HW2 Review

Zeyi Liu, 02/24/2023

Reminders

- Due Date:
  - It’s due on Thursday **March 23th** at 5:00 PM EST.
  - Late submission policy is on the courseworks website.
- Submissions:
  - Please submit on courseworks.
  - Make sure your written PDF is in the folder.
  - Submit only the requested files as listed on the handout
- Check Ed often

GCloud Setup

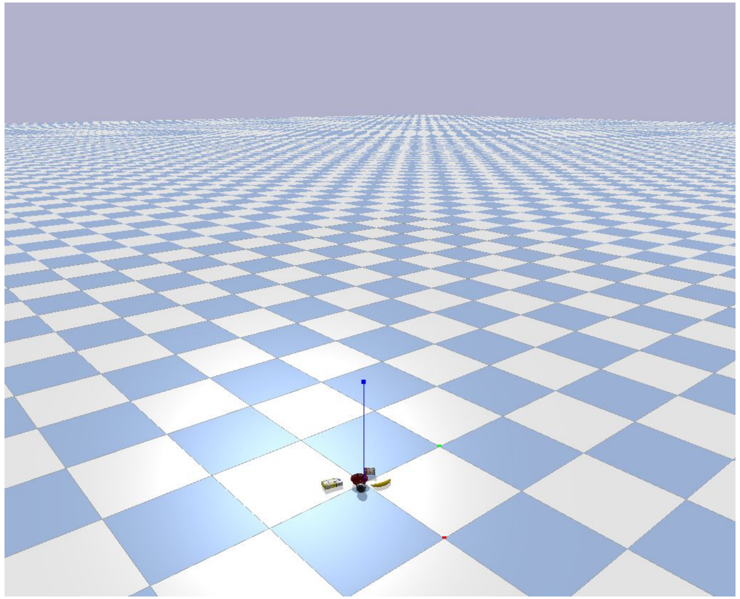
- If you have not started yet, please do so as soon as possible.

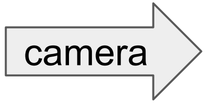
Problem #1

TODO list:

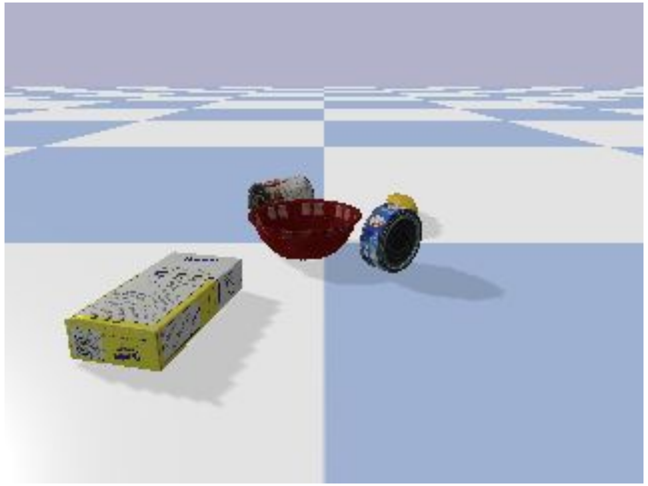
```
camera.py
    compute_camera_matrix() -- 5 points
```

Simulated env

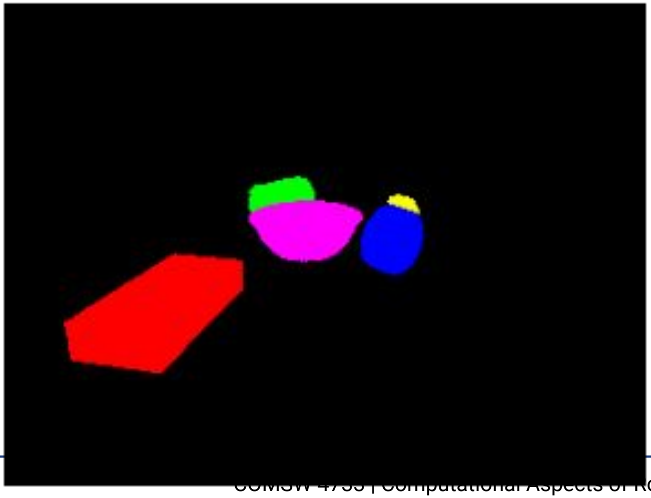




RGB



mask
(ground truth)



