NViSII: A Scriptable Tool for Photorealistic Image Generation

Nathan Morrical[1,2], **Jonathan Tremblay**[1], **Yunzhi Lin**[1,3],
Stephen Tyree[1], **Stan Birchfield**[1], **Valerio Pascucci**[2], **Ingo Wald**[1]
NVIDIA[1], University of Utah[2], Georgia Institute of Technology[3]

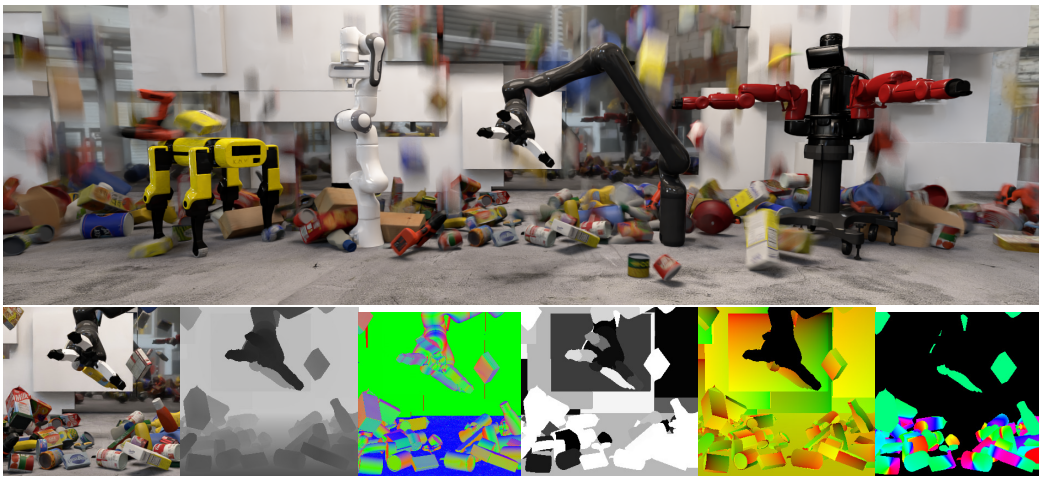

Figure 1: Our scriptable tool leverages hardware-accelerated path tracing to generate photorealistic images. Annotations are shown along the bottom row, from left to right: image with no motion blur, depth, surface normals, segmentation labels, texture coordinates, and optical flow.

ABSTRACT

We present a Python-based renderer built on NVIDIA's OptiX ray tracing engine and the OptiX AI denoiser, designed to generate high-quality synthetic images for research in computer vision and deep learning. Our tool enables the description and manipulation of complex dynamic 3D scenes containing object meshes, materials, textures, lighting, volumetric data (*e.g.*, smoke), and backgrounds. Metadata, such as 2D/3D bounding boxes, segmentation masks, depth maps, normal maps, material properties, and optical flow vectors, can also be generated. In this work, we discuss design goals, architecture, and performance. We demonstrate the use of data generated by path tracing for training an object detector and pose estimator, showing improved performance in sim-to-real transfer in situations that are difficult for traditional raster-based renderers. We offer this tool as an easy-to-use, performant, high-quality renderer for advancing research in synthetic data generation and deep learning. `https://github.com/owl-project/NVISII`

1 INTRODUCTION

For many computer vision tasks, it is challenging or even impossible to obtain labeled real-world images for use in training deep neural networks. For example, labeled ground truth of rare events like car crashes, or for dense high-dimensional data like optical flow vectors, are not easy to obtain. To overcome these limitations, researchers have explored synthetic data for a variety of applications: object pose estimation (Tremblay et al., 2018b; Denninger et al., 2019), depth estimation of transparent objects (Sajjan et al., 2019), scene segmentation (Handa et al., 2015; Roberts & Paczan, 2020), optical flow (Dosovitskiy et al., 2015), autonomous vehicles (Ros et al., 2016; Prakash et al., 2019), robotic control (Tobin et al., 2017), path planning and reasoning in 3D scenes (Kolve et al., 2017; Xia et al., 2020), and so forth.

