1. More Experiments

Here, we provide more experiments and analyses. Results are reported for 3DPW.

**Visualization of the distribution denoising.** In Fig. 1, we visualize the step-by-step denoising of the distributions of our method with DAT and without DAT. Using DAT, we can quickly reach the input-specific manifold, where the distribution resembles the target distribution, which shows our method can converge fast with DAT.

![Visualization for DAT](image)

Figure 1. Visualization for DAT. For simplicity, we only show the distribution of a part of the mesh vertices.

**Evaluation on occlusion benchmark.** To further evaluate the effectiveness of our method in reducing uncertainty, we conduct additional experiments following [14, 6]. Specifically, we evaluate our method on 3DPW-PC [6], which is a subset of the 3DPW dataset and contains many occluded samples with high levels of uncertainty. As shown in Tab. 1, our method significantly outperforms existing methods and achieves state-of-the-art results, which shows the effectiveness of our method at handling uncertainty.

![Evaluation of hyperparameters K and N.](image)

Table 1. Comparison on occlusion setting.

<table>
<thead>
<tr>
<th>Method</th>
<th>MPVE</th>
<th>MPJPE</th>
<th>PA-MPJPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>[14]</td>
<td>152.8</td>
<td>119.7</td>
<td>79.7</td>
</tr>
<tr>
<td>[6]</td>
<td>149.6</td>
<td>117.5</td>
<td>77.1</td>
</tr>
<tr>
<td>Ours</td>
<td>143.1</td>
<td>114.2</td>
<td>73.5</td>
</tr>
</tbody>
</table>

**Impact of the total number of diffusion steps K and number of samples N.** To further investigate our HMDiff, we also conduct additional ablation experiments that vary the total number of diffusion steps $K$ and number of samples $N$. From Fig. 2, we can observe that the quality of the reconstructed human mesh tends to improve when $K$ is increased, and shows only minor improvements when $K$ is above 200. Besides, we also find that the output mesh quality improves when we increase the number of samples $N$ till $N = 25$, and stays roughly consistent thereafter. Therefore, we set $K = 200$ and $N = 25$.

![More visualization results.](image)

**More visualization results.** In Fig. 2 of the main paper, we displayed some qualitative results of our method. Here, we visualize more examples in Fig. 3. These examples show that our method effectively recovers the human mesh even in scenarios with high uncertainty, such as occlusions and background noise.

**Impact of acceleration technique.** In our work, we accelerate the reverse diffusion process via the acceleration technique from DDIM [13], which uniformly skips diffusion steps. For more implementation details, refer to Sec. 3.2 of Supplementary. Our experiment results are presented in Tab. 2. As shown in Tab. 2, when we adopt the acceleration technique, the efficiency improves (i.e., FPS increases significantly), while the performance is similar.

![Evaluation of acceleration technique.](image)

Table 2. Evaluation of acceleration technique.

<table>
<thead>
<tr>
<th>Method</th>
<th>MPVE</th>
<th>MPJPE</th>
<th>PA-MPJPE</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours w/o acc</td>
<td>82.1</td>
<td>72.3</td>
<td>44.1</td>
<td>5</td>
</tr>
<tr>
<td>Ours w/ acc</td>
<td>82.4</td>
<td>72.7</td>
<td>44.5</td>
<td>18</td>
</tr>
</tbody>
</table>

† Corresponding author
Impact of DAT threshold $r$. In Fig. 4, we evaluate the impact of different settings of the DAT threshold $r$. By decreasing $r$ starting from $r = 1$ (which is equivalent to not using DAT), we observe that performance steadily improves to a peak at around $r = 0.05$, and then slightly drops. This is because, when $r$ is high, decreasing $r$ will lead to the distribution $H_k$ getting aligned closer to $U$ and $H_0$ (before DAT is deactivated), thus it will benefit performance. However, when $r$ is already small (e.g., $r = 0.05$), decreasing $r$ further will cause $H_k$ to align towards $U$ even at the later steps (when $H_k$ is already relatively accurate and certain). Since the prior distribution $U$ contains uncertainty and noise, this alignment between $H_k$ and $U$ at the later steps will not benefit the performance. Therefore, we fix the threshold $r$ at 0.05.

2. Discussion on Theoretical Aspects

In this section, we discuss some of the theoretical aspects related to our work.

Rather than directly minimizing the gap between $h_k$ and $U$ by using the gradient $\nabla_{h_k} \log p(U|h_k)$, we instead compute the Distribution Alignment Gradient $\nabla_{h_k} \log p(U|h_0(h_k))$ by using $h_0$ as an intermediate step (as mentioned in Sec. 4.2 of the main paper). Specifically, we draw $N$ samples from $U$ (i.e., $u_i \sim U$) and take an update to minimize the sum of $L_2$ norms between $\{u_i\}_{i=1}^N$ and $f(h_0(h_k))$. In other words, we represent $\nabla_{h_k} \log p(U|h_0(h_k))$ with $\sum_{i=1}^N \nabla_{h_k} \log p(u_i|h_0(h_k))$. We remark that the theoretical bounds of using $p(u|h_0(h_k))$ to approximate $p(u|h_k)$ have been well-explored in [4].

