CS4182 Computer Graphics Single-View Human Reconstruction Tutorial

2024/25 Semester A

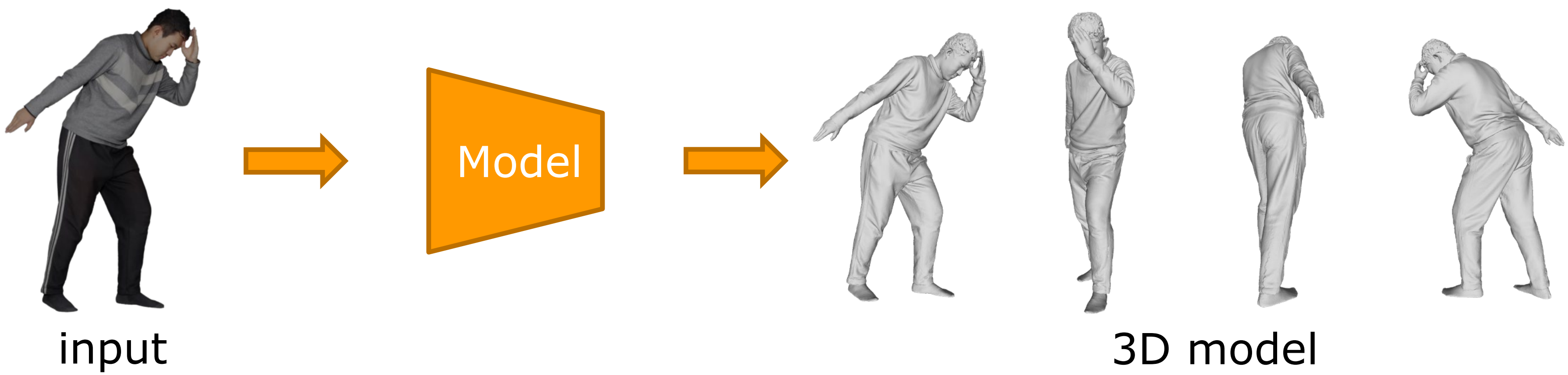
City University of Hong Kong (DG)

Background

- Reconstructing human model from the single-view image plays important roles in various applications, e.g., gaming, film production, and sports event broadcasting.

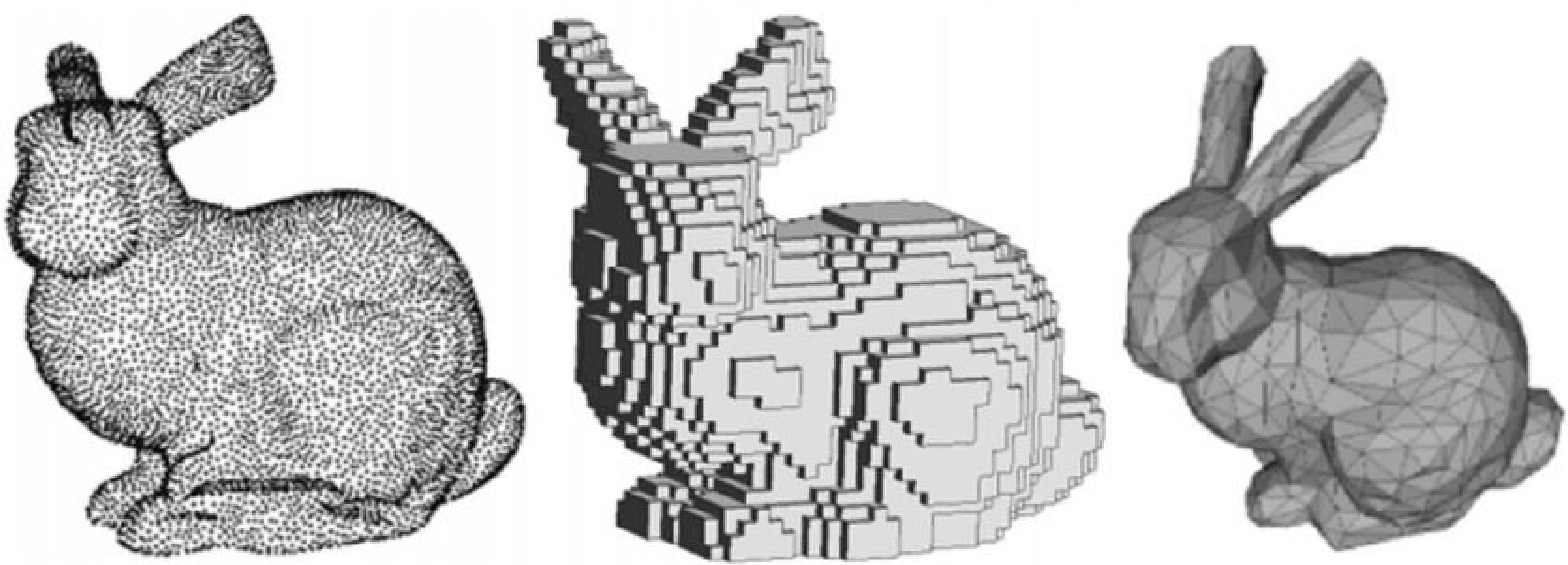
Background

- Problem Formulation:
  - Given a single-view RGB image, we aims to reconstruct the surface of the human body.



Background

- 3D representations
  - Point clouds
  - Voxel
  - Mesh
  - Implicit-Functions (SDF, Occupancy, UDF)



How to learn 3D information from 2D images?

- Implicit-based methods

  Given a point in the 3D space, determine whether it is in the surface or out the surface.

  - PIFu[1]
  - ICON[2]

