Imagining Hidden Supporting Objects using Volumetric Conditional GANs and Differentiable Stability Scores

Supplementary Material

Hector Basevi
h.r.a.basevi@bham.ac.uk
Aleš Leonardis
a.leonardis@bham.ac.uk

School of Computer Science
University of Birmingham
Birmingham, United Kingdom

1 Contents

This supplementary material document contains additional information about this work. It contains:

Section 2	Additional details on the parsing algorithm used to convert the voxel representation into a set of object poses.
Section 3	Additional details on the form of the voxel representations used in this work.
Section 4	Additional details on the network architectures used in this work.
Section 5	Additional details on the dataset created and used in this work.
Section 6	Additional details on the software and hardware used in this work.
Section 7	Larger and additional examples of imagined scenes.
Section 8	Larger and additional examples of varying the latent representation.
Section 9	Larger and additional examples of varying the stability context.

This document is accompanied by a video file `supplementaryVideo.mp4` which visualises the stability of imagined systems through physics simulation. Please see this video for examples of the physical behaviour of imagined scenes and for a general summary.

© 2022. The copyright of this document resides with its authors.
It may be distributed unchanged freely in print or electronic forms.

2 Parsing algorithm

Our voxel parsing algorithm (see alg. 1) converts the voxel representation produced by the generator into a set of discrete object poses defined in terms of object class, and translations and rotations on the canonical YCB [2] mesh representations. These transform the object mesh representation from a canonical coordinate system into a world coordinate system.

Data: Voxel representation V
Data: Canonical object point sample set C
Result: Object pose set O
foreach *Class voxel representation* V_i *in* V **do**
  Extract boundary voxel locations P_i from V_i
  Extract connected segments S_{P_i} from P_i
  foreach *Connected segment* $s_{P_i,j}$ *in* S_{P_i} **do**
    while *Size of* $s_{P_i,j} > \frac{1}{16}$ *size of object* c_i **do**
      Fit object c_i to component $s_{P_i,j}$ via ICP to produce object pose $o_{i,j}$
      Add $o_{i,j}$ to O
      Remove points in $s_{P_i,j}$ which agree with $o_{i,j}$
    end
  end
end

Algorithm 1: Algorithm for parsing semantic voxel representation into a set of voxel poses

3 Representations

The input voxel representations used in this work have dimensions of $64 \times 64 \times 32$ voxels. The channel structure consists of 1 channel per object class (14 in total), 1 to encode the shape of the table, and 1 channel to encode the visibility of the voxel (i.e. whether the voxel belongs to the visible surfaces and free space, or lies *behind* the visible surfaces). When utilising a latent vector, we augment the latent representation with 32 additional channels for the latent representation. The latent vector is identical for all voxels. Finally, 1 additional channel is added if using stability context to hold the stability score.

The output voxel representations have dimensions of $64 \times 64 \times 32$ voxels, and the channel structure consists of 1 channel per object class including the table (15 in total).

4 Network architectures

4.1 Generator

The generator has a U-Net [10] design and consists of 5 stride-2 three dimensional convolutional layers with 128, 192, 288, 432 and 648 channels respectively followed by 5 transposed convolutional layers with 648, 432, 288,192 and 128 channels respectively. All convolutional layers use a kernel size of $3 \times 3 \times 3$, and are followed by leaky ReLU activation layers [8]. The final layer in the network is a convolutional layer with a kernel size of $1 \times 1 \times 1$ which outputs a representation with 15 channels (objects and table). All layers are initialised

by Xavier uniform initialisation [6]. Due to the U-Net design there are skip connections between layers of the same spatial resolution.

4.2 Discriminator and stability scoring networks

These networks consist of 5 stride-2 three dimensional convolutional layers with 128, 192, 288, 432 and 648 channels respectively. All convolutional layers use a kernel size of $3 \times 3 \times 3$, and are followed by leaky ReLU activation layers. The final layer in the network is a fully connected layer which outputs a scalar number. All layers are initialised by Xavier uniform initialisation. These networks use weight normalisation [11].

5 Dataset

Scenes were generated using the Bullet physics engine [5]. For scenes generated by sampling up to 14 object classes without replacement, a simple flat table was used. For scenes generated by sampling up to 25 object classes with replacement, a table with a raised rim was used to prevent objects from being pushed off the table. If objects fell off the table, additional objects were added and simulated until the table contained the desired number of objects. Reference object properties from the YCB object set [2] were used for physical simulations.

During training scenes were augmented by applying random rotations about the axis of gravity, in integer multiples of 90°, consistently to all scene data.

The datasets are accessible from the project webpage[1].

6 Software and hardware

The dataset generation software was implemented in MATLAB [7]. Physics simulation was performed via the Bullet physics engine [5]. Rendering was performed via a custom renderer implemented in MATLAB which made use of some transformation functions from the Camera Calibration Toolbox for MATLAB [1]. Neural networks and training were implemented in PyTorch [9].

This work was executed on a computer with an Intel i7-6700k [3] processor, 64 GiB of random access memory, and an Nvidia GeForce GTX 1070 [4] graphics processing unit.