Table 1: Related work compared to our proposed system. '✓' refers to fully supported, '✗' not supported, and '–' partially supported or it is complicated.

	AI2-Thor	iGibson	NDDS	Unity3D	Sapien	BlenderProc	Ours
path tracing	✗	✗	✗	✓	✓	✓	✓
easy installation	✓	✓	✗	–	–	–	✓
cross platform	✓	✓	✗	✓	✗	✓	✓
Python API	✓	✓	✗	✗	–	–	✓
headless rendering	✓	✓	✗	✗	✓	✓	✓

To generate such datasets, a variety of tools have been developed, including AI2-Thor (Kolve et al., 2017), iGibson (Xia et al., 2020), NDDS (To et al., 2018), Unity3D (Crespi et al., 2020), Sapien (Xiang et al., 2020), BlenderProc (Denninger et al., 2019), and others. See Table 1. Although AI2-Thor and iGibson come with powerful Python APIs, both of them are based on classic raster scanning. On the other hand, the more recent tools capable of photorealistic imagery via path tracing (such as Unity3D, Sapien, and BlenderProc) do not come with scriptable interfaces. To overcome this limitation, in this work we introduce NViSII: a scriptable tool for path-traced image generation. With our tool, users can construct and manipulate complex dynamic 3D scenes containing object meshes, materials, textures, lighting, volumetric data (*e.g.*, smoke), and cameras—all potentially randomized—using only Python code. This design choice ensures that users have full control, allowing scenes to be permuted on-the-fly according to the needs of the problem being considered. By leveraging path tracing, photorealistic images are produced, including physically-based materials, lighting, and camera effects. All of this can be achieved while maintaining interactive frame rates via NVIDIA's OptiX library and hardware accelerated ray tracing. Our tool is easily accessible via the `pip` packaging system[1].

We offer this tool to the community to enable researchers to procedurally manage arbitrarily complex scenes for photorealistic synthetic image generation. Our contributions are as follows: 1) An open source, Python-enabled ray tracer built on NVIDIA's OptiX, with a C++/CUDA backend, to advance sim-to-real and related research. 2) A demonstration of the tool's capabilities in generating synthetic images by training a DOPE pose estimator network (Tremblay et al., 2018c) and a 2D bounding detector (Zhou et al., 2019) for application to real images. 3) An investigation into how physically-based material definitions can increase a pose estimator's accuracy for objects containing specular materials.

2 Path tracer with Python interface

We developed the tool with three goals in mind: 1) ease of installation, 2) speed of development, and 3) rendering capabilities. For ease of installation, we ensured that the solution is accessible, open source, and cross platform (Linux and Windows), with pre-compiled binaries (thus obviating the need to build the tool) that are distributed using a package manager. For speed of development, the solution provides a comprehensive and interactive Python API for procedural scene generation and domain randomization. The tool does not require embedded interpreters, and it is well-documented with examples. Finally, we wanted a solution that supports advanced rendering capabilities, such as multi-GPU enabled ray tracing, physically-based materials, physically-based light controls, accurate camera models (including defocus blur and motion blur), native headless rendering, and various metadata (segmentation, motion vectors, optical flow, depth, surface normals, albedo, and so forth). See Figure 2 for some example renders.

2.1 Tool architecture

Our rendering tool follows a data driven entity component system (ECS) design. Such ECS designs are commonly used for game engines and 3D design suites, as they help keep the scene description intuitive and flexible by avoiding complex multiple-inheritance hierarchies required by object-oriented designs. This flat design allows for simpler procedural generation of scenes when compared to object-oriented designs.

[1] `pip install nvisii`

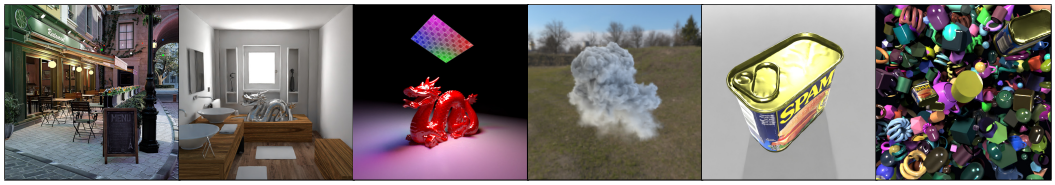

Figure 2: Example renders. TOP: bistro scene[1] (zoom for details), indoor scene[2], texture used as light. BOTTOM: smoke with 3D volume, reflective metal, domain randomization.