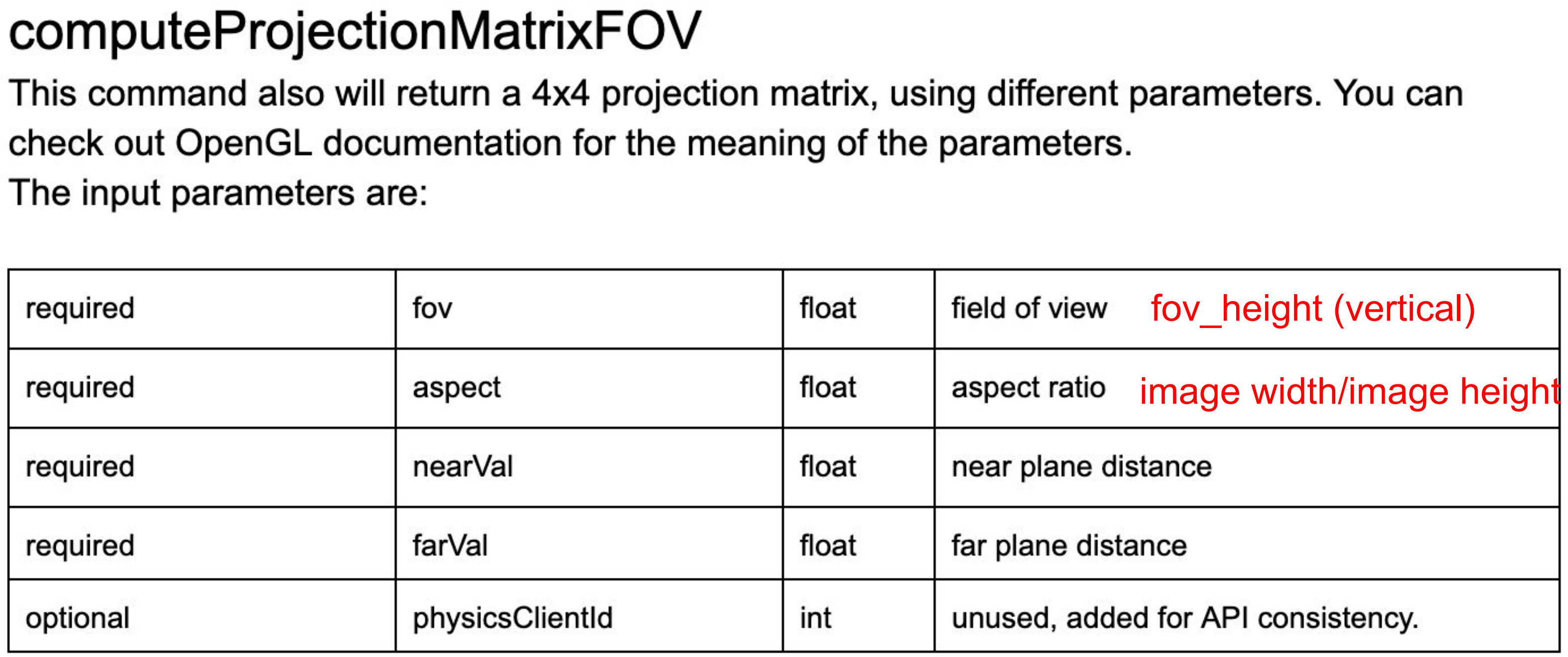
Camera Intrinsics

$$K = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix}$$

Convert camera coordinates to image coordinates

- fx, fy: focal length
- ox, oy: principle point, center of the image

Projection Matrix



computeProjectionMatrixFOV

This command also will return a 4x4 projection matrix, using different parameters. You can check out OpenGL documentation for the meaning of the parameters.
The input parameters are:

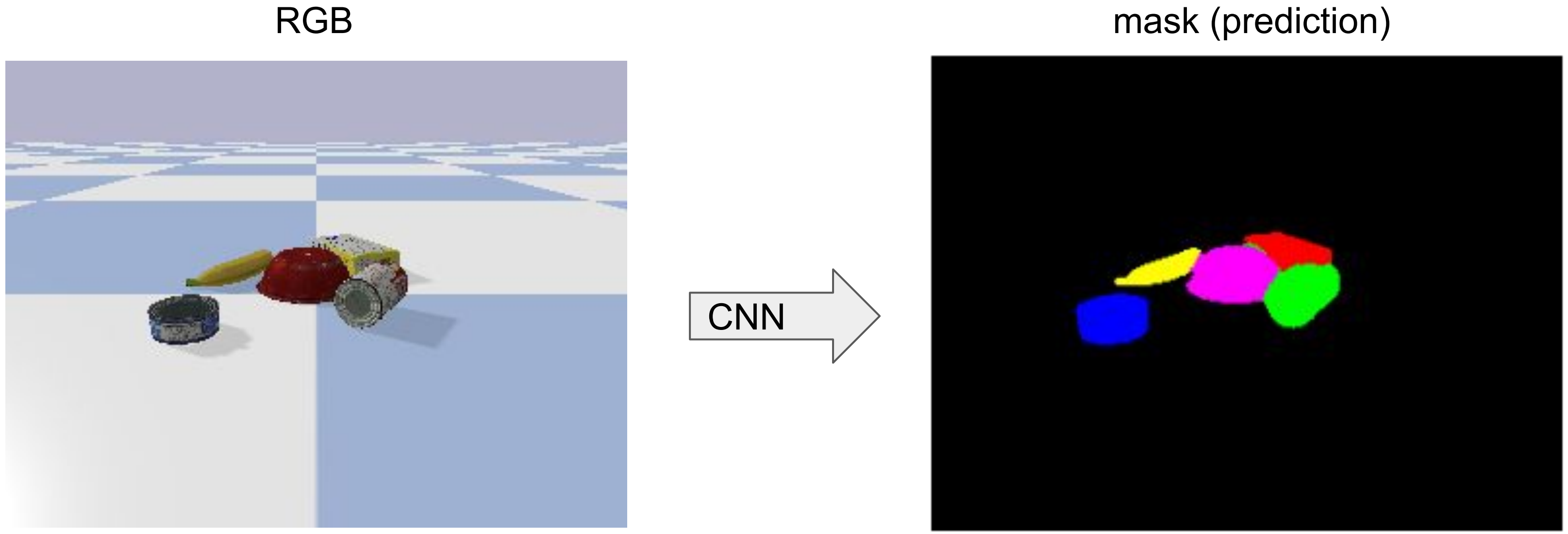
required	fov	float	field of view fov_height (vertical)
required	aspect	float	aspect ratio image width/image height
required	nearVal	float	near plane distance
required	farVal	float	far plane distance
optional	physicsClientId	int	unused, added for API consistency.

