CV HW5

Lifan Yu (lifany)

December 2023

Question 1: Loading and nderstanding the dataset

Setup

```
%matplotlib inline

import numpy as np
import torch
import matplotlib.pyplot as plt
import cv2
import sys
import os

def show_mask(mask, ax, random_color=False):
    # This function is used to visualize the mask on the image in a matplotlib axis.
    # bool mask: (H, W). True for each pixel that belongs to the object.
    # ax: matplotlib axis
    # random_color: if True, use a random color for the mask. Otherwise, use blue.

    if random_color:
        color = np.concatenate([np.random.random(3), np.array([0.6])], axis=0)
    else:
        color = np.array([30/255, 144/255, 255/255, 0.6])
    h, w = mask.shape[-2:]
    mask_image = mask.reshape(h, w, 1) * color.reshape(1, 1, -1)
    ax.imshow(mask_image)

def show_box(box, ax):
    # This function is used to visualize the bounding box on the image in a matplotlib axis.
    # box: (4,) array. [x0, y0, x1, y1]
    # (x0, y0): top-left corner
    # (x1, y1): bottom-right corner
    # ax: matplotlib axis

    x0, y0 = box[0], box[1]
    w, h = box[2] - box[0], box[3] - box[1]
    ax.add_patch(plt.Rectangle((x0, y0), w, h, edgecolor='green', facecolor=(0,0,0,0), lw=2))
```

1.1: Comopute the camera matrix K

```
import json

# dataset class provided to load extrinsics, intrinsics and image paths.
class Dataset:
    def __init__(self, json):
        self.json = json # the json file containing the extrinsics, intrinsics and image paths.
        self.load_extrinsics()
        self.load_intrinsics()
        self.compute_intrinsics()

    def load_extrinsics(self):
        # This function loads the extrinsics parameters from the json file.

        with open(self.json) as f:
            self.data = json.load(f)
        self.frames = self.data['frames'] # 'frames' in the json contains the extrinsics and image
                                                path for each image.
        self.transforms = np.array([frame['transform_matrix'] for frame in self.frames]) # extrinsic
                                                matrix for each image, shape (N, 4, 4)
        self.file_paths = np.array([frame['file_path'] for frame in self.frames]) # path to each
                                                image, shape (N,)

    def load_intrinsics(self):
        # This function loads the intrinsics parameters from the json file.
        self.f_x = self.data['fl_x'] # focal length in x
        self.f_y = self.data['fl_y'] # focal length in y
        self.w = self.data['w'] # image width
        self.h = self.data['h'] # image height
        self.cx = self.data['cx'] # principal point in x
        self.cy = self.data['cy'] # principal point in y

    def compute_intrinsics(self):
        self.K = None # K: the intrinsic matrix, shape (3, 3)
        # compute the K matrix form the intrinsic parameters computed in load_intrinsics() : [2 pts]
        # TODO: YOUR CODE HERE
        self.K = np.array([[self.f_x, 0, self.cx],
                           [0, self.f_y, self.cy],
                           [0, 0, 1]])


dataset = Dataset('images/dataset/transforms_train.json') # load the dataset
np.set_printoptions(precision=3, suppress=True) # do NOT remove this line when you print matrices
                                                for grading

print('Shape of extrinsic matrices: {}'.format(dataset.transforms.shape)) # all extrinsic matrices,
                                                shape (N, 4, 4)
#TODO: print the intrinsic matrix and add to your gradescope submission.
print('K matrix {}'.format(dataset.K)) # The intrinsic matrix K you computed.
```

Output

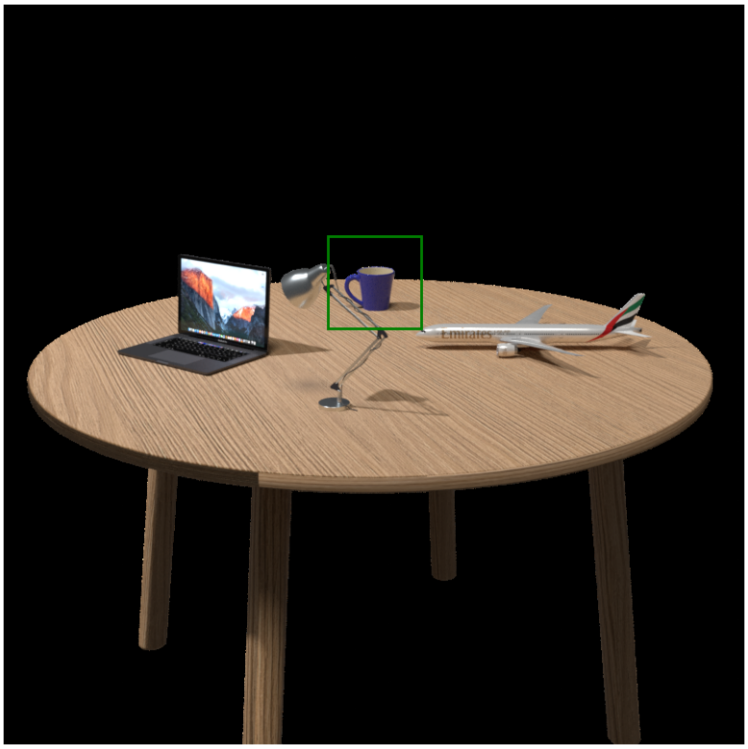
```
Shape of extrinsic matrices: (100, 4, 4)
K matrix [[1111.111    0.     400.   ]
 [   0.    1111.111  400.   ]
 [   0.       0.       1.   ]]
```

