Learning Compositional Radiance Fields of Dynamic Human Heads – Supplemental Document –

Ziyan Wang[1,3] Timur Bagautdinov[3] Stephen Lombardi[3] Tomas Simon[3]
Jason Saragih[3] Jessica Hodgins[1,2] Michael Zollhöfer[3]
[1]Carnegie Mellon University [2]Facebook AI Research [3]Facebook Reality Labs Research

A. Video Results

Please refer to the `videos` directory for video results and `index.html` for a video navigation. The results can also be found on our project page [1].

B. Training Details

B.1. Network Architecture

There are three main neural networks used in our methods: 1) **Encoder**, that regresses image input to the statistics $\boldsymbol{\mu}, \boldsymbol{\sigma}$ of a latent space vector $\mathbf{z} \in \mathbb{R}^{256}$; 2) **Decoder**, a 3D convolutional network that regresses the latent vector $\mathbf{z}$ to a coarse-level volume $\mathbf{V_p}$ of log differential opacity $\widetilde{\sigma}_\mathbf{p}$, color $\widetilde{\mathbf{c}}_\mathbf{p}$, and spatial scene features $\mathbf{f_p}, \mathbf{f}^v_\mathbf{p}$; 3) **Refinement MLP**, that takes in the coordinate of a spatial location $\mathbf{p}$ as well as its corresponding spatial local feature from the coarse-level volume $\mathbf{f_p}, \mathbf{f}^v_\mathbf{p}$ and outputs the fine-level log differential opacity $\sigma_\mathbf{p}$ and color $\mathbf{c_p}$.

For the image encoder and volume decoder, please refer to Table 1 and Table 2 for their architecture. To better model the view-dependent effects, we employ two decoders to regress the color and opacity at the coarse level. The common structure of each decoder is shown in Table 2. For the color decoder, the input is the concatenation of the latent vector $\mathbf{z}\in\mathbb{R}^{256}$ and the camera view direction $\mathbf{v}\in\mathbb{R}^3$, thus the final input size is $N^c_{in} = 256 + 3$ and the output size is $N^c_{out} = 3$, with a parallel branch producing view-dependent spatial scene features $\mathbf{f}^v_\mathbf{p}\in\mathbb{R}^{32}$. Similarly, the opacity decoder only takes the latent vector $\mathbf{z}\in\mathbb{R}^{256}$ as input and regresses opacity $\sigma_p\in\mathbb{R}$ and view-independent spatial scene features $\mathbf{f_p}\in\mathbb{R}^{32}$ from its two branches respectively. To restrict the regressed color $\widetilde{\mathbf{c}}_\mathbf{p}$ to be non-negative, we apply a ReLU function after the last layer that directly outputs it.

In Figure B.1, we show the structure of the Refinement MLP. The spatial scene features $\mathbf{f_p}, \mathbf{f}^v_\mathbf{p}$ are extracted from the feature voxel $\mathbf{V_p}$ with a continuous coordinate $\mathbf{p} \in \mathbb{R}^3$ using tri-linear interpolation. Log differential opacity $\sigma_\mathbf{p} \in \mathbb{R}$ is regressed from the last fully-connected layer of the top branch and no non-linearity is applied. The spatial color value $\mathbf{c_p}$ is the output of the bottom branch and ReLU is applied afterwards to guarantee the regressed value is non-negative. At the beginning of the refinement network, a concatenation of a positional encoding of position $\mathbf{p}$ and its corresponding view-independent spatial scene feature $\mathbf{f_p}$. The color branch network learns to explain view-dependent effects by having additional inputs in addition to the positional encoding, such as the camera view $\mathbf{v}$ and view-dependent spatial scene feature $\mathbf{f}^v_\mathbf{p}$ at position $\mathbf{p}$. Note that the adapted version of NeRF, which we us as a baseline, shares exactly the same architecture as shown in Figure B.1, except that instead of $\mathbf{f_p}, \mathbf{f}^v_\mathbf{p}$ it uses the global latent vector $\mathbf{z}$ as additional input.

	Encoder	
1	`Conv2d`(9, 32)	
2	`Conv2d`(32, 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`(256x4x2, 512)	
10	`Linear`(512, 256)	`Linear`(512, 256)

Table 1. **Encoder architecture**. Each `Conv2d` layer in the encoder has a kernel size of 4, stride of 2 and padding of 1. After each layer, except for the last two parallel fully-connected layers, a Leaky ReLU [2] activation with a negative slope of 0.2 is applied. The last two parallel fully-connected layers produce, respectively, $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$.