Problem #2

Problem 2

TODO list:

```
written question -- 5 points
dataset.py
    class RGBDataset() -- 10 points
model.py
    class MiniUNet() -- 20 points
segmentation.py
    train() -- 5 points
    val() -- 5 points
    main() -- 5 points
```



Dataset

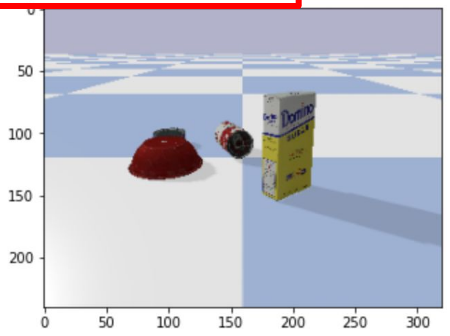
- def __init__(self, dataset_dir, has_gt)
  - Define transformation to apply
    - transforms.ToTensor() and transforms.Normalize()
  - Compute dataset length
- def __len__()
  - Just return dataset length
- def __getitem__(self, idx)
  - Read rgb image ({idx}_rgb.png), apply transformation
    - If gt mask exists, pair them together as a sample
      - sample = {‘input’: rgb_img, ‘target’: gt_mask}

Sanity check

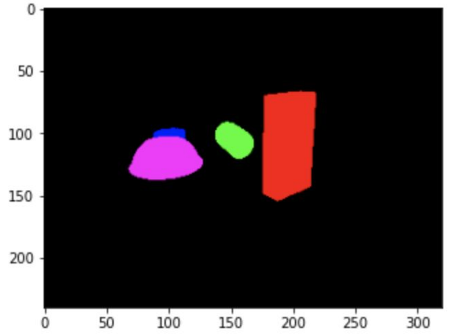
Dataset size:
train 300 images
val and test 5 images

Dataloader: shuffle and batch the data

```
device: cuda
dataset size: 300
input shape: (3, 240, 320)
```

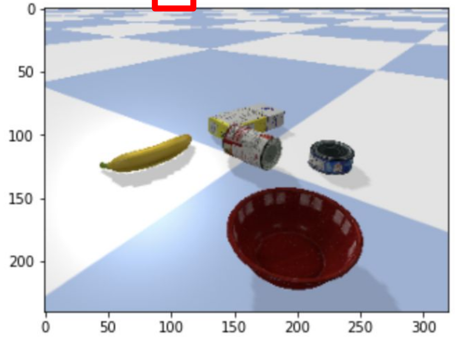


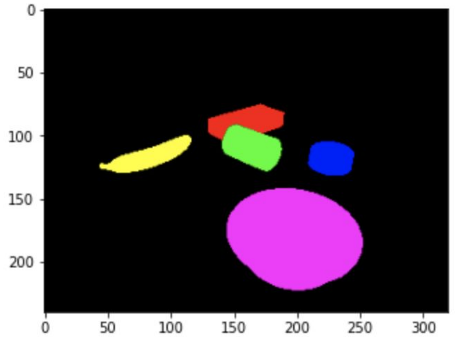
```
target shape: (240, 320)
```



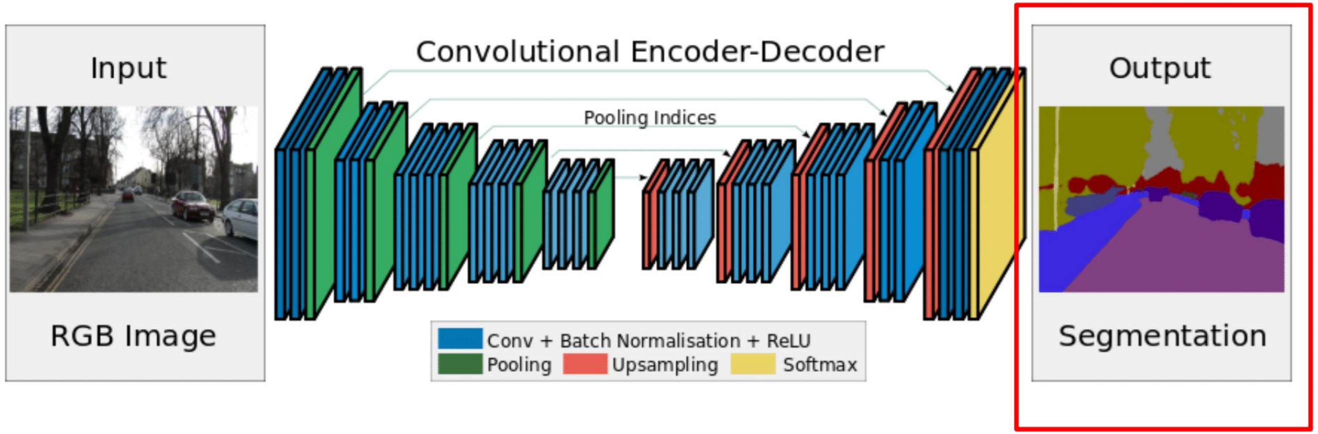
```
dataset size: 300
input shape: (4, 3, 240, 320)
target shape: (4, 240, 320)
```