Visualize the dataset

```
image = cv2.imread(os.path.join('images/dataset',dataset.file_paths[0]))
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(10,10))
plt.imshow(image)
plt.axis('on')
plt.show()


def depthmap_viz(depth,min_d=0.0,max_d=3.5):
    # depth: (H,W,3) - depth map, every channel contains the same depth values for that pixel. 2
                                                    channels are redundant.

    # min_d: minimum depth value to visualize
    # max_d: maximum depth value to visualize

    depth = np.clip(depth,min_d,max_d)

    depth = (depth-min_d)/(max_d - min_d)

    image = depth

    plt.clf()
    plt.imshow(depth,cmap='magma', vmin=min_d,vmax=max_d)

depth_location = 'images/dataset/train/depth.npy' # location of ground truth depth maps.
depths = np.load(depth_location) # load the depth maps

depthmap_viz(depths[0]) # visualize the first depth map
plt.show() # show the plot

import sys
from segment_anything import sam_model_registry, SamPredictor

sam_checkpoint = "ckpts/sam_vit_h_4b8939.pth" # the checkpoint loaded in the setup section.
model_type = "vit_h"

device = "cuda" # loading to GPU.

sam = sam_model_registry[model_type](checkpoint=sam_checkpoint)
sam.to(device=device)

predictor = SamPredictor(sam)
```

1 Question 2: Designing our prompt

```
#TODO: YOUR CODE HERE
# add this plot to gradescope submission.
input_box = np.array([350, 250, 450, 350]) # choose correct bounding box [2 pts]

plt.figure(figsize=(10, 10))
plt.imshow(image)
show_box(input_box, plt.gca())
plt.axis('off')
plt.show()
```

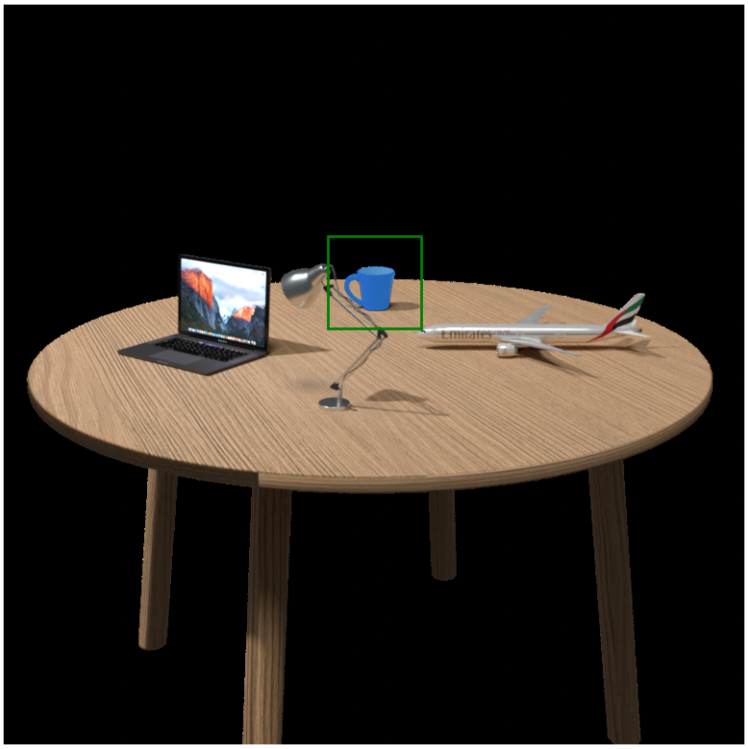


```
# Here we're running SAM on the image with the bounding box.
predictor.set_image(image) # loading the image to the predictor.
masks, _, _ = predictor.predict(
    point_coords=None,
    point_labels=None,
    box=input_box[None, :],
    multimask_output=False,
)
# Calling the predictor with the bounding box.
# You will not need to change any of the other arguments in this homework.
```

Continued on the next page ————

Visualizing the mask

```
mask = masks[0]
h, w = mask.shape[-2:]
mask_image = mask.reshape(h, w, 1)

plt.figure(figsize=(10, 10))
plt.imshow(image)
show_mask(mask, plt.gca())
show_box(input_box, plt.gca())
plt.axis('off')
plt.show()
```



