NeuWigs: A Neural Dynamic Model for Volumetric Hair Capture and Animation Supplementary Material

1. Appendix

1.1. Network Architecture

Here we provide details about how we design our neural networks and further information about training.

Encoder. As training a point cloud encoder solely is extremely unstable, we first train an image encoder and use it as a teacher model to train the point cloud encoder. In practice, we train two encoders for our hair branch together. Here we first illustrate the structure of both encoders. We will go back to how we train them and use them later. One of the encoders is an image encoder which is a convolutional neural network (CNN) that takes multiple view images as input. We denote the image encoder as $\mathcal{E}_{img}$ The other one is $\mathcal{E}$ which is a PointNet encoder that takes either an unordered hair point cloud p_t or a tracked hair point cloud q_t as input. Positional encoding [4] is applied to the raw point cloud coordinate before it is used as the input to the network. We find this is very effective to help the network in capturing high frequency details. In practice, we use frequencies of x^2 where x ranges from 1 to 7. We show the detailed architecture of $\mathcal{E}_{img}$ in Tab 1. The architecture of the point cloud encoder $\mathcal{E}$ is shown in Tab 2. Both of the two encoders $\mathcal{E}$ and $\mathcal{E}_{img}$ can produce a latent vector in size of 256, which are supposed to describe the same content. Their output will be passed to $\mathcal{E}_\mu$ and $\mathcal{E}_\sigma$ which are two linear layers that produce $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ of $\boldsymbol{z_t}$ respectively.

Point Decoder. We use a 3-layer MLP as the point decoder $\mathcal{D}$, which takes a 1d latent code $\boldsymbol{z_t}$ as input and outputs the coordinate of the corresponding tracked point cloud q_t. We show the architecture of $\mathcal{D}$ in Tab 3.

Volume Decoder. The volumetric model is a stack of 2D deconv layers. We align the x-axis and y-axis of each volume and put them onto a 2D imaginary UV-space. Then we convolve on them to regress the z-axis content for each of the x,y position. We show the architecture of the volume decoder in Tab 4. In our setting, we have two seperate volume decoder for both RGB volume and alpha volume.

Dynamic Model. We use three different inputs to the dynamic model T^{2}M, namely the hair encoding $\boldsymbol{z}_{t-1}$ at the previous frame $t-1$, the head velocity $\boldsymbol{h}_{t-1}$ and $\boldsymbol{h}_{t-2}$ from the previous two frames $t-1$ and $t-2$, and the head relative gravity direction $\boldsymbol{g}_t$ at the current frame t. We first encode $\{\boldsymbol{h}_{t-1}, \boldsymbol{h}_{t-2}\}$ and $\boldsymbol{g}_t$ into two 1d vectors with 128 dimensions respectively. Then, we concatenate them together with encoding $\boldsymbol{z}_{t-1}$ as the input to another MLP to regress the next possible hair state encoding $\boldsymbol{z}_t$. As in Tab. 5, we show the flow of T^{2}M. For the head velocity branch, we first extract the per-vertex velocity $\boldsymbol{h}_{t-1} = \boldsymbol{x}_t - \boldsymbol{x}_{t-1}$ where $\boldsymbol{x}_t$ is the coordinate of the tracked head mesh at frame t. To be noted, here the $\boldsymbol{h}_{t-1}$ contains only the information of the rigid head motion but not any other non-rigid motion like expression change. This representation of head motion is redundant theoretically, but we find it helps our network to converge better when compared to just using the pure 6-DoF head rotation and translation. We then reshape it and use it as the input to a two layer MLP to extract a 1d encoding of size 128. For the gravity branch, we first encode the gravity direction $\boldsymbol{g}_t$ with cosine encoding [4]. The output of the dynamic model is the mean μ_{t+1} and standard deviation σ_{t+1} of the predicted hair state $\boldsymbol{z}_{t+1}$.

