SceneNet RGB-D: 5M Photorealistic Images of Synthetic Indoor Trajectories with Ground Truth

John McCormac, Ankur Handa, Stefan Leutenegger, Andrew J. Davison
Dyson Robotics Laboratory at Imperial College, Department of Computing,
Imperial College London
{brendan.mccormac13,s.leutenegger,a.davison}@imperial.ac.uk, handa.ankur@gmail.com

Abstract

We introduce SceneNet RGB-D, expanding the previous work of SceneNet to enable large scale photorealistic rendering of indoor scene trajectories. It provides pixel-perfect ground truth for scene understanding problems such as semantic segmentation, instance segmentation, and object detection, and also for geometric computer vision problems such as optical flow, depth estimation, camera pose estimation, and 3D reconstruction. Random sampling permits virtually unlimited scene configurations, and here we provide a set of 5M rendered RGB-D images from over 15K trajectories in synthetic layouts with random but physically simulated object poses. Each layout also has random lighting, camera trajectories, and textures. The scale of this dataset is well suited for pre-training data-driven computer vision techniques from scratch with RGB-D inputs, which previously has been limited by relatively small labelled datasets in NYUv2 and SUN RGB-D. It also provides a basis for investigating 3D scene labelling tasks by providing perfect camera poses and depth data as proxy for a SLAM system. We host the dataset at http://robotvault.bitbucket.io/scenenet-rgbd.html.

1. Introduction

A primary goal of computer vision research is to give computers the capability to reason about real world images in a human-like manner. This includes a semantic understanding of the objects present in the scene, their locations, 6-DoF poses, and an intuitive grasp of the physics involved. Recent years have witnessed a huge interest in scene understanding, largely sparked by the seminal work of AlexNet [14] and the increasing popularity of Convolutional Neural Networks (CNNs). That work highlighted the importance of large scale labelled datasets when working with data-hungry supervised learning algorithms. In this work we focus on the challenge of obtaining large quantities of labelled data with the aim of alleviating the need for collecting datasets through manual effort.

In particular, we are motivated by tasks which require more than a simple text label for an image. For tasks such as semantic labelling and instance segmentation, obtaining accurate per-pixel ground truth annotations by hand is prohibitively expensive. In other cases it can be almost impossible, such as for fine-grained optical flow data, or metrically accurate 3D pose information for an object. Inspired

	Stanford Scenes	NYUv2	SUN RGB-D	SceneNet	sceneNN	SUN CG*	**SceneNet RGB-D**
RGB-D videos available	✗	✓	✗	✗	✓	✗	✓
Per-pixel annotations	NA	Key frames	Key frames	Key frames	Videos	Key Frames	Videos
Trajectory ground truth	✗	✗	✗	✗	✓	✗	✓
RGB texturing	Non-photorealistic	Real	Real	Non-photorealistic	Real	Non-photorealistic	Photorealistic
Number of layouts	1723	464	-	57	100	45,622	57
Number of configurations	1723	464	-	1000	100	45,622	16,895
3D models available	✓	✗	✗	✓	✓	✓	✓
Method of design	Manual	Real	Real	Manual and Random	Real	Manual	Random

Table 1. A comparison table of indoor scene datasets and their differing characteristics. SceneNN is an example of a real world dataset that does provide 3D models, however the models are not water-tight. Stanford Scenes database [6] does not provide any explicit ground truth as it is primarily designed for scene retrieval. SUN RGB-D captures short video clips but only release single image annotations. *At the time of this manuscript's publication the SUN CG dataset was not released. They provide dense volumetric annotations, however only for single images.

by the recent success of synthetic data for training scene understanding systems, our goal has been to generate a large scale dataset of photorealistic RGB-D videos which provide perfect and complete ground truth for a wide range of problems.

Our dataset has several key strengths relative to other publicly available datasets for indoor scene understanding that make it especially useful for training computer vision models, which could be used for real-world applications in robotics and augmented reality. We have used ray-tracing to generate high quality synthetic RGB images, aiming towards photorealism with full lighting effects and elements such as motion blur, as well as accompanying synthetic depth images. The images are rendered from randomly generated smooth trajectories to create sequential video clips from a moving virtual camera, opening up research on temporal fusion for high quality labelling. Our process to generate the contents of the synthetic scenes observed has relied to the greatest degree possible on fully automatic methods, with object distributions statistically sampled from publicly available real-world scene repositories and randomly positioned within a physics simulation that then ensures feasible configurations. This means that our pipeline can produce a greater degree of variety of scene configurations than others, enabling a potentially much larger dataset without the need for human scene design or annotation.

In Section 3 we discuss the overall dataset pipeline and available ground truth labels. In Section 4 below, we describe the process of obtaining metric scales of objects from SUN RGB-D. Section 5 provides a detailed explanation on random scene generation, and Section 6 talks about generating random trajectories.

2. Background

A growing body of research has highlighted that carefully synthesised artificial data with appropriate noise models can be an effective substitute for real-world labelled data in areas that ground-truth data is difficult to obtain. Aubry *et al.* [1] used synthetic 3D CAD models for learning visual elements to do 2D-3D alignment in images, and similarly, Gupta *et al.* [7] trained on rendering of synthetic objects to do alignment of 3D models with RGB-D images. Peng *et al.*[17] augmented small datasets of objects with renderings of synthetic 3D objects with random textures and backgrounds to improve object detection performance. FlowNet [5] and recently FlowNet 2.0 [12] showed that remarkable improvements can be made with training data obtained from synthetic scenes for optical flow estimation. de Souza *et al.* [4] use procedural generation of human actions with computer graphics to generate large dataset of videos for human action recognition.

As a precursor to the present work, Handa *et al.* [9] produced SceneNet, a repository of labelled synthetic 3D scenes from five different categories. This repository was used to generate per-pixel semantic segmentation ground truth for depth only images from random viewpoints. They demonstrate that a network trained on 10K images of synthetic depth data and fine-tuned on the original NYUv2 [22] and SUN RGB-D [23] datasets shows an increase in the performance on the task of semantic segmentation when compared to the network trained on just the original datasets.

