Occupancy Plane Learning for 3D Human Reconstruction from RGB-D Images

Viktor B. Ladics
University of Illinois Urbana-Champaign

Jon Vincent Medenilla
University of Illinois Urbana-Champaign

Rishabh Garg
University of Illinois Urbana-Champaign

Abstract

Reconstruction of shapes is an interesting research area and it has wide applications across domains such as robotics, medical imaging, fashion etc. Over the years, there have been several advanced techniques that have been discovered for 3D reconstruction. In this paper, we will discuss a method for reconstruction of human images, where occupancy planes are used to slice a given 2D RGB image. We consider a range of depth values in front of the camera over which the occupancy plane slices the image and the output indicates the presence of human at that depth. We closely follow the work by Zhao et al. [1] to generate occupancy planes though we do not continue to full 3D reconsutrction.

1 Introduction and Related Work

Human shape reconstruction from images has garnered substantial attention. It has useful applications in areas such as image and video editing, AR/VR content, and virtual try-on in the fashion industry. This work focuses on the initial steps of 3D human reconstruction from camera parameters and a single RGB-D image. Specifically, we aim to return occupancy planes for specified depths. Such planes can then be used to reconstruct a human using further processing which this work does not examine.

Methods to perform 3D reconstruction of human body can be fairly classified into two categories, template based and non-parametric based. Our initial exploration included study of advanced techniques such as Skinned Multi- Person Linear (SMPL), variants of SMPL - SMPLify-X and SMPL-X, Pixel-Aligned Implicit Function (PIFu), and Implicit Clothed humans Obtained from Normals (ICON).

PIFuHD is a neural network based approach that was developed by Facebook AI team, it captures fine details of a human accurately, however, this technique fails to perform well at low resolution. The core principle of this technique is similar to PIFu and has an additional layer at the output end to generate details of 3D human model [2].

ICON is a deep learning based model that takes in an RGB image of segmented clothed human and output the 3D reconstructed model. It supports two types of modelling, full body or local feature modelling. ICON relies on SMPL for modelling and estimation, and the modelling happens in a local manner where it does not consider the overall body posture and focuses only of local parts. This helps in achieving state-of-the-art results and 3D poses [3].

SMPL is another human body reconstruction method which considers shape of the body, joint parameters and rotation of joints to build a mesh. It builds close to 7000 points on the body to reconstruct a mesh along with a different loss funtion for each body part for example, face, knee, arm etc.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

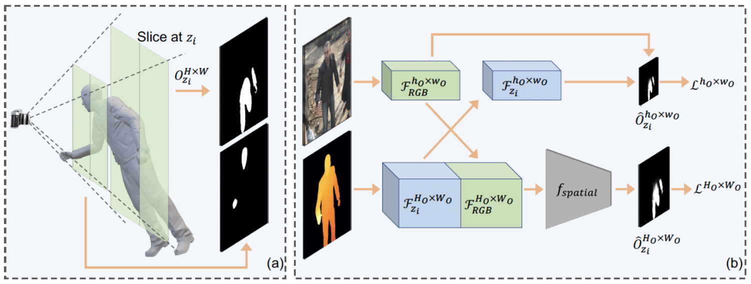
Another successful approach is using occupancy planes which are parallel slices through the 3D space captured by the camera image. One of the merits of this method is that the number of such slices can be varied as per application, hence providing flexibility during training. Furthermore, the model is benefited by the correlation between the planes that collectively predict the neighbouring space of a plane.

In this work, we closely follow the structure outlined in Zhao et al.'s paper on Occupancy Planes for Single-view RGB-D Human Reconstruction [1]. The following sections of this paper outlines in greater detail the method and our specific implementation used to predict the occupancy planes. We then provide some experimental results to showcase our final results.

2 Model Architecture

For this procedure, we are given a depth image, an RGB image, a mask highlighting the person of interest in these images, and a 3D mesh representing the ground truth 3D reconstruction. Using this information, we train a model to generate a set of occupancy planes for a desired set of depth values. Each plane is a slice of the camera's view frustrum at the specified depth and is a binary image where a 0 means the pixel is not within the target person, and a 1 means the pixel is within the target person.

Next we outline the model architecture. At a high level, the processing is separated into two. One pipeline processes the RGB feature while the other process the depth feature. These intermediate results are then combined for one last step of processing which results in the final occupancy plane prediction.