	Encoder $\mathcal{E}_{img}$
1	`Conv2d`(3, 64)
2	`Conv2d`(64, 64)
3	`Conv2d`(64, 128)
4	`Conv2d`(128, 128)
5	`Conv2d`(128, 256)
6	`Conv2d`(256, 256)
7	`Conv2d`(256, 256)
8	`Flatten`()
9	`Linear`(256×n_{inpimg}×15, 256)

Table 1. **Encoder $\mathcal{E}_{img}$ architecture**. Each `Conv2d` layer in the encoder has a kernel size of 3, stride of 1 and padding of 1. Weight normalization [9] and untied bias are applied. After each layer, except for the last two parallel fully-connected layers, a Leaky ReLU [3] activation with a negative slope of 0.2 is applied. Then a downsample layer with a stride of 2 is applied after every conv2d layer. The first linear layer takes the concatenation of all towers from different image views as input. n_{inpimg} stands for much many views we take.

	Encoder $\mathcal{E}$
1	`Conv2d`(3, 128)
2	`Conv2d`(128, 256)
3	`Conv2d`(256, 256)
4	`Conv2d`(256, 256)
5	`Conv2d`(256, 512)
6	`Conv2d`(512, 512)
7	`Conv2d`(512, 512)
8	`Conv2d`(512, 1024)
8	`MAM pooling()`
9	`Linear`(1024×3, 512)
10	`Linear`(512, 256)
11	`Linear`(256, 256)

Table 2. **Encoder $\mathcal{E}$ architecture**. We use a $\mathcal{E}$ structure similar to PointNet [6]. All `Conv2d` uses a kernel of 1 and stride of 1, which serves as a shared MLP. We only use `Conv2d` for simpler implementation. After each `Conv2d` layer, a Leaky ReLU [3] activation with a negative slope of 0.2 is applied. Then we use a MAM pool layer to aggregate features from all points. MAM stands for min, avarage and max pooling, which concatenates the results of min, average and max pooling into one. Then, two linear layers are applied to the output of MAM pooling and generate a 256 latent vector.

	Decoder $\mathcal{D}$
1	`Linear`(256, 256)
2	`Linear`(256, 256)
2	`Linear`(256, 4096×3)

Table 3. **Decoder $\mathcal{D}$ architecture**. We use an MLP with three `Linear` layers as the decoder $\mathcal{D}$. After each layer except the last layer, a Leaky ReLU [3] activation with a negative slope of 0.2 is applied.

1.2. Training details

Dataset and Capture Systems. Following the setting in HVH [10], we also captured several video sequences with scripted hair motion performed under different hair styles for animation tests. During the capture, we ask the participants to put on different kind of hair wigs and perform a variety of head motions like nodding, swinging and tilting. For each action, they performed multiple times and at both slow and fast speed. To collect a demonstration set for animation, we also ask the participants to put on a hair net (bare head) and perform the same set of motions as when they are wearing a hair wig.

Hair Point Flow Estimation. There are three steps for computing the hair point flow, namely per-point feature descriptor extraction, feature matching and flow filtering. In the first step, we compute a per-point feature descriptor based on the distribution of each point's local neighboring points. In the second step, we match the points from two adjacent time steps based on the similarity between their feature descriptor. In the last step, we filter out outlier flows that are abnormal.

To compute the point feature descriptor, we construct Line Feature Histograms (LFH) inspired by Point Feature Histograms (PFH) [8]. The LFH is a historgram of a 4-tuple that describes the spatial relationship between a certain point $\boldsymbol{p}_1^t$ and its neighboring poit $\boldsymbol{p}_2^t$. As shown in Fig. 1, we visualize two points $\boldsymbol{p}_1^t \in \mathbb{R}^3$ and $\boldsymbol{p}_2^t \in \mathbb{R}^3$ from the same time step t. Given $\boldsymbol{p}_1^t$ and $\boldsymbol{p}_2^t$, we define the following four properties that describe their spatial relationship. The first one is the relative position of $\boldsymbol{p}_2^t$ with respect to $\boldsymbol{p}_1^t$, which is $\boldsymbol{d}_{1,2}^t = \boldsymbol{p}_2^t - \boldsymbol{p}_1^t$. Then we can

