Taskonomizer Demo!

Installation

We provide a docker that contains all the necessary libraries. It's simple to install and run.

1. Clone the repo:

```
git clone ...
cd taskonomizer
```

2. Simply run:

```
./install.sh
```

Now the installation is complete, and all the necessary libraries are installed in the docker.

Run Demo

Next, we show how to generate data for different tasks. You can see the list of tasks in the table below:

```
RGB (8-bit)                  Surface-Normals (8-bit)     Curvature (8-bit)
Reshading (8-bit)            Depth ZBuffer (16-bit)      Depth Euclidean (16-bit)
Edge 2D (16-bit)             Edge 3D (16-bit)            Keypoints 2D (16-bit)
Keypoints 3D (16-bit)        Segmentation 2D (8-bit)     Segmentation 2.5D (8-bit)
Semantic Segmentation (8-bit)
```

Task Name	Output Dimension	Number of Bits
RGB	(512, 512, 3)	8-bit
Surface Normals	(512, 512, 3)	8-bit
Curvature	(512, 512, 3)	8-bit
Reshading	(512, 512, 3)	8-bit
Depth ZBuffer	(512, 512)	16-bit
Depth Euclidean	(512, 512)	16-bit
Edge 2D	(512, 512)	16-bit
Edge 3D	(512, 512)	16-bit
Keypoints 2D	(512, 512)	16-bit
Keypoints 3D	(512, 512)	16-bit
Segmentation 2D	(512, 512)	8-bit
Segmentation 2.5D	(512, 512)	8-bit

Task Name	Output Dimension	Number of Bits
Semantic Segmentation	(512, 512, 3)	8-bit

```
model_path
|   mesh.ply
|   camera_poses.json
└─── point_info
└─── rgb
└─── normals
└─── depth_zbuffer
|   ...
|
└─── pano
|   └─── point_info
|   └─── rgb
|   └─── normal
|   └─── depth_euclidean
```

```
model_path/
    mesh.ply
    camera_poses.json
    point_info/
    rgb/
    normals/
    depth_zbuffer/
    ...
    pano/
        point_info/
        rgb/
        normal/
        depth_euclidean/
```

To run the taskonomizer for a specific task:

```
./taskonomizer.sh --model_path=$PATH_TO_FOLDER_OF_MODEL --task=$TASK with {SETTING=VALUE}*
```

The `--model_path` tag specifies the path to the folder containing the mesh, where the data from all other tasks will also be saved. The `--task` tag specifies the target task for generating the data. You can generate all the tasks using `--task=all` . You can also specify different setting values for each task in the command. The list of all settings defined for different tasks are in `settings.py`.

Now, we run taskonomizer for some sample tasks.

1. Generating points:

In order to generate the points, we need camera poses. You can either generate new camera poses or read them from a `json` file.

Read camera poses from json file:

The following command generates 100 points using the camera poses defined in `camera_poses.json`.

```
./taskonomizer.sh --model_path=replica/room_1 --task=points \
   with GENERATE_CAMERAS=False CAMERA_POSE_FILE=camera_poses.json \
        MIN_VIEWS_PER_POINT=3  NUM_POINTS=100 \
        MODEL_FILE=mesh.ply    POINT_TYPE=CORRESPONDENCES
```

In order to read the camera info from the json file, you should specify `GENERATE_CAMERAS=False`. This json file should contain location and quaternion rotation (wxyz) for a list of cameras.
Below, you can see how this information should be saved for each camera.

```
{
    "camera_id": "0000",
    "location": [0.2146, 1.2829, 0.2003],
    "rotation_quaternion": [0.0144, -0.0100, -0.0001,-0.9998]
}
```

You can specify the type of generated points. `POINT_TYPE=CORRESPONDENCES` is used for generating fixated photos. Switch to `POINT_TYPE=SWEEP` in case you want to generate panoramas.