Slice at z_i
$O_{z_i}^{H\times W}$
(a)
$\mathcal{F}_{RGB}^{h_O\times w_O}$
$\mathcal{F}_{z_i}^{h_O\times w_O}$
$\mathcal{L}^{h_O\times w_O}$
$\hat{O}_{z_i}^{h_O\times w_O}$
$\mathcal{F}_{z_i}^{H_O\times W_O}$
$\mathcal{F}_{RGB}^{H_O\times W_O}$
$f_{spatial}$
$\mathcal{L}^{H_O\times W_O}$
$\hat{O}_{z_i}^{H_O\times W_O}$
(b)

Figure 1: Architecture and Occupancy Planes overview. **(a)** Occupancy planes shown in black and white for slices at depths z_i. **(b)** Pipelines for processing the RGB image (top) and the depth image (bottom).

2.1 RGB Image Processing

First, the original RGB image is concatenated with two one-channel features resulting in an image feature with 5 channels. These additional features consist of an edge image computed using a Farid filter and a distance feature which, for each pixel, shows the distance to the visibility mask's boundary. To calculate the edges, we import filters from *scikit-image*, but we first convert the original RGB image into greyscale then feed it into the imported function: *filters.farid(grey_img)*. Similary, we use the greyscale images to calculate the euclidean distance between each pixel and the closest non-zero value of the mask. To calculate this, we use *scipy.ndimage.distance_transform_edt(grey_img)*.

The image feature is processed with two networks:

$$F_{RGB} = f_{RGB}(f_{FPN}(I_{RGB}))$$

Here I_{RGB} is the image feature with f_{FPN} being a feature pyramid network and f_{RGB} being a simple convolution neural network (CNN).

The feature pyramid network (FPN) [5] allows the model to detect objects at various scales. For our implementation, we used ResNet50 [4] as the pyramid backbone (as per Zhao et al.) to allow for thorough processing at a large range of scales. Other pyramid networks incur a large processing cost, however, the use a top-down processing in an FPN allows for high-resolution features to contain more semantic information that is normally only present at lower resolutions following processing. The large number of features that result from the ResNet (256) allows for thorough investigation of the input data to generate a range of features. The 256 features from the the high-resolution layer of the FPN are input into the CNN which is a 3-layer network with convolutional layers of dimension (256, 128, 3),(128, 128, 3), and (128, 128, 1). These tuples represent the number of input channels, number of output channels, and filter width respectively. A Group norm and ReLU activation function is used between each convolutional layer. Here we opted for group norm since the batch size will be low.

2.2 Depth Image Processing

In parallel to the image feature, the depth feature is created by processing the depth image.

$$I_{z_i}[x,y] = PE(z_i - Depth[x,y])$$
$$F_{z_i} = f_{depth}(I_{z_i})$$

Here $Depth[x,y]$ is the depth image for a pixel coordinate (x,y). This single channel feature undergoes positional encoding to form the depth difference image. This is effectively a measure of how far each point on the plane at a depth of z_i is behind (or in front) of the front surface of the person of interest. The positional encoding is done as follows:

$$PE(pos) = (PE_0(pos), PE_1(pos), ..., PE_{63}(pos)$$
$$PE_{2t}(pos) = sin(\frac{50 \cdot pos}{200^{2t/64}})$$
$$PE_{2t+1}(pos) = cos(\frac{50 \cdot pos}{200^{2t/64}})$$

This depth difference image consists of the 64 channels associated with the positional encoding. This is then fed into f_{depth} which is another CNN with two convolutional layers with dimensions (64,128, 1) and (128, 128, 1). Group norm and ReLU activation functions have been used here as well between convolutional layers.

2.3 Combining Features

With the image feature and the depth feature resulting from the previous processing steps, one final processing step occurs to combine the results:

$$\hat{O}_{z_i} = f_{spatial}([F_{RGB}; F_{z_i}])$$

The occupancy plane for the given depth of z_i is $\hat{O}_{z_i}$. $f_{spatial}$ is another CNN with a configuration of (256,128,3), (128,128,3) and (128,1,1) with group norm and ReLU activations between the convolutional layers. This network takes a concatenation ($[;]$) of the image feature f_{RGB} and the depth feature f_{z_i} as input. This forward pass occurs for each value of z_i that is provided for a particular input scene.