	Volume Decoder	
	global encoding $\boldsymbol{z}_t$	per-point hair feature
	repeat	
	concat	
1	`Linear`(320, 512)	
2	`deconv2d`(512, 256)	
3	`conv2d`(256, 256)	
4	`deconv2d`(256, 256)	
5	`conv2d`(256, 256)	
6	`deconv2d`(256, 128)	
7	`conv2d`(128, 128)	
8	`deconv2d`(128, 16×ch)	

Table 4. **Architecture of the Volume Decoder**. We first repeat the global encoding $\boldsymbol{z}_t$ into the shape of the per-point hair feature. The per-point hair feature is a tensor that is shared across all time frames. We then concatenate those two into one. Each layer except for the last one is followed by a Leaky ReLU layer with a negative slope of 0.2. Each `deconv2d` layer has a filter size of 4, stride size of 2 and padding size of 1. Each `conv2d` layer has a filter size of 3, stride size of 1 and padding size of 1. ch stands for the channel size of the output. It is set to 3 if it is an rgb decoder and 1 for a alpha decoder.

	Temporal Transfer Module (`T`^{2}`M`)		
1	head velocity $\{\boldsymbol{h}_{t-1}, \boldsymbol{h}_{t-2}\}$	head relative gravity $\boldsymbol{g}_t$	hair state $\boldsymbol{z}_{t-1}$
2	`Linear`(7306×3, 256)	cosine encoding	
3	`Linear`(256, 128)		
4	`Linear`(539, 256)		
5	`Linear`(256, 256)		
6	`Linear`(256, 256)		
7	`Linear`(256, 256)	`Linear`(256, 256)	

Table 5. **Temporal Transfer Module** (`T`^{2}`M`). We first encode the head velocity $\{\boldsymbol{h}_{t-1}, \boldsymbol{h}_{t-2}\}$ and head relative gravity $\boldsymbol{g}_t$ into 1d vectors, with a 2-layer MLP and cosine encoding respectively. Then we concatenate hair state $\boldsymbol{z}_{t-1}$ with those vectors to serve as the input to another MLP. The last two layers will be regressing the mean μ_{t+1} and standard deviation σ_{t+1} of the predicted hair state $\boldsymbol{z}_{t+1}$. All `Linear` except for the last two are followed by a Leaky ReLu activation with a negative slope of 0.2.

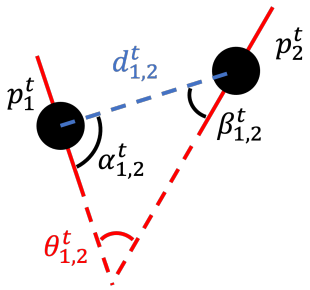

Figure 1

compute the relative distance as $||\boldsymbol{d}_{1,2}^t||_2 \in \mathbb{R}$. The second term is the angle $\theta_{1,2}^t$ between $\boldsymbol{dir}(\boldsymbol{p}_1^t)$ and $\boldsymbol{dir}(\boldsymbol{p}_2^t)$, where $\boldsymbol{dir}(\boldsymbol{x})$ is the line direction of $\boldsymbol{x}$ from [5]. The last two terms are the angles $\alpha_{1,2}^t$ and $\beta_{1,2}^t$ between $(\boldsymbol{dir}(\boldsymbol{p}_1^t), \boldsymbol{d}_{1,2}^t)$ and $(\boldsymbol{dir}(\boldsymbol{p}_2^t), \boldsymbol{d}_{1,2}^t)$ respectively. For all intersections, we take the acute angle, which means $\theta_{1,2}^t, \alpha_{1,2}^t$ and $\beta_{1,2}^t$ are in $[0, \pi/2]$. Thus, the 4-tuple we used to create $LFH(\boldsymbol{p}_1^t)$ is $(||\boldsymbol{d}_{1,2}^t||_2, \theta_{1,2}^t, \alpha_{1,2}^t, \beta_{1,2}^t)$ and we normalize the histogram by its l2 norm. The designed LFH has three good properties. As we use the normalized feature, it is density invariant. As $\theta_{1,2}^t, \alpha_{1,2}^t$ and $\beta_{1,2}^t$ are always acute angles, the feature is also rotation and flip invariant, meaning that if we flip or rotate $\boldsymbol{dir}(\boldsymbol{p}_1^t)$ the histogram will remain unchanged. This design helps us to get a more robust feature descriptor for matching. We set the resolution for each entry of the 4-tuple to be 4 and it results in a descriptor in size of 256.