Question 3: Project to 3D

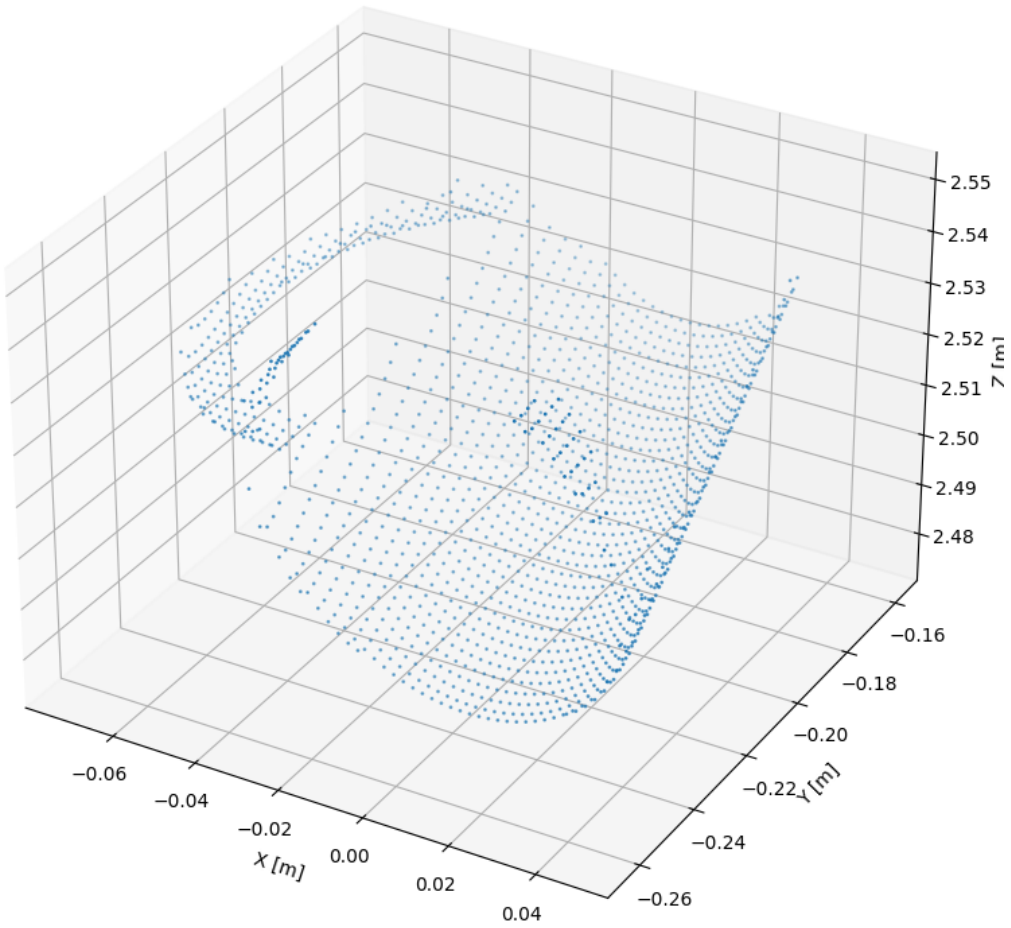
3.1: Image frame to camera frame

```python
def img2cam(points, K, depths=None):
    # project the points from image coordinates to camera coordinates [5 pts]
    cam_3d = None

    # TODO: YOUR CODE HERE
    # (1) Use the intrinsic matrix K to convert the points from image coordinates to a point cloud
                                                        in the camera frame.
    # (2) Normalize the points to a plane with z=1.
    # (3) Use depths to scale the points to be at the correct distance from the camera.
    points[:, [1, 0]] = points[:, [0, 1]] # matrix coordinates xy = image coordinates yx
    pointsT = np.vstack((points.T, np.ones(len(points))))
    points3dT = np.matmul(np.linalg.inv(K), pointsT)
    points3dT[-1, :] = depths.T
    points3dT[:2, :] = points3dT[:2, :]*points3dT[-1, :]
    cam_3d = points3dT

    return cam_3d

def filter_points(coords,depths,thresh=2.55):
    # filter out points that are too far away in the first mask, this first mask will be very
                                                        important! [2 pts]
    # return filtered coords and depths
    # don't make it too complicated, this should be a one-liner.
    # TODO: YOUR CODE HERE

    in_coords, _ = np.where(depths<=thresh)
    coords_f = coords[in_coords]
    depths_f = depths[in_coords]

    return coords_f, depths_f

def mask2cam(mask, K, depths,thresh=None):
    # project mask points to camera frame [3 pts]
    # steps todo:
    # (1) get all coordinates where the mask is True, this should be N x 2
    # (2) get the depth values for these coordinates, Nx1
    # (3) call filter_points to filter out points that are too far away, with depth above the
                                                        treshold.
    # Here far away means the depth is above a certain threshold.
    # (4) call img2cam to convert the points to camera frame using intrinsics and depth.
    # TODO: YOUR CODE HERE
    Xs, Ys, _ = np.where(mask == True)
    coords = np.hstack((Xs.reshape(-1, 1), Ys.reshape(-1, 1)))

    l = len(coords)
    Ds = np.zeros((l, 1))
    for i in range(l):
        Ds[i] = depths[coords[i][0], coords[i][1]][0]

    coords_f, Ds_f = filter_points(coords,Ds,thresh)
    points3d = img2cam(coords_f, K, Ds_f)

    return points3d
```

```python
cam_pnts_3d = mask2cam(mask_image,dataset.K,depths[0],thresh=2.55)
print(cam_pnts_3d)
```

Continued on the next page

```python
def viz_pts_3d(pts,xrange=None,yrange=None,zrange=None,title=None):
    # viz the 3D points
    fig = plt.figure(figsize=(10,10))
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(pts[0,:],pts[1,:],pts[2,:],s=1)
    ax.set_xlabel('X [m]')
    ax.set_ylabel('Y [m]')
    ax.set_zlabel('Z [m]')

    if xrange is not None:
        ax.set_xlim(xrange)
    if yrange is not None:
        ax.set_ylim(yrange)
    if zrange is not None:
        ax.set_zlim(zrange)

    if title is not None:
        ax.set_title(title)
    plt.show()

np.save('cam_pnts_3d.npy',cam_pnts_3d)

#TODO: add this plot to gradescope submission
viz_pts_3d(cam_pnts_3d)
```



3.2: Camera frame to world frame

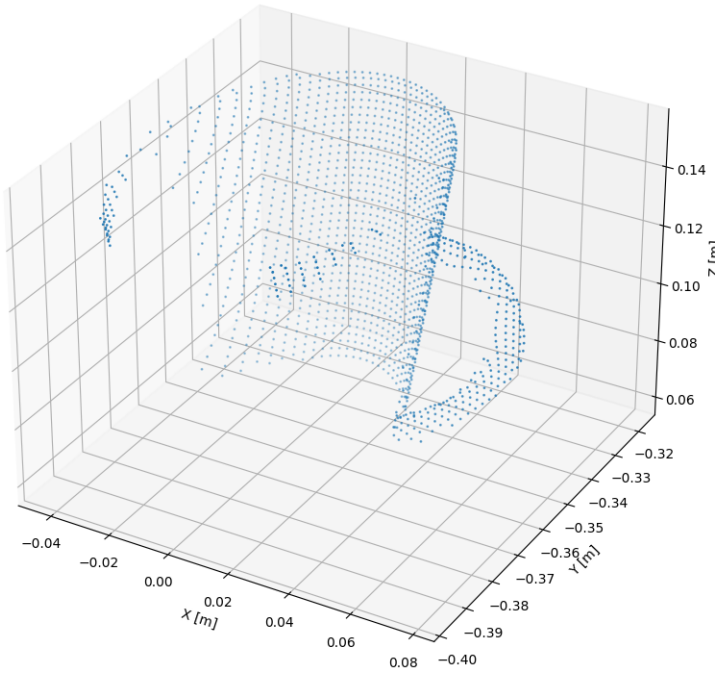
```
from IPython.display import Image
img_size = 800
Image(filename="images/cam_frames.png", width=img_size)
```