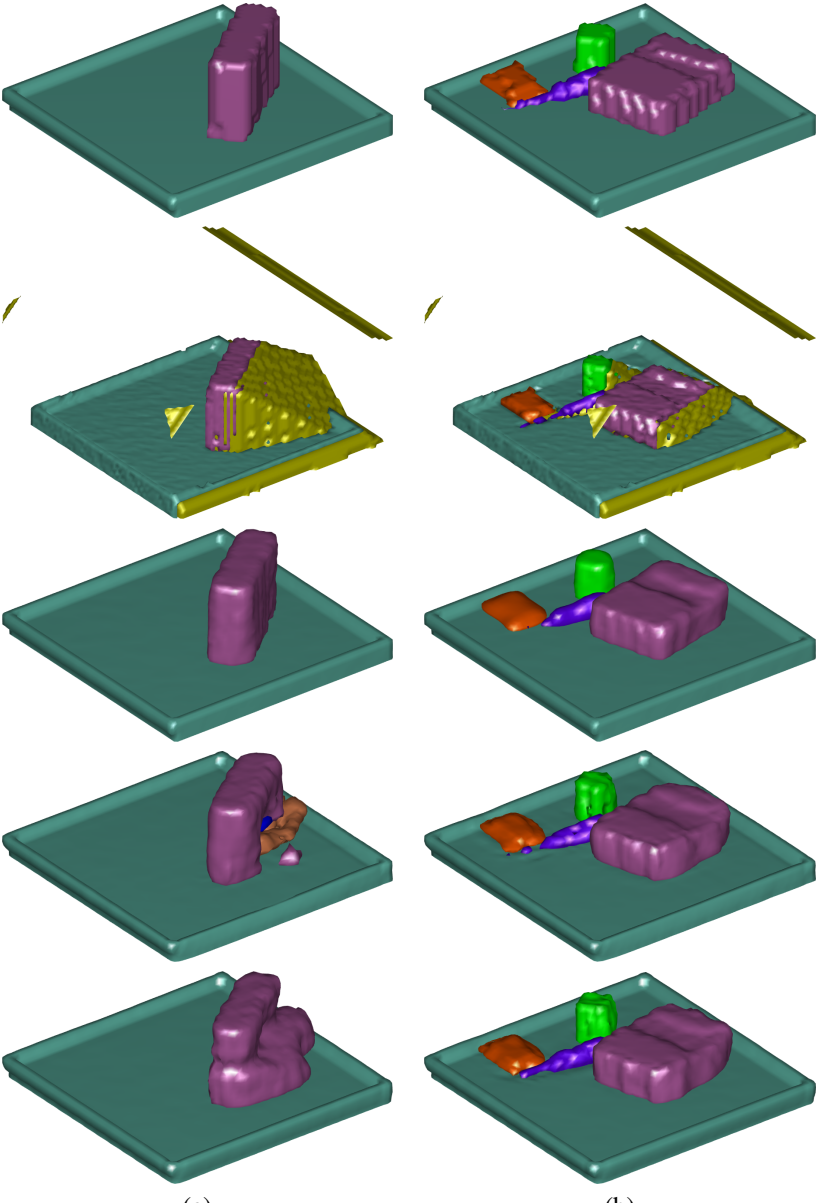
7 Imagined scenes

Larger and additional imagined scenes can be found in figures 1, 2, 3, 4, 5, and 6. Please see section 5.6 of the paper for discussion.

8 Varying the latent representation

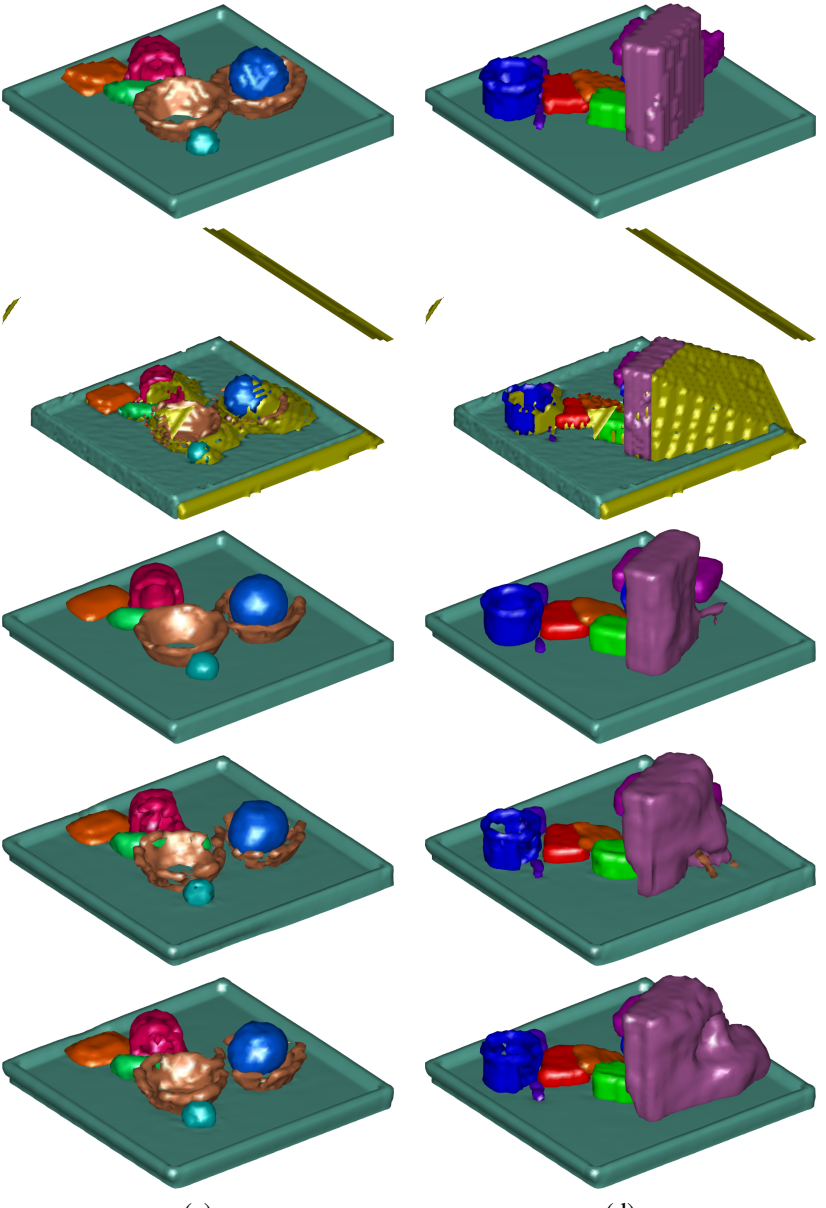
Larger and additional imagined scenes generated using different latent input samples can be found in figures 7, 8, and 9. Please see section 5.5 of the paper for discussion.

