method can render the full cost image at an average runtime of $0.192s$ at a resolution of 640×480 . In comparison, [2] render an image of 512×512 in $0.99s$.

Cost Function Although related approaches use a depth loss to optimise their latents [1, 3], we observe that for our TSDF representation using a depth-based loss leads to an incorrect definition at occlusion boundaries. We design an SDF-based cost function composed of two parts: one describing the loss at the visible surface and one for the visible, unoccupied region of the scene. Let π^{-1} be the function which backprojects a pixel \mathbf{u}_i of the depth image into the 3D scene and I is the trilinear interpolation function which obtains the TSDF value at that point. The surface loss is:

$$L_{\text{surface}} = \sum_{\Omega} I(\pi^{-1}(\mathbf{u}_i)). \quad (1)$$

The current TSDF is optimised towards alignment with this surface data, but this loss doesn't constrain on visible regions of empty space. We therefore define an empty space loss which penalises the code if it produces a negative TSDF value in observed empty space. We sample at regular intervals along rays in all regions of observed empty space and define the empty space loss, which has a 'space carving' effect: $L_{\text{empty_space}} = \sum_s L_{\text{empty_space},s}$, where:

$$L_{\text{empty_space},s} = \begin{cases} |I(s)| & \text{if } I(s) < 0 \\ 0 & \text{if } I(s) > 0 \end{cases}. \quad (2)$$

Here s is a sampled TSDF value. Our final cost function is the simple sum of both losses:

$$L = L_{\text{surface}} + L_{\text{empty_space}}. \quad (3)$$

Optimisation (implementation details) We use first-order optimisation to optimise our latent code against additional depth images: Once the loss L is computed, we backpropagate the gradients into the voxel-grid using inverse raytracing and trilinear interpolation, parallelised on the GPU. For multiple depth images, we accumulate the gradients of all views. We then leverage PyTorch autograd to backpropagate the gradients through the generative decoder of our C-VAE and use Adam to generate gradient updates.

Runtime Our multi-view conditioning method's runtime only depends on the time to generate a partial TSDF ($5.2s$ for 6 views using our TSDF Fusion method) from multiple views as the forward pass time stays constant. It clearly outperforms our multi-view optimisation method which takes $21s$ and $75s$ to optimise against 1 and 6 views respectively, for 30 iterations.

3. Network Architecture Details

C-VAE We provide a detailed overview of our C-VAE in Figure 5. **Conditioning AutoEncoder** Our conditioning network is trained as an autoencoder, encoding and decoding the partial TSDF generated from the input depth view. Encoder and Decoder each have 5 convolutional layers with 2 linear layers compressing the feature maps into a 1D bottleneck of 96. The encoder feature maps are used to condition the encoder and decoder of the shape and instance VAE. **Shape and Instance VAE** Our VAE's encoder maps input TSDF and instances to a common feature space using two convolutional layers for each modality. The resulting feature maps are concatenated and further compressed using 5 joint encoding layers. One linear layer maps our 3D feature space to our 1D latent code of size 96 while 2 linear layers map it back to 3D. Our VAE 3D decoder mirrors our encoder. We use Batch Normalisation (BN) and PRelu activations in all of our hidden layers.

SSCNet Baseline We provide a detailed overview of our adapted version of SSCNet in Figure 6. To fit this baseline to our task which requires same-resolution output and TSDF prediction, we adapt the SSCNet architecture by (1) remove downsampling in most layers by adding padding (2) adding an upsampling (deconv) layer at the end to generate a prediction at the same resolution as the input and (3) predicting TSDF values instead of semantic labels.