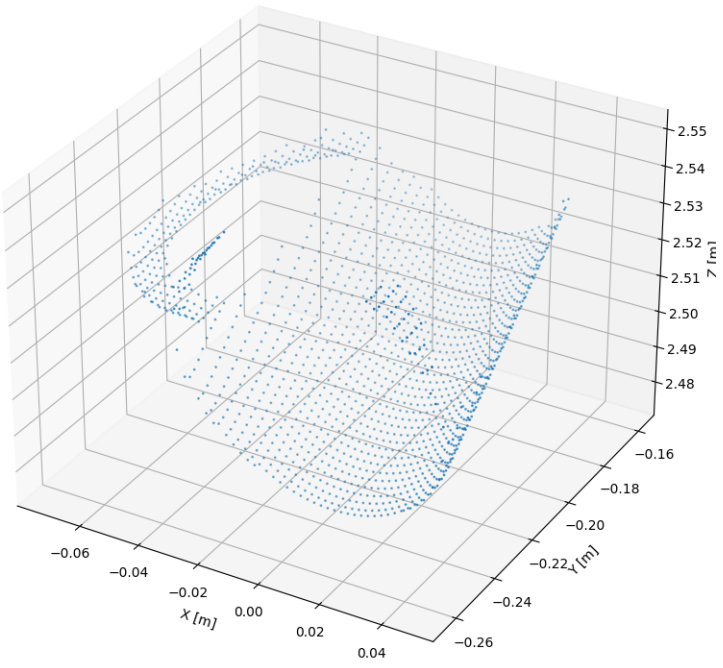
```
def cam2world(points,transform):
    # project camera coordinates to world coordinates [5 pts]
    # NOTE: transform is the transformation from the blender camera frame to the world frame.
    # TODO: YOUR CODE HERE
    # -- cv to blender
    points[1, :] = -points[1, :]
    points[2, :] = -points[2, :]
    # -- then to world
    points = np.vstack((points, np.ones(points.shape[1])))
    res = np.matmul(transform, points)
    res = res[:3, :]

    return res

def world2cam(points,transform):
    # project world coordinates to camera coordinates [5 pts]
    # NOTE: do not use np.linalg.inv to compute the inverse of transform, we will award only partial
                                                        credit.
    # There is an intuitive and elegent way to compute the inverse of transform.
    # NOTE: do not forget about blender coordinates!

    # TODO: YOUR CODE HERE
    points = np.vstack((points, np.ones(points.shape[1])))
    pseudoInv = np.matmul(np.linalg.inv(np.matmul(transform.T, transform)), transform.T)
    res = np.matmul(pseudoInv, points)[:3, :]
    # -- blender to cv
    res[1, :] = -res[1, :]
    res[2, :] = -res[2, :]

    return res
```

```
def show_mask(mask, ax, random_color=False):
    # This function is used to visualize the mask on the image in a matplotlib axis.
    # bool mask: (H, W). True for each pixel that belongs to the object.
    # ax: matplotlib axis
    # random_color: if True, use a random color for the mask. Otherwise, use blue.

    if random_color:
        color = np.concatenate([np.random.random(3), np.array([0.6])], axis=0)
    else:
        color = np.array([30/255, 144/255, 255/255, 0.6])
    h, w = mask.shape[-2:]
    mask_image = mask.reshape(h, w, 1) * color.reshape(1, 1, -1)
    ax.imshow(mask_image)

def show_box(box, ax):
    # This function is used to visualize the bounding box on the image in a matplotlib axis.
    # box: (4,) array. [x0, y0, x1, y1]
    # ax: matplotlib axis

    x0, y0 = box[0], box[1]
    w, h = box[2] - box[0], box[3] - box[1]
    ax.add_patch(plt.Rectangle((x0, y0), w, h, edgecolor='green', facecolor=(0,0,0,0), lw=2))
```

Continued on the next page

```
transform_0 = dataset.transforms[0]

world_pts = cam2world(cam_pnts_3d,transform_0)


np.save('world_pts.npy',world_pts)
#TODO: add this plot to gradescope submission
viz_pts_3d(world_pts)

cam_pts_test = world2cam(world_pts, transform_0)
viz_pts_3d(cam_pts_test)
```





Question 4: Casting masks to new frames

```python
def cam2img(points, K):
    # project camera coordinates to image coordinates [3 pts]
    # output should be pixel coordinates in the correct range with shape (2, N)
    # TODO: YOUR CODE HERE
    res = np.matmul(K, points)
    res = res/res[-1, :]
    res = res[:2, :]

    return res
```

```python
# (1) Filter out masks that have a confidence that's too low. [0 pts]

score_thresh = 0.85
def keep_score(score):
    return score > score_thresh
```