For outdoor scenes, Ros *et al.* generated the SYNTHIA [20] dataset for road scene understanding, and work by Richter *et al.* [19] produced synthetic training data from a photorealistic gaming engine. This is an exciting avenue, however it is not always possible to obtain the required data from gaming engines, which due to proprietary issues lack the flexibility of a fully open-source alternative. SceneNet RGB-D uses open-source scene layouts [9] and 3D object repositories [3] that provide textured objects. We have also built upon an open-source ray-tracing framework which allows significant flexibility in the ground truth data we can collect and visual effects we can simulate.

For indoor scenes, recent work by Qui *et al.* [18] called UnrealCV provides a plugin to generate ground truth data and photorealistic images from UnrealEngine. However, they do not provide any labelled dataset and their plugin uses scene assets created by artists, which assists in the ap-

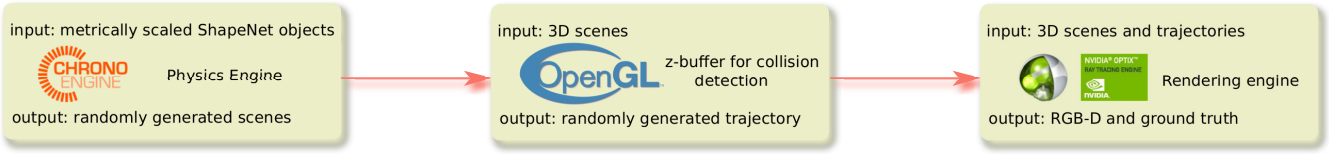


Figure 1. Flow chart of the different stages in our pipeline. Physically realistic scenes are created using Chrono Engine by dropping objects from the ceiling. These scenes are used to generate automated camera trajectories simulating human hand-held motion and both are passed on the rendering engine — inspired by OptiX — to produce RGB-D ground truth.

parent photorealism with high quality assets. Assets of this quality are often proprietary, and difficult to source in large quantities. Finally, they do not explore random scene generation systems as we do here.

Our dataset, SceneNet RGB-D, samples random layouts from SceneNet [9] and objects from ShapeNet [3] to create a potentially unlimited number of scene configurations. As shown in Table 1, there are a number of key differences between our work and other available datasets. It is one of the first to provide large quantities of photorealistic renderings of indoor scenes. Hua *et al.* provide sceneNN [11], a dataset of 100 labelled meshes of real world scenes, obtained with a reconstruction system with objects labelled directly in 3D for semantic segmentation ground truth. Such real-world datasets are often limited in scale due to the amount of manual effort required.

Recently, Song *et al.* released the SUN-CG dataset [24] which consists of 45,622 synthetic scene layouts created using Planner5D. There are a few key differences between this and our work. First, they have not aimed towards rendering photorealistic images of their scenes. Second, our dataset explicitly provides a sequential video trajectory within a scene, allowing 3D correspondences between viewpoints for 3D scene understanding tasks, with the ground truth camera poses acting in lieu of a SLAM system[15]. Third, their approach to scene generation is quite different. While they have many examples of natural looking manually designed scenes, our approach produces more chaotic configurations that can be generated on-the-fly with almost no chance of repeating. Moreover, since layout textures, positions of light sources, and camera trajectories are all randomised we are able to generate a wide variety of geometrically identical but visually differing renders as shown in Figure 10.

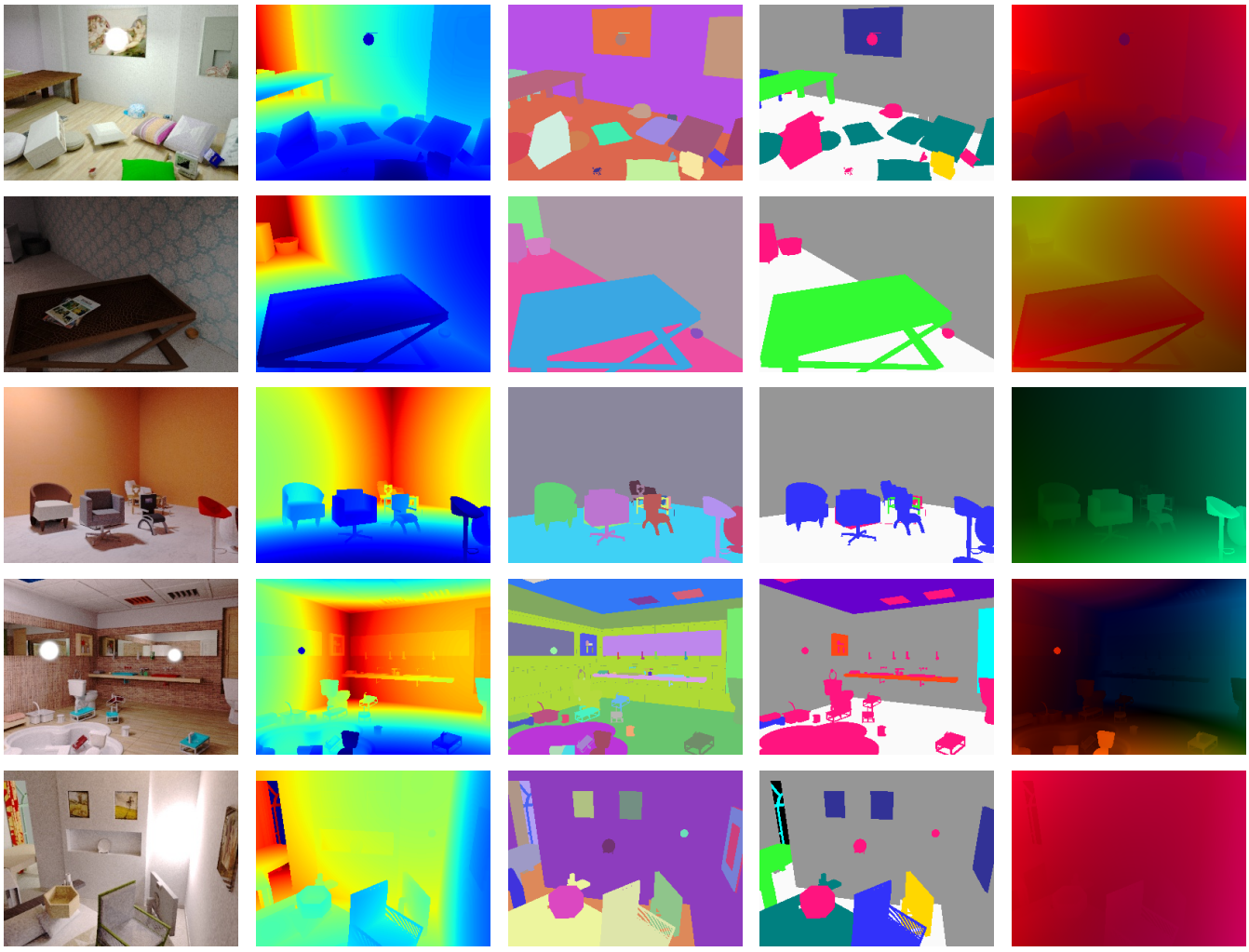
We believe such randomness could help prevent overfitting by providing a significantly less predictable set of training examples with high instructional value. It remains an open question whether randomness is preferrable to well designed scenes for learning algorithms, but the recent works of FlowNet and FlowNet 2.0 [5, 12] seem to suggest that randomness is potentially helpful. Randomness also leads to a simpler data generation pipeline and, given a sufficient computational budget, allows for dynamic on-the-fly generated training examples suitable for active machine learning. A combination of these two approaches, with a reasonable manually designed scene layouts and added physically simulated noise and clutter may in the end provide the best of both worlds.