After initialization, the scene entities and components are created at runtime, as opposed to an offline scene description format like many prior solutions. See Figure 3 for a simple example. By enabling runtime scene edits, our solution more effectively leverages modern rendering performance capabilities during synthetic image generation. These components can be created using either the C++ or Python API, and as these components are created and connected together using entities, the out-of-date components are asynchronously uploaded to a collection of GPUs in a data-replicated pattern for interactive rendering.

Any given entity can be attached to any combination of the following component types: transform, mesh, material, light, camera, and volume. Transforms refer to the entity's SE(3) behaviour, *e.g.,* 3D location, 3D rotation, motion blur, *etc.* Meshes describe the 3D points and triangles to be rendered. Materials refer to the entity's physically-based rendering material definition. Lights define the energy emitted by the entity, which requires a mesh since point lights are physically impossible. Cameras refer to scene views, along with focal length and aperture. Volumes can be used in place of meshes to represent voxel data like smoke. In addition to the aforementioned components, textures can be used to drive any material properties, and when a texture is used in concert with a light, the texture RGB value defines the color of the light. Once connected to an entity, components immediately affect the appearance of the scene. These components can also be reused across entities as a basic form of instancing. Swapping out and modifying components additionally serves as an effective way to randomize the scene.

2.2 RENDERING CAPABILITIES

The material definition follows the Principled BSDF (Burley, 2015). This model consists of sixteen "principled" parameters, where all combinations result in physically-plausible, or at least well-behaved, results. Among these are base color, metallic, transmission, and roughness parameters. All parameters can be driven using either constant scalar values or texture components. With these parameters, scenes can represent smooth and rough plastics, metals, and dielectric materials (*e.g.*, glass). In addition, material components also accept an optional "normal map" texture to allow for more details on surface geometry.

```python
import nvisii
nvisii.initialize()
# Create camera
my_camera = nvisii.entity.create(
    name      = 'cam',
    transform = nvisii.transform.create('c_tfm'),
    camera    = nvisii.camera.create('c_cam')
)
my_camera.get_transform().look_at(
    eye = [3, 3, 3], at = [0, 0, 0], up = [0, 0, 1]
)
nvisii.set_camera_entity(my_camera)
# Create object
my_object = nvisii.entity.create(
    name      = 'obj'
)
my_object.set_transform(
    transform = nvisii.transform.create('o_tfm'),
    mesh      = nvisii.mesh.create_sphere('o_mesh'),
    material  = nvisii.material.create('o_mat')
)
material.get('o_mat').set_base_color([1, 0, 0])
# Render image
nvisii.render_to_file(
    width = 512, height = 512,
    samples_per_pixel = 1024,
    file_path = 'image.png'
)
nvisii.deinitialize()
```

Figure 3: A minimal Python script example

Direct and indirect lighting is computed using NVIDIA's OptiX ray tracing framework (Parker et al., 2010) through a standard path tracer with next event estimation. Multi-GPU support is enabled using the OptiX Wrapper Library (OWL) developed by Wald et al. (2020). These frameworks enable real-world effects like reflections, refractions, and global

[1] https://developer.nvidia.com/orca/amazon-lumberyard-bistro

[2] https://blendswap.com/blend/12584

Table 2: Sim-to-real 2D bounding box detection experiment on HOPE dataset.

Methods	AP	AP50	AP75
DOME + MESH + FAT	**47.8**	70.0	**47.7**
DOME + MESH	46.2	**70.1**	47.2
DOME	44.2	66.0	45.4
MESH	36.1	59.4	36.2
FAT	33.7	49.2	34.9

illumination, while simultaneously benefiting from hardware-accelerated ray tracing. Finally, the OptiX denoiser is used to more quickly obtain clean images for use in training.

3 EVALUATION

In this section we explore the use of our tool to generate synthetic data for training neural networks.