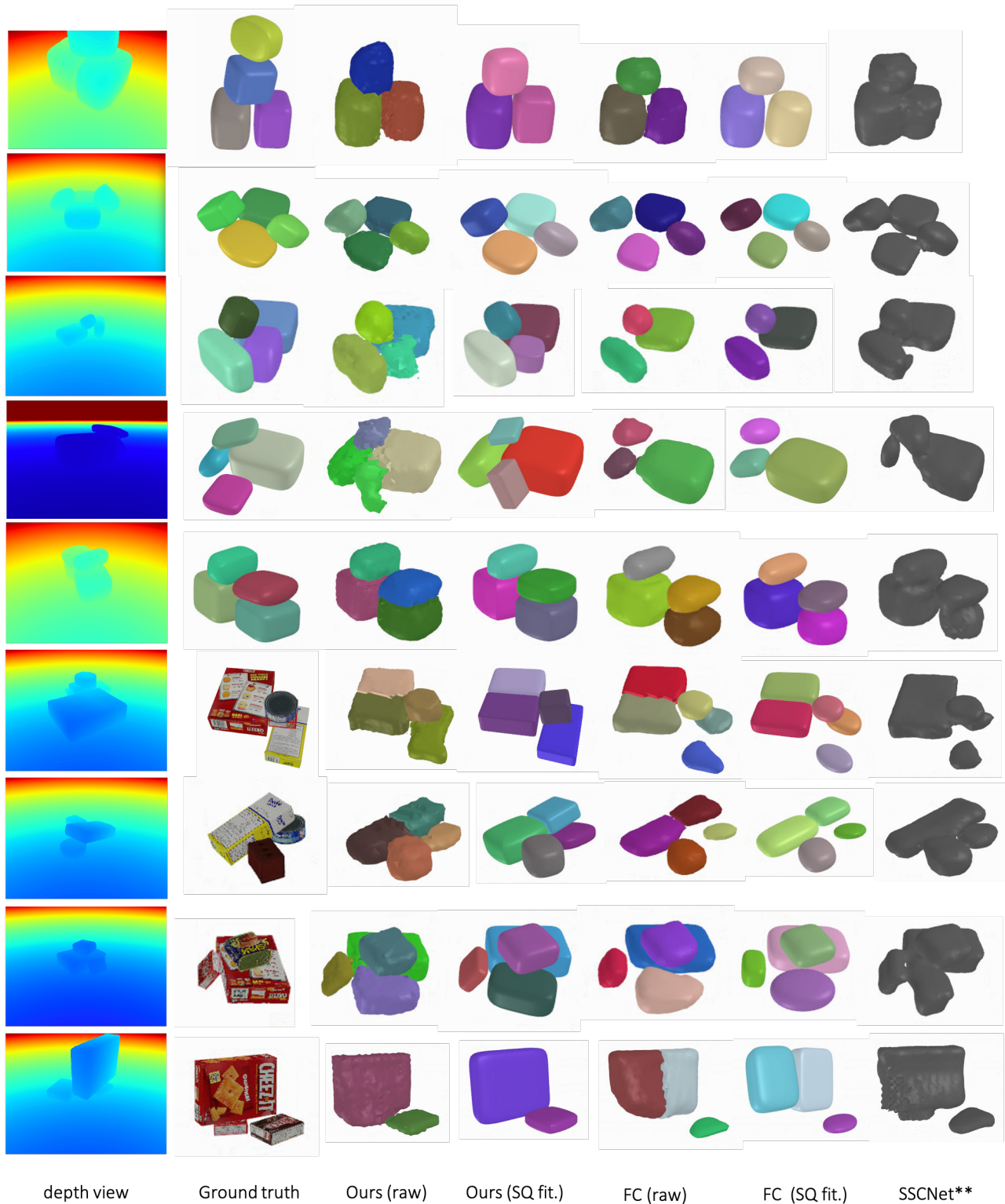


Figure 4: Qualitative results comparing SIMstack (Ours) to our fully convolutional (FC) baseline and our SSCNet baseline (*SSCNet***). We select a random viewpoint for each scene and display a random latent code sample for SIMstack.