Generate new camera poses:

You can generate new camera poses before generating the points using `GENERATE_CAMERAS=True`.

```
./taskonomizer.sh --model_path=replica/room_1 --task=points \
   with GENERATE_CAMERAS=True     MODEL_FILE=mesh.ply \
        MIN_CAMERA_HEIGHT=1       MAX_CAMERA_ROLL = 10 \
        MIN_CAMERA_DISTANCE = 1   MIN_CAMERA_DISTANCE_TO_MESH=0.3 \
        MIN_VIEWS_PER_POINT=3     POINTS_PER_CAMERA=5 \
        POINT_TYPE=CORRESPONDENCES
```

Camera locations are sampled inside the mesh using **Poisson Disc Sampling**. Minimum distance between cameras is specified by `MIN_CAMERA_DISTANCE`. `MIN_CAMERA_DISTANCE_TO_MESH` also defines the minimum distance of each camera to the closest point of the mesh. More camera settings are defined in **settings.py**.
You can specify the number of generated points either by `NUM_POINTS` or `NUM_POINTS_PER_CAMERA`. In case we have `NUM_POINTS=None`, the number of generated points will be **NUM_POINTS_PER_CAMERA * number of cameras**.

2. Surface Normals:

In order to generate surface normal images simply run:

```
./taskonomizer.sh --model_path=replica/apartment_0 --task=normal \
    with MODEL_FILE=mesh.ply  CREATE_FIXATED=True
```

This will generate fixated photos.
In case you want to generate panoramas :

```
./taskonomizer.sh --model_path=replica/apartment_0 --task=normal \
    with MODEL_FILE=mesh.ply CREATE_FIXATED=False CREATE_PANOS=True
```

3. Depth ZBuffer:

To generate depth zbuffer images :

```
./taskonomizer.sh --model_path=replica/apartment_0 --task=depth_zbuffer \
    with MODEL_FILE=mesh.ply  DEPTH_ZBUFFER_MAX_DISTANCE_METERS=16
```

ZBuffer depth is defined as the distance to the camera plane.
You can specify the depth sensitivity by `DEPTH_ZBUFFER_MAX_DISTANCE_METERS`. With 16m and and 16-bit images, we will have depth sensitivity equal to $$\frac{16}{2^{16}} = \frac{1}{4096}$$ meters.

4. Depth Euclidean:

To generate depth euclidean images :

```
./taskonomizer.sh --model_path=replica/apartment_0 --task=depth_euclidean \
    with MODEL_FILE=mesh.ply  DEPTH_EUCLIDEAN_MAX_DISTANCE_METERS=16
```

Euclidean depth is measured as the distance from each pixel to the camera's optical center.

5. Reshading:

To generate reshading images :

```
./taskonomizer.sh --model_path=replica/apartment_0 --task=reshading \
    with MODEL_FILE=mesh.ply  LAMP_ENERGY=2.5
```

6. Keypoints 2D:

To generate 2D keypoints images :

```
./taskonomizer.sh --model_path=replica/apartment_0 --task=keypoints2d with MODEL_FILE=mesh.ply
```

7. Keypoints 3D:

To generate 3D keypoints images :

```
./taskonomizer.sh --model_path=replica/apartment_0 --task=keypoints3d \
    with MODEL_FILE=mesh.ply  KEYPOINT_SUPPORT_SIZE=0.3
```

`KEYPOINT_SUPPORT_SIZE` specifies the diameter of the sphere around each 3D point that is used to decide if the point should be an
interest point. 0.3 meters is suggested for indoor spaces.

8. Edge 2D:

To generate 2D edge images :

```
./taskonomizer.sh --model_path=replica/apartment_0 --task=edge2d \
    with MODEL_FILE=mesh.ply  CANNY_RGB_BLUR_SIGMA=3.0
```

`CANNY_RGB_BLUR_SIGMA` specifies the sigma in Gaussian filter used in **Canny edge detector**.

9. Edge 3D:

To generate 3D edge images :

```
./taskonomizer.sh --model_path=replica/apartment_0 --task=edge3d with MODEL_FILE=mesh.ply
```

10. Segmentation 2D:

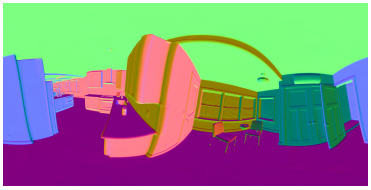
To generate 2D segmentation images :

```
./taskonomizer.sh --model_path=replica/apartment_0 --task=segment2d with MODEL_FILE=mesh.ply
```

11. Segmentation 2.5D:

To generate 2.5D segmentation images :

```
./taskonomizer.sh --model_path=replica/apartment_0 --task=segmentation25d with MODEL_FILE=mesh.ply
```

Surface Normals	Euclidean Depth	Semantics
 />