B.2. Hyperparameter Settings

We use Adam [1] with a learning rate $1\mathrm{e}{-4}$, and $\beta_1 = 0.9, \beta_2 = 0.999$. All the models are trained for approximately $70 - 100$K iterations, each batch containing 64×64 rays. For each ray, we then uniformly sample 128 query locations for the coarse level, and 32 more locations for

[1] https://ziyanw1.github.io/hybrid_nerf/

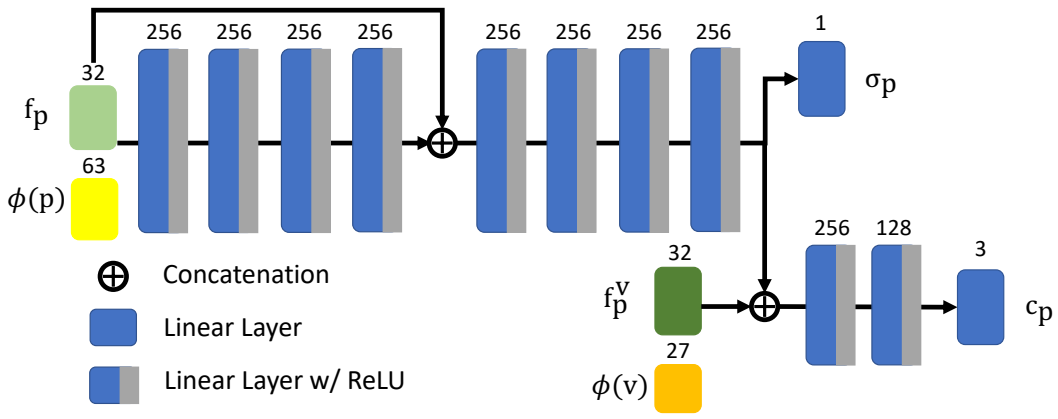

Figure 1. **Refinement MLP architecture**. Each blue box is a fully-connected layer and the number on top of each box is the output size of that layer. Blue box with gray tail is a linear layer with ReLU activation. Boxes in other color stand for different inputs and the size is marked on top.

	Decoder	
1	`Linear`(N_{in}^X, 1024)	
2	`Reshape`(1024, 1, 1, 1)	
3	`ConvTrans3d`(1024, 512)	`ConvTrans3d`(1024, 512)
4	`ConvTrans3d`(512, 512)	`ConvTrans3d`(512, 512)
5	`ConvTrans3d`(512, 256)	`ConvTrans3d`(512, 256)
6	`ConvTrans3d`(256, 256)	`ConvTrans3d`(256, 256)
7	`ConvTrans3d`(256, 128)	`ConvTrans3d`(256, 128)
8	`ConvTrans3d`(128, N_{out}^X)	`ConvTrans3d`(128, 32)

Table 2. **Decoder architecture**. Each layer is followed by a Leaky ReLU [2] activation with a negative slope of 0.2 except for the last two parallel layers. Each `ConvTrans3d` layer has a kernel size of 4, a stride of 2 and a padding of 1. N_{in}^X stands for the input feature size and N_{out}^X is the output size. X here is a placeholder for color or opacity, $X \in \{c, \sigma\}$.

the fine level using our sampling scheme. We set $\lambda_f = 0.1, \lambda_c = 0.1$ and $\lambda_{KL} = 0.001$ Training on a sequence of 360 frames under 93 camera views with 1024×667 resolution takes approximately 3-4 days on a single NVidia-V100-32GB GPU. All our models are implemented in PyTorch.

C. Novel View Synthesis

C.1. Qualitative Results

We show more qualitative results in a larger size than in the main document in Figure 2 and Figure 3. Please also refer to `videos/rot_zoom` for more video results.

D. Animation

D.1. Latent Space Sampling

We show more results of expression sampling in Figure 4 and Figure 5. Please see `videos/sample_interp_latent/interp_X.mp4` and `videos/sample_interp_latent/sample_X.mp4` for video results of sampling in latent space and interpolation. `X` here could be *subj1* or *subj2*. The first video contains 12 uniform keyframe expressions that are directly sampled from the latent space. Then, between each keyframe, we linearly interpolate 10 more frames to create the video. The second videos contains free view rendering of several sampled expressions.

D.2. Landmark-driven Animation

We used a PointNet [3]-like encoder as a base architecture for the keypoint encoder. Compared to the original work, our inputs are different in three aspects: 1) The points are in 2D, 2) The order of each point is fixed rather than arbitrary, 3) All points are roughly aligned to a canonical pose. To simplify the problem, we use the T-Net in the PointNet as the encoder that regresses the latent code from a set of points. We show the architecture in Table 3. More results of keypoint-driven animation can be found in Figure 6. Please refer to `videos/kps_render/subj1.mp4` more video results. In the video, the 2d keypoints in the blue bounding box are used as input to the keypoint encoder. The image in the middle is the output of our model and the image on the right most column is the ground truth. As we can see, the decoder in our method can also be driven by inputs from other modalities.