- Explicit-based methods

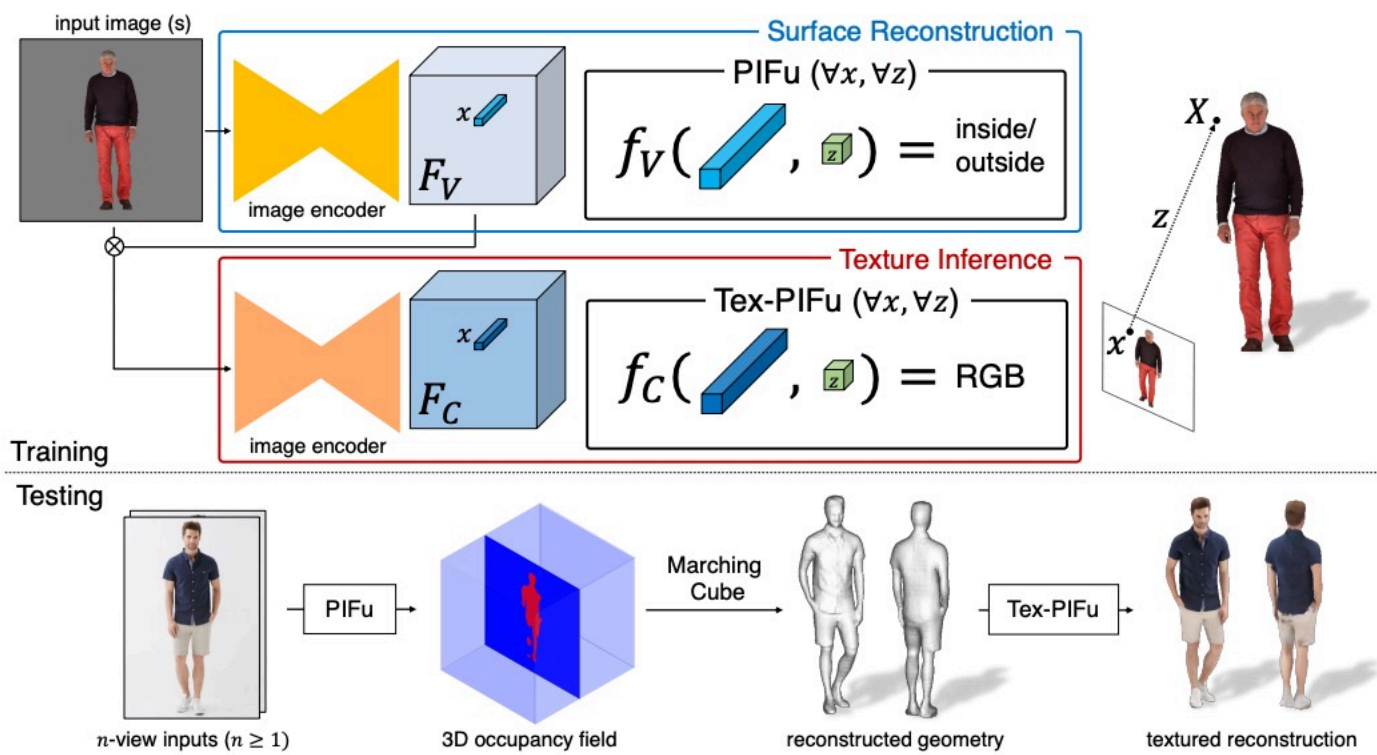
  Explicitly learn the position of the surface.

  - HaP[3]

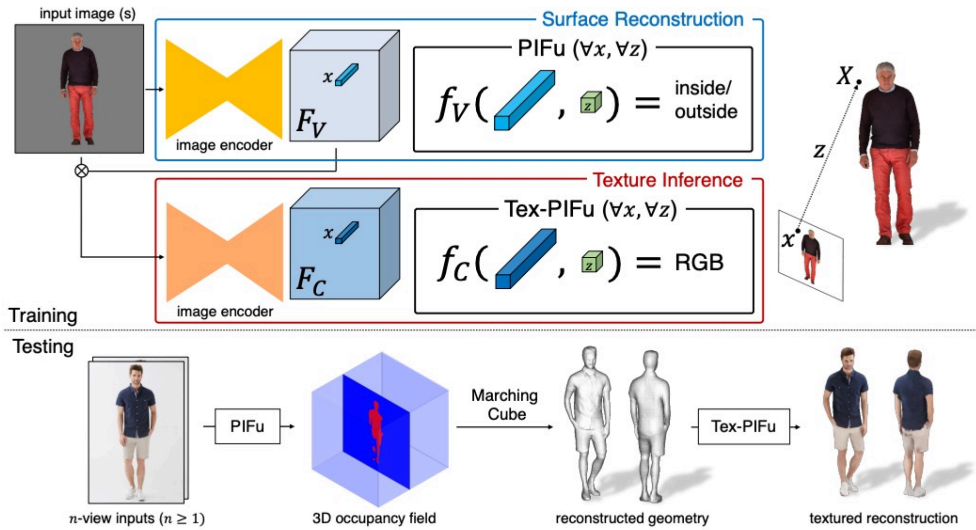
1. Saito S, Huang Z, Natsume R, et al. Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization[C]//Proceedings of the IEEE/CVF international conference on computer vision. 2019: 2304-2314.
2. Xiu Y, Yang J, Tzionas D, et al. Icon: Implicit clothed humans obtained from normals[C]//2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, 2022: 13286-13296.
3. Tang Y, Zhang Q, Hou J, et al. Human as Points: Explicit Point-based 3D Human Reconstruction from Single-view RGB Images[J]. arXiv preprint arXiv:2311.02892, 2023.

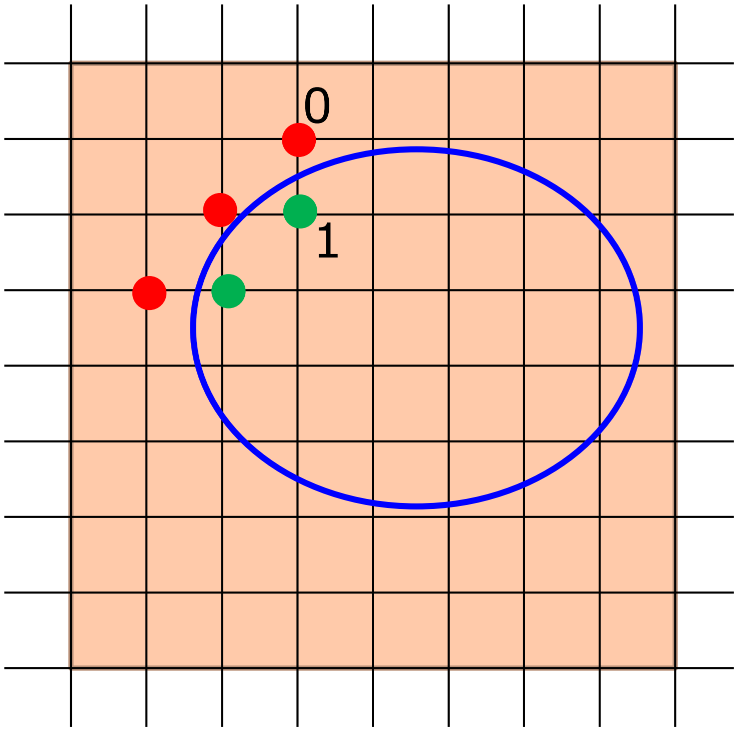
Implicit-based: PIFu

- Feature Preparation: Project points on the images, to achieve the pixel-level features.
- Input: The (x,y,z) axises of the query point; the pixel-level features (rgb, network feature).
- Output: Binary Occupancy values.