[1] https://hectorbasevi.github.io/imagining-hidden-supporting-objects



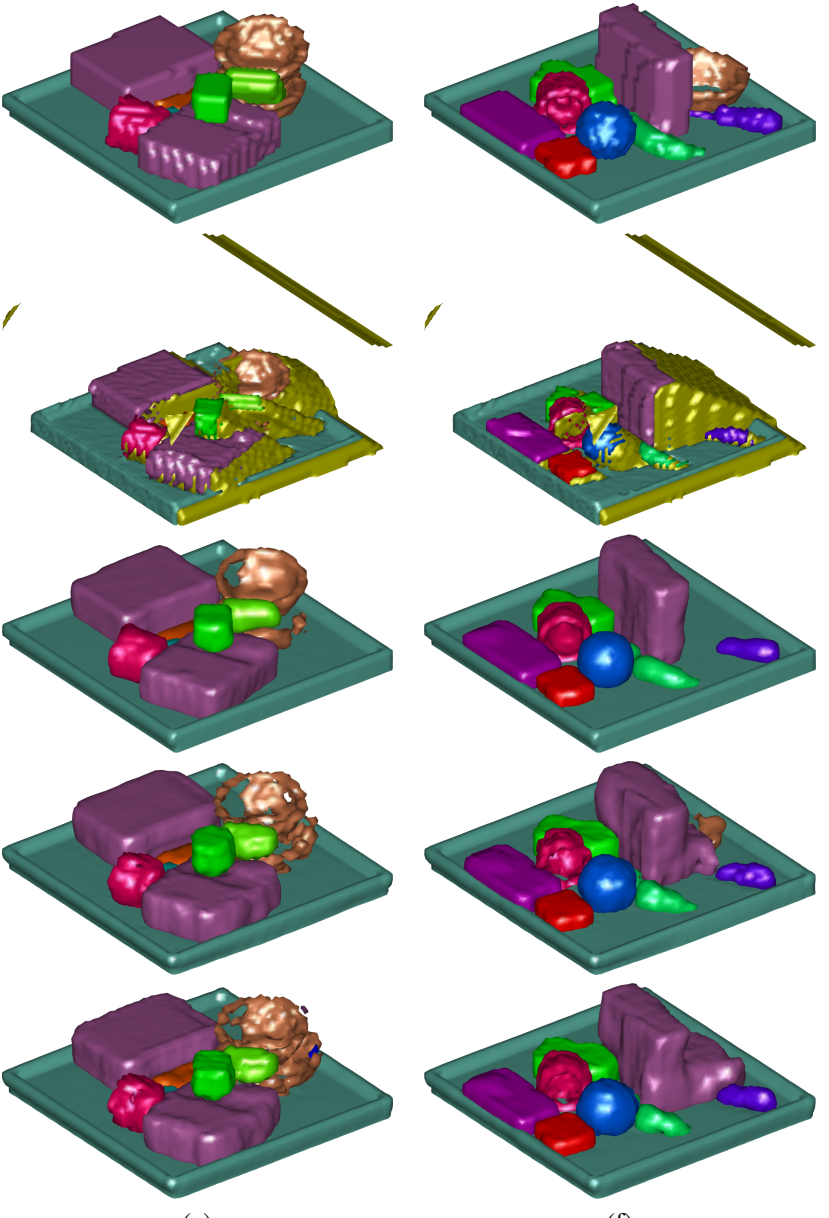
(a) (b)

Figure 1: Two examples of imagined scenes produced by the three models. The first row shows ground truth scenes. The second row shows visible surfaces (yellow regions are non-visible). The third row shows explanations from the regression model. The fourth row shows explanations from the generative model trained without stability context. The fifth row shows explanations from the full system. The results in this figure are explained and discussed in section 5.6 of the paper.



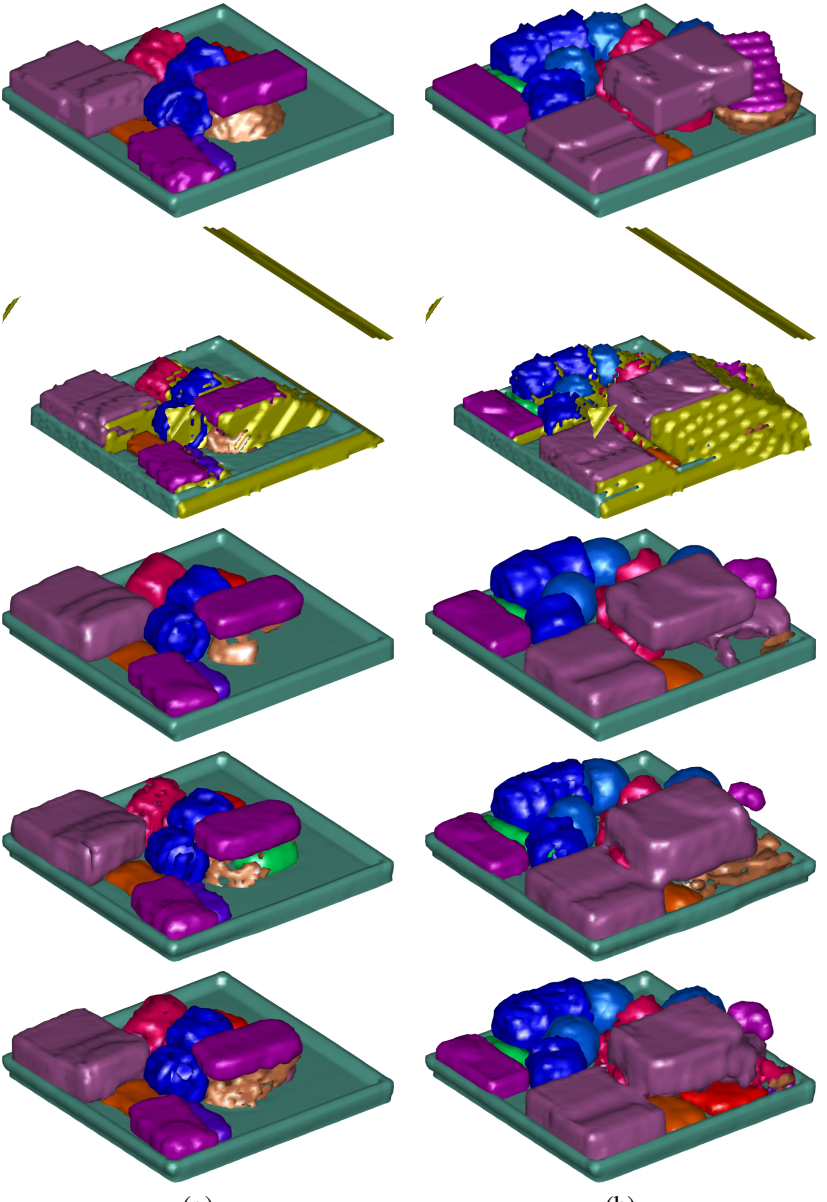
(c) (d)

Figure 2: Two examples of imagined scenes produced by the three models. The first row shows ground truth scenes. The second row shows visible surfaces (yellow regions are non-visible). The third row shows explanations from the regression model. The fourth row shows explanations from the generative model trained without stability context. The fifth row shows explanations from the full system. The results in this figure are explained and discussed in section 5.6 of the paper.