3. Dataset Overview

The overall process from sampling objects to rendering RGB-D frames is shown in Figure 1. For the dataset, we had to balance the competing requirements of frame-rates for video sequences with the computational cost of rendering many very similar images, which would not provide significant variation in the training set for CNNs. We decided upon 5 minute trajectories at 320×240 image resolution, but with a single frame per second, resulting in 300 images per trajectory. Each render takes 2–3 seconds on an NVIDIA GTX 1080 GPU. There is also a trade off between rendering time and quality of renders (See Figure 9 in Section 7.2).

Our trajectory is calculated with a frame-rate of 25Hz, however we only render every 25th pose. Each pose consists of a pair of poses, which define the shutter open and shutter close of the camera. We sample from poses linearly interpolated between the two points to produce motion blur artefacts for simulating any rapid camera shaking. Different ground truth labels can be obtained with an extra rendering pass *e.g.* instance labels are obtained by assigning indices to each object and rendering for each pixel the index of each object instead of RGB values. Depth is defined as the ray length from the camera origin to the first surface it intersects with, this provides perfect depth information even in the case of reflections and motion blur. For ground truth we do not sample multiple points for each pixel as we do for RGB, instead a single ray is emitted from the center of the pixel.

From these ground truth images it is possible to calculate a number of other pieces of ground truth information. For example, in accompanying datafiles for each trajectory we store a mapping from each instance label to a semantic label. These semantic labels are defined with a WordNet id, which provides a useful network structure for semantic links and hierarchies. In total we have 255 different WordNet semantic categories, including 40 WordNet ids outside of the normal corpus, which were added by the ShapeNet dataset. Given the static scene assumption, the instantaneous optical



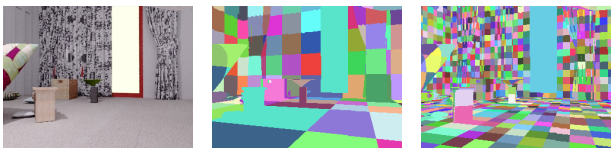
(a) photo (b) depth (c) instance (d) class segmentation (e) optical flow

Figure 2. Hand-picked examples from our dataset. Rendered images on the left and the available ground truth information on the right.

flow can be calculated using the camera pose along with the depth map. Some examples of the available ground-truth information for a corresponding image is shown in Figure 2.

Using the inverse camera model reprojection and the perfect depth map, it is also possible to calculate the 3D position of each surface in the scene. We use this to calculate voxel correspondence indices (for some arbitrarily selected voxel size) for an entire trajectory, to mimic the correspondences available in a perfect SLAM system. For an example colorisation of this correspondence system see Figure 3.

Our dataset is separated into train, validation, and test sets. Each of these sets has a unique set of layouts, objects, and trajectories particular to the set. However the parameters for randomly choosing lighting and trajectories remains the same. We selected two layouts from each type (bathroom, kitchen, office, living room, and bedroom) for the validation and test sets making the layout split 37-10-10.



(a) Photo (b) 0.5m voxels (c) 0.15m voxels

Figure 3. On the left is the original photo, on the right are unique randomly coloured voxels that remain the same throughout a trajectory. Outside the window there is no depth reading so we assign all of these areas the same default identifier.

For ShapeNet objects within a scene we randomly divide the objects within each wordnet class into 80-10-10% splits for train-val-test. This ensures that some of each type of object are in each training set. Our final training set has 5M images from 16K images, our validation and test set have 300K images from 1K different room layouts. Each layout

has a single accompanying trajectory through it.

4. Obtaining metric scales of CAD models

The majority of 3D models in CAD repositories or open-source libraries are created by 3D designers and artists without any explicit designation of metric-scale information. However, it is desirable that the objects placed in our synthetic scenes have similar statistics to their corresponding real world counterparts in terms of their physical dimensions. Fortunately, datasets like SUN RGB-D [23] are captured with a depth camera and provide metric 3D bounding boxes for each labelled object in the scene. We leverage this information to obtain the height distribution of object categories, and then randomly sample metric heights from this distribution to scale each object before placing it in the scene. We maintain the aspect ratio of these objects during this scaling procedure. Figure 4 shows probability distribution of heights of some objects as obtained from SUN RGB-D.

This simple approach is not entirely without drawbacks. The lack of granularity within classes can lead to multi-modal height distributions. For example bedside lamps and floor lamps both are within the same 'lamp' class for our purposes, however their heights vary significantly. If the height of a floor lamp is applied to a squat bedside lamp, the resulting object can appear closer in its dimensions to a refrigerator. Tackling this is a significant problem and some work has been done which could be useful in future iterations [21].

5. Generating random scenes with physics

We use an off-the-shelf physics engine, Project Chrono[1], to dynamically simulate the scene. We opted for this rather than a computationally more efficient static geometric analysis system for a number of reasons. Firstly, the computational bottleneck in our system was the rendering pipeline, the physics engine uses the CPU which leaves the GPU free for rendering and can simulate many scenes in the time it takes to render one. Secondly, off-the-shelf physics software was readily available and quite easy to use, and resulted in reasonable looking layouts. Finally, a full physics simulator leaves open the potential for physically simulated dynamic scenes in future work.