In the second step, we compute the correspondence between points from adjacent time frames t and $t + t_\delta$ where $t_\delta \in \{-1, 1\}$. We use the method from Rusu *et al.* [7] to compute the correspondence between two point clouds from t and $t + t_\delta$. To further validate the flow we get, we use several heuristics to filter out obvious outliers. We first discard all the flows that have a large magnitude. As the flow is computed between two adjacent frames, it should not be large. The second heuristic we use to filter the outliers is cycle consistency, where we compute the flow both forward and backward to see if we can map back to the origin. If the mapped back point departs too far away from the origin, we discard that flow.

Training of Encoder. As mentioned before, we train two encoders $\mathcal{E}$ and $\mathcal{E}_{img}$ together. In practice, we find that directly training $\mathcal{E}$ is not very stable and might not lead to convergence. Thus, we learn the two encoders in a teach-student manner, where we use $\mathcal{E}_{img}$ as a teach model to train $\mathcal{E}$. We denote $\boldsymbol{x}_{img,t}$ as the output of $\mathcal{E}_{img}$ and $\boldsymbol{x}_{pt,t}$ as the output of $\mathcal{E}$. Then, we formulate the following MSE loss to enforce the $\mathcal{E}$ to output similarly to $\mathcal{E}_{img}$:

$$\mathcal{L}_{ts} = ||\boldsymbol{x}_{img,t} - \boldsymbol{x}_{pt,t}||_2,$$

where we restraint the gradient from $\mathcal{L}_{ts}$ from back-propagating to $\mathcal{E}_{img}$ while training.

	MSE↓	PSNR↑	SSIM↑	LPIPS↓
$\mathcal{E}$ on SEEN	29.48	34.05	0.9657	0.1109
$\mathcal{E}_{img}$ on SEEN	29.44	34.05	0.9657	0.1109
$\mathcal{E}$ on UNSEEN	34.97	33.21	0.9587	0.1209
$\mathcal{E}_{img}$ on UNSEEN	37.36	32.94	0.9559	0.1333

Table 6. **Metrics on Novel Views.** We show quantitative results of different encoders under both SEEN and UNSEEN sequence of the same hair styles.

1.3. Ablation on Different Encoders

We show quantitative evaluations on rendering quality of different encoders on both the SEEN and UNSEEN sequences in Tab. 6. Our $\mathcal{E}$ performs similarly to the $\mathcal{E}_{img}$ on the novel views of the SEEN sequence. This result is as expected due to the nature of teach-student model and we train our model on the SEEN sequence with the training views. On the UNSEEN sequence, we find our $\mathcal{E}$ performs better than $\mathcal{E}_{img}$. We hypothesis that this is because there is a smaller domain gap between the point clouds from SEEN sequence and UNSEEN sequence while the multi-view images vary a lot due to the head motion. The CNN is not good for handling such changes as the head motion is not 2D translation invariant after projection while point encoder can process point clouds with better 3D structure awareness.

1.4. Ablation on Different Designs of the Dynamic Model

We show the comparisons of different dynamic models and per-frame driven models in Tab. 7. We find that our model offers a significant improvement over the per-frame driven model that takes head pose or motion as input. This result is because the hair motion is not only determined by the head pose or the previous history of head pose but also the initial states of the hair. In Fig. 2, we visualize how each model drifts by plotting the Chamfer distance between the regressed point cloud and the ground truth point cloud. We find that adding head relative gravity direction can improve the model performance on slow motions.