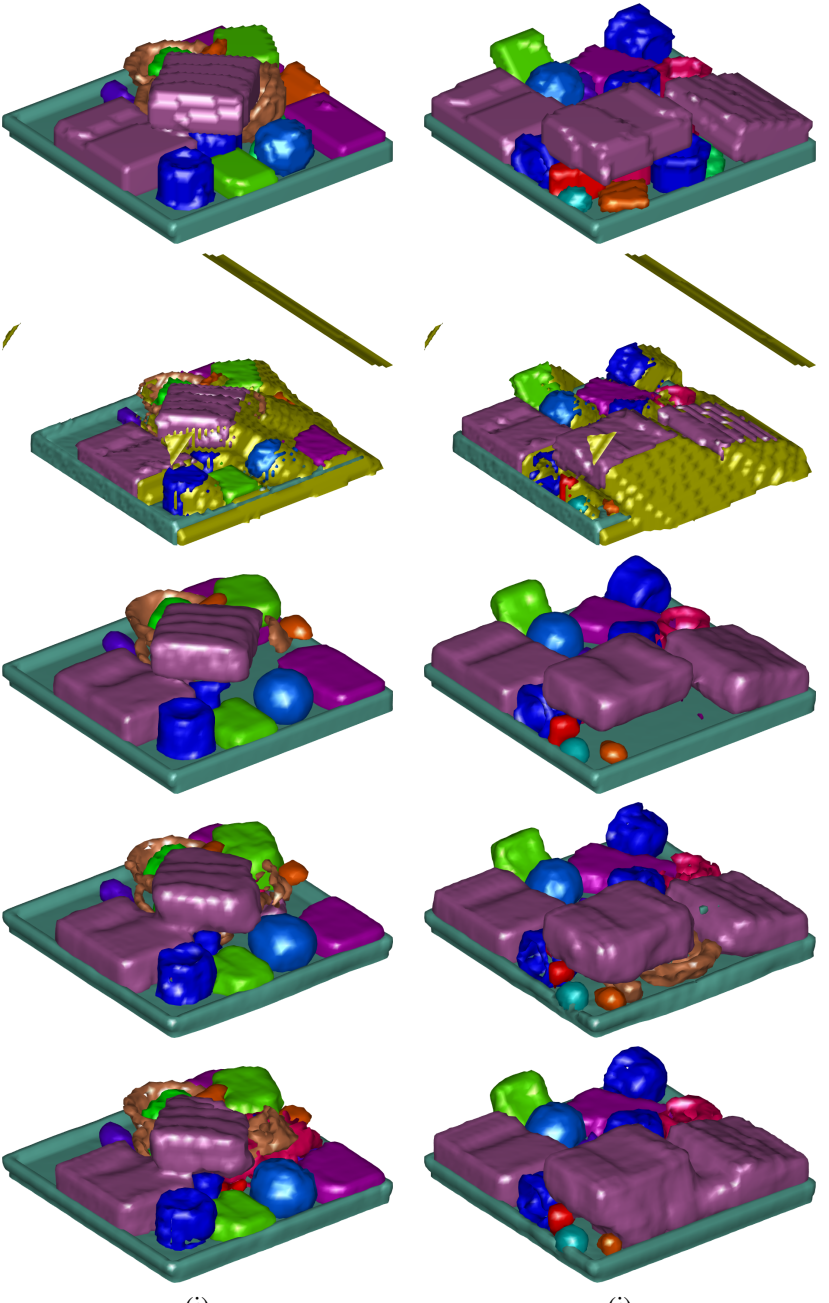
(e) (f)

Figure 3: Two examples of imagined scenes produced by the three models. The first row shows ground truth scenes. The second row shows visible surfaces (yellow regions are non-visible). The third row shows explanations from the regression model. The fourth row shows explanations from the generative model trained without stability context. The fifth row shows explanations from the full system. The results in this figure are explained and discussed in section 5.6 of the paper.



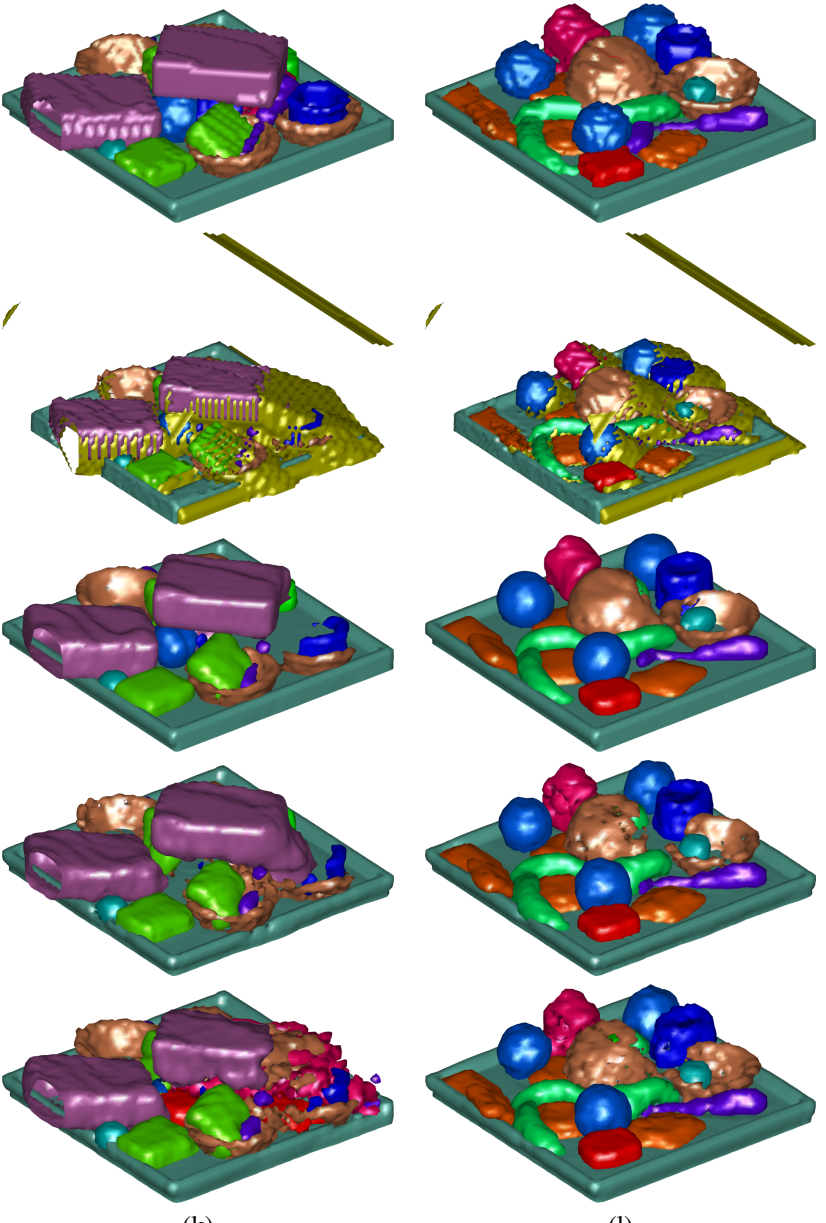
(g) (h)