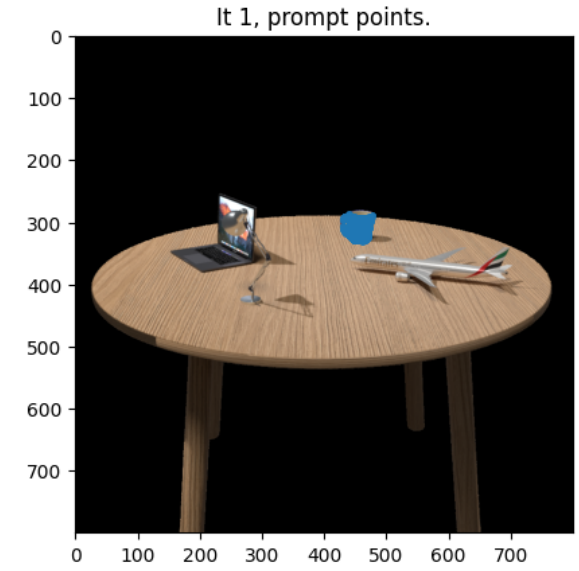
```python
# (2) Filter out points that are too far away from anything else we've seen so far.

dist_thresh = 0.15 # decide on a good distance threshold [2 pts]

def keep_dist(new_pts,existing_pts):
    # TODO: YOUR CODE HERE
    med = np.median(existing_pts, axis=1).reshape(3, 1)
    dist = new_pts - med
    n = np.linalg.norm(dist, axis = 0, ord = 1)
    condition = n < dist_thresh
    new_pts = new_pts[:, condition]

    return new_pts
```

```python
# (3) Carefully choose the bounding box dimensions to avoid noise to have a big effect on the result
                                                    .

# filter out points n_std away from mean of all points [3 pts]
def filter_for_box(world_points,transform, n_std=2):
    # TODO: YOUR CODE HERE
    mean_pt = np.mean(world_points, axis=1)
    std_pt = np.std(world_points, axis = 1)
    condition1 = (np.abs(world_points[0, :] - mean_pt[0]) < n_std*std_pt[0])
    condition2 = (np.abs(world_points[1, :] - mean_pt[1]) < n_std*std_pt[1])
    condition3 = (np.abs(world_points[2, :] - mean_pt[2]) < n_std*std_pt[2])
    condition = np.logical_and(np.logical_and(condition1, condition2), condition3)
    world_points_f = world_points[:, condition]
    cam = world2cam(world_points_f, transform)
    res = cam2img(cam, dataset.K)
    return res

# based on the filtered points, compute the bounding box [2 pts]
def prompt_points_to_box(prompt):
    # TODO: YOUR CODE HERE
    # output: np.array([x0, y0, x1, y1]])
    # (x0, y0): top-left corner
    # (x1, y1): bottom-right corner
    # prompt[:2, :] = prompt[:2, :] / prompt[-1, :]
    min_pt = np.min(prompt, axis = 1)
    max_pt = np.max(prompt, axis = 1)
    res = np.array([min_pt[0], min_pt[1], max_pt[0], max_pt[1]])
    return res
```

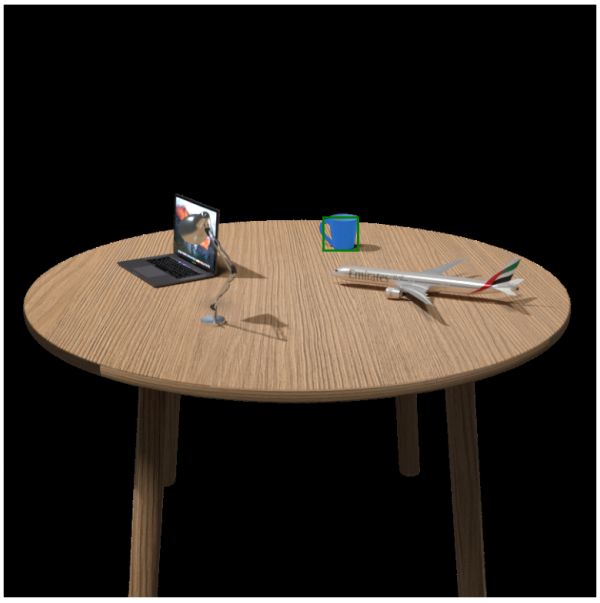
Continued on the next page

```
import os
import copy
from sklearn.cluster import KMeans
all_world_pts = copy.deepcopy(world_pts)

it = 1

# show_its = [1,4,10]
show_its = [1,4,10,50,75,99]

end_idx = -1 # set this to a different number, e.g. 10, for faster debugging

#TODO: add all plots generated to gradescope submission
for transform, file_path, depth in zip(dataset.transforms[1:end_idx], dataset.file_paths[1:end_idx],
                                       depths[1:end_idx]):
    # compute 3d points
    image = cv2.imread(os.path.join('images/dataset/', file_path))
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    prompt_points = filter_for_box(all_world_pts,transform) # (3) Carefully choose the bounding box
                                                            dimensions to avoid noise to have a big effect
                                                             on the result.


    if it in show_its:
        plt.imshow(image)
        # scatter plot the img_pts
        plt.scatter(prompt_points[0,:],prompt_points[1,:],s=5)
        plt.title('It {}, prompt points.'.format(it))
        plt.show()

    predictor.set_image(image)

    prompt_box = prompt_points_to_box(prompt_points) # (3) Choose the bounding box points based on
                                                     the filtered points.


    masks, scores, _ = predictor.predict(
        box=prompt_box,
        point_labels=[1],
        multimask_output=False,
    )

    if it in show_its:
        plt.figure(figsize=(10,10))
        plt.imshow(image)
        show_mask(masks, plt.gca())
        show_box(prompt_box, plt.gca())
        plt.axis('off')
        plt.title('It {}, mask.'.format(it))
        plt.show()

    mask = masks.reshape((h, w, 1))
    cam_pnts_3d = mask2cam(mask,dataset.K,depth, 2.55)
    tmp_world_pts = cam2world(cam_pnts_3d,transform)

    if keep_score(scores): # (1) Filter out masks that have a score that's too low.
        tmp_world_pts = keep_dist(tmp_world_pts,all_world_pts) # (2) Filter out points that are too
                                                               far away from anything else we've seen so
                                                               far.
        all_world_pts = np.hstack([all_world_pts,tmp_world_pts])
        print(tmp_world_pts)

    if it in show_its:
        viz_pts_3d(all_world_pts,title='It {}, all points found so far.'.format(it),xrange=[-0.1,0.1
                                                               ],yrange=[-0.45,-0.25],zrange=[0.06,0.15])


    it+=1
```