	MSE(↓)	PSNR(↑)	SSIM(↑)	LPIPS(↓)	ChamDis(↓)
pf w/ hair img	37.36	32.94	0.9559	0.1333	10.47
pf w/ hair pts	34.97	33.21	0.9587	0.1209	10.46
pf w/ head pos	47.43	32.01	0.9458	0.1522	18.94
pf w/ head mot	40.25	32.64	0.9508	0.1333	13.31
dyn w/o cos	44.96	32.26	0.9458	0.1327	25.49
dyn w/o cyc	45.22	32.23	0.9453	0.1335	26.79
dyn w/o grav	40.12	32.64	0.9504	0.1268	13.76
dyn	38.49	32.80	0.9532	0.1211	11.12

Table 7. **Ablation of Different Dynamic Models.**

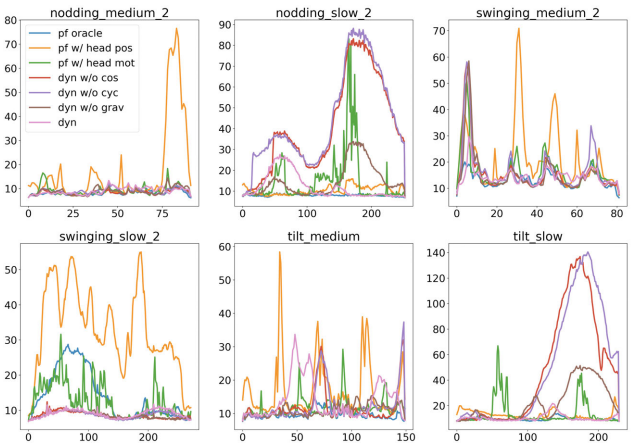

Figure 2. **ChamDist v.s. time.** We plot Chamfer distance vs. time of different dynamic models to show drifting.

1.5. Effect of the Initialization

We test how robust our model is to the initialization of the hair point cloud. In Fig. 3, we show animation results from models of two different hair styles **(hs)** with different initialization hair point clouds. The results look sharp when the model is matched with the correct hair style, but blurry when we use mismatched hair point clouds for initialization. However, we find our model self-rectifies and returns to a stable state after a certain number of iterations. This ability of stablization could be partially due to the model prior stored in the point encoder as well as the pooling operation in the point encoder that denoise the inputs.

Figure 3. **Effect of Initialization.** We initialize two different models (hs1 mod. and hs2 mod.) with two different hair point clouds (hs1 and hs2) in two time steps. The green box indicates matched initialization while orange indicates mismatched initialization. Although the mismatched initialization shows blurry results at first, the model automatically corrects itself when there is no head motion.

	Seq01	Seq02	Seq03
HVH [10]	0.6685	0.4121	0.3766
Ours	**0.8289**	**0.9243**	**0.8571**

Table 8. **IoU**(↑) between rendered hair silhouette and ground truth hair segmentation.

	MSE(↓)	PSNR(↑)	SSIM(↑)	LPIPS(↓)
MVP	66.21	30.36	0.9291	0.2830
Ours	**29.44**	**34.05**	**0.9657**	**0.1109**

Table 9

1.6. Effect of Segmentation loss

In HVH [10], certain level of hair and head disentanglement is achieved even without using any supervision like segmentation. In Tab 8, we show the IoU between the rendered silhouette of hair volumes and ground truth hair segmentatation of the different methods.

1.7. Further Ablation on the Point Encoder

To further study the point encoder $\mathcal{E}$'s ability to denoise the encoding, we tested the encoder with inputs that containing different level of noise. Similar to the study in the main paper, we first extract a fixed encoding z and add noise n to it as $\hat{z}=z+n$ and do noise removal as $\bar{z}=\mathcal{E}(\mathcal{D}(\hat{z}))$. We extend this by adding different levels of noise by multiplying n with different scalars. Please refer to the video navigation page[1] for more details.