Figure 4: Two examples of imagined scenes produced by the three models. The first row shows ground truth scenes. The second row shows visible surfaces (yellow regions are non-visible). The third row shows explanations from the regression model. The fourth row shows explanations from the generative model trained without stability context. The fifth row shows explanations from the full system. The results in this figure are explained and discussed in section 5.6 of the paper.



(i) (j)

Figure 5: Two examples of imagined scenes produced by the three models. The first row shows ground truth scenes. The second row shows visible surfaces (yellow regions are non-visible). The third row shows explanations from the regression model. The fourth row shows explanations from the generative model trained without stability context. The fifth row shows explanations from the full system. The results in this figure are explained and discussed in section 5.6 of the paper.



(k) (l)

Figure 6: Two examples of imagined scenes produced by the three models. The first row shows ground truth scenes. The second row shows visible surfaces (yellow regions are non-visible). The third row shows explanations from the regression model. The fourth row shows explanations from the generative model trained without stability context. The fifth row shows explanations from the full system. The results in this figure are explained and discussed in section 5.6 of the paper.

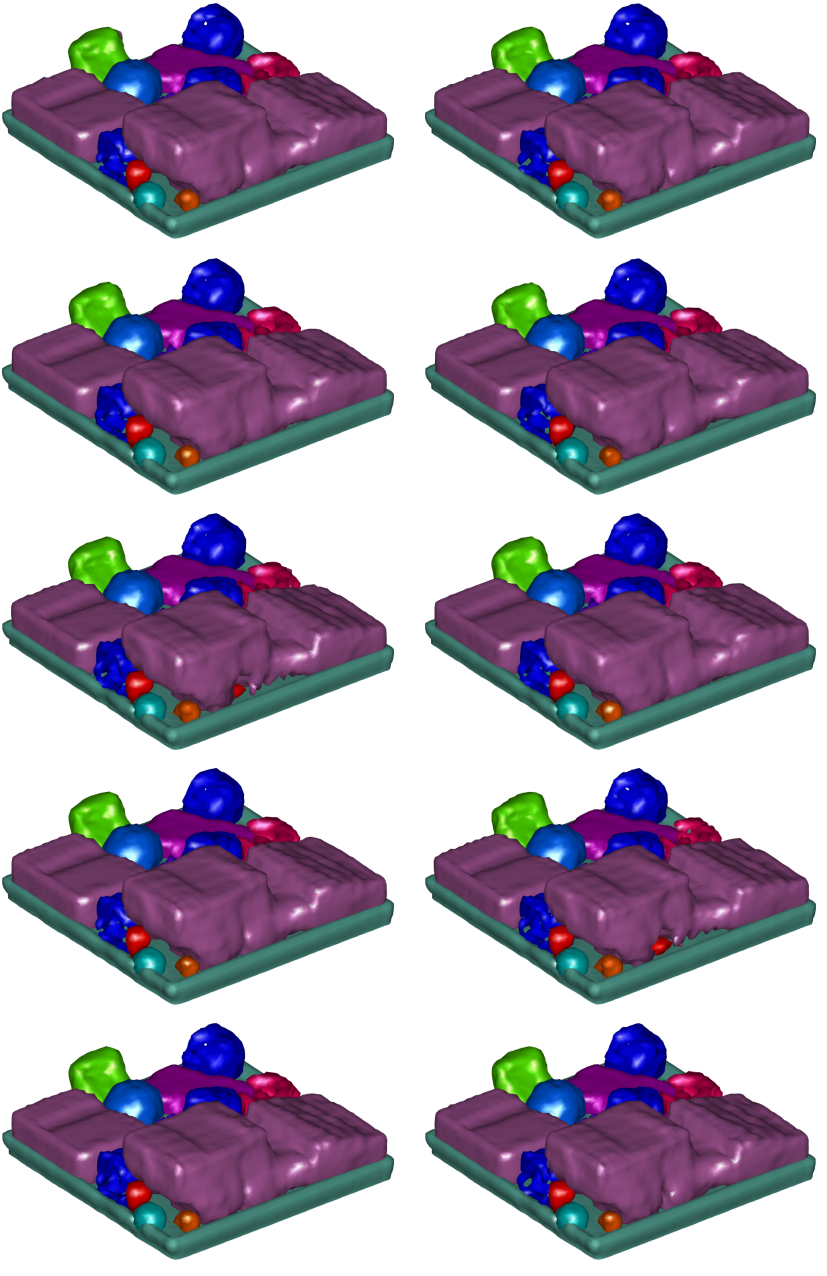


Figure 7: Example 1 of the effect of varying the latent representation, showing the effect of different random samples of the latent representation on scene imagination for the full system. Varying the latent representation is discussed in section 5.5 of the paper.