batch size





Case study: Image Segmentation

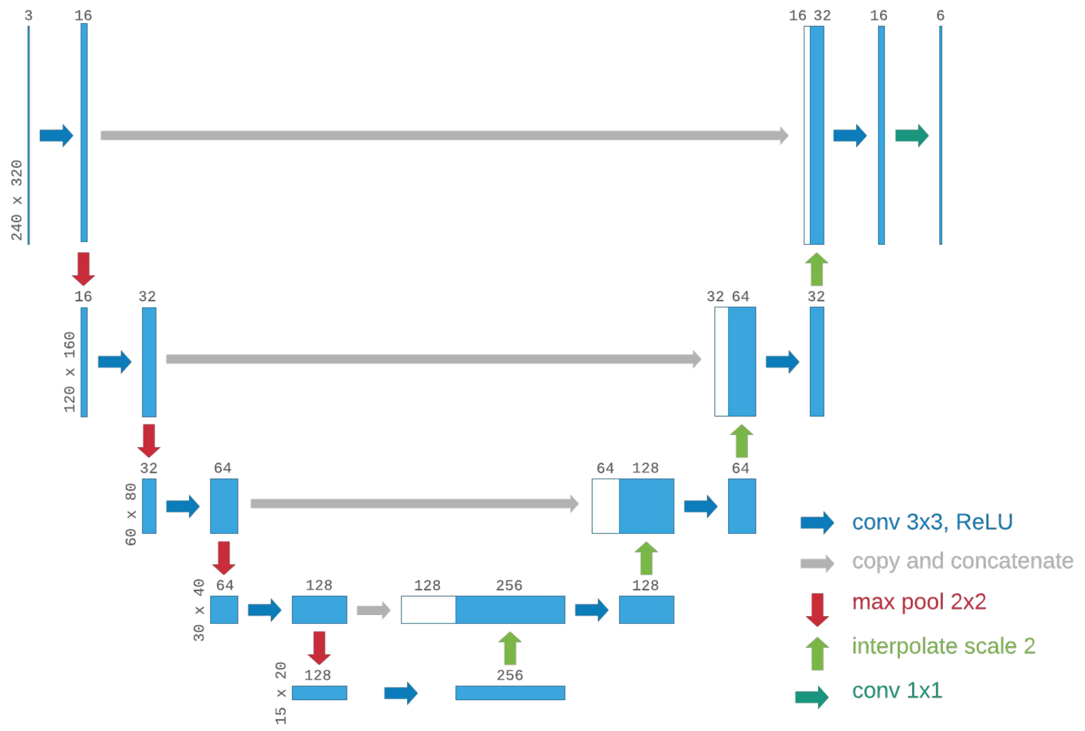


Output 2D image with pixel-wise label

Spatial pixel-wise loss (classification on each pixel instead of whole image)

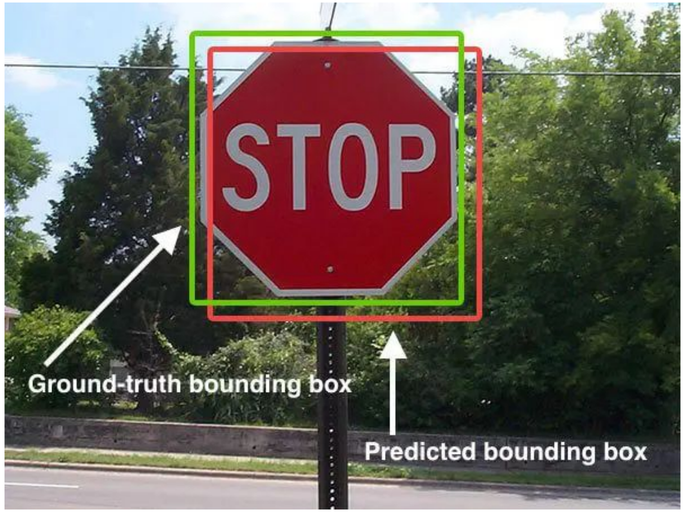
MiniUNet

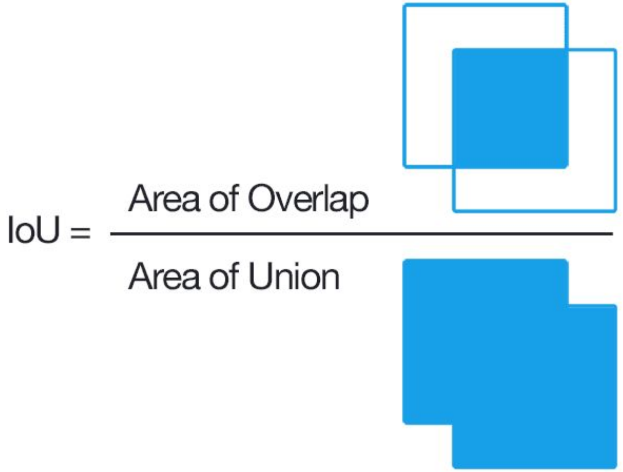
```
import torch.nn as nn
import torch.nn.functional as F
```



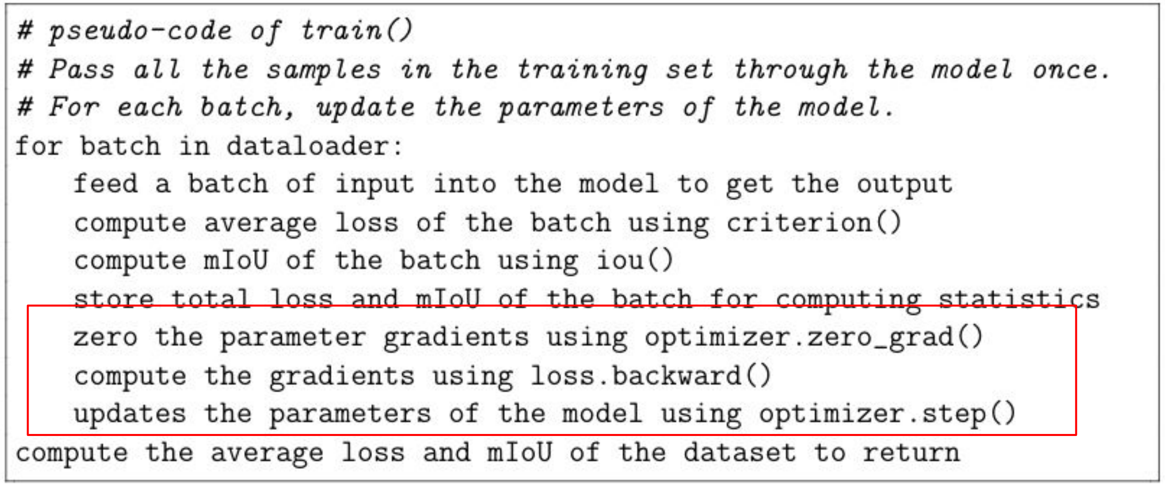
Train and validate the model

Metrics: Cross entropy loss, mIoU (intersection over union, larger is better)