To create scenes, we first of all randomly choose the density of objects per square meter. In our case we have two of these densities. For large objects we choose a density between 0.1 and 0.5 objects/m^2, and for small objects ($<$0.4m) we choose a density between 0.5 and 3.0 objects/m^2. Given the floor area of a scene, we then can easily calculate the number objects needed. We sample objects for a given scene according to the distribution of objects categories in that scene-type in the SUN-RGBD real-world dataset. We do this with the aim of including relevant objects within a context *e.g.* a bathroom is more likely to contain a sink or toilet than a microwave (see Figure 5 for an object breakdown by scene type). We then randomly choose an object class according to the scene type and pick a random instance uniformly from the available models for that object type.

The objects are provided with a constant mass (10kg) and convex collision hull and positioned uniformly within the 3D space of the layouts axis aligned bounding box. To slightly bias objects towards maintaining a correctly orientated upwards direction, we offset the center of gravity on the objects to be below the mesh. Without this, we found that very few objects such as chairs were in their normal upright position after the physics simulation had completed. One drawback of the convex collision hull is that, for example, a whole table can sometimes be propped up by a small object underneath the middle of it.

The physics engine models the movement of objects using Newtonian laws, and their interactions with each other and the layout (which is properly modelled as a static non-convex collision object). We simulate 60s of the system, leaving the objects to settle to a physically realistic configuration. It is important to note that the scene is not necessarily organised and structured in a human manner. It contains objects in random poses and locations but the overall configuration is physically plausible *i.e.* we will not have configurations where an object cannot physically support another, and unrealistic object intersections are avoided.

6. Generating random trajectories

As we aim to render videos at a large scale, it is imperative that the trajectory generation be automated to avoid costly manual labour. The majority of previous works have used a SLAM system operated by a human to collect hand-held motion: the trajectory of the camera poses returned by the SLAM system is then inserted in a synthetic scene and the corresponding data is rendered at discrete or interpolated poses of the trajectory [8, 10]. However, such reliance on humans to collect trajectories quickly limits the potential scale of the dataset.

We automate this process using a simple random camera trajectory generation procedure which we have not found in any previous synthetic dataset work. For our trajectories, we have the following desiderata. Our generated trajectories should be random, but slightly biased towards looking into central areas of interest, rather than, for example panning along a wall (See Figure 6 for an analysis on the number of instances visible for any given image in our final dataset). It should contain a mix of fast and slow rotations such as those of a human operator focussing on nearby and far away points. It should also have limited rotational freedom that emphasises yaw and pitch rather than rolling,

[1] https://projectchrono.org/

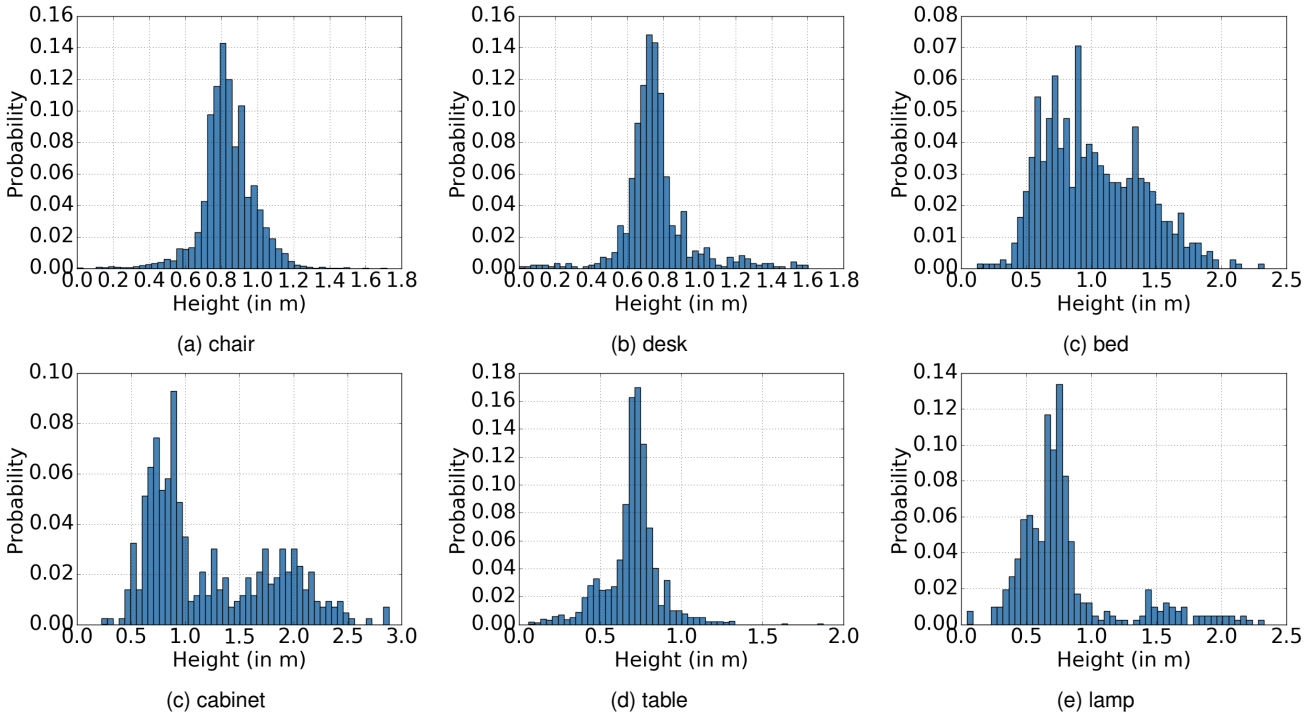


Figure 4. Probability distributions of heights (in m) of different objects as obtained from SUN RGB-D. It is interesting to see that some objects like cabinets and lamps clearly do have multimodal height distributions.

which is a less prominant motion in human trajectories.