HOPE object detection. We trained CenterNet (Zhou et al., 2019) to perform 2D detection of known objects from the HOPE dataset (Tyree et al., 2019) consisting of 28 toy grocery items with associated 3D models. Since the dataset does not contain any training images, it is an ideal candidate for sim-to-real transfer. The test set contains 238 unique images with 914 unique object annotations. To train the detector we generated three datasets. 1) *DOME*, a domain randomization set of images similar to those generated by Tremblay et al. (2018c), consists of 3D objects flying in front of a real image background. 2) *MESH*, inspired by recent work by Hinterstoisser et al. (2019), consists of random object shapes as background. 3) *FAT*, similar to Tremblay et al. (2018b), allows objects to freely fall within a photorealistic scene. We simplify the prior work by using a dome texture to create photorealistic backgrounds and create natural lights, then apply a random floor texture onto the plane to simulate the surface. Please consult the supplemental section for greater details on the datasets, with examples.

We use a collection of 6k rendered images to train the object detector, and we compare using a single type of dataset with a mixture of datasets, while keeping the training size constant. We trained CenterNet[2] from scratch, using SGD optimizer with learning rate of 0.02 reduced by 10X at 81k and 108k steps, and a batch size of 128. The network was trained for 126k iterations on eight NVIDIA V100s for about 12 hours. Table 2 shows the results, specifically that the DOME dataset performs better than DR. These results also confirm the observation from Tremblay et al. (2018c) that mixing different dataset generation methods outperfoms using a single one.

Metallic material object pose estimator

Tremblay et al. (2018c) hypothesized that the somewhat disappointing pose estimation accuracy of the YCB potted meat model was caused by inaccurate synthetic representation of the object's real-world material properties. The object is metallic on the top and bottom, and is wrapped with a plastic-like label. Additionally, the original 3D model provided by YCB (Calli et al., 2015) has lighting conditions (highlights) baked into the model's texture. This results in unrealistic appearance, especially under variable lighting conditions. To test this hypothesis, we modified the original base color texture of the model to remove all baked highlights. Next, we manually segmented the different material properties of the object (specifically the metallic *vs.* non-metallic regions) to create a more physically-accurate material description. Similar to the previous experiment, we generated 60k domain randomized synthetic images for training using NViSII. Compared with the original DOPE weights available online, which scores 0.314 for area under the ADD threshold curve on YCB-video (Xiang et al., 2017), our proposed solution gets an improved 0.462.

4 CONCLUSION

We have presented an open-source Python-enabled ray tracer built on top of NVIDIA's OptiX with a C++/CUDA backend to advance sim-to-real and related research. The tool's design philosophy is easy install, accessible hardware requirements, enable scenes to be created through scripting, and rendering of photorealistic images. We release this tool in the hope that it will be helpful to the community.

[2]We used the repo at `github.com/FateScript/CenterNet-better`

REFERENCES

Brent Burley. Extending the Disney BRDF to a BSDF with integrated subsurface scattering. *Physically Based Shading in Theory and Practice SIGGRAPH Course*, 2015.

B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar. The YCB object and model set: Towards common benchmarks for manipulation research. In *Intl. Conf. on Advanced Robotics (ICAR)*, 2015.

Erwin Coumans and Yunfei Bai. PyBullet, a Python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2019.

Adam Crespi, Cesar Romero, Srinivas Annambhotla, Jonathan Hogins, and Alex Thaman. Unity perception, 2020. `https://blogs.unity3d.com/2020/06/10/`.

Maximilian Denninger, Martin Sundermeyer, Dominik Winkelbauer, Youssef Zidan, Dmitry Olefir, Mohamad Elbadrawy, Ahsan Lodhi, and Harinandan Katam. Blenderproc. *arXiv preprint arXiv:1911.01911*, 2019.

A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. Smagt, D. Cremers, and T. Brox. FlowNet: Learning optical flow with convolutional networks. In *ICCV*, 2015.

A. Handa, V. Pătrăucean, V. Badrinarayanan, S. Stent, and R. Cipolla. SceneNet: Understanding real world indoor scenes with synthetic data. In *arXiv 1511.07041*, 2015.