2.4 Loss Functions

For training, all the parameters across all the previously mentioned networks are processed in a single loss function. This ensures that the entire pipeline is optimized based on the accuracy of the predicted occupancy plane with respect to the ground truth occupancy planes.

The total loss consists of a weighted combination of a binary cross entropy (BCE) loss and a DICE loss. Furthermore, losses are considered at two different resolutions. In our implementation, one loss measurement occurs at the original input image resolution of 800x1280. Another intermediate loss is calculated for the intermediate features at a resolution of h0xw0 which is 400x640.

$$Loss_{total} = Loss_{lowres} + Loss_{highres}$$

$$Loss = Loss_{BCE} + Loss_{DICE}$$

The total loss that is being minimized is the sum of the high resolution and low resolution losses. Each of these are a weighted sum of the BCE and DICE losses. In our case, both have an equal weighting equal to 1.

Intermediate Resolution Features

The occupancy planes at this intermediate resolution are calculated using a dot product of the lower resolution image feature and a lower resolution depth feature. This encourages the two features to be strongly correlated during training.

$$\hat{O}_{z_i}^{h_0 x w_0} =< F_{RGB}^{h_0 x w_0}, F_{z_i}^{h_0 x w_0}[x, y, \cdot] >$$

To get the low resolution depth feature, the original depth difference image is downsampled to the low resolution and is fed into f_{depth}. The low resolution image feature is obtained directly from the output of $f_R GB$. The higher resolution image feature is the result of bilinearly upsampling this output.

BCE and Dice Loss

Equations for the BCE and DICE loss are shown below. Z_N is the set of depth values being processed, $mask$ is the provided mask which highlights the person of interest in the image, and $\hat{O}$ and O represent the predicted and ground truth occupancy planes respectively.

$$Loss_{BCE} = \frac{1}{|Z_N| \cdot Sum(mask)} \sum_{z_i \in Z_n} mask[x,y](O_{z_i}[x,y] \cdot log\hat{O}_{z_i}[x,y] + (1 - O_{z_i}[x,y]) \cdot log(1 - \hat{O}_{z_i}[x,y]))$$

$$Loss_{DICE} = \frac{1}{|Z_N|} \sum_{z_i \in Z_n} \frac{2 \cdot Sum(mask \cdot O_{z_i} \cdot \hat{O}_{z_i})}{Sum(mask \cdot O_{z_i}) + Sum(mask \cdot \hat{O}_{z_i})}$$

For the BCE loss, the sum occurs over the pixels where the $mask[x, y] = 1$. During implementation we achieved this by multiplying the interior of the sum by the mask value at that pixel since the mask is binary.

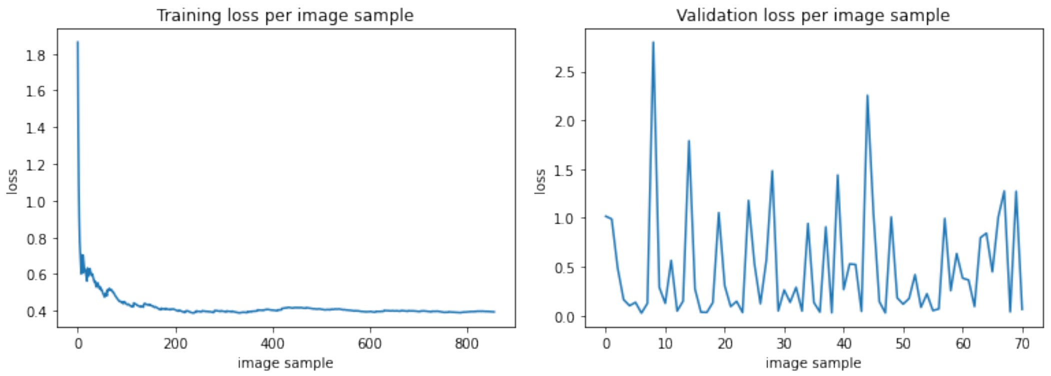
During training, we were provided with code to process the mesh of each image. From this mesh and the camera properties, we were able to extract the set of ground truth occupancy values given a set of randomly selected depth values from a range $[z_{min}, z_{max}]$. This closest distance to the camera and the depth of the mesh were gathered from the mesh using the provided code. We then randomly sampled 5 different depth values for each input data point. One forward pass produced an occupancy plane for each depth. These 5 planes were then collectively processed to produce the losses using the equations above. We used an Adam optimizer with a learning rate of 0.001.