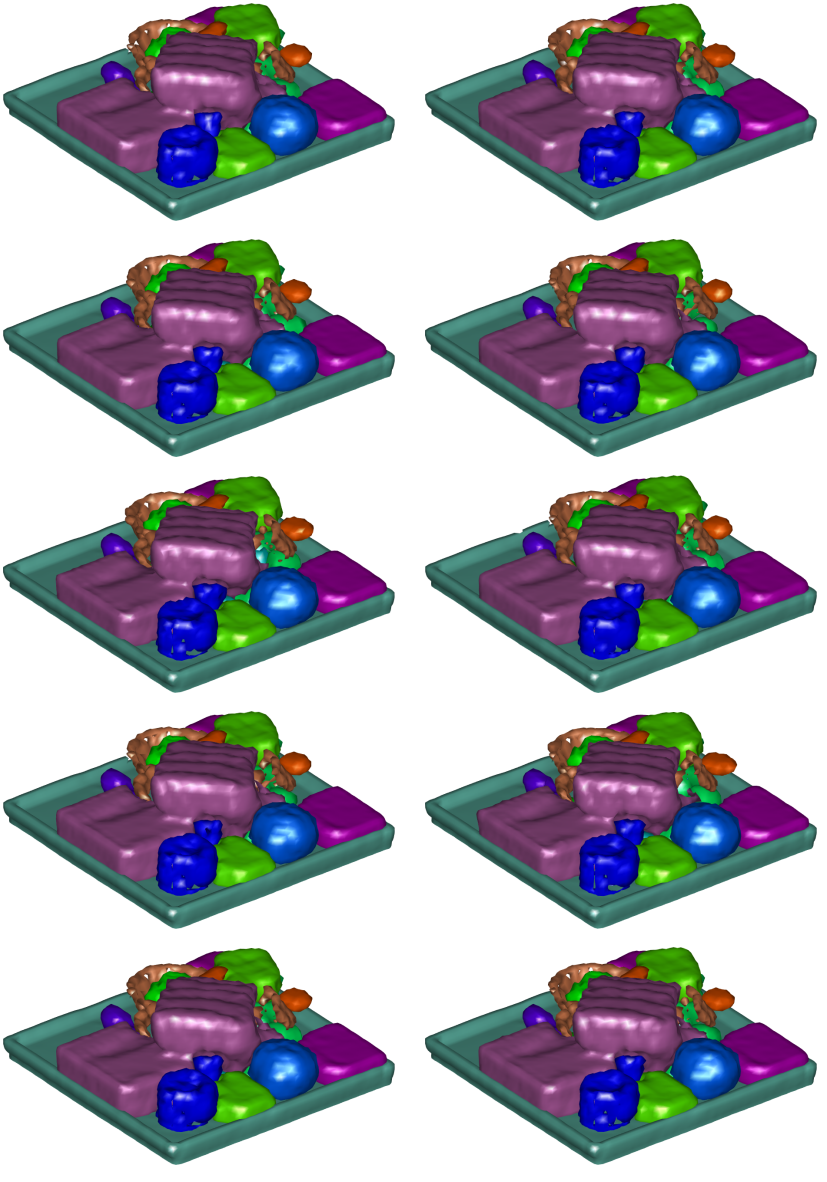


Figure 8: Example 2 of the effect of varying the latent representation, showing the effect of different random samples of the latent representation on scene imagination for the full system. Varying the latent representation is discussed in section 5.5 of the paper.

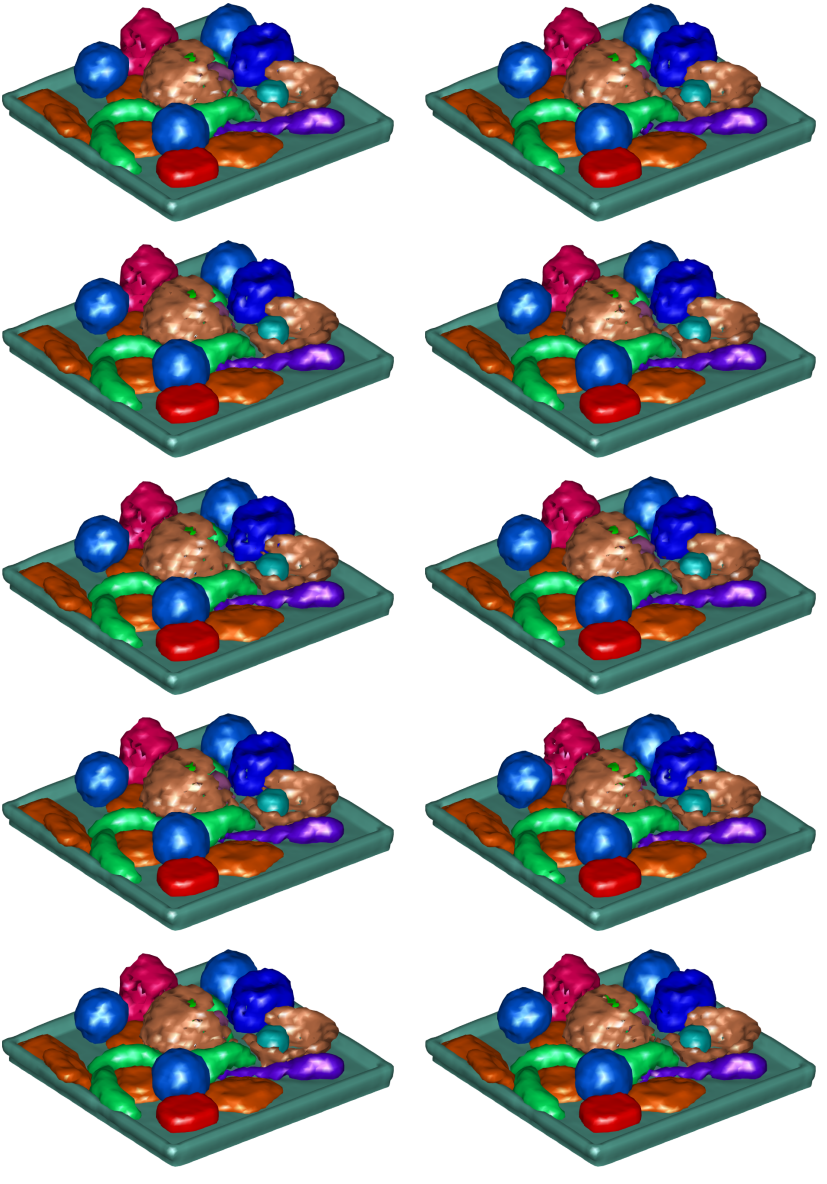


Figure 9: Example 3 of the effect of varying the latent representation, showing the effect of different random samples of the latent representation on scene imagination for the full system. The effect of varying the latent representation is discussed in section 5.5 of the paper.