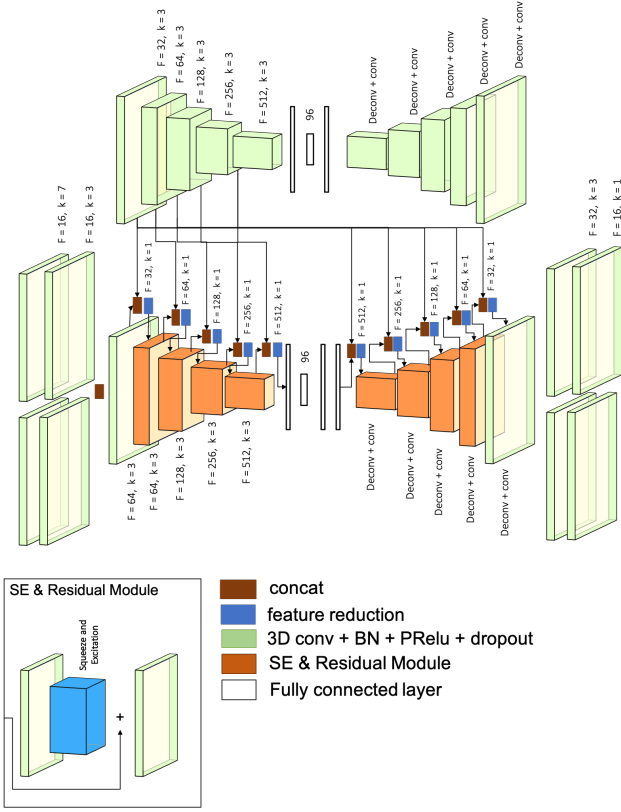


Figure 5: C-VAE architecture

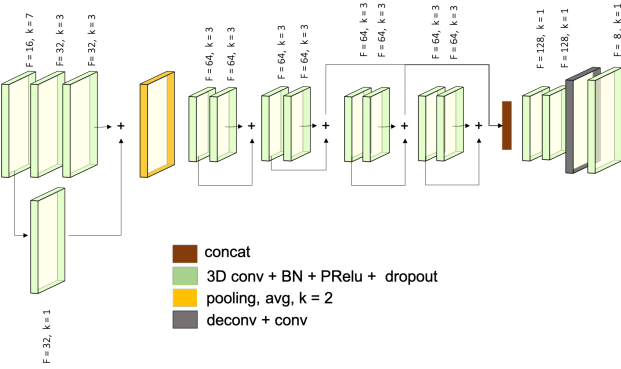


Figure 6: *SSCNet*** architecture

FC Baseline Figure 7 shows our fully convolutional (FC) baseline architecture. We use a partial TSDF encoder branch with the same SE and Residual units used in our C-VAE, which splits into two task specific decoder branches.

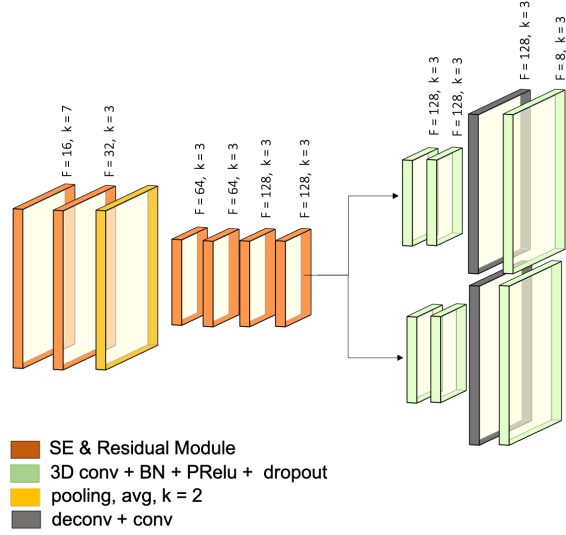


Figure 7: FC baseline architecture

4. Additional Experiments

4.1. Decomposing more complex scenes

We test our method on real scenes with a collection of non-convex objects and find that although trained on convex shapes only, our approach (excluding parametric fitting) shows some ability to generalise to such scenes. Our reconstructions from different viewpoints for each scene show that although instance decomposition varies across viewpoints and in some cases instances are oversegmented, our C-VAE is able to distinguish individual instances and generate a rough reconstruction of cups, bottles and even part of a drill (see Figure 8). This suggests that our method could be extended to more complex scenes with non-convex objects; steps to achieve this could include augmenting the dataset with non-convex shapes and using a different shape refinement step.

4.2. Latent Code Analysis

In this final section, we provide additional experiments which demonstrate the smoothness and consistency of our joint shape and instance encoding.

Sampling from the latent space Given the design of our C-VAE, sampling from its latent space generates different proposals for occluded regions, while reconstruction of the visible surface stays constant. We show example of this on our Superquadrics test dataset in Figure 9. We show the mean scene (zero code scene) along with random latent code samples for a random viewpoint.



Figure 8: Qualitative results on real scenes with non-convex objects. The raw mesh segmentation of SIMstack is able to estimate shape and decomposition of scenes with non-convex objects.