Implicit-based: PIFu





- Convert the implicit-function to surface with marching-cube.

Implicit-based: PIFu

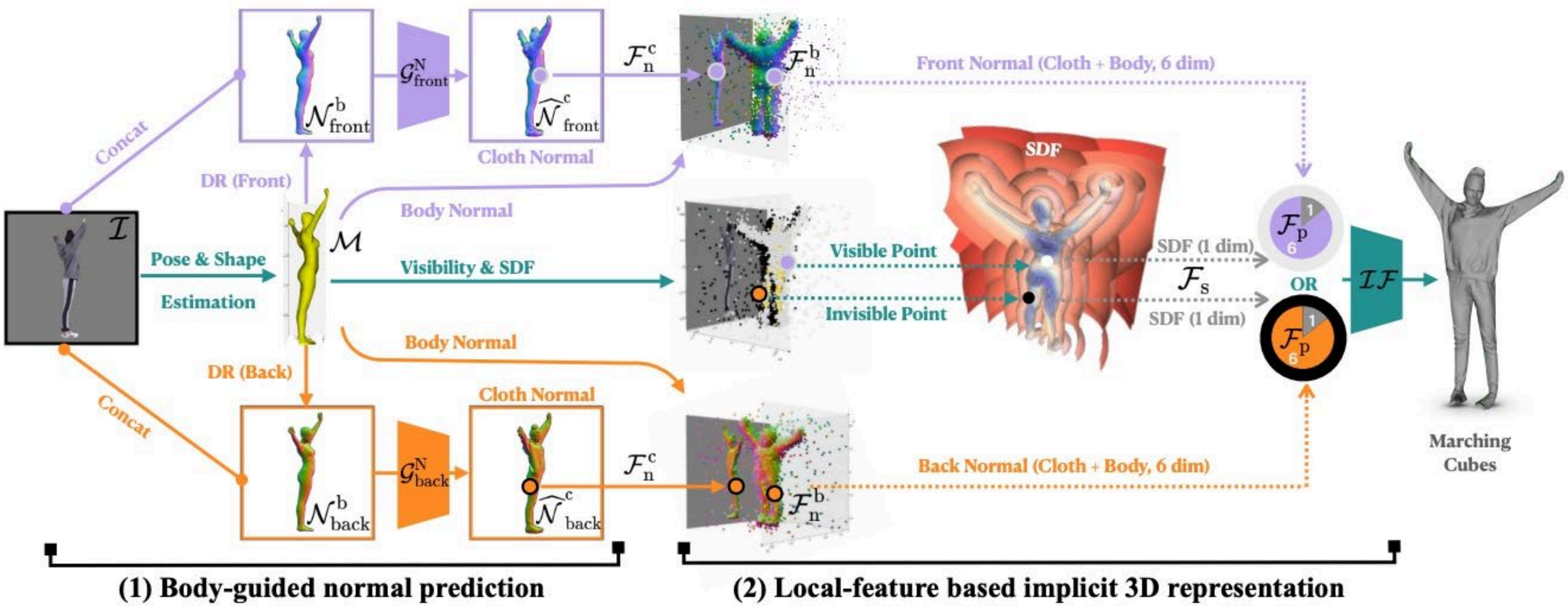
- Defect: cannot tackle with the occlusion situation