9 Varying the stability context

Larger and additional imagined scenes generated using different stability contexts can be found in figures 10, 11, and 12. Please see section 5.5 of the paper discussion.

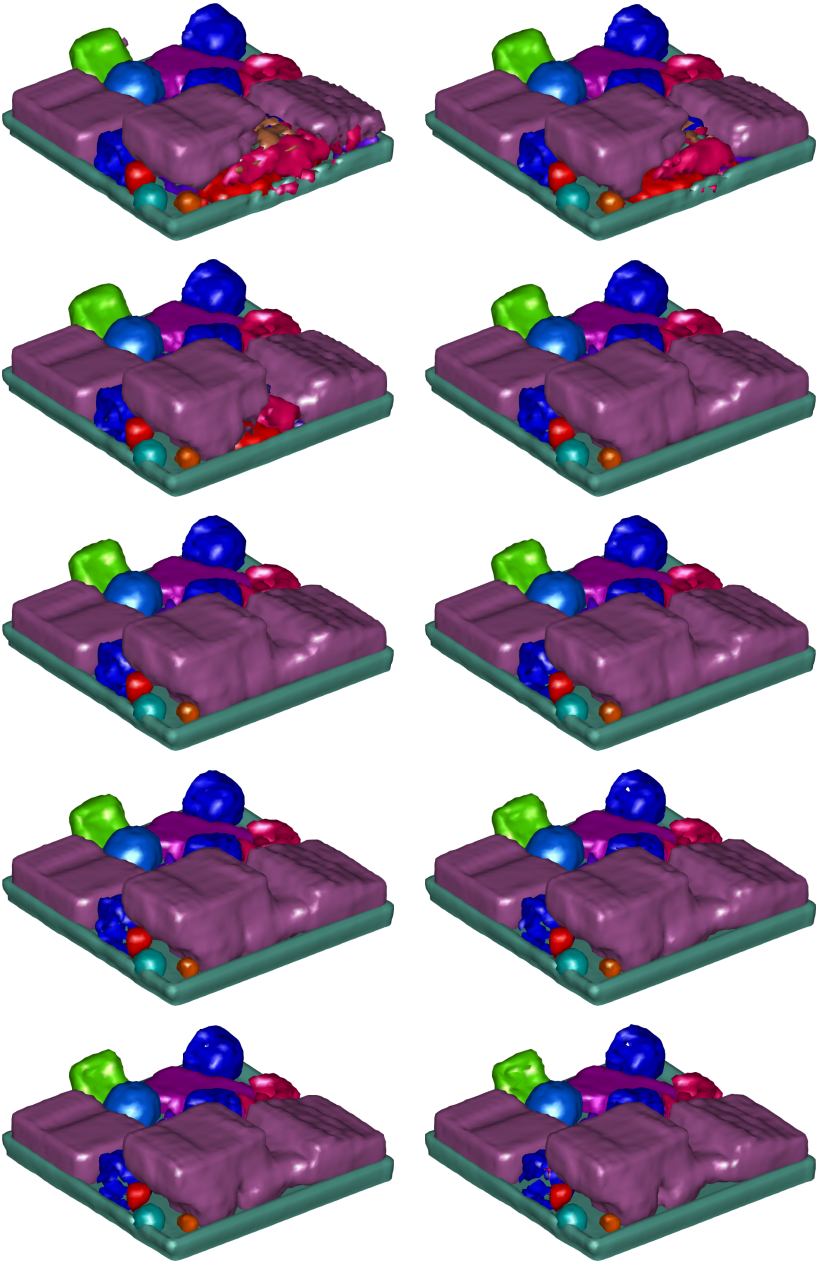


Figure 10: Example 1 of the effect of varying the stability context, showing the effect of varying the stability context on scene explanation for the full system. The stability context changes from instability to stability moving from left to right and then down the page (as if reading text). The effect of varying the stability context is discussed in section 5.5 of the paper.