3 Experimentation

Here, we introduce training details and procedure. First, we split the dataset into training (80%) and validation (20%). To make data sampling easier, we used *SubsetRandomSampler* as the sampler for the Dataloader we imported from *torch.data.utils*. Initially, sampling was taking 10 minutes per image to just print out samples. Sampling sped up once we set *persisent_workers=True* and *num_workers=4*.

Using a custom dataloader, we obtain the images, ground truth O-planes, depth array, mask, low resolution mask, and the z-values to be used for slicing the ground-truth O-planes. We pass these data to the step function to obtain the losses.

We did our code development both on VSCode and Google Colab, with most model integration done on Google Colab. Due to the GPU limitation in Google Colab, we moved our actual training to Kaggle. Using 2 Tesla 4 GPU's, we trained our model for 3 epochs, with a total run time of 6.5 hours.



(a) Training losses

(b) Validation losses

Figure 2: Losses

4 Results and Discussion

In this section, we show our results for training, validation, and testing. Figure 2 shows our training and validation set losses. Figure 2a shows the average loss per input over the course of training up to 800 images. We can see that the losses swiftly went down from 1.8 to 0.4 in a span of 200 image samples. It also seems to have converged just below 0.4 which is a decent metric given that the total loss was a sum of two losses.

For the validation loss, we measured the loss for each image in the validation set as shown in Figure 2b. These losses remain very large compared to our training losses which suggests the model is overfitting. The fast convergence of the losses followed by minimal variation also suggests that the learning process can be improved. One way we may be able to alleviate this is to introduce a learning rate decay, i.e. divide our original value of 0.001 by 10 after each epoch. Furthermore, the large variation in the losses on the validation set suggests that the model has not generalized well. Although our training losses converged quickly, increased training time could help alleviate this issue since we only trained for 3 epochs and only 5 planes per epoch. Zhao et al. had a similar finding. In their training, they used 10 planes per input and found that for a lower number (5) resulted in poorer performance. Another technique that could have improved our training is increasing the batch size to 2 or 4. We chose our batch size to 1, due to the design of our overall network. However, using a larger batch size could have helped our model generalize better.

Initially, we found that our backpropagation was very slow. This was in large part due to inefficiencies in our PyTorch code. After some optimization we found significant improvements which helped reduce this issue and make our training time feasible. Even still, we opted to train at 5 planes instead of 10 to make better use of our image dataset. We found that performance of Kaggle was better than Google Colab. It takes 40 seconds per image per step on Colab, while it only takes 30 seconds on Kaggle. With some more PyTorch experience, our code could likely be optimized further to allow more processing.

Another consideration is that we conducted our processing and loss calculations at higher resolutions than in previous work. We opted to maintain the high resolution for ground truth occupancy planes and upsample our predictions for loss calculation, as opposed to downsampling the ground truth occupancy planes. However, reducing the resolution of the images being processed during the forward pass would surely reduce training time and potentially improve performance due to more semantic information.

5 Conclusions

In this project we conducted of partial investigation of 3D human reconstruction by creating a model for generating occupancy planes. These planes serve as an alternate mechanism for reconstruction compared with previous work. The prediction of an entire plane allows for the model to benefit from correlations between predictions. With these occupancy planes, algorithms such as the marching cube algorithm can be used to generate the full reconstruction. The prediction of entire planes can allow this solution to be better suited in challenging situations where occlusion occurs. Further work could investigate the use of more complex CNNs for processing both the RGB and depth features. In our case, further training could also be beneficial to provide more reliable and robust results compared with what were shown here.

References

[1] Park, Y., Dang, L. M., Lee, S., Han, D., Moon, H. (2021). Multiple object tracking in deep learning approaches: A survey. Electronics, 10(19), 2406.

[2] Saito, S., Simon, T., Saragih, J., & Joo, H. (2020). Pifuhd: Multi-level pixel-aligned implicit function for high-resolution 3d human digitization. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 84-93).

[3] Xiu, Y., Yang, J., Tzionas, D., & Black, M. J. (2022, June). Icon: Implicit clothed humans obtained from normals. In 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 13286-13296). IEEE.

[4] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. CVPR. 2016. arXiv preprint arXiv:1512.03385.

[5] Lin, T. Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature pyramid networks for object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2117-2125).