Input

PIFu

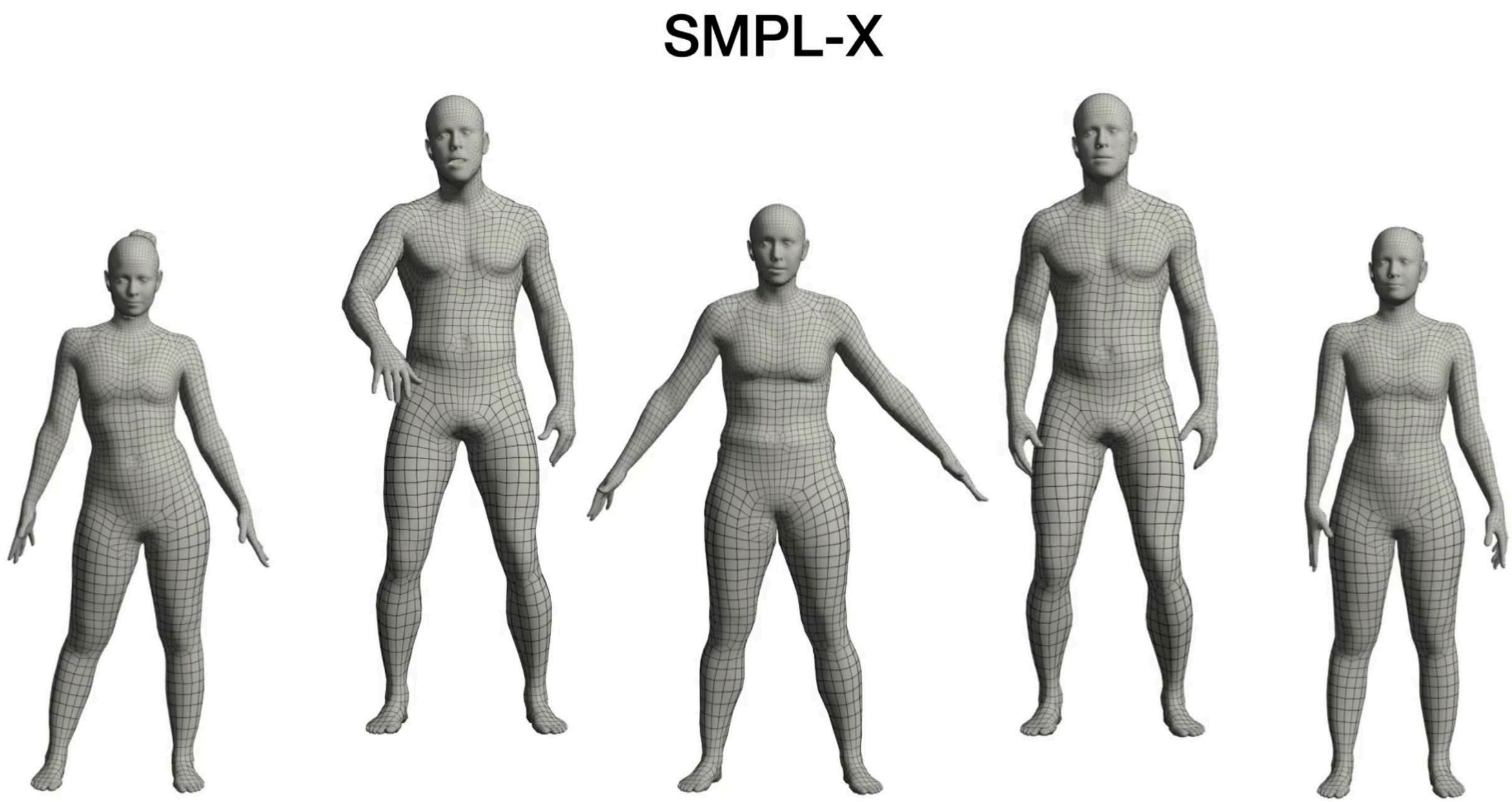
Implicit-based: ICON

- Introduce human prior SMPL into the model



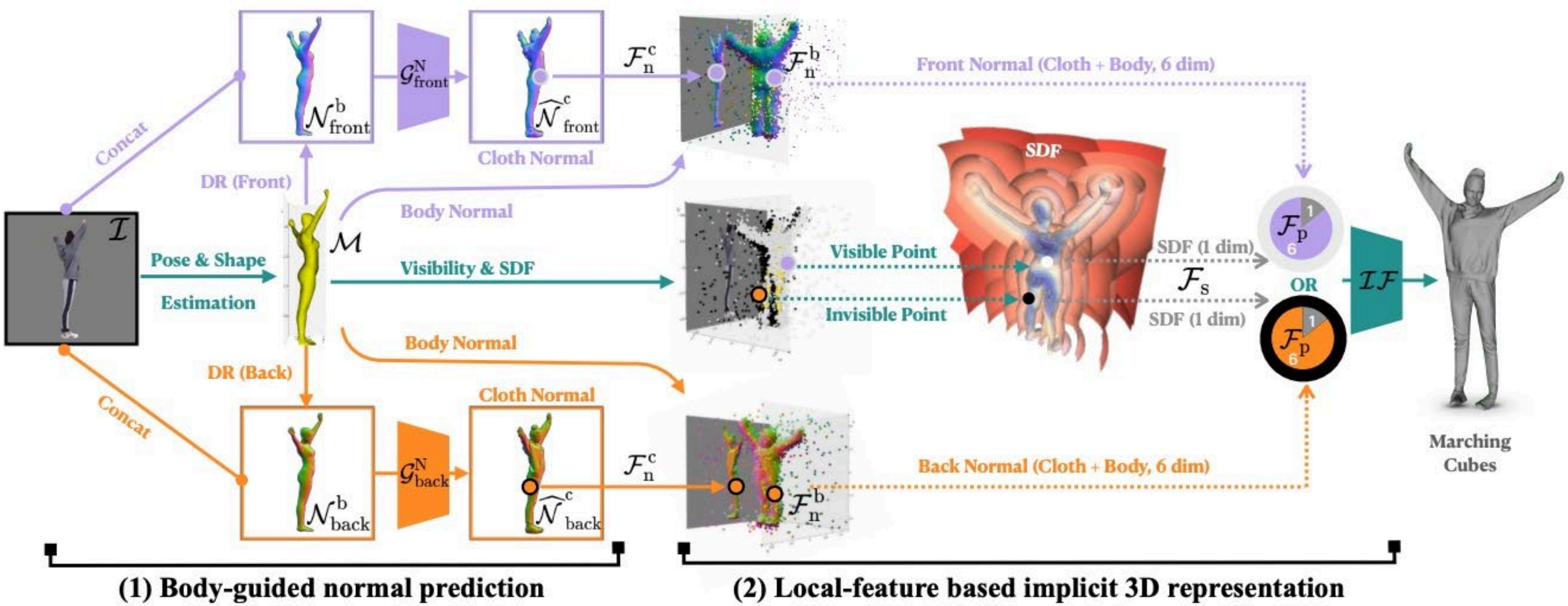
Implicit-based: ICON

- Introduce human prior SMPL into the model

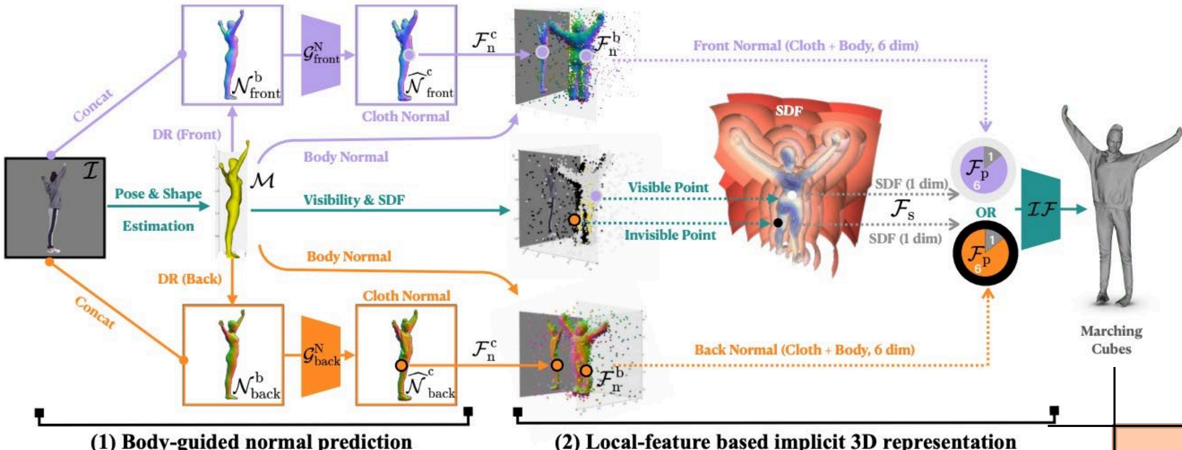


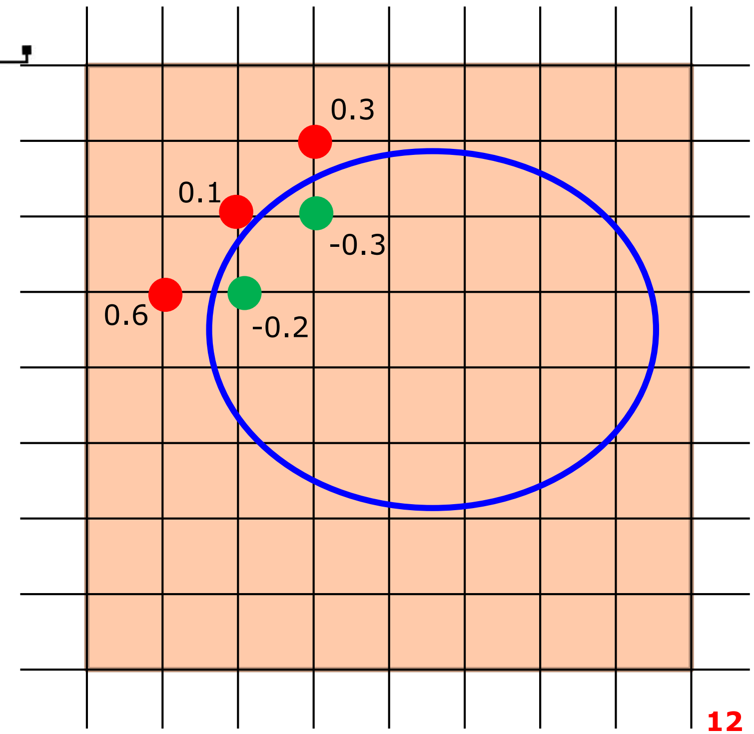
Implicit-based: ICON

- Input: sdf value, smpl normal feature and cloth normal feature
- Output: sdf value