D.3. Fitting New Sequences

Please see `videos/sequence_fitting/X_noft.mp4` for rendering results of the model without finetuning and `videos/sequence_fitting/X_enc.mp4` for rendering results of the model with encoder-only finetuning. `X` could be either *subj1* or *subj2*.

	Kps Encoder
1	`Conv1d`(2, 64)
2	`Conv1d`(64, 128)
3	`Conv1d`(128, 256)
4	`Conv1d`(256, 512)
5	`Conv1d`(512, 1024)
6	`MaxPool1d`()
7	`Flatten`()
8	`Linear`(1024, 512)
9	`Linear`(512, 512)
10	`Linear`(512, 256)

Table 3. **Keypoint Encoder architecture**. Each layer is followed by a ReLU except for the last fully-connected layer. Each `Conv1d` layer has a kernel size of 1, a stride of 1 and a padding of 0.

In both videos, the images in the red bounding boxes serve as inputs. The image in the middle is the output of our model and the image on the right most column is the ground truth. As we can see, the model without finetuning can achieve reasonable performance on fitting the new sequence. And with only encoder finetuning, the encoder quickly adapts to the latent space of the decoder on the novel sequence and creates much smoother results. For free view rendering results of both models, please see `videos/sequence_fitting/X_enc_fr.mp4` and `videos/sequence_fitting/X_noft_fr.mp4`

References

[1] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

[3] 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.

Figure 2. **Qualitative comparison of rendered images**.