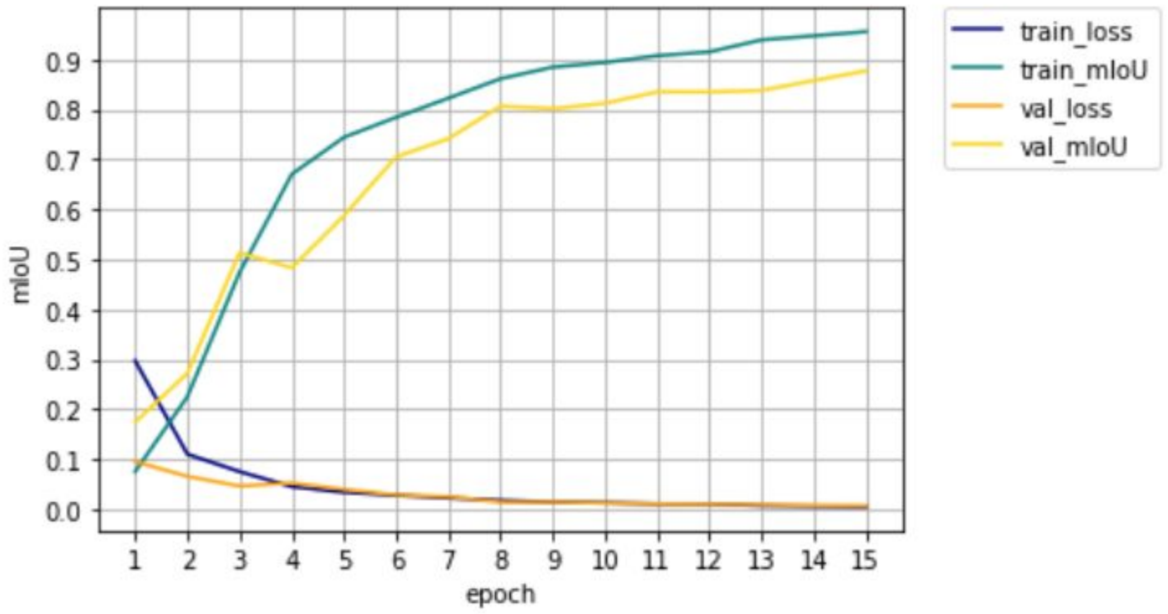
Pseudocode



```
# pseudo-code of train()
# Pass all the samples in the training set through the model once.
# For each batch, update the parameters of the model.
for batch in dataloader:
    feed a batch of input into the model to get the output
    compute average loss of the batch using criterion()
    compute mIoU of the batch using iou()
    store total loss and mIoU of the batch for computing statistics
    zero the parameter gradients using optimizer.zero_grad()
    compute the gradients using loss.backward()
    updates the parameters of the model using optimizer.step()
compute the average loss and mIoU of the dataset to return
```

- There might be less data than batch size in the last batch, so don't average over batch, average over the whole dataset.
- Validation is the same without back propagation.

Result


train_loss
train_mIoU
val_loss
val_mIoU
mIoU
epoch
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Problem #3

Pose Estimation - Prepare point clouds

- First point cloud is on us!

    def obj_mesh2pts(obj_id, point_num, transform=None)

- The second point cloud is your turn

    def gen_obj_depth(obj_id, depth, mask)

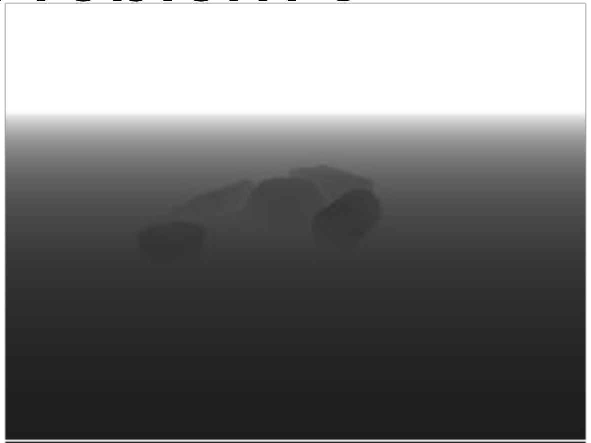
        Generate depth image that only contains the specific object given by obj_id

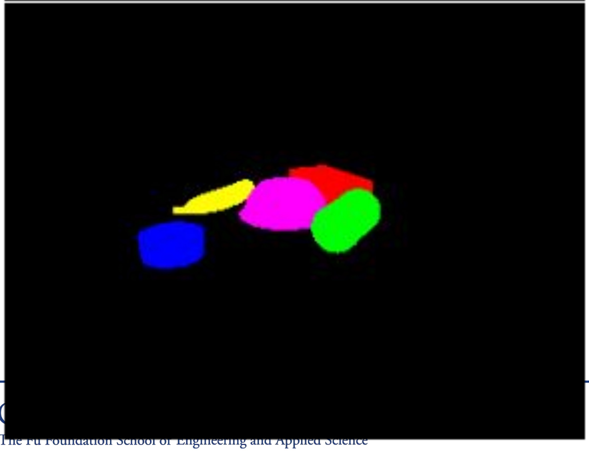
    def obj_depth2pts((obj_id, depth, mask, camera, view_matrix)

