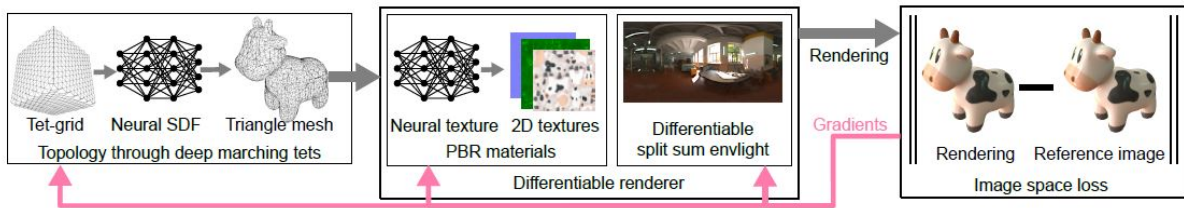

nvdiffrrec



Joint optimization of topology, materials and lighting from multi-view image observations as described in the paper [Extracting Triangular 3D Models, Materials, and Lighting From Images](#).

For differentiable marching tetrahedons, we have adapted code from NVIDIA's Kaolin: A Pytorch Library for Accelerating 3D Deep Learning Research.

News

- **2023-10-20** : We added a version of the renderutils library written in slangpy to leverage the autodiff capabilities of slang instead of CUDA extensions with manually crafted forward and backward passes. This simplifies the code substantially, with the same runtime performance as before. This version is available in the [slang](#) branch of this repo.
- **2023-09-15** : We added support for the FlexiCubes isosurfacing technique. Please see the config [configs/bob_flexi.json](#) for a usage example, and refer to the FlexiCubes documentation for details.

Citation

```
1 @inproceedings{Munkberg_2022_CVPR,  
2   author    = {Munkberg, Jacob and Hasselgren, Jon and Shen,  
3               Tianchang and Gao, Jun and Chen, Wenzheng  
4               and Evans, Alex and Müller, Thomas and Fidler,  
5               Sanja},  
6   title     = "{Extracting Triangular 3D Models, Materials, and  
7               Lighting From Images}",  
8   booktitle = {Proceedings of the IEEE/CVF Conference on Computer  
9               Vision and Pattern Recognition (CVPR)},  
10  month     = {June},  
11  year      = {2022},  
12  pages     = {8280-8290}  
13 }
```

Licenses

Copyright © 2022, NVIDIA Corporation. All rights reserved.

This work is made available under the Nvidia Source Code License.

For business inquiries, please visit our website and submit the form: NVIDIA Research Licensing.

Installation

Requires Python 3.6+, VS2019+, Cuda 11.3+ and PyTorch 1.10+

Tested in Anaconda3 with Python 3.9 and PyTorch 1.10

One time setup (Windows)

Install the Cuda toolkit (required to build the PyTorch extensions). We support Cuda 11.3 and above. Pick the appropriate version of PyTorch compatible with the installed Cuda toolkit. Below is an example with Cuda 11.6

```
1 conda create -n dmodel python=3.9
2 activate dmodel
3 conda install pytorch torchvision torchaudio cudatoolkit=11.6 -c
  pytorch -c conda-forge
4 pip install ninja imageio PyOpenGL glfw xatlas gdown
5 pip install git+https://github.com/NVlabs/nvdiffrast/
6 pip install --global-option="--no-networks" git+https://github.com/
  NVlabs/tiny-cuda-nn#subdirectory=bindings/torch
7 imageio_download_bin freeimage
```

Every new command prompt

```
activate dmodel
```

Examples

Our approach is designed for high-end NVIDIA GPUs with large amounts of memory. To run on mid-range GPU's, reduce the batch size parameter in the .json files.

Simple genus 1 reconstruction example:

```
1 python train.py --config configs/bob.json
```