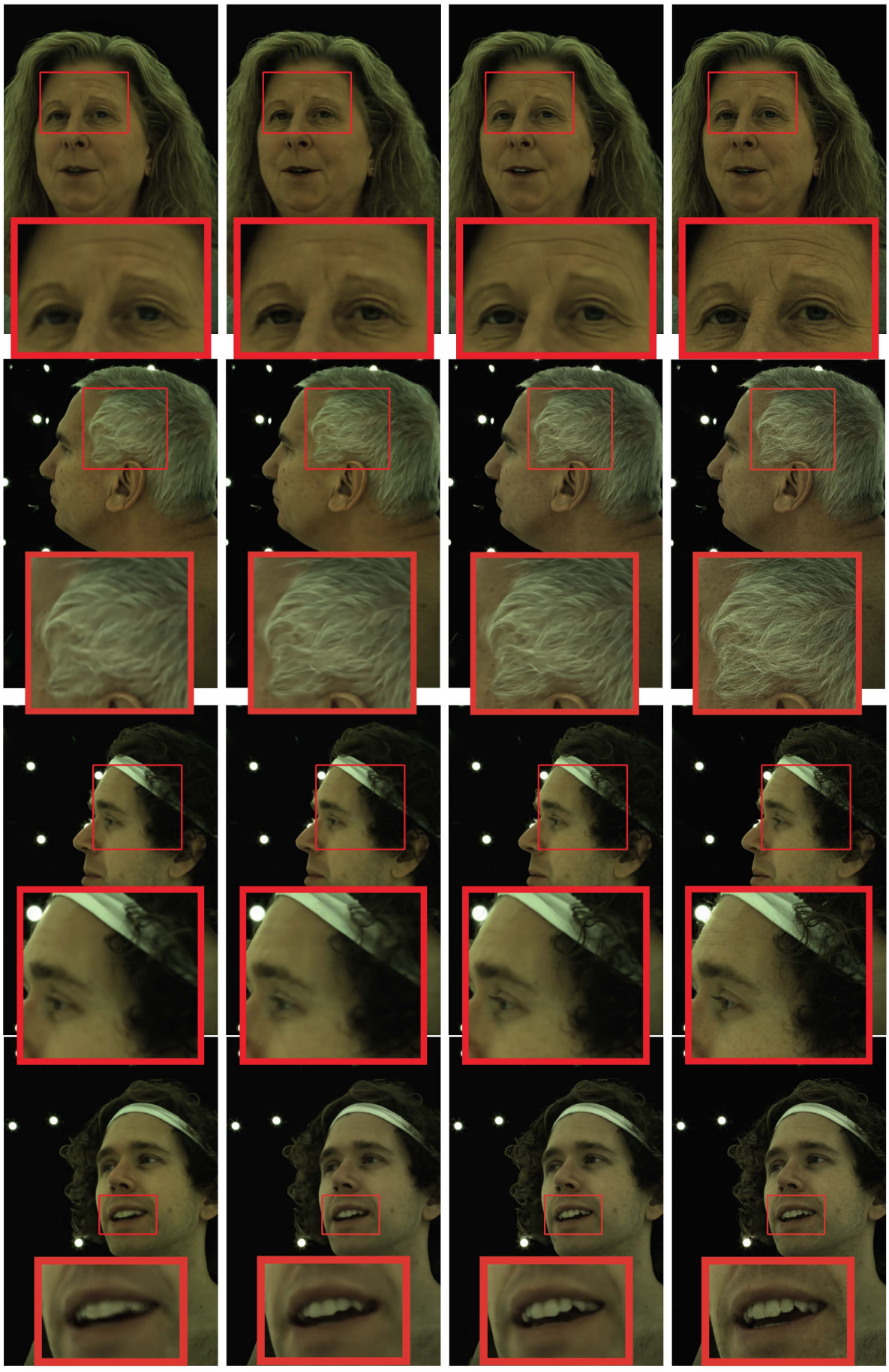

Figure 3. **Qualitative comparison of rendered images**.

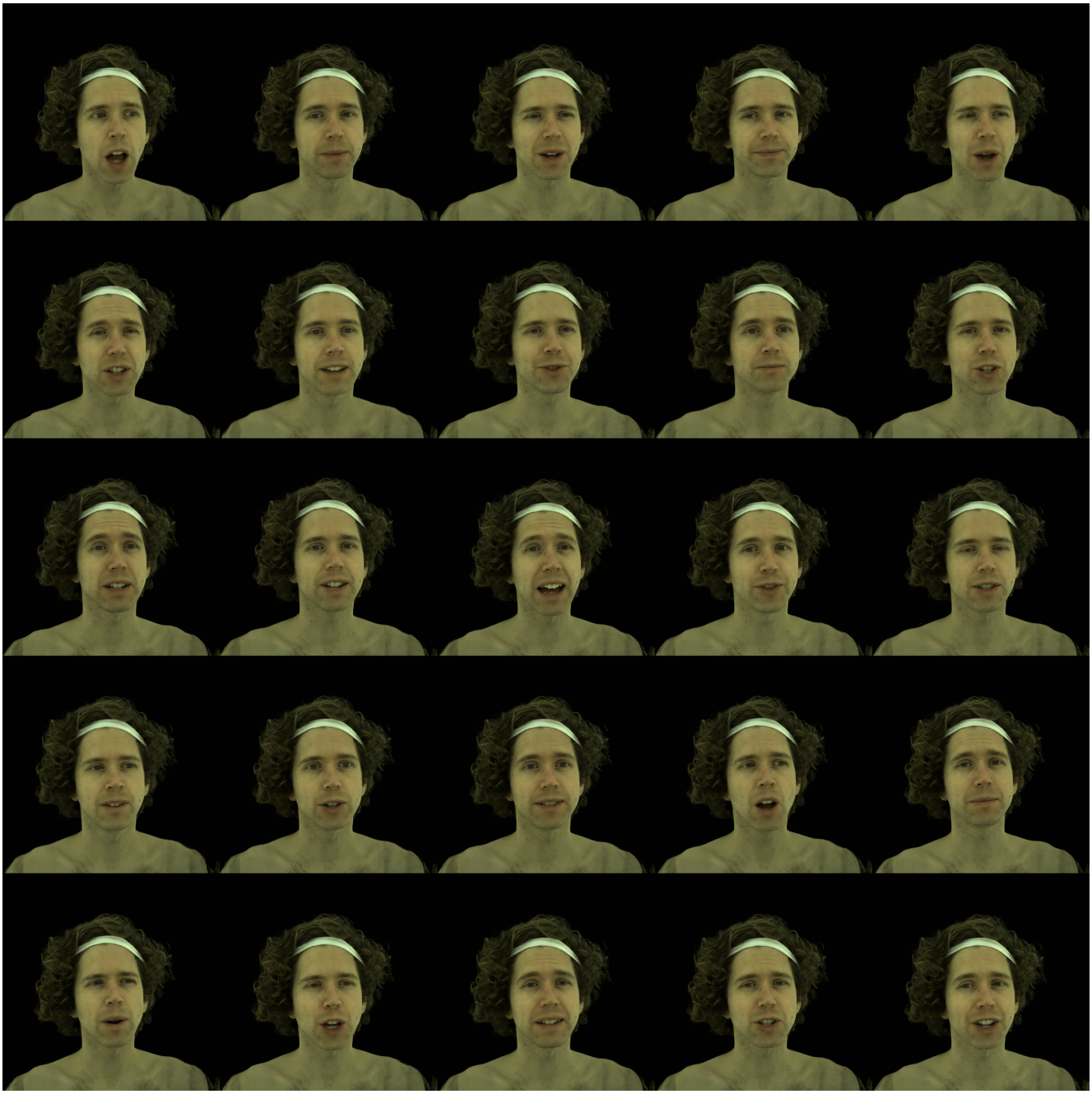

Figure 4. **Rendering results of direct sampling in latent space**.