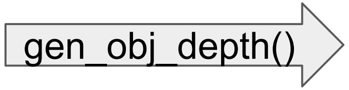
        depth → point cloud in camera coordinates → world coordinates

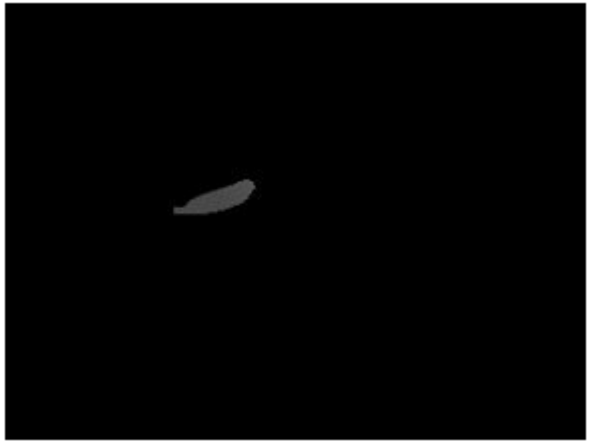
        gen_obj_depth → depth_to_point_cloud → apply camera pose matrix

Problem 3


depth (scene)


mask



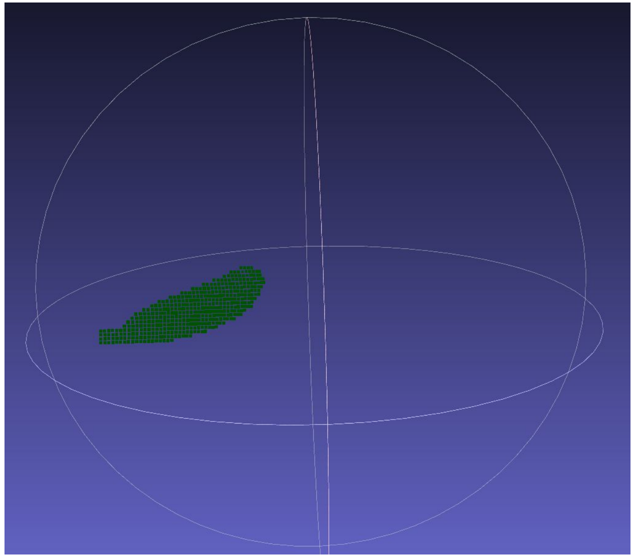

depth (object)

Problem 3

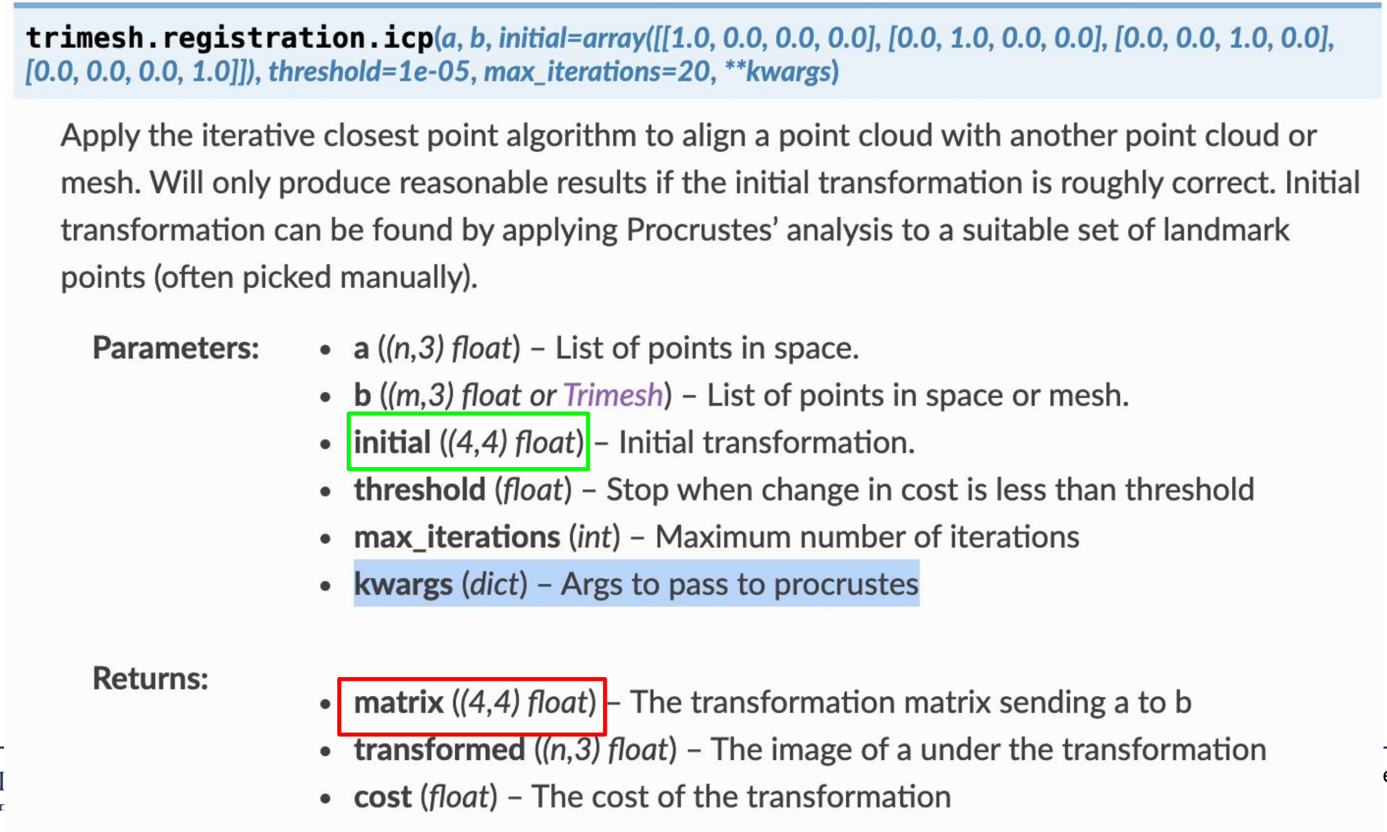
depth (object)