Implicit-based: ICON





Implicit-based: ICON

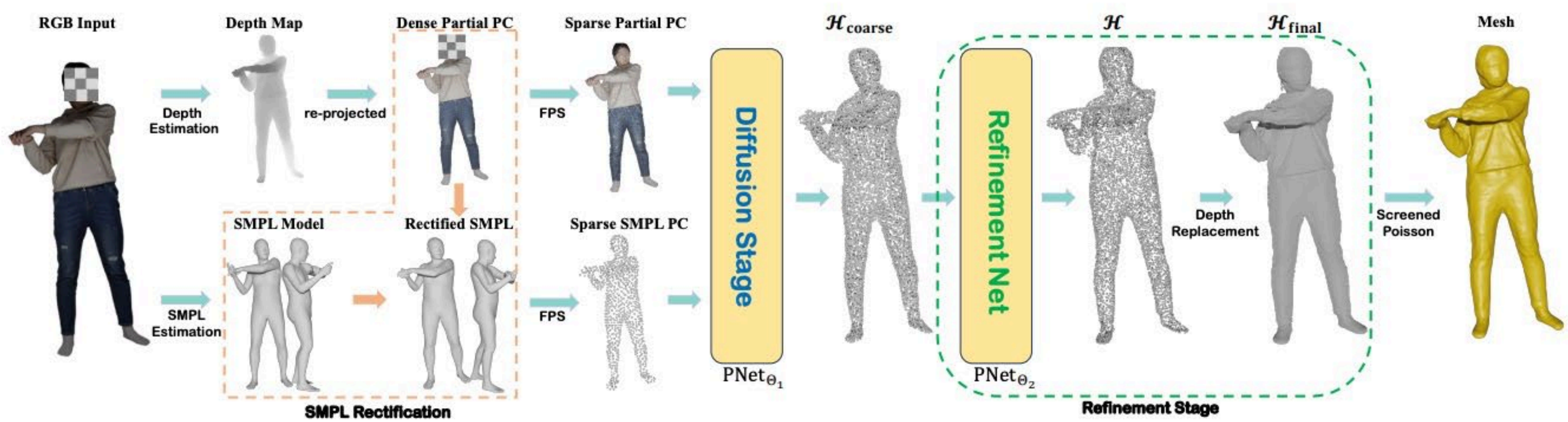
- Defect: cannot recover loosing clothes.