Figure 5. **Rendering results of direct sampling in latent space**.

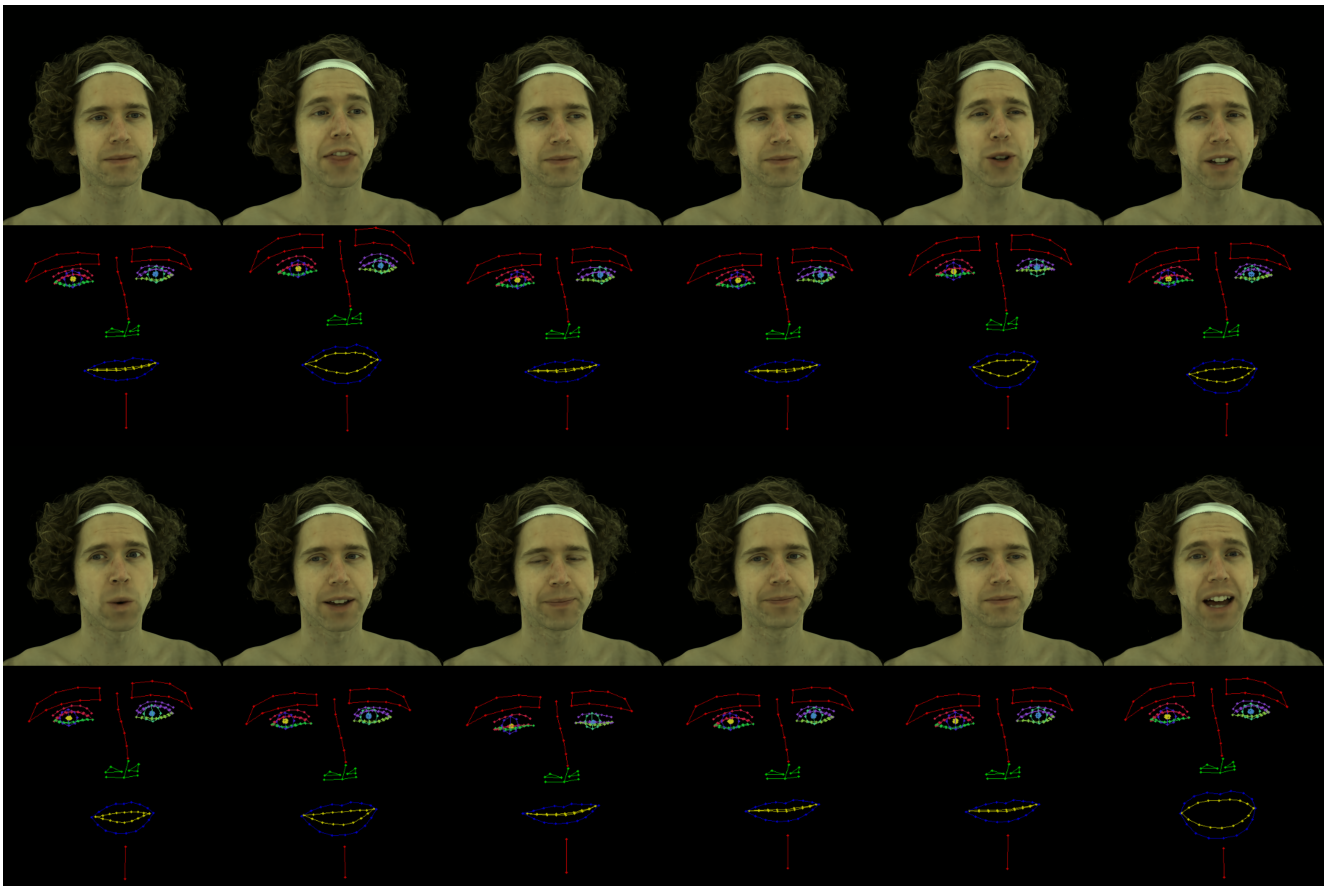

Figure 6. **Keypoint-driven animation**.