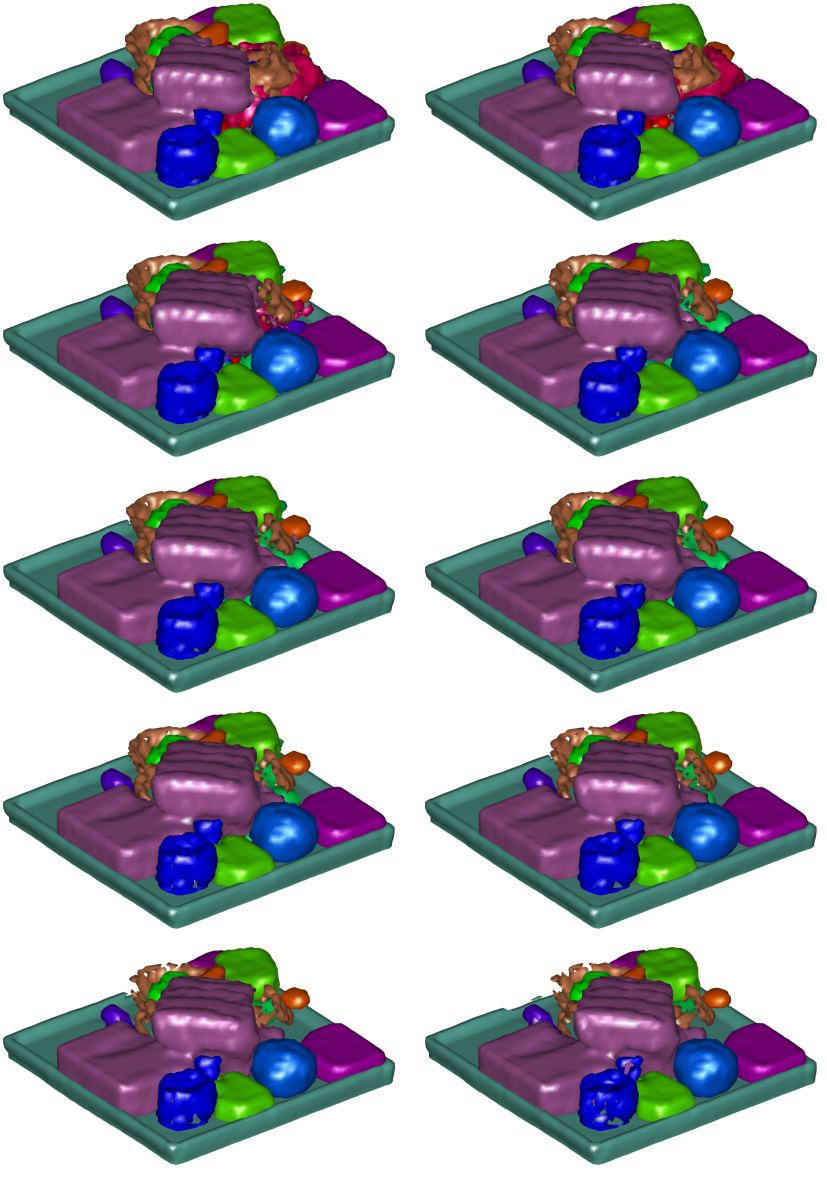


Figure 11: Example 2 of the effect of varying the stability context, showing the effect of varying the stability context on scene explanation for the full system. The stability context changes from instability to stability moving from left to right and then down the page (as if reading text). The effect of varying the stability context is discussed in section 5.5 of the paper.

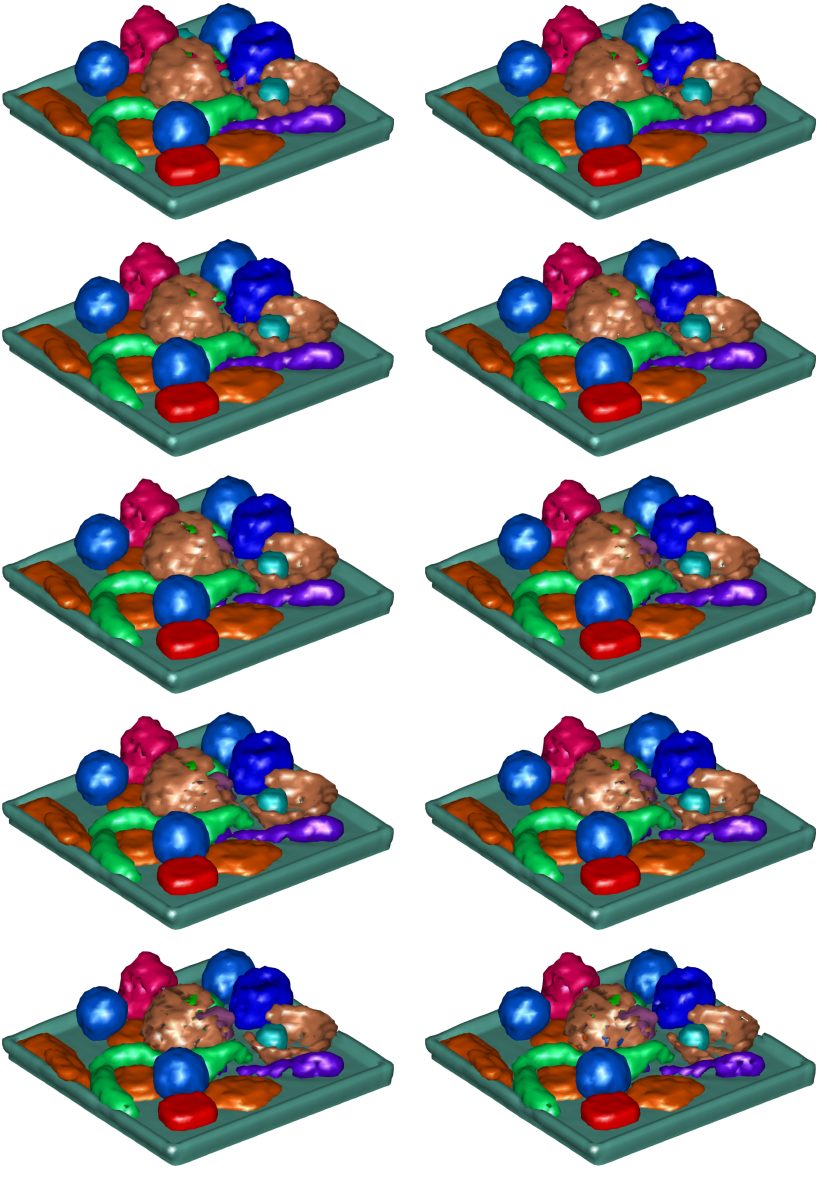


Figure 12: Example 3 of the effect of varying the stability context, showing the effect of varying the stability context on scene explanation for the full system. The stability context changes from instability to stability moving from left to right and then down the page (as if reading text). The effect of varying the stability context is discussed in section 5.5 of the paper.

References

[1] Jean-Yves Bouguet. Camera Calibration Toolbox for MATLAB, 2003. URL http://robots.stanford.edu/cs223b04/JeanYvesCalib.

[2] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar. Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set. *IEEE Robotics Automation Magazine*, 22(3):36–52, 2015.

[3] Intel Corporation. Intel Core i7-6700k Processor, 2022. URL https://www.intel.co.uk/content/www/uk/en/products/sku/88195/intel-core-i76700k-processor-8m-cache-up-to-4-20-ghz/specifications.html.

[4] NVIDIA Corporation. GeForce GTX 1070 Specifications, 2021. URL https://www.nvidia.com/en-gb/geforce/graphics-cards/geforce-gtx-1070/specifications.

[5] Erwin Coumans et al. Bullet physics library. *Open source: bulletphysics. org*, 15:49, 2013.

[6] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[7] The MathWorks Inc. MATLAB, 2022. URL https://www.mathworks.com/products/matlab.html.

[8] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[9] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[10] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

[11] Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909, 2016.