Furthermore, following up upon the thermodynamics analogy used in our paper, more theoretical aspects regarding the thermodynamics analogy can be found in [12]. For instance, upper and lower bounds on the entropy of each reverse diffusion step have been derived in [12].
3. More Details

3.1. More network architecture details

In Sec. 4.3 of the main paper, we presented the main aspects of our network architecture. Here, we describe our network architecture in more detail.

**Image feature f_{i}.** Firstly, we go into more details regarding how we encode positional information for the image feature f_{i}, which is important for the transformer-based diffusion model g. Recall that we first extract a context feature f_{c} ∈ R^{2048×7×7} from our CNN backbone φ_{c}. Then, we perform average pooling on f_{c} and flatten it, to obtain a feature of shape 128 × 49. To encode positional information, we generate a position encoding map E_{s} ∈ R^{128×49} – which is generated by the sinusoidal function – to add to the feature. Specifically, for the position encoding map E_{s}, at each index i ∈ [1, ..., 49], E_{s}[i] is a vector of length 128. At each even (2j) index of E_{s}[i], we set the element E_{s}[i, 2j] to sin(i/49^{2j}/128), while at each odd (2j + 1) index, we set the element E_{s}[i, 2j + 1] to cos(i/49^{2j}/128). After the position encoding map E_{s} is added, we obtain the image feature f_{i} ∈ R^{128×49}, which contains rich semantic features (and positional information) and is fed to the diffusion model g at every step.

**Pose estimator head φ_{P}.** Our 3D pose estimator head φ_{P} (along with our CNN backbone φ_{c}) are obtained off-the-shelf from [15], where they have been pre-trained on Human3.6M [5], UP-3D [7], MuCo-3DHP [11], COCO [10] and MPII [1]. Overall, we utilize φ_{P} to generate a xy heatmap E_{x,y} ∈ R^{J×56×56} and a depth heatmap E_{z} ∈ R^{J×56×56}, where J is the number of joints. We follow previous works [2, 9, 8] to set J = 14. The pose estimator head φ_{P} is a lightweight module consisting of three de-convolutional layers, in which the input feature shape in each layer is 7 × 7 × 1024, 14 × 14 × 512, and 28 × 28 × 256 respectively.

Next, we present how to obtain a 3D human pose distribution U from φ_{P} to guide the reverse diffusion process. We note that φ_{P} allows us to generate an xy heatmap E_{x,y} ∈ R^{J×56×56} and a depth heatmap E_{z} ∈ R^{J×56×56}, where J is the number of joints. Then, we normalize the obtained heatmaps. These normalized heatmaps can naturally be regarded as a probability map that characterizes the distribution of predicted 3D human pose. In implementation, to efficiently calculate the distribution gap between U and the intermediate distribution H_{k}, we sample 25 3D human poses based on these heatmaps to approximate U.

**Diffusion step embedding E_{d,k}.** Moreover, to assist the diffusion model to learn the reverse diffusion process effectively, we also build a diffusion step embedding E_{d,k} ∈ R^{61} for each k-th diffusion step. Specifically, at each even (2j) index of E_{d,k}, we set the element E_{d,k}[2j] to sin(k/200^{2j}/61), while at each odd (2j + 1) index, we set the element E_{d,k}[2j + 1] to cos(k/200^{2j}/61).

**Diffusion model g.** Overall, as described in Sec. 4.3 of the main paper, our diffusion network g is transformer-based and consists of a single vertex self-attention layer, a single vertex-image cross-attention layer, and a linear layer. V vertex tokens \{x_{1}, x_{2}, ..., x_{V}\} are input to the network, where each token x_{v} ∈ R^{128} represents the v-th vertex of the sample h_{k} at step k. When the V vertex tokens \{x_{1}, x_{2}, ..., x_{V}\} are input to the network, the vertex-self-attention layer performs the self-attention operation with an adjacency matrix to generate the intermediate tokens \{b_{1}, b_{2}, ..., b_{V}\}. Then, the vertex-image cross-attention layer performs the cross-attention operation between \{b_{1}, b_{2}, ..., b_{V}\} and f_{i}. Specifically, f_{i} ∈ R^{128×49} is treated as 49 tokens of length 128, and this cross-attention mechanism takes \{b_{1}, b_{2}, ..., b_{V}\} as the query components and f_{i} tokens as the key and value components. Next, we take the output of the cross-attention layer (of shape V × 128), and feed it into a linear layer to predict h_{k−1} ∈ R^{V×3}. Then, we modify h_{k−1} via DAT (if it is activated).

**MLP.** Finally, after the K diffusion steps, we obtain the high-quality mesh distribution H_{0}, which is represented by N samples (h_{0}). We take the mean of H_{0} by averaging the N samples, and feed it into a 3-layer MLP to obtain the final prediction h_{m}. More specifically, the MLP takes in a coarse human mesh sample R^{491×3} and outputs a fine-grained one R^{1723×3}.