obj_depth2pts()

point cloud (projected)



Pose Estimation – Align Pts



trimesh.registration.icp(*a, b, initial=array([[1.0, 0.0, 0.0, 0.0], [0.0, 1.0, 0.0, 0.0], [0.0, 0.0, 1.0, 0.0], [0.0, 0.0, 0.0, 1.0]]), threshold=1e-05, max_iterations=20, **kwargs*)

Apply the iterative closest point algorithm to align a point cloud with another point cloud or mesh. Will only produce reasonable results if the initial transformation is roughly correct. Initial transformation can be found by applying Procrustes' analysis to a suitable set of landmark points (often picked manually).

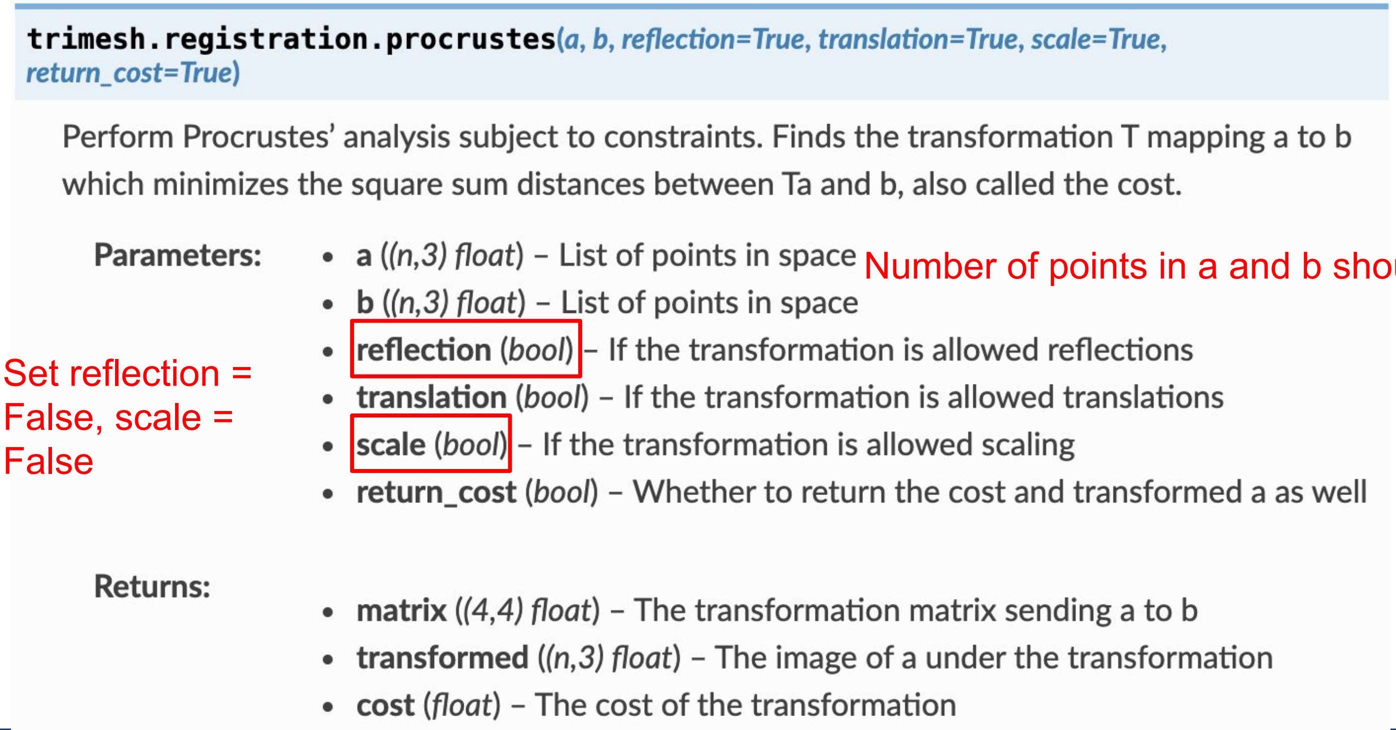
Parameters:

- **a** (*(n,3) float*) – List of points in space.
- **b** (*(m,3) float or Trimesh*) – List of points in space or mesh.
- **initial** (*(4,4) float*) – Initial transformation.
- **threshold** (*float*) – Stop when change in cost is less than threshold
- **max_iterations** (*int*) – Maximum number of iterations
- **kwargs** (*dict*) – Args to pass to procrustes

Returns:

- **matrix** (*(4,4) float*) – The transformation matrix sending a to b
- **transformed** (*(n,3) float*) – The image of a under the transformation
- **cost** (*float*) – The cost of the transformation

Pose Estimation – Align Pts



trimesh.registration.procrustes(*a, b, reflection=True, translation=True, scale=True, return_cost=True*)

Perform Procrustes' analysis subject to constraints. Finds the transformation T mapping a to b which minimizes the square sum distances between Ta and b, also called the cost.