1.8. Further Ablation on Novel View Synthesis

We compare our method with MVP [2] on the longer sequences we captured with scripted head motion. Reconstruction related metrics are shown in Tab. 9. We found that NeRF-based methods can not fit to longer sequences properly. This problem might be due to the large range of motion exhibited in the videos as well as the length of the video. HVH is not applicable because it does not support hair tracking across segmented sequences of different hair motion. Compared to MVP, we achieve better reconstruction accuracy and improved perceptual similarity between the rendered image and ground truth with the hair specific modeling in our design.

1.9. Animation on Bald Head Sequences

We show animation results driven by head motions both on lab multi-view video captures and in-the-wild phone video captures. For results on in-the-wild phone video captures, please refer to the supplemental videos[1]. For phone

[1] https://ziyanw1.github.io/neuwigs/resources/index.html

Figure 4. **Animation on Bald Sequence.** We animate a straight brown hair with a nodding head.

Figure 5. **Animation on Bald Sequence.** We animate short blue pigtails with a nodding head.

captures, we ask the participants to face the frontal camera of the phone and perform different head motions. Then, we apply the face tracking algorithm in [1] to obtain face tracking data that serves as the input to our method. The initial hair state of the phone animation is sampled from the lab captured dataset. We find our model generates reasonable motions of hair under head motions like swinging and nodding.

We also test our model on multi-view video captures from the lab. As shown in Figs. 4 and 5, our model generates reasonable hair motions with respect to the head motion while preserving multi-view consistency.

References

[1] Chen Cao, Tomas Simon, Jin Kyu Kim, Gabe Schwartz, Michael Zollhoefer, Shun-Suke Saito, Stephen Lombardi, Shih-En Wei, Danielle Belko, Shoou-I Yu, Yaser Sheikh, and Jason Saragih. Authentic volumetric avatars from a phone scan. *ACM Trans. Graph.*, 41(4), jul 2022. 5

[2] Stephen Lombardi, Tomas Simon, Gabriel Schwartz, Michael Zollhoefer, Yaser Sheikh, and Jason Saragih. Mixture of volumetric primitives for efficient neural rendering. *ACM Transactions on Graphics (TOG)*, 40(4), July 2021. 4

[3] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, 2013. 1, 2

[4] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision*. Springer, 2020. 1

[5] Giljoo Nam, Chenglei Wu, Min H Kim, and Yaser Sheikh. Strand-accurate multi-view hair capture. In *Proceedings of*

Figure 6. **Animation on Bald Sequence.** We animate curly blonde pigtails with a rotating head.

Figure 7. **Animation on Bald Sequence.** We animate burly blonde pigtails with a nodding head.

the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 155–164, 2019. 3

[6] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017. 2

[7] Radu Bogdan Rusu, Nico Blodow, Zoltan Csaba Marton, and Michael Beetz. Aligning point cloud views using persistent feature histograms. In *2008 IEEE/RSJ international conference on intelligent robots and systems*, pages 3384–3391. IEEE, 2008. 3

[8] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Persistent point feature histograms for 3d point clouds. In *Proc 10th Int Conf Intel Autonomous Syst (IAS-10), Baden-Baden, Germany*, pages 119–128, 2008. 2

[9] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29, 2016. 1

[10] Ziyan Wang, Giljoo Nam, Tuur Stuyck, Stephen Lombardi, Michael Zollhoefer, Jessica Hodgins, and Christoph Lassner. Hvh: Learning a hybrid neural volumetric representation for dynamic hair performance capture, 2021. 2, 4

Figure 8. **Animation on Bald Sequence.** We animate curly blonde pigtails with a rotating head.

Figure 9. **Animation on Bald Sequence.** We animate a curly ash blonde hair with a nodding head.