3D Prior is All You Need: Cross-Task Few-shot 2D Gaze Estimation

Supplementary Material

7. Summary of Evaluation Datasets

Our work conducts experiments on three datasets: MPIIGaze, which is also referred to as MPIIFaceGaze, EVE, and GazeCapture. We preprocess these datasets for evaluation. The dataset statistics are summarized in Table 6. Notably, GazeCapture includes over 1,000 subjects, making it impractical to evaluate all subjects. Therefore, we sort all subjects based on their identifiers, e.g., sub_00001, and select the 20 subjects in ascending order of their identifiers. We exclude subjects with fewer than 500 images to ensure a convincing evaluation.

Overall, under the experimental settings, our method was evaluated on 74 subjects across three different platforms, demonstrating its advantages and robustness.

Table 6. Dataset statistics on our experiment

	Devices	# Subjects	# Images per subject
EVE	Desktop computer	39	$\sim$ 1800
MPIIGaze	Laptop	15	1500
GazeCapture	Phone & tablet	20	$\sim$ 1200

Besides, we perform image normalization during data preprocessing. The original method requires camera intrinsic parameters for normalization, which conflicts with one of our motivations: enabling quick adaptation for non-expert users. In our method, this issue is resolved by using estimated camera intrinsic parameters. For the GazeCapture dataset, we apply estimated parameters for normalization. We do not provide detailed explanations of this in our manuscript, as it is not a central focus of our work and can be addressed effectively.

8. Implementation Details

Our work is primarily implemented using two libraries: PyTorch and PyTorch3D. Most of our modules are developed using PyTorch, while the Rodrigues transformation is implemented using PyTorch3D. The Rodrigues transformation ensures that $\mathbf{R} \in SO(3)$, facilitating the computation of the inverse matrix for coordinate transformations. We initialize the rotation matrix $\mathbf{R}$ as diag(-1, 1, -1) and the translation vector $\mathbf{t}$ as (0, 0, 0), where the value of $\mathbf{r}$ could be computed using Rodrigues formula. This represents a basic transformation between the camera coordinate system and the screen coordinate system, i.e., we assume that the origins of the two systems overlap, and the x-y planes of the two systems are parallel.

This is also why we claim that the initial screen pose happened to be same as the actual screen pose in the GazeCapture dataset. The dataset collects images using mobile devices, where the x-y planes of the embedded camera are typically parallel to the screen. More importantly, the authors of GazeCapture precisely measure the camera placement and screen dimensions to define a unified prediction space, setting the origin of the defined screen coordinate system at the camera position. However, this setup is atypical, as manually measuring the camera placement is equivalent to manually calibrate screen pose.

9. Implementation on the Real-World Device

We also evaluate our method in a real-world environment. Specifically, we implement our method at a desktop computer and invite a volunteer for testing. Our implementation is conducted on the machine equipped with an NVIDIA RTX 3090 GPU in a Python environment. We use a 1080p webcam to capture the face images and applied our method to estimate the user's 2D gaze on the screen. The experimental setup is illustrated in Figure 8, with the user positioned approximately 90 cm from the camera.

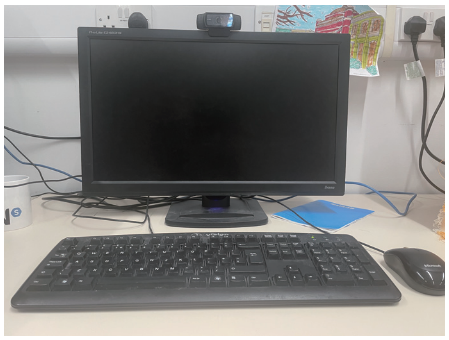

Figure 8. Setup for the implementation on the real-world device.

The process involves the following steps:

1. Calibration: The volunteer is required to look at four calibration points on the screen. For each point, we collect 40 face images for model adaptation while the volunteer focuses on each point for 1-2 seconds. We collect 40 images from each point to minimize the impact of noise data such as blink.
2. Data Pre-Processing: From these images, we detect human face landmarks and estimate the 3D head pose to compute the 3D face centers. Image normalization is then applied to obtain the processed face images.
3. Model Adaptation: Using our method, we adapt the 3D gaze estimation model for the real-world 2D gaze estimation.

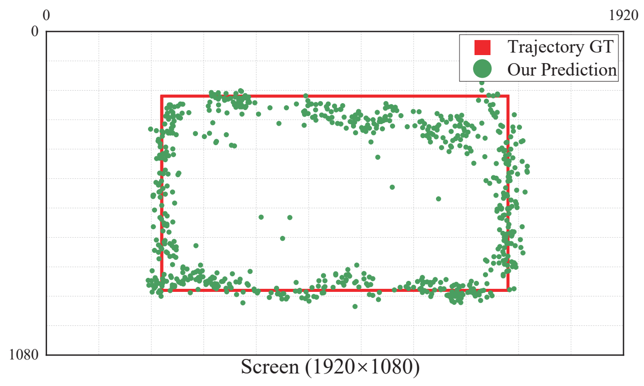

Figure 9. The visualization of our estimation in a real-world implementation. The red line represents the trajectory of a moving dot that we pursued, while the blue dots indicate our gaze predictions. This result demonstrates the effectiveness of our method. The jitter is due to various environmental and personal factors, such as eye blinking and unstable face detection. These issues can be easily addressed using post-processing methods.

4. Evaluation: The volunteer is instructed to continuously focus on a moving dot. Our method estimates the gaze trajectory from face images.

We visualize the results in Figure 9, where the red line represents the trajectory of the moving dot, and the green dots indicate the gaze estimation results. It is important to note that we do not pre-calibrate the camera intrinsic matrix or the screen pose. Besides, we do not apply post-processing methods, such as blink detection or filtering, to the estimation results. A video of the entire process is provided as supplementary material.