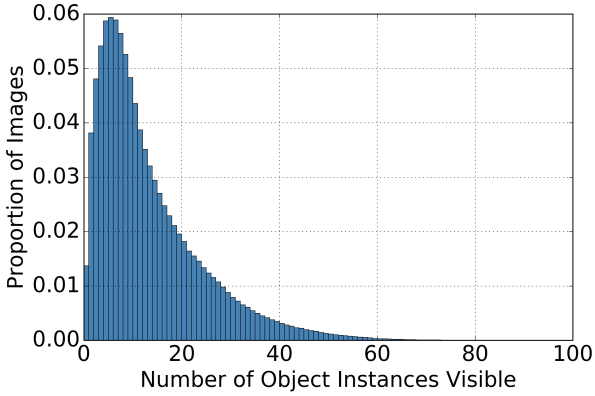


Figure 6. The frequency of images with a certain number of object instances visible in the dataset.

6.1. Two body camera trajectories

To achieve the desired trajectory paths we simulate two physical bodies in space. One defines the location of the camera, and another, the point in space that it is focussing on as a proxy for a human focussing on random points in a scene. We take the simple approach of locking roll entirely, by setting the up vector to alway be along the positive y-axis, these two points then completely define the camera coordinate system.

The physical approach has a number of benefits. Firstly, it provides an intuitive set of metric physical properties we can set to achieve a desired trajectory, such as the strength of the force in Newtons and the drag coefficients. Secondly, it naturally produces smooth trajectories. Finally, although not provided in this dataset, it automatically provides a set of IMU style accelerometer measurements, which could in future prove useful for Visual-Inertial systems.

We initialise the pose and "look-at" point from a uniform random distribution within the bounding box of the scene, ensuring they are less than 50cm apart. As not all scenes are convex, it is possible to initialise the starting points outside of a layout, for example in an 'L'-shaped room. Therefore, we have two simple checks. The first is to restart the simulation if either body leaves the bounding volume. The second is that within the first 500 poses at least 10 different object instances must have been visible. This prevents trajectories external to the scene layout with only the outer wall visible.

We use simple Euler integration to simulate the motion of the bodies and apply random force vectors and drag to them independently. The body is initialized with a position, $\boldsymbol{p}$, sampled as described above, and a velocity, $\mathbf{v} = \mathbf{0}$. We begin by sampling from a uniform spherical distribution. We achieve this by sampling from a 3-dimensional multivariate gaussian, with $\boldsymbol{\mu} = \mathbf{0}$ and $\boldsymbol{\Sigma} = \mathbf{I}$,

$$\mathbf{u} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \tag{1}$$

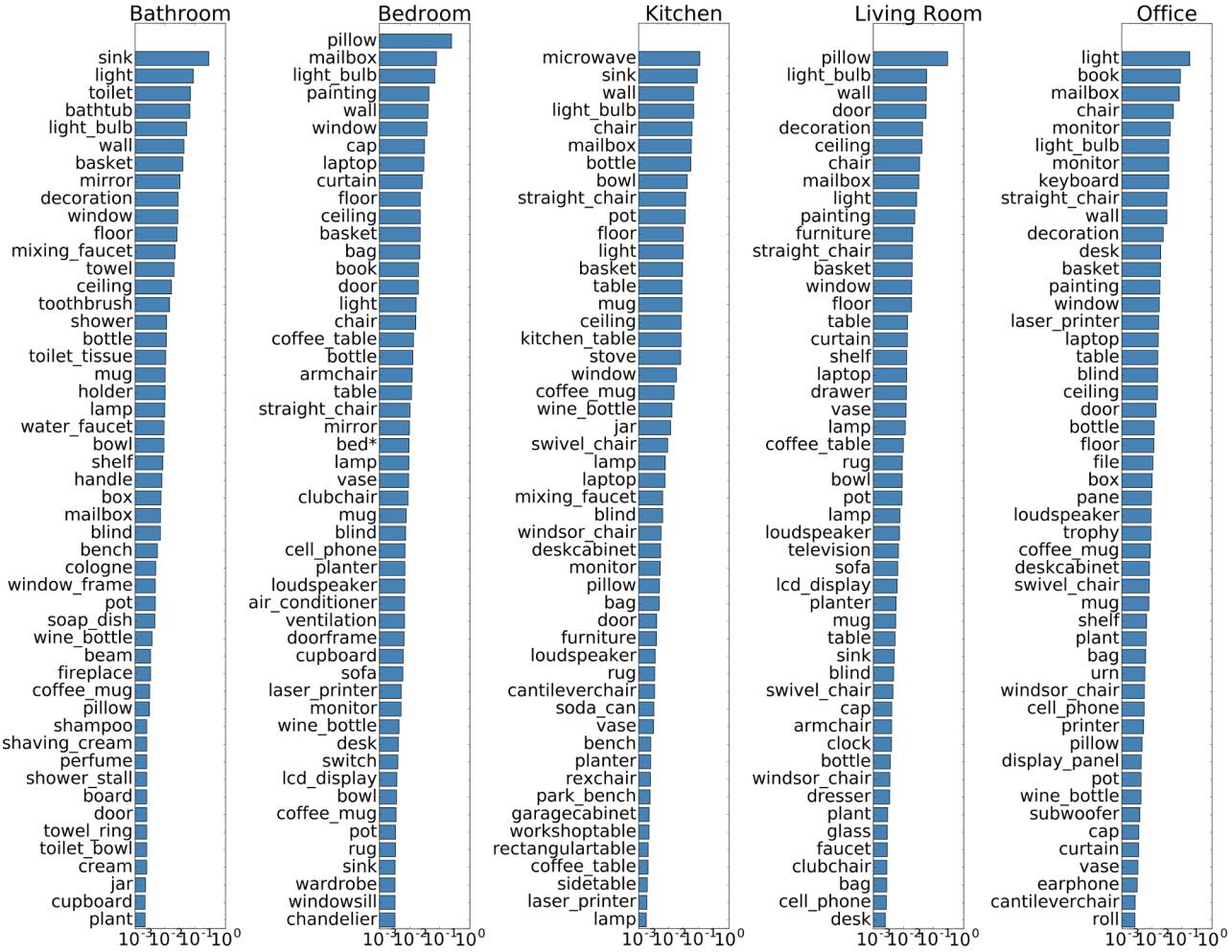


Figure 5. Top 50 objects and their log proportions by scene type. The unfortunate number of mailboxes is a result of a mistaken mapping of the 'box' class in SUN RGB-D to a class defined as box in ShapeNets, but which contains primarily mailboxes. This is an unfortunate mishap that serves to highlight some of the difficulties inherent in working with large quantities of objects and labels in an automated way. *beds are subdivided into a number of similar classes such as miscbeds, kingsized beds, and here we combine these into a coherent group.