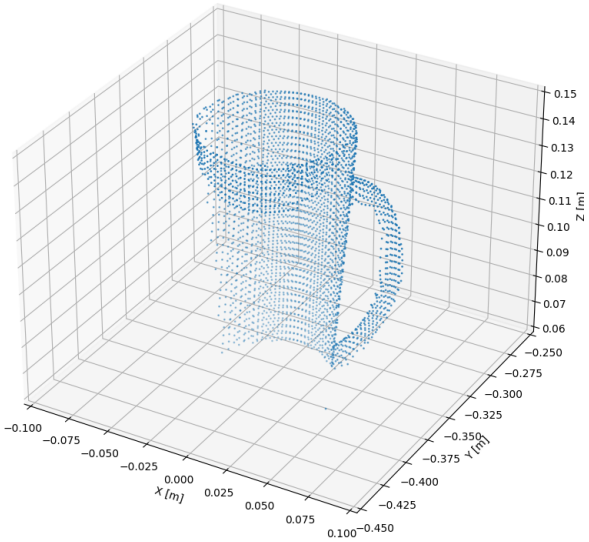
Images are on the next page

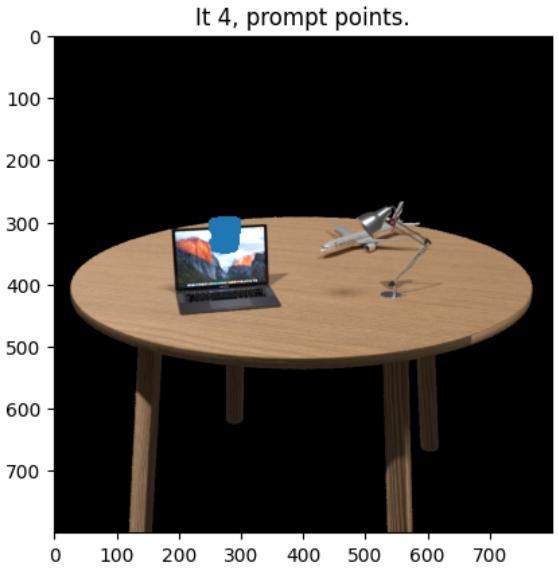


It 1, mask.



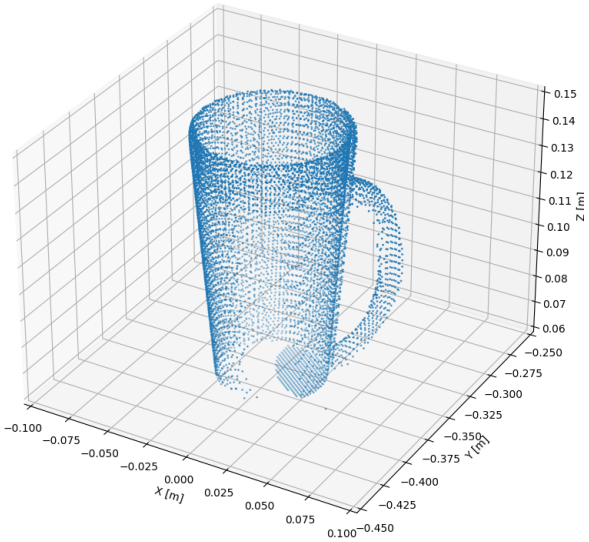
It 1, all points found so far.

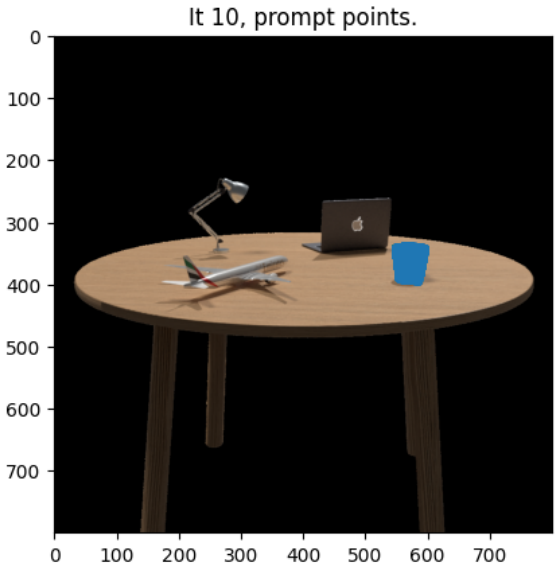




It 4, mask.

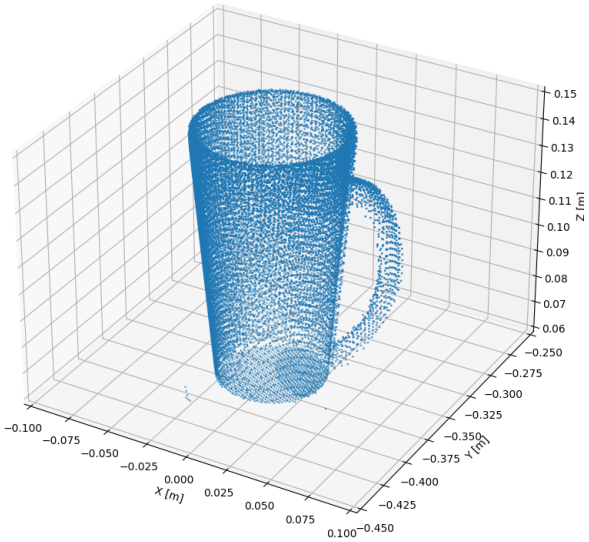
It 4, all points found so far.