Stefan Hinterstoisser, Olivier Pauly, Hauke Heibel, Martina Marek, and Martin Bokeloh. An annotation saved is an annotation earned: Using fully synthetic training for object instance detection. *arXiv preprint arXiv:1902.09967*, 2019.

Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An interactive 3D environment for visual AI. *arXiv preprint arXiv:1712.05474*, 2017.

Steven G Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, and Austin Robison. OptiX: A General Purpose Ray Tracing Engine. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, 2010.

Aayush Prakash, Shaad Boochoon, Mark Brophy, David Acuna, Eric Cameracci, Gavriel State, Omer Shapira, and Stan Birchfield. Structured domain randomization: Bridging the reality gap by context-aware synthetic data. In *ICRA*, 2019.

Mike Roberts and Nathan Paczan. Hypersim: A photorealistic synthetic dataset for holistic indoor scene understanding, 2020. URL `https://arxiv.org/pdf/2011.02523.pdf`.

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.

Shreeyak S Sajjan, Matthew Moore, Mike Pan, Ganesh Nagaraja, Johnny Lee, Andy Zeng, and Shuran Song. Cleargrasp: 3d shape estimation of transparent objects for manipulation. *arXiv preprint arXiv:1910.02550*, 2019.

Thang To, Jonathan Tremblay, Duncan McKay, Yukie Yamaguchi, Kirby Leung, Adrian Balanon, Jia Cheng, and Stan Birchfield. NDDS: NVIDIA deep learning dataset synthesizer, 2018. `https://github.com/NVIDIA/Dataset_Synthesizer`.

Joshua Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IROS*, 2017.

Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. In *CVPR Workshop on Autonomous Driving (WAD)*, 2018a.

Jonathan Tremblay, Thang To, and Stan Birchfield. Falling things: A synthetic dataset for 3D object detection and pose estimation. In *CVPR Workshop on Real World Challenges and New Benchmarks for Deep Learning in Robotic Vision*, 2018b.

Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. Deep object pose estimation for semantic robotic grasping of household objects. In *CoRL*, 2018c.

Stephen Tyree, Jonathan Tremblay, Thang To, Jia Cheng, Terry Mossier, and Stan Birchfield. 6-DoF pose estimation of household objects for robotic manipulation: an accessible dataset and benchmark. In *ICCV Workshop on Recovering 6D Object Pose*, 2019.

I. Wald, N. Morrical, and E. Haines. OWL – The OptiX 7 Wrapper Library, 2020. URL `https://github.com/owl-project/owl`.

Fei Xia, William B Shen, Chengshu Li, Priya Kasimbeg, Micael Edmond Tchapmi, Alexander Toshev, Roberto Martín-Martín, and Silvio Savarese. Interactive gibson benchmark: A benchmark for interactive navigation in cluttered environments. *IEEE Robotics and Automation Letters*, 5(2): 713–720, 2020.

Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. SAPIEN: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11097–11107, 2020.

Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox. PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes. In *arXiv 1711.00199*, 2017.

Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. In *arXiv preprint arXiv:1904.07850*, 2019.

SUPPLEMENTAL

In this section we explore the use of our tool to generate synthetic data for training neural networks. We first explore the problem of 2D detections and then explore how different material propreties can be used to enhance pose estimation of metallic objects.

HOPE OBJECT DETECTION

Using NViSII we generated 3 different dataset types: *DOME*, *MESH*, and *FAT*.

DOME. Tobin et al. (2017) demonstrated that *domain randomization* can be used to train a model fully on synthetic data to be used on real data. Tremblay et al. (2018a) extended the framework for 2D car detection. In this work we generate similar images, see Figure 4 first row for examples. In our tool, we leverage its capacity to use HDR dome texture to illuminate a scene, which offers a natural looking light that mimics interreflections. Similar to Tremblay et al. (2018a) we randomize the object poses within a volume and add flying distractors. See Figure 4 first row.

MESH. Hinterstoisser et al. (2019) introduced the concept that using other 3D meshes as background could potentially lead to better sim-to-real. As such, we use our tool capacity to generate random 3D meshes and applied random material to these, we used around a 1000 moving meshes as background as seen in Figure 4 second row. For illumination we used 2 to 6 random lights (random color and intensity) placed behind the camera to generate random light context. See Figure 4 second row.