we normalise $\boldsymbol{u}$ to be on the unit sphere and then scale it by a force constant, f, which we set to 2.5 N, to arrive at our force vector $\mathbf{f}$

$$\mathbf{f} = f\frac{\mathbf{u}}{||\mathbf{u}||}. \tag{2}$$

We also apply a drag force to dampen fast motions. We roughly model this as air drag at 20° with a 30 cm ball. With a cross-sectional area $A = 0.09\,\mathrm{m}^2$, drag coefficient $C_D = 0.1$, and air density $\rho = 1.204\,\mathrm{kg\,m}^{-3}$,

$$\mathbf{d} = -\frac{\mathbf{v}}{2||\mathbf{v}||}\rho A C_D ||\mathbf{v}||^2 \tag{3}$$

To calculate the acceleration we assign the body a mass, m, of 1.0 kg. We use simple Euler integration over a timestep, τ, which here we set to $\frac{7}{300}$s for the period between shutter close and shutter open, and $\frac{1}{60}$s for the shutter open and shutter close exposure time.

$$\mathbf{v}_t = \mathbf{v}_{t-1} + \tau\left(\frac{\mathbf{d}+\mathbf{f}}{m}\right) \tag{4}$$

We also limit the maximum speed of the body to $s_{\max}$,

$$\mathbf{v}_t = \begin{cases} s_{\max}\frac{\mathbf{v}_t}{||\mathbf{v}_t||}, & \text{if } ||\mathbf{v}_t|| > s_{\max} \\ \mathbf{v}_t, & \text{otherwise} \end{cases}$$

Finally, to avoid collisions with the scene or objects we render a depth image using the z-buffer of OpenGL. If a collision occurs, the velocity is simply negated in a 'bounce', which simplifies the collision by assuming the surface normal is always the inverse of the velocity vector.

6.2. Scene and Trajectory Description

Listing 1 shows a sample trajectory file defining the scene and camera trajectory. We provide the WordNet id,

and also save the object height in meters and the full 3×4 transformation of the object.

```
layout_file:./bedroom/bedroom3_layout.obj
object
03938244/218f86362028a45b78f8b40f4a2ae98a
wnid
03938244
scale
0.416493
transformation
0.999238  -0.00604157 -0.0385491   1.46934
0.00627241 0.999963    0.00587011 -0.0346129
0.0385122 -0.00610744  0.999239   -1.00603

object
03938244/ac2477b9b5d3e6e6c7c8ce3bef5c2aa9
wnid
03938244
scale
0.169709
transformation
 0.505633   0.123627  0.853845  3.57641
-0.00155019 0.989809 -0.142395 -0.0223919
-0.862747   0.070676  0.500672 -0.377113
....

# Poses come in alternating pairs. With shutter open
# on the first line then shutter close on the next.
# Each line has a timestamp in seconds as well as
# the camera position and look at position both defined
# in world coordinates. The layout is as follows:
# time cam_x cam_y cam_z lookat_x lookat_y lookat_z

# frame rate (Hz):      25
# shutter duration (s): 0.0166667

0.0000 -2.157 1.234 2.384  -0.5645 2.491 0.5848
0.0167 -2.157 1.233 2.384  -0.5646 2.490 0.5847

0.0400 -2.156 1.232 2.384  -0.5647 2.489 0.5843
0.0567 -2.156 1.232 2.384  -0.5648 2.489 0.5841
```

Listing 1. Partial scene layout document after trajectory generation

7. Rendering photorealistic RGB frames

The rendering engine used was version of the Opposite Renderer[2] [16], a flexible open-source ray-tracer built on top of the NVIDIA OptiX framework. We added certain extra features such as phong specular materials, ground truth materials, and multiple photon maps which can be stored in CPU memory and swapped unto the GPU. Although there were other open-source alternatives that we considered *e.g.* POVRay, Blender and OpenGL, each one had their own limitations. For instance, though POVRay is able to use multi-threading on the CPU, it does not have GPU support. It is not easy to render high quality visual artefacts such as global illumination, caustics, and reflections and transparency in OpenGL and we did not find Blender as flexible for customised rendering as OptiX.

We do not have strict real-time constraints to produce photorealistic rendering, but the scale and quality of images required does mean the computational cost is an important factor to consider. Since OptiX allows rendering on the GPU it is able to fully utilise the parallelisation offered by modern day graphics cards. This framework also provides us with significant flexibility with our rendering pipeline, enabling us to obtain ground truth information of various kinds such as depth and object instance number quite conveniently. Moreover, in future it could also allow for more complicated BRDF surface properties to be easily modelled.

[2] http://apartridge.github.io/OppositeRenderer/

7.1. Photon Mapping

We use a process known as photon mapping to approximate the rendering equation. Our static scene assumption makes photon mapping particularly efficient as we can produce photon maps for a scene which are maintained throughout the trajectory. A good tutorial on photon mapping is given by its creator Jensen *et al.*[13].

(a) Direct & specular (b) Surface radiance (c) Combined

Figure 7. Comparison of direct and indirect photon mappings

As a quick summary, this technique works via a two-pass process. In the first pass, simulated photons are emitted from light sources accumulating global illumination information and storing this information in a photon map. In the second pass radiance information from this photon map is gathered along with direct illumination from light sources and specular reflections using ray-tracing to produce the final render, these separate and combined images can be seen in Figure 7. Normal ray-tracing allows for accurate reflections and transparency renderings such as those in Figure 8, but photon mapping provides a global illumination model that also approximates indirect illumination, colour-bleeding from diffuse surfaces, and caustics (this effect can be seen through the transparent shower enclosure).