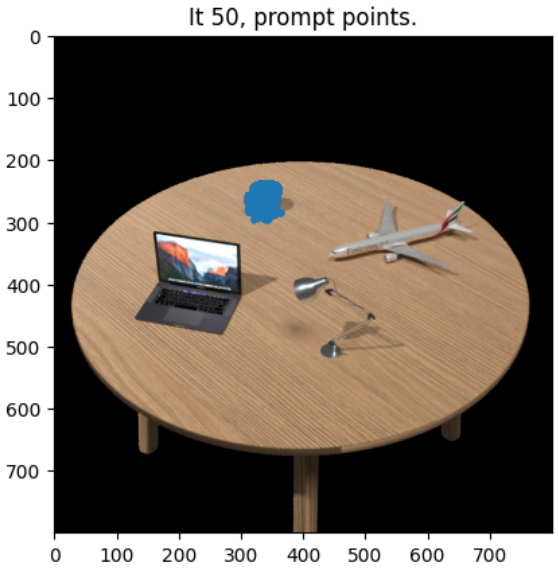




It 10, mask.

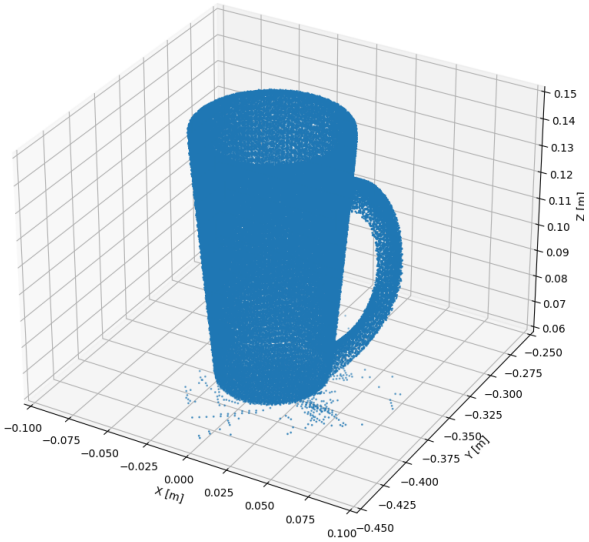
It 10, all points found so far.

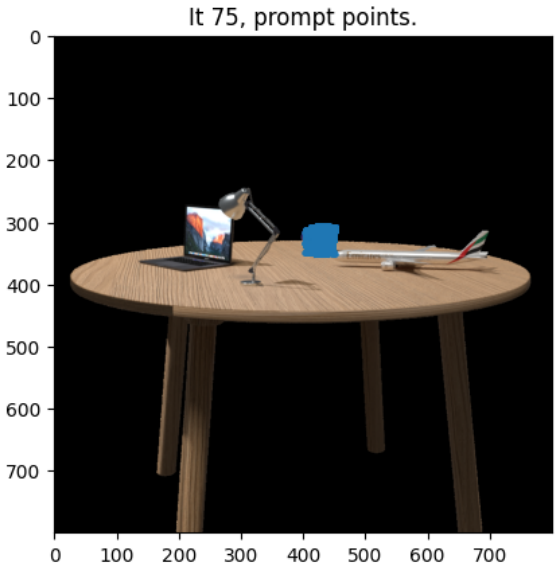




It 50, mask.

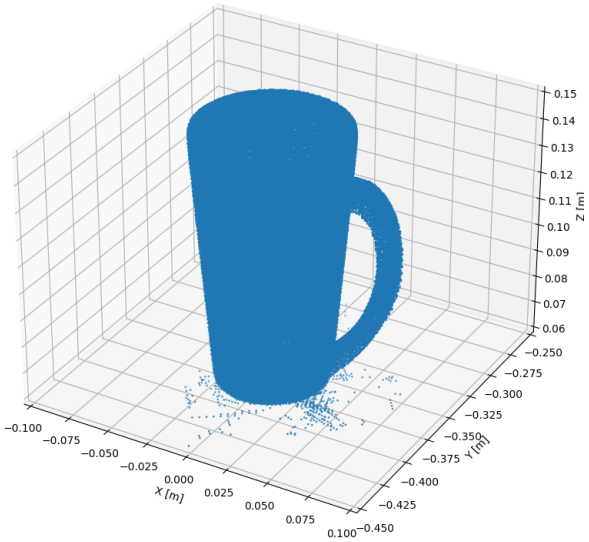
It 50, all points found so far.





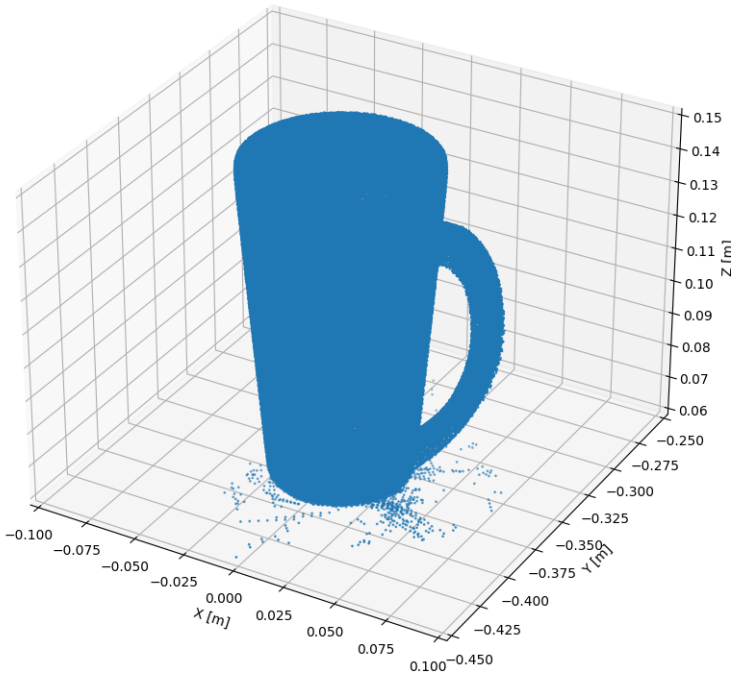
It 75, mask.