Input

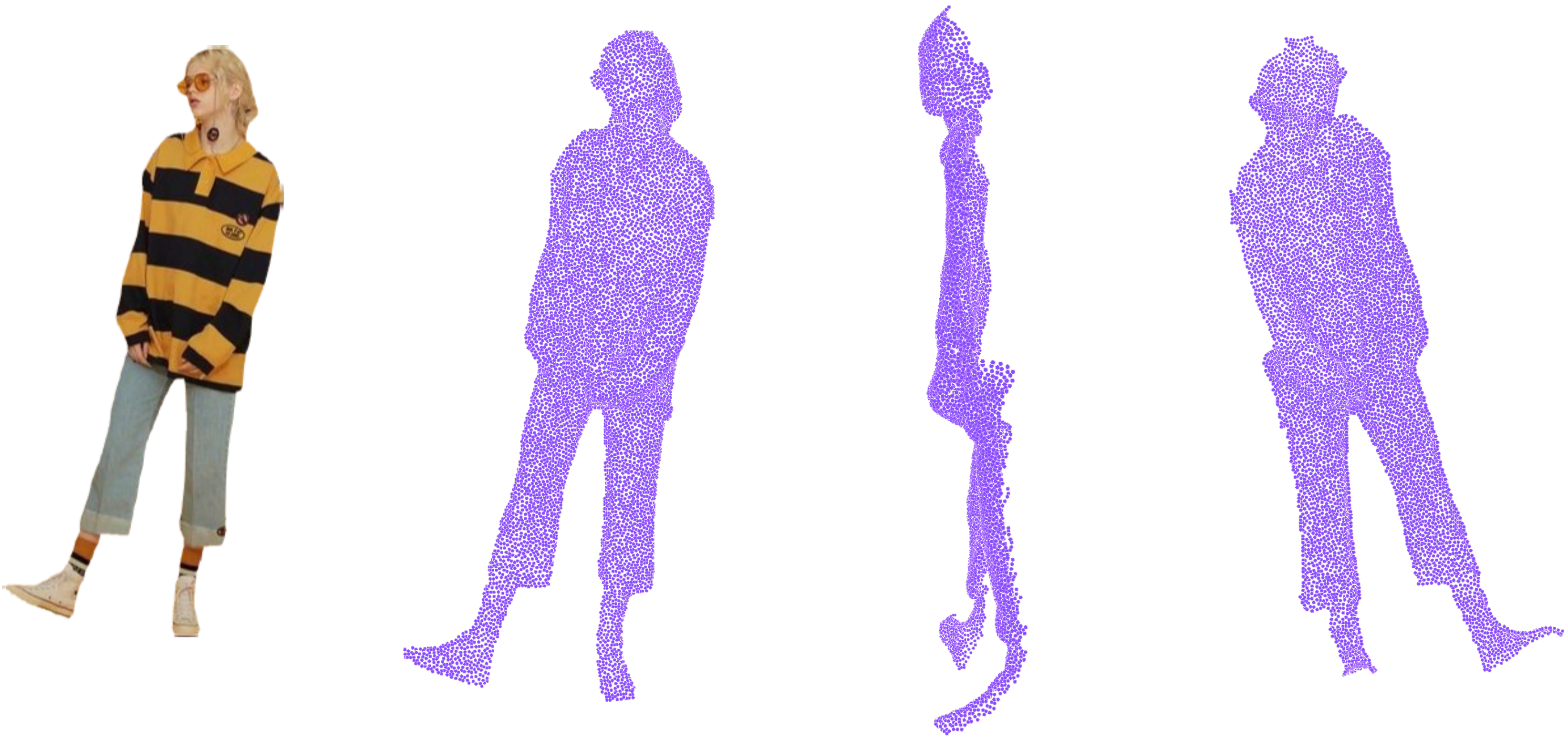
ICON

Explicit-based: HaP

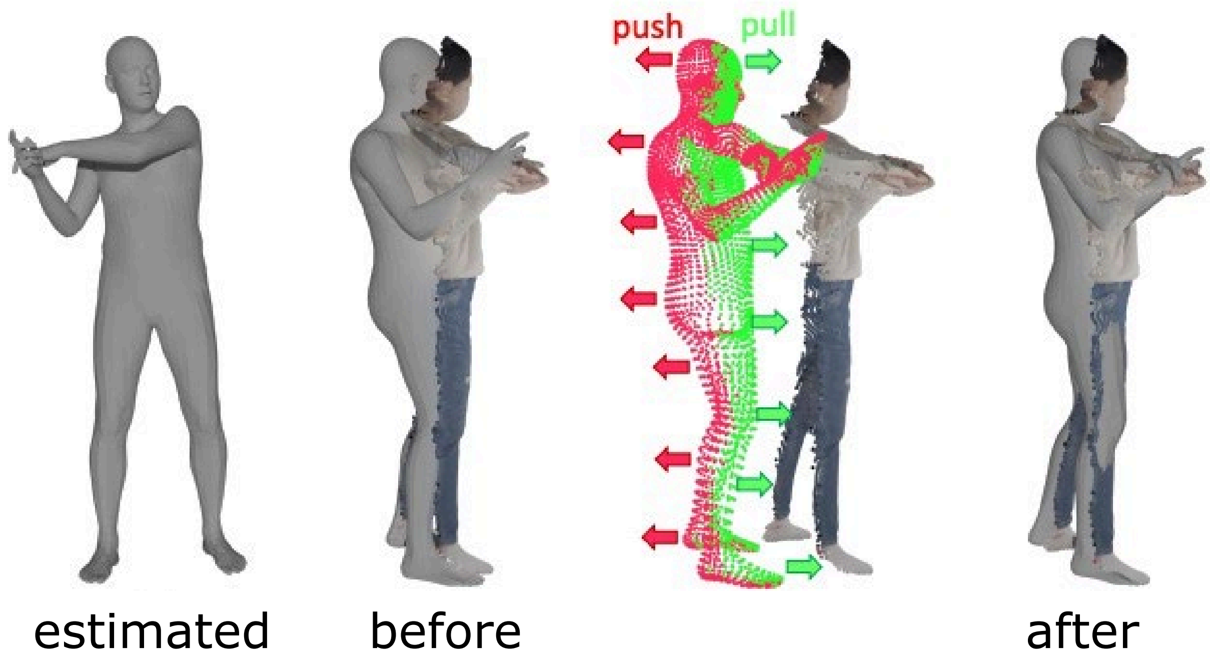
- Explicitly generating the human body point cloud.
- Reconstruct the human surface from the point cloud.