(a) No reflections & transparency (b) With reflections & transparency

Figure 8. Reflections and transparency

7.2. Rendering Quality

Rendering over 5M images requires a significant amount of computation. We rendered our image on 4-12 GPUs for approximately one month. An important tradeoff in this cal-

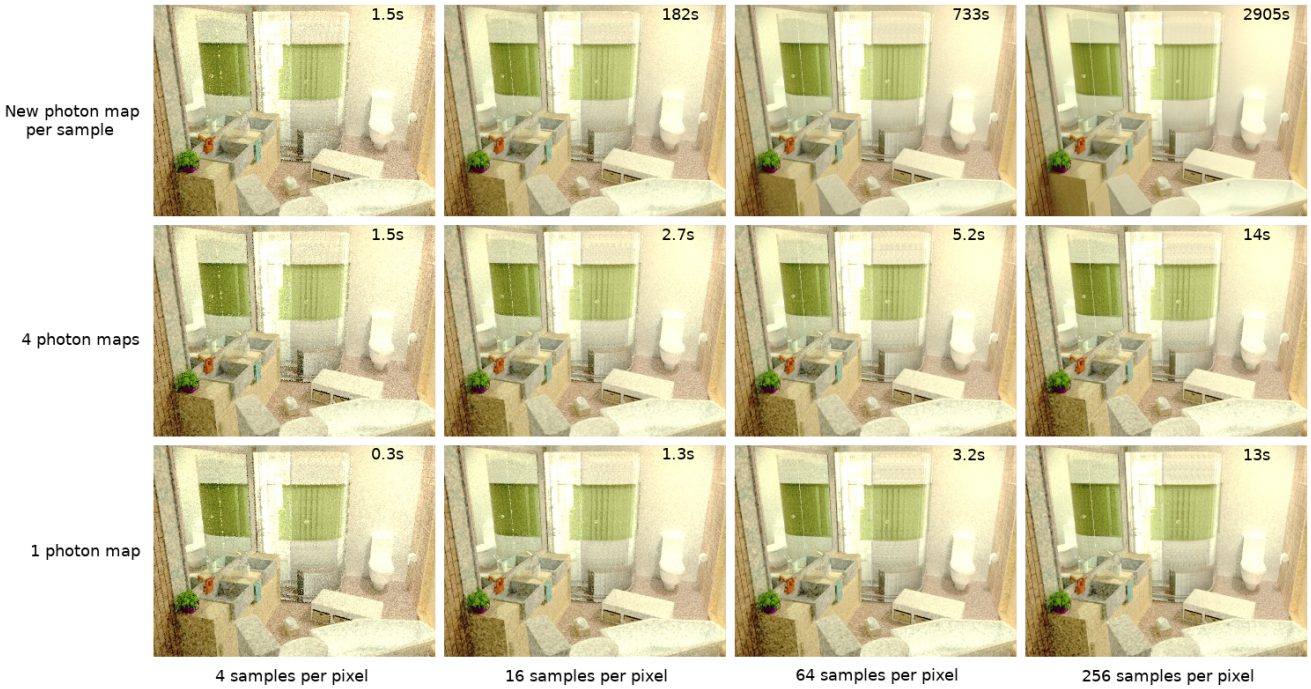


Figure 9. Trade off between rendering time and quality. Each photon map contains approximately 3M stored photons.

culation is between the quality of the renders and the quantity of images. Figure 9 shows two of the most important variables dictating this balance within our rendering framework. Our final dataset was rendered with 16 samples per pixel and 4 photon maps. This equates to approximately 3s per image on a single GPU.

An important threshold for the purposes of photon-mapping is that anymore than 8 photon maps exceeds the available 32GB memory. For less than 8 photon maps, we can precalculate the photon map once, and the computational cost is amortised across a trajectory. More than this and we must either store to disk or recompute a new set of photon maps for each frame in a trajectory.

7.3. Random Layout Textures and Lighting

To improve the variability within our 57 layouts, we randomly assign textures to each of its constituent components. Each layout object has a material type, which then gives a number of random texture images for that type. For example, we have a large number of different seamless wall textures, floor textures, and curtain textures.

As well as this, we add random lighting to the scene. A number of lights between 1 and 5 is selected. We have two types of lights, spherical orbs, which serve as point light sources and parallelograms which act as area lights. We randomly pick a hue and power of each light and then add them to a random location within the scene. We bias this location to be within the upper half of the scene.

(a) Version 1 (b) Version 2

Figure 10. Different renderings of the same geometric scene with different lighting and layout textures.

This approach allows an identical geometric layout to result in renders with different visual characteristics, see Figure 10. In this work we have only rendered a single version of each layout, however the availability of such pairs could prove an interesting facet of such randomisation in future.

7.4. Camera Model and CRF

Our camera is a simple global shutter pinhole model, with a focal length of 20cm, a horizontal FoV of 60°and vertical FoV of 40°. In order to make sure the rendered images are a faithful approximation to the real-world images, we also apply a non-linear Camera Response Function (CRF) that maps the irradiance to quantised brightness as in a real camera. We use a hard coded CRF in our case as shown in Figure 11, however it would be relatively simple to also randomise these parameters.

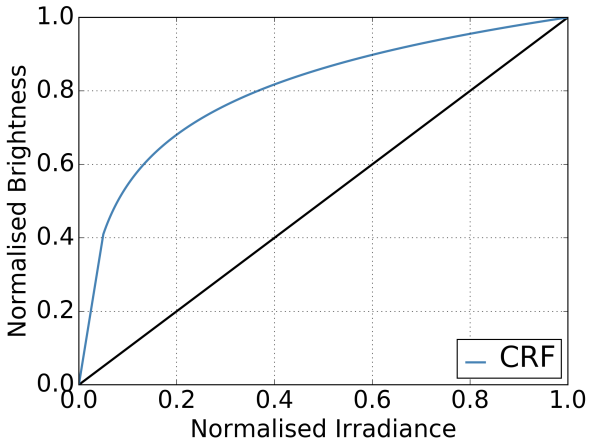
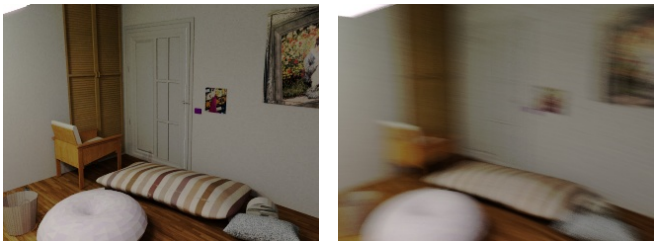


Figure 11. The Camera Response Function used by our renderer.