Parameters:

- **a** (*(n,3) float*) – List of points in space
- **b** (*(n,3) float*) – List of points in space
- **reflection** (*bool*) – If the transformation is allowed reflections
- **translation** (*bool*) – If the transformation is allowed translations
- **scale** (*bool*) – If the transformation is allowed scaling
- **return_cost** (*bool*) – Whether to return the cost and transformed a as well

Returns:

- **matrix** (*(4,4) float*) – The transformation matrix sending a to b
- **transformed** (*(n,3) float*) – The image of a under the transformation
- **cost** (*float*) – The cost of the transformation

Number of points in a and b should be the same

Set reflection = False, scale = False

Note

The predicted mask might not have enough points to solve the transformation matrix.

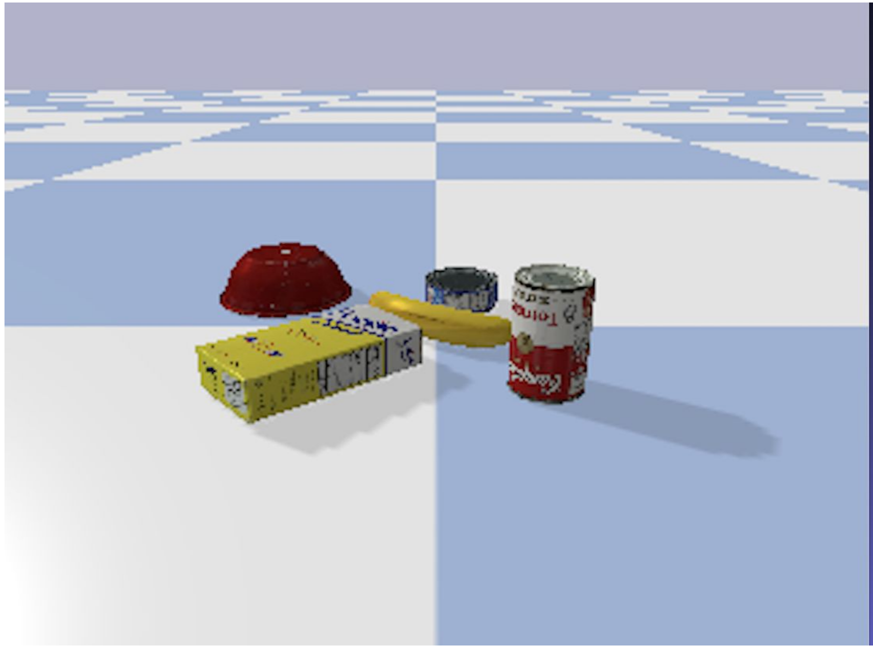
```
try:
    procrustes()
except numpy.linalg.LinAlgError:
    return None
```

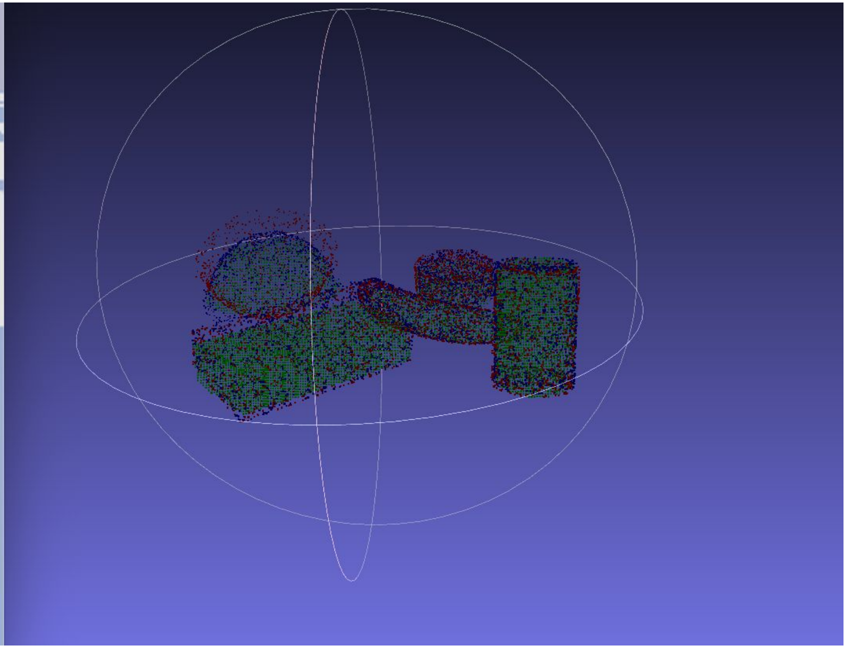
The same for icp()

Pose Estimation

- In estimate_pose(), for each object
  - Prepare two point clouds
  - Align points and get the transformation matrix
- In main(),
  - Use estimate_pose() to estimate the pose of the objects in each scene
  - For the validation set, use both ground truth mask and predicted mask.
  - For the test set, use the predicted mask.
  - Use save_pose(), export_gt_ply() and export_pred_ply() to generate files to be submitted.

Problem 3





Improvement (Optional)

- Better segmentation model
- Use multi-view images to deal with occlusion
- Try different/multiple initial transformations and optimize
- Denoise the predicted mask
- Denoise the projected point cloud when using predicted mask
- Encode more information than just x,y,z location

Thank you, Q&A time