FAT. Tremblay et al. (2018b) introduced a dataset where objects were allowed to freely fall into a complex 3D scene. Following on that work Tremblay et al. (2018c) proposed to mix falling dataset and domain randomization dataset to solve the sim-to-real problem. As such we integrated our tool with PyBullet (Coumans & Bai, 2016–2019) to let objects fall onto a simple plane, see Figure 4 third row. We simplify the prior work were we use a dome texture to create photorealistic backgrounds and create natural lights, and we then simply apply a random floor texture onto the plane to simulate the surface. See Figure 4 third row. These renders are also similar to images generated by BlenderProc (Denninger et al., 2019). Detections from the better model trained on using all the presented datasets can be seen on Figure 5.

METALLIC MATERIAL OBJECT POSE ESTIMATOR

Figure 6 compares the physically-correct material *vs.* the original material with baked lighting. These images demonstrate how the metallic texture causes a highlight from the lights along the reflection direction under certain view angles, whereas the original texture is flat and contains unnatural highlights that do not match the surrounding synthetic scene. Using the approach proposed by Hinterstoisser et al. (2019) we generated 60k domain randomized synthetic images using path raytracer for training with random meshes and random material as background. Two to six lights were placed randomly behind the camera with randomized position, temperature, and intensity.

DOME (Tremblay et al., 2018a)

MESH (Hinterstoisser et al., 2019)

FAT (Tremblay et al., 2018b)

Figure 4: Training images for pose estimation and object detection. For each row, images were generated using a procedure similar to that of the reference shown.

Figure 5: Detections on real HOPE images (Tyree et al., 2019) using CenterNet (Zhou et al., 2019) trained on synthetic data generated by NViSII. On these images, 68.2% of objects were detected, with no false positives.

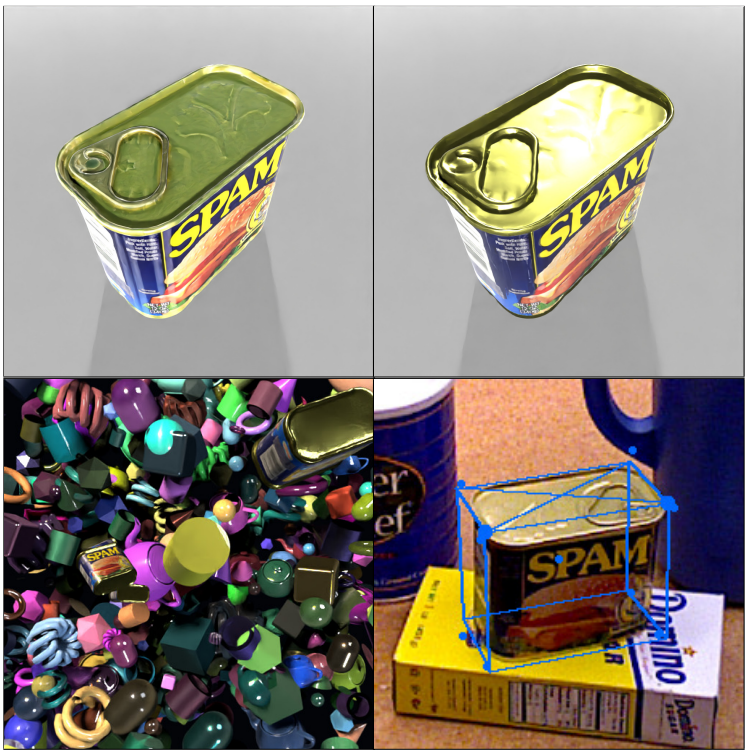

Figure 6: TOP-LEFT: Original YCB texture with baked-in highlights. TOP-RIGHT: Corrected flat texture and properly associated metallic material used by our tool. BOTTOM-LEFT: DR image rendered by our tool. BOTTOM-RIGHT: Pose prediction from our trained model. (Note that the model trained on the original YCB texture does not detect the meat can in this image.)