It 75, all points found so far.



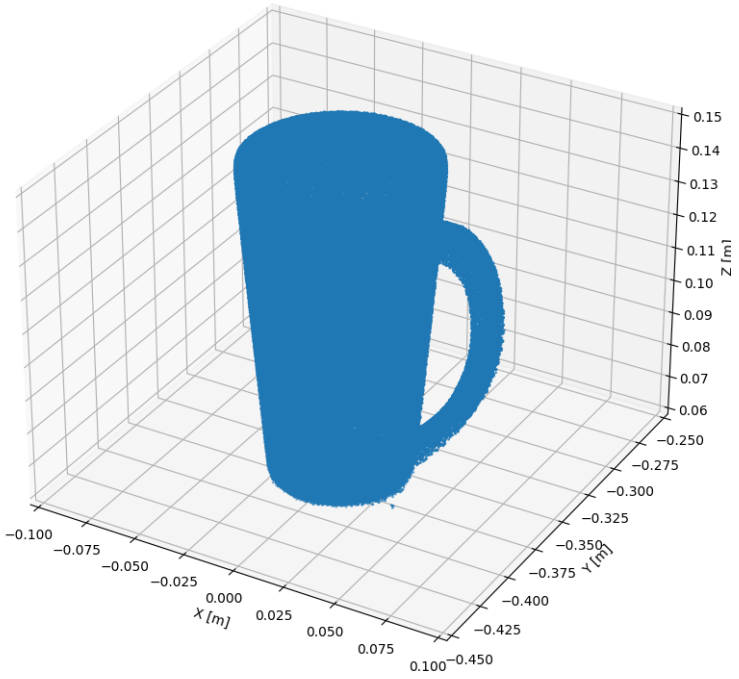
Visualize all 3D points

```
#TODO: do NOT need to add to gradescope submission
viz_pts_3d(all_world_pts,xrange=[-0.1,0.1],yrange=[-0.45,-0.25],zrange=[0.06,0.15])
```



Question 5: Statistical outlier removal

```
def filter_points(points,nb_neighbors=20,std_ratio=2.0):
    import open3d as o3d
    # filter points using open3d statistical outlier removal. [3 pts]
    # TODO: YOUR CODE HERE
    pcd = o3d.geometry.PointCloud()
    pcd.points = o3d.utility.Vector3dVector(points.T)
    cl, ind = pcd.remove_statistical_outlier(nb_neighbors=nb_neighbors, std_ratio=std_ratio)
    points_f = points[:, ind]
    return points_f

all_world_pts_filtered = filter_points(all_world_pts[:3,:])

#TODO: add this plot to gradescope submission
viz_pts_3d(all_world_pts_filtered,xrange=[-0.1,0.1],yrange=[-0.45,-0.25],zrange=[0.06,0.15])
```



Question 6

Done in a study group with roshsanr, bjhamb

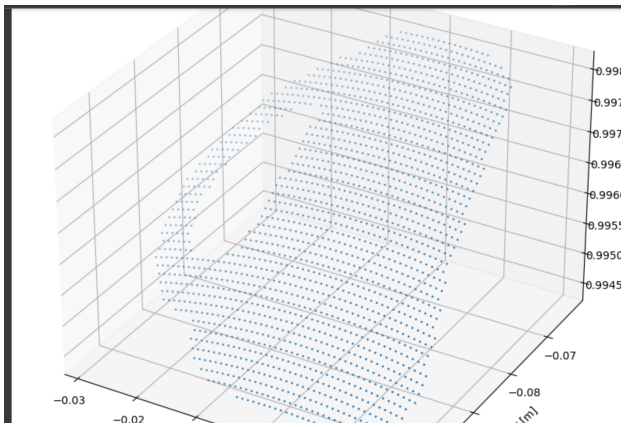
Weights file: mydepths.npy included in code submission

I made another notebook for this. The changes below are made:

```
depth_location = '/content/mydepths/mydepths.npy' # location of ground truth depth maps.
depths = np.load(depth_location) # load the depth maps

depthmap_viz(depths[0]) # visualize the first depth map
plt.show() # show the plot
```

```
def img2cam(points, K, depths=None):
    # project the points from image coordinates to camera coordinates [5 pts]
    cam_3d = None

    # TODO: YOUR CODE HERE
    # (1) Use the intrinsic matrix K to convert the points from image coordinates to a point cloud
                                                    in the camera frame.
    # (2) Normalize the points to a plane with z=1.
    # (3) Use depths to scale the points to be at the correct distance from the camera.
    points[:, [1, 0]] = points[:, [0, 1]] # matrix coordinates xy = image coordinates yx
    pointsT = np.vstack((points.T, np.ones(len(points))))
    points3dT = np.matmul(np.linalg.inv(K), pointsT)
    points3dT[-1, :] = depths.T
    points3dT[:2, :] = points3dT[:2, :]*points3dT[-1, :]
    cam_3d = points3dT

    cam_3d = cam_3d / np.linalg.norm(cam_3d, axis=0).reshape(-1)

    return cam_3d
```

Output



More output can be seen in the HW6-extraCredit.ipynb