Explicit-based: Depth Estimation

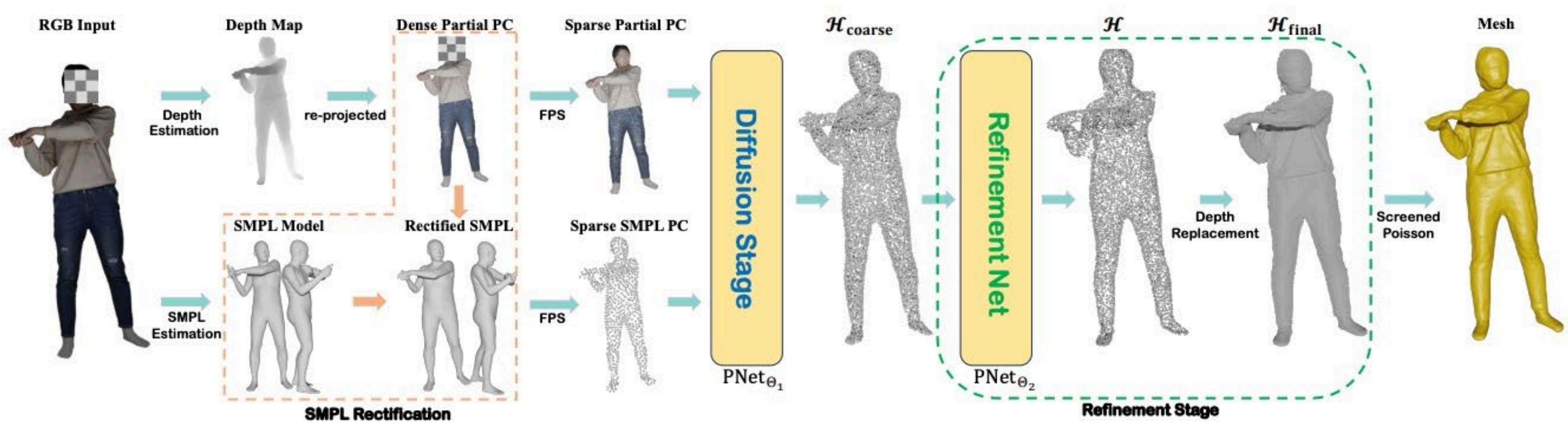


Explicit-based: SMPL Estimation

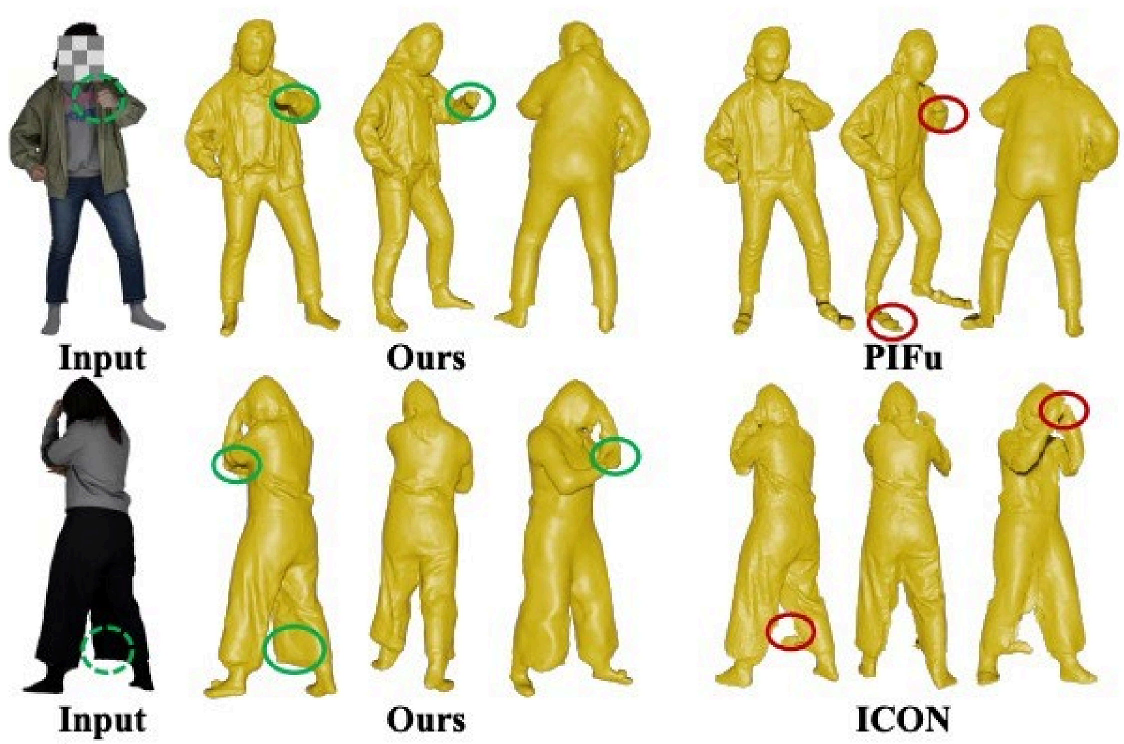


Explicit-based: HaP

- Depth Estimation
  - Convert the RGB input to depth map, and then project the depth map to partial point cloud
- SMPL Rectification
  - Rectify the SMPL pose and shape based on the partial point cloud
- Point Cloud Generation
  - Generate human point cloud conditioned on the partial point cloud and the rectified SMPL



Explicit-based: HaP



Code: Occupancy Network

- Three Elements in Training a Neural Network
  - DataLoader (prepare data for the training process)
  - Network Architecture (use the network to predict)
  - Optimization Objective (the loss to supervise the training)

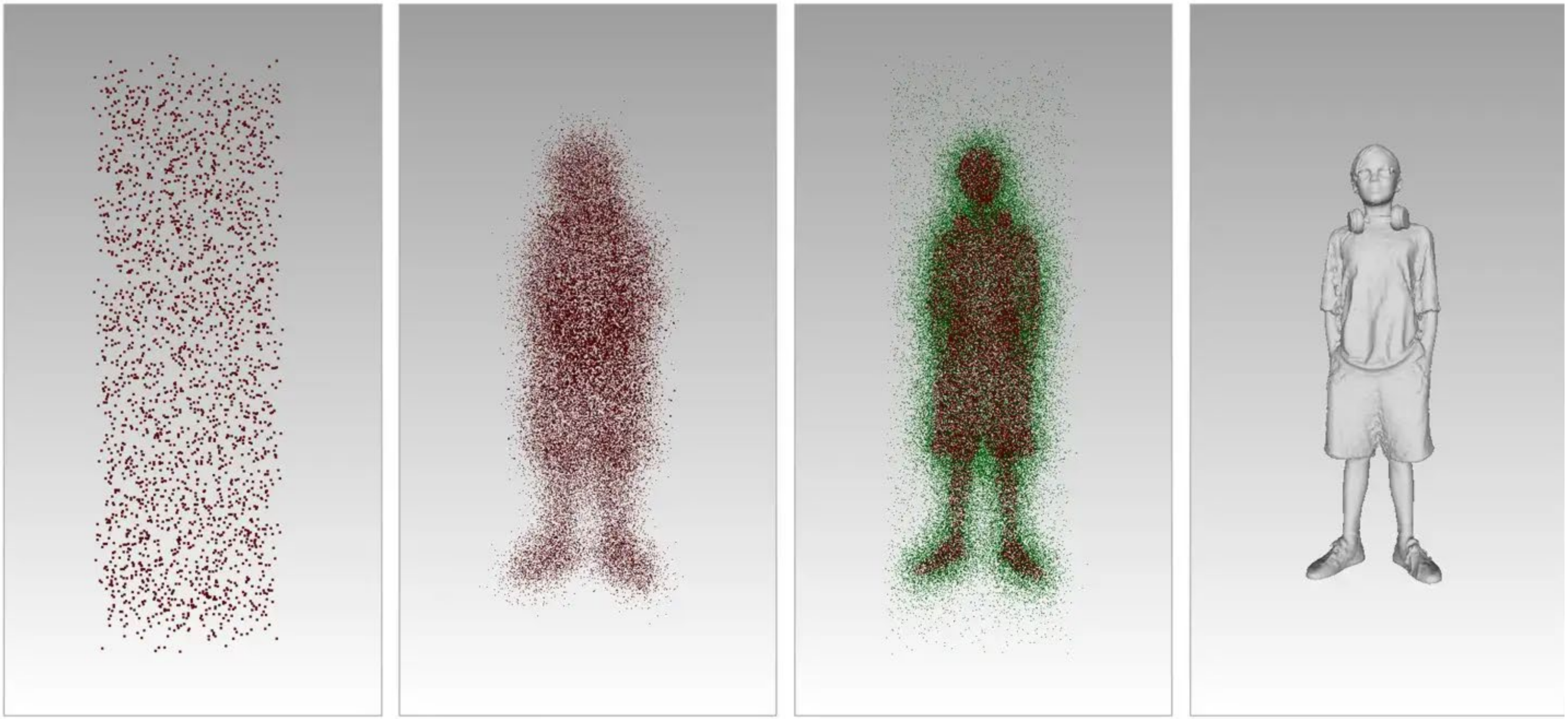
Code: Occupancy Network

- Dataloader
  - __init__
    - Prepare the data paths
  - __len__
    - Return the number of samples
  - __getitem__
    - Get the data and label

