Supplemental:
Scaling Neural Face Synthesis to High FPS and Low Latency by Neural Caching

Frank Yu
Sid Fels
University of British Columbia (UBC)
Helge Rhodin
frankyu@cs.ubc.ca
ssfels@ece.ubc.ca
rhodin@cs.ubc.ca

This document supports the main paper, including how processing is parallelized over multiple GPUs, limitations of the used head reconstruction algorithm, and additional ablation studies.

1. Implementation details

Meshes are generated using the forward function of the FLAME model implemented in python [3]. UV maps are generated by rendering the FLAME head model using the PyTorch3D rasterizer, which implements rasterization on the GPU sing custom cuda kernels but is not as fast as classical rasterization. The explicit warp is implemented using the pytorch *grid_sample* function, which is the same as used by [1] for their explicit warp.

Generating Novel Views Our method relies on both the processed UV maps, initialized with the 3D mesh estimated by DECA [2]. Therefore, to generate novel views for our method we require to modify these parameters. In our tests, we found that the best generalization to novel views is obtained by rotating the vertices outputed by the FLAME model, as modifying the joint angles in FLAME model to extreme poses may lead to parameter ranges not seen during training.

Multithreading Library Our multi-gpu operation modes are implemented using the PyTorch multithreading library and two threads. For synchronization and communication between threads we use the *torch.multiprocessing.mp.Queue* ques: one input queue for each thread (two for two GPUs), one image output que, and one que to signal completion of caching. Listing 1 provides pseudocode for how caching and warping threads are orchestrated from the main thread, making sure that the output images arrive in sequence and that the caching completes before warping on the same worker while also ensuring that the parallel workers are independent otherwise for maximal performance. For simplicity, the pseudocode is simplified for 2x GPU and 2x warping for every caching operation. A different number of GPUs and warp operations only requires to change the condition in line 4 and line 8 that distribute caching and warping operations over worker threads. We will publish our multithreading code so others can build upon it with their own neural models.

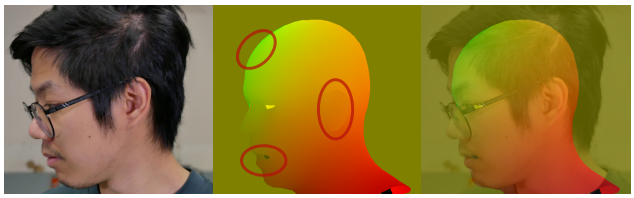

Figure 1: **Limitations of Head Reconstruction** We can see that at extreme angles and glasses present a problem to the capture algorithm and produces inaccurate UV maps and parameters.

Algorithm Time Complexity Using the code provided by pytorch-OpCounter, we measured the FLOPs for both the generator and warping network which are 364G and 87G FLOPs respectively. This factor of 4x corresponds well with the 3.2x latency improvement that we reported. Note that our approach decouples the complexity of the generator and warp networks, i.e., the warp keeps the same time complexity when coupled with deeper generators (c.f. experiment with ResNet in ablation study).

2. Limitations of Capture Algorithm

Due to limitations with the DECA approach [2], extreme head poses and cases when the subject is wearing glasses are not accurately reconstructed. Figure 1 shows such a case with glasses and a perpendicular angle to the camera, leading to a misalignment of model and image around the mouth, ears, and forehead.

Algorithm 1: Parallel Implementation (2x GPUs, 2x warp)

```
/* Main thread                                                              */
Queue viewpoint_queue[NGPU=2], cache_queue, out_queue   // Queues used for synchronization
cache_tid ← 0                          // Pointer to assign worker responsible for caching
for t=0; t++; do
    Input: viewpoint, expr
    if t % NUM_WARPS == 0 then
        cache_queue.get()                                  // Wait for caching to complete
        cache_tid ← (cache_tid+1) % NGPU                       // Switch role of threads
        // Post viewpoint to worker and assign caching role
        (viewpoint, expr, warp=False) → viewpoint_queue[cache_tid]
    // Post viewpoint to the other worker and assign warping role
    (viewpoint, expr, warp=True) → viewpoint_queue[(cache_tid+1) % NGPU]
    // Display output image once available
    Output: out_queue.get()
/* Worker thread, one for each GPU                                          */
do in parallel
    NeuralCache cache                                  // variable local to each thread
    (viewpoint, expression, warp) ← viewpoint_queue.get()   // Wait for new input to process
    if warp = True then
        image ← runWarpNet(viewpoint, expression, cache)              // Generate the image
        image → out_queue                                  // Return output image via queue
    else
        mesh ← runFaceModel(viewpoint, expression)      // Face mesh from expr. parameters
        cache ← runDecoder(viewpoint, expression, mesh)        // Cache neural representation
        "caching done" → cache_queue                  // Notify cache completion via queue
```

C3	C4	C5	L1 ↓	PSNR ↑	SSIM ↑
✓			0.0261	26.13	0.9048
✓	✓		0.0247	26.48	0.9096
✓	✓	✓	**0.0246**	**26.54**	**0.9102**

Table 1: **Ablating caching layers** for our warping network using the sequential variant (1x warp), showing the improvement after adding each component on the *beard* dataset.

3. Additional Ablation on Caching Layers

In this additional ablation, we sequentially remove the caching layers (C3, C4, C5) and measure the reconstruction quality in terms of L1 loss, PSNR, and SSIM. Each configuration was trained on the *beard* dataset for 50 epochs, and metrics were computed on the held-out test set. Table 1 shows that adding the C4 cache layer is the most impactful, but since performance increases after adding all layers, we include all cache layers in our final configuration.

4. Additional Ablation on the Warp Distance

In the experiments in the main document, we present the results of our warping network by warping either 1 or 2 frames ahead. This ablation study measures the performance drop from warping more frames ahead. Figure 2 shows that the reconstruction quality decreases linearly with the warp distance on the *beard* dataset, which was recorded at 30fps. Please note that this increase of error is expected as the proposed neural caching and warping is a local approximation designed for high-fps videos where the warp distance is small.

References

[1] Xiangxiang Chu, Bo Zhang, Hailong Ma, Ruijun Xu, and Qingyuan Li. Fast, accurate and lightweight super-resolution with neural architecture search. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 59–64. IEEE, 2021.

[2] Yao Feng, Haiwen Feng, Michael J. Black, and Timo Bolkart. Learning an animatable detailed 3D face model from in-the-wild images. volume 40, 2021.

[3] Tianye Li, Timo Bolkart, Michael J Black, Hao Li, and Javier Romero. Learning a model of facial shape and expression

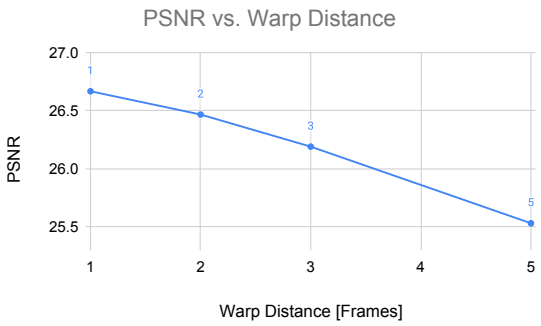

Figure 2: **Ablation on warp distance** as we increase the warp distance, or the number of warping operations per caching operation, the reconstruction quality decreases linearly.

from 4d scans. *ACM Trans. Graph.*, 36(6):194–1, 2017.