7.5. Motion Blur

For fast motion we integrate incoming rays throughout a shutter exposure to approximate motion blur — this can be efficiently performed within the rendering process by changing the poses from which samples are drawn for each pixel and integrating the irradiance value rather than for example averaging RGB values after rendering. To calculate the motion blur we draw linearly interpolated lines between the camera position and look-at position at both shutter open and shutter close. Then when rendering we uniformly sample a value from $\mathcal{U}(0,1)$ for different camera and "look-at" positions and then render in those sampled poses. For an example rendering using this technique see Figure 12.

The motion blur does not affect the ground truth outputs of depth or instance segmentations. For these images we set the pose to be the exact midpoint of the shutter exposure.



(a) without motion blur (b) with motion blur

Figure 12. Motion blur examples.

8. Conclusion

We have tackled the problem of producing realistic synthetic per-pixel labelled data, and we anticipate that the scale and quality of this dataset could help bridge the gap between simulations and reality and be suitable for domain adaption tasks [2]. We highlight some of the problems we have successfully tackled such as physically realistic scene layouts, sensible random camera trajectory generation, and photorealistic rendering. We also note certain areas where more work is needed. The primary challenges still to be faced include curating accurately metric scaled objects, and insuring accurate and consistent labels on object datasets. As mentioned in Figure 5, our automated systems mistakenly found mailboxes from ShapeNets when searching for the object category 'box'. This unfortunately led to large numbers of mailboxes in indoor scenes. At present even a synthetic dataset requires significant manual intervention in cases such as this to prevent mistakes.

Although immediately useful for many computer vision tasks, the present work has a number of limitations. Firstly, the scenes are static. This allows us to take advantage of efficient rendering techniques, but dynamic scenes, including soft bodies, would provide a more faithful representation of the real world. Secondly, we do not have certain intrinsic physical attributes of objects, such as mass or friction coefficients. Both of these limitations mean that the dataset is not immediately applicable to active agents in an interactive physically realistic dynamic scene. However, given enough compute power, our rendering pipeline could potentially provide rendering data on-the-fly for these sorts of systems.

The randomness inherent in our pipeline also allows for a continuous stream of unseen training examples, dynamically designed to target current limitations of a model being trained. In the future, it is likely that the generation of training data and the training of models will become more tightly interleaved, and the advantages of automatically generated training data becomes clear.

9. Acknowledgements

Research presented in this paper has been supported by Dyson Technology Ltd. We would also like to thank Patrick Bardow for providing optical flow code.

References

[1] M. Aubry, D. Maturana, A. Efros, B. Russell, and J. Sivic. Seeing 3D chairs: exemplar part-based 2D-3D alignment using a large dataset of CAD models. In *CVPR*, 2014.

[2] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, and D. Erhan. Domain Separation Networks. In *NIPS*, 2016.

[3] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An Information-Rich 3D Model Repository. *CoRR*, abs/1512.03012.

[4] C. R. de Souza, A. Gaidon, Y. Cabon, and A. M. López Peña. Procedural Generation of Videos to Train Deep Action Recognition Networks. *ArXiv e-prints*, 2016.

[5] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox.

FlowNet: Learning Optical Flow with Convolutional Networks. In *ICCV*, 2015.

[6] M. Fisher, D. Ritchie, M. Savva, T. Funkhouser, and P. Hanrahan. Example-based synthesis of 3d object arrangements. In *ACM SIGGRAPH Asia*, SIGGRAPH Asia, 2012.

[7] S. Gupta, P. A. Arbeláez, R. B. Girshick, and J. Malik. Aligning 3D models to RGB-D images of cluttered scenes. In *CVPR*, 2015.

[8] A. Handa, R. A. Newcombe, A. Angeli, and A. J. Davison. Real-Time Camera Tracking: When is High Frame-Rate Best? In *ECCV*, 2012.

[9] A. Handa, V. Pătrăucean, V. Badrinarayanan, S. Stent, and R. Cipolla. SceneNet: Understanding Real World Indoor Scenes With Synthetic Data. *arXiv preprint arXiv:1511.07041*, 2015.

[10] A. Handa, T. Whelan, J. B. McDonald, and A. J. Davison. A Benchmark for RGB-D Visual Odometry, 3D Reconstruction and SLAM. In *ICRA*, 2014.

[11] B.-S. Hua, Q.-H. Pham, D. T. Nguyen, M.-K. Tran, L.-F. Yu, and S.-K. Yeung. Scenenn: A scene meshes dataset with annotations. In *3DV*, 2016.

[12] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. *ArXiv e-prints*, Dec 2016.

[13] H. W. Jensen and N. J. Christensen. A practical guide to global illumination using photon maps. *Siggraph 2000 Course 8*, 2000.

[14] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[15] J. McCormac, A. Handa, A. J. Davison, and S. Leutenegger. SemanticFusion: Dense 3D Semantic Mapping with Convolutional Neural Networks. abs/1609.05130, 2016.

[16] S. A. Pedersen. Progressive photon mapping on gpus. *Master's Thesis, NTNU*, 2013.

[17] X. Peng, B. Sun, K. Ali, and K. Saenko. Learning Deep Object Detectors from 3D Models. *ArXiv e-prints*, 2014.

[18] W. Qiu and A. Yuille. UnrealCV: Connecting computer vision to unreal engine. *arXiv preprint arXiv:1609.01326*, 2016.

[19] S. R. Richter, V. Vineet, S. Roth, and V. Koltun. Playing for data: Ground truth from computer games. In *ECCV*, 2016.

[20] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. Lopez. The SYNTHIA Dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *CVPR*, 2016.

[21] M. Savva, A. X. Chang, G. Bernstein, C. D. Manning, and P. Hanrahan. On being the right scale: Sizing large collections of 3D models. In *SIGGRAPH Asia 2014 Workshop on Indoor Scene Understanding: Where Graphics meets Vision*, 2014.

[22] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from RGBD images. In *ECCV*, 2012.

[23] S. Song, S. P. Lichtenberg, and J. Xiao. SUN RGB-D: A RGB-D scene understanding benchmark suite. In *CVPR*, pages 567–576, 2015.

[24] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser. Semantic Scene Completion from a Single Depth Image. *arXiv preprint arXiv:1611.08974*, 2016.