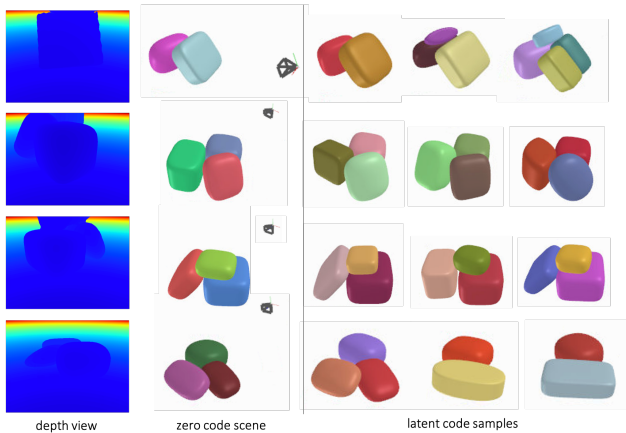


Figure 9: Sampling from the latent code of our C-VAE. Given a single depth image of our SQ test dataset (left), we generate the zero code scene and three latent code samples from our C-VAE.

Sampling with multi-view information We condition our VAE on depth information to improve reconstruction in visible regions and to allow the latents to focus on occluded regions; sampling from those latents generates a variety of propositions for shape and instance segmentation of those hidden regions. Adding additional views increases the information about 3D space and should show a decreasing variety in latent space samples. We demonstrate this on an example from our Superquadrics test dataset in Figure 10.



Figure 10: How sampling variety changes as views are added. **Left:** Ground truth and visible mesh area overlay (red). **Right:** 3 random latent space samples (without SQ fitting) for each view; as data is added the samples are increasingly constrained.

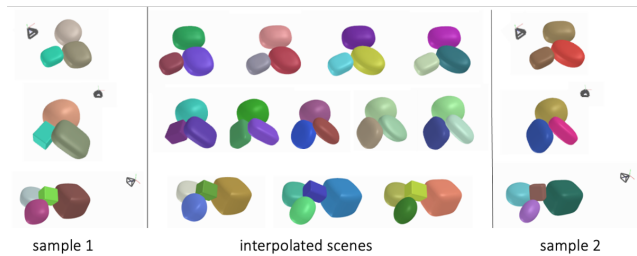


Figure 11: Latent code interpolations between two random latent code samples, conditioned on one view. We show three examples of interpolating between scenes with the *same number of instances*.

Latent code interpolation We qualitatively evaluate the smoothness of our latent code by interpolating between random latent code samples of a depth-conditioned 3D reconstruction. Given a test scene and one viewpoint, we interpolate between two latent code samples to generate intermediary scenes. Our interpolations show a visibly smooth transition between scenes with the same number of instances (see Figure 11) as well as realistic intermediary scenes for interpolations between scenes of varying numbers of instances (see Figure 12): in the first example, the small object on the

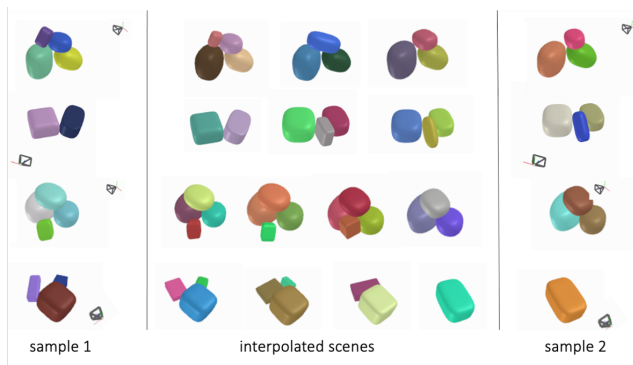


Figure 12: Latent code interpolations between two random latent code samples, conditioned on one view. We show four examples of interpolating between scenes of *varying numbers of instances*.

top left present in sample 1 becomes smaller, then merges into a long object which then shortens as interpolation approaches sample 2.

References

- [1] Yue Jiang, Dantong Ji, Zhizhong Han, and Matthias Zwicker. Sdfdiff: Differentiable rendering of signed distance fields for 3d shape optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 1, 2
- [2] Shaohui Liu, Yinda Zhang, Songyou Peng, Boxin Shi, Marc Pollefeys, and Zhaopeng Cui. Dist: Rendering deep implicit signed distance function with differentiable sphere tracing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020. 2
- [3] Edgar Sucar, Kentaro Wada, and Andrew J. Davison. NodeSLAM: Neural object descriptors for multi-view shape reconstruction. In *Proceedings of the International Conference on 3D Vision (3DV)*, 2016. 2