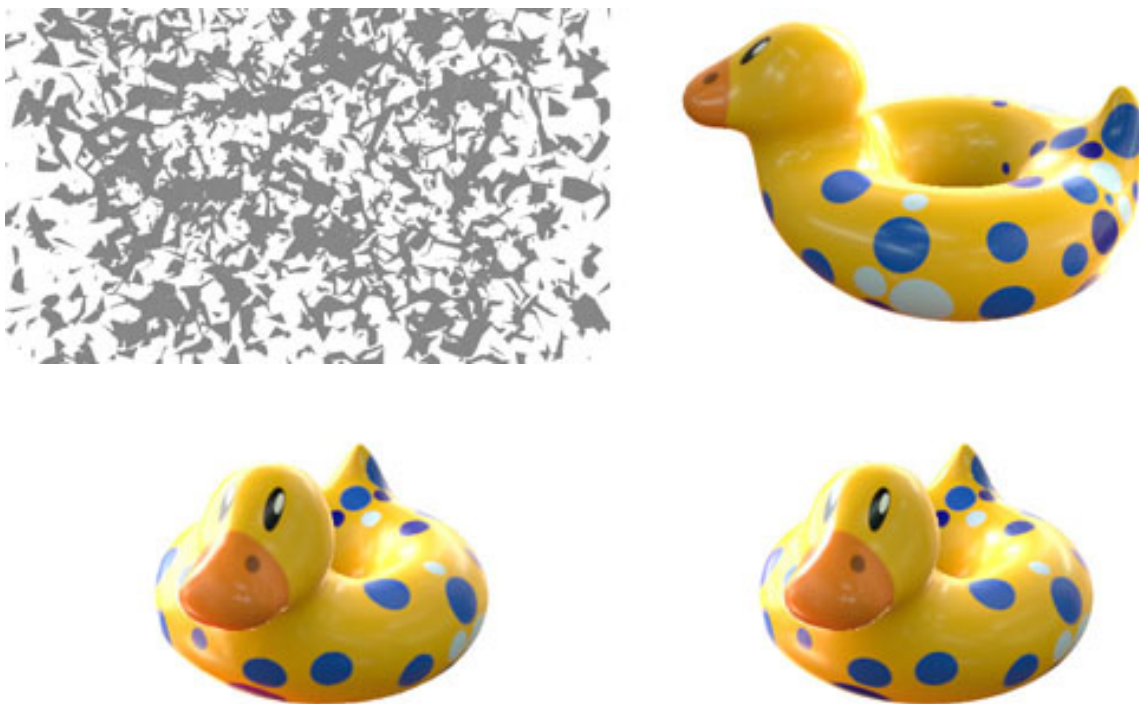
Visualize training progress (only supported on Windows):

```
1 python train.py --config configs/bob.json --display-interval 20
```

Multi GPU example (Linux only. *Experimental: all results in the paper were generated using a single GPU*), using PyTorch DDP

```
1 torchrun --nproc_per_node=4 train.py --config configs/bob.json
```

Below, we show the starting point and the final result. References to the right.



The results will be stored in the `out` folder. The Spot and Bob models were created and released into the public domain by Keenan Crane.

Included examples

- `spot.json` - Extracting a 3D model of the spot model. Geometry, materials, and lighting from image observations.
- `spot_fixlight.json` - Same as above but assuming known environment lighting.
- `spot_metal.json` - Example of joint learning of materials and high frequency environment lighting to showcase split-sum.
- `bob.json` - Simple example of a genus 1 model.

Datasets

We additionally include configs (`nerf_*.json`, `nerd_*.json`) to reproduce the main results of the paper. We rely on third party datasets, which are courtesy of their respective authors. Please note that individual licenses apply to each dataset. To automatically download and pre-process all datasets, run the `download_datasets.py` script:

```
1 activate dmodel
2 cd data
3 python download_datasets.py
```

Below follows more information and instructions on how to manually install the datasets (in case the automated script fails).

NeRF synthetic dataset Our view interpolation results use the synthetic dataset from the original NeRF paper. To manually install it, download the NeRF synthetic dataset archive and unzip it into the `nvdiffrrec/data` folder. This is required for running any of the `nerf_*.json` configs.

NeRD dataset We use datasets from the NeRD paper, which features real-world photogrammetry and inaccurate (manually annotated) segmentation masks. Clone the NeRD datasets using git and rescale them to 512 x 512 pixels resolution using the script `scale_images.py`. This is required for running any of the `nerd_*.json` configs.

```
1 activate dmodel
2 cd nvdiffrrec/data/nerd
3 git clone https://github.com/vork/ethiopianHead.git
4 git clone https://github.com/vork/moldGoldCape.git
5 python scale_images.py
```

Server usage (through Docker)

- Build docker image.

```
1 cd docker
2 ./make_image.sh nvdiffrrec:v1
```

- Start an interactive docker container: `docker run --gpus device=0 -it --rm -v /raid:/raid -it nvdiffrrec:v1 bash`
- Detached docker: `docker run --gpus device=1 -d -v /raid:/raid -w=[path to the code] nvdiffrrec:v1 python train.py --config configs/bob.json`