Code: Occupancy Network

```
class Shapes3dDataset(data.Dataset):
    ''' 3D Shapes dataset class.
    '''

    def __init__(self, dataset_folder, fields, split=None,
                 categories=None, no_except=True, transform=None):
        ''' Initialization of the the 3D shape dataset.

        Args:
            dataset_folder (str): dataset folder
            fields (dict): dictionary of fields
            split (str): which split is used
            categories (list): list of categories to use
            no_except (bool): no exception
            transform (callable): transformation applied to data points
        '''
        # Attributes
        self.dataset_folder = dataset_folder
        self.fields = fields
        self.no_except = no_except
        self.transform = transform

        # If categories is None, use all subfolders
        if categories is None:
            categories = os.listdir(dataset_folder)
            categories = [c for c in categories
                          if os.path.isdir(os.path.join(dataset_folder, c))]

        # Read metadata file
        metadata_file = os.path.join(dataset_folder, 'metadata.yaml')

        if os.path.exists(metadata_file):
            with open(metadata_file, 'r') as f:
                self.metadata = yaml.load(f)
        else:
            self.metadata = {
                c: {'id': c, 'name': 'n/a'} for c in categories
            }

        # Set index
        for c_idx, c in enumerate(categories):
            self.metadata[c]['idx'] = c_idx

        # Get all models
        self.models = []
        for c_idx, c in enumerate(categories):
            subpath = os.path.join(dataset_folder, c)
            if not os.path.isdir(subpath):
                logger.warning('Category %s does not exist in dataset.' % c)

            split_file = os.path.join(subpath, split + '.lst')
            with open(split_file, 'r') as f:
                models_c = f.read().split('\n')

            self.models += [
                {'category': c, 'model': m}
                for m in models_c
            ]

    def __len__(self):
        ''' Returns the length of the dataset.
        '''
        return len(self.models)

    def __getitem__(self, idx):
        ''' Returns an item of the dataset.

        Args:
            idx (int): ID of data point
        '''
        category = self.models[idx]['category']
        model = self.models[idx]['model']
        c_idx = self.metadata[category]['idx']

        model_path = os.path.join(self.dataset_folder, category, model)
        data = {}

        for field_name, field in self.fields.items():
            try:
                field_data = field.load(model_path, idx, c_idx)
            except Exception:
                if self.no_except:
                    logger.warn(
                        'Error occured when loading field %s of model %s'
                        % (field_name, model)
                    )
                    return None
                else:
                    raise

            if isinstance(field_data, dict):
                for k, v in field_data.items():
                    if k is None:
                        data[field_name] = v
                    else:
                        data['%s.%s' % (field_name, k)] = v
            else:
                data[field_name] = field_data

        if self.transform is not None:
            data = self.transform(data)

        return data
```

Code: Occupancy Network



Sample in bbox　　Sample near surface　　Combine　　Ground truth

Outside the surface: 0
Inside the surface: 1

Code: Occupancy Network

- Network Architecture
  - __init__
    - Define the network architecture
  - __forward__
    - Forward the input into the network and achieve the prediction

Code: Occupancy Network

```
class ResnetPointnet(nn.Module):
    ''' PointNet-based encoder network with ResNet blocks.

    Args:
        c_dim (int): dimension of latent code c
        dim (int): input points dimension
        hidden_dim (int): hidden dimension of the network
    '''

    def __init__(self, c_dim=128, dim=3, hidden_dim=128):
        super().__init__()
        self.c_dim = c_dim

        self.fc_pos = nn.Linear(dim, 2*hidden_dim)
        self.block_0 = ResnetBlockFC(2*hidden_dim, hidden_dim)
        self.block_1 = ResnetBlockFC(2*hidden_dim, hidden_dim)
        self.block_2 = ResnetBlockFC(2*hidden_dim, hidden_dim)
        self.block_3 = ResnetBlockFC(2*hidden_dim, hidden_dim)
        self.block_4 = ResnetBlockFC(2*hidden_dim, hidden_dim)
        self.fc_c = nn.Linear(hidden_dim, c_dim)

        self.actvn = nn.ReLU()
        self.pool = maxpool
```

```
def forward(self, p):
        batch_size, T, D = p.size()

        # output size: B x T X F
        net = self.fc_pos(p)
        net = self.block_0(net)
        pooled = self.pool(net, dim=1, keepdim=True).expand(net.size())
        net = torch.cat([net, pooled], dim=2)

        net = self.block_1(net)
        pooled = self.pool(net, dim=1, keepdim=True).expand(net.size())
        net = torch.cat([net, pooled], dim=2)

        net = self.block_2(net)
        pooled = self.pool(net, dim=1, keepdim=True).expand(net.size())
        net = torch.cat([net, pooled], dim=2)

        net = self.block_3(net)
        pooled = self.pool(net, dim=1, keepdim=True).expand(net.size())
        net = torch.cat([net, pooled], dim=2)

        net = self.block_4(net)

        # Recude to  B x F
        net = self.pool(net, dim=1)

        c = self.fc_c(self.actvn(net))

        return c
```

Code: Occupancy Network

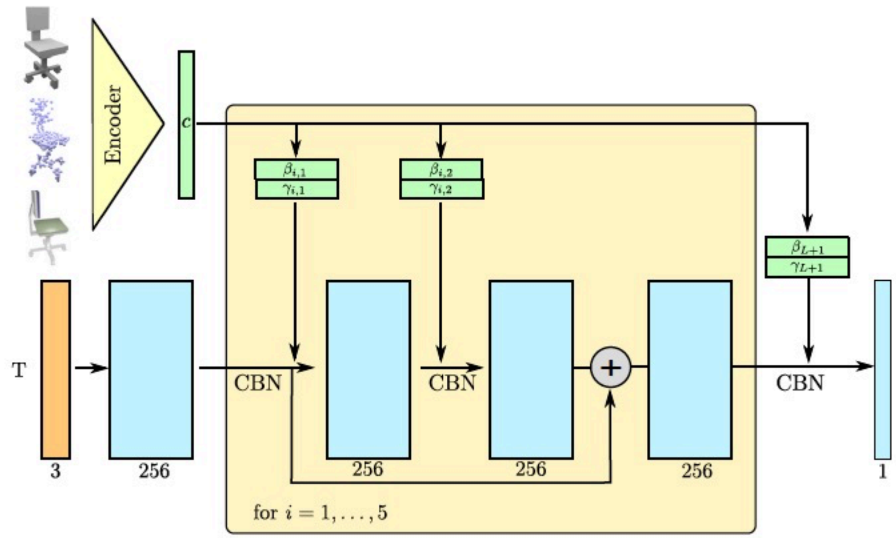


Figure 1: **Occupancy Network Architecture.** We first compute an embedding c of the input. We then feed the input points through multiple fully-connected ResNet-blocks. In these ResNet-blocks, we use Conditional Batch-Normalization (CBN) to condition the network on c. Finally, we project the output of our network to one dimension using a fully-connected layer and apply the sigmoid function to obtain occupancy probabilities.

Code: Occupancy Network

- Loss Function

$$\mathcal{L}_{\mathcal{B}}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \sum_{j=1}^{K} \mathcal{L}(f_\theta(p_{ij}, x_i), o_{ij})$$