3.2. More implementation details

In the forward diffusion process, we set the number of total diffusion steps K at 200 and generate the decreasing sequence α_{1;K} via the formula:

\[ α_{k} = \prod_{i=1}^{k}(1 − β_{i}) \]  

where β_{1;K} is a sequence from 1e − 4 to 2e − 2, which is interpolated by the linear function. Moreover, we utilize the acceleration technique DDIM [13] to speed up our diffusion inference procedure. Specifically, we accelerate our diffusion process by tuning the value of σ_{k} in Eq. 6 in the main paper to skip a certain number of reverse diffusion steps. Following [13], for each k-th step, we generate the acceleration metric σ_{k} via the formula below:

\[ σ_{k} = η \cdot \sqrt{(1 − \frac{α_{k}}{α_{k+n_{i}}+1}) \cdot \frac{(1 − α_{k+n_{i}+1})}{1 − α_{k}}} \]  

where n_{i} is the number of skipped steps between the current and next diffusion step, and η is the hyperparameter that controls the variance of σ_{k}. In our implementation, we follow [13] to set η to 0.8 and n_{i} to 4.

Our method is implemented using PyTorch, and can be trained on a powerful workstation with four NVIDIA RTX A5000 GPU cards within 48 hours.
In our forward and reverse process, we add noise to the mesh vertex coordinates and denoise the mesh vertex coordinates respectively. In both of these processes, the topology of the mesh vertices stays fixed. In other words, to link the mesh vertices together to produce the mesh surfaces, we follow previous works [2, 9] to define an adjacency matrix between the vertices, and this adjacency matrix is kept fixed throughout the diffusion process.

3.3. More training details

**Learning Mesh Geometry.** To reconstruct an accurate and natural human mesh, we also optimize our diffusion model via geometric constraints of human mesh. As mentioned in Sec. 4.4 of our main paper, following previous work [3, 8, 9], we optimize our model by incorporating four kinds of losses to learn the mesh geometry: 3D Vertex Regression Loss \( \mathcal{L}_v \), 3D Joint Regression Loss \( \mathcal{L}_j \), Surface Normal Loss \( \mathcal{L}_n \), and Surface Edge Loss \( \mathcal{L}_e \).

Specifically, we obtain the estimate of diffusion target \( \hat{h}_0 \) at each diffusion step and then utilize these losses to optimize the diffusion model. Below we introduce more details regarding each loss.

3D Vertex Regression Loss \( \mathcal{L}_v \) is used to optimize our model to learn how to regress 3D mesh vertices. It is computed by:

\[
\mathcal{L}_v = \frac{1}{V} \| \hat{h}_0 - h_0 \|_1,
\]

where \( V \) is the number of vertices and \( h_0 \in \mathbb{R}^{V \times 3} \) denotes the ground-truth 3D vertex coordinates.

We can also regress 3D joints \( \hat{h}_0 \) from the estimated mesh \( h_0 \) via a linear mesh-to-pose function \( f \). Thus, we can also use a 3D Vertex Regression Loss \( \mathcal{L}_j \) to train our model for the regression of 3D body joints. Specifically, this loss \( \mathcal{L}_j \) applies an \( L_1 \) loss, and is computed by:

\[
\mathcal{L}_j = \frac{1}{J} \| \hat{h}_0 - h_0 \|_1,
\]

where \( J \) is the number of joints and \( h_0^j \) denotes the ground-truth 3D joint coordinates.

Moreover, we optimize our model to reconstruct a consistent mesh surface via a Surface Normal Loss \( \mathcal{L}_n \), which is defined as follows:

\[
\mathcal{L}_n = \sum_{t} \sum_{\{i,j\} \subset t} \left| \left\langle \frac{v_i - v_j}{\|v_i - v_j\|_2}, n^*_t \right\rangle \right|,
\]

where \( t \) and \( n^*_t \) denote a triangle face in the human mesh and the ground-truth unit normal vector of \( t \) respectively; \( \langle \cdot, \cdot \rangle \) denotes a dot product, and \( v_i \) denotes the \( i \)-th vertex in \( t \).

Finally, we utilize a Surface Edge Loss \( \mathcal{L}_e \) to smooth the surfaces with dense vertices, such as face, hand and feet, The loss \( \mathcal{L}_e \) is formulated as:

\[
\mathcal{L}_e = \sum_{t} \sum_{\{i,j\} \subset t} \| v_i - v_j \|_2 - \| \hat{v}_i^* - \hat{v}_j^* \|_2 ,
\]

where \( t \) and the asterisk (*) denote a triangle face in the human mesh and the ground-truth respectively, and \( v_i \) denotes the \( i \)-th vertex in \( t \).

In this paper, we follow previous work [3] to set the loss coefficient values as: \( \lambda_v = 0.1, \lambda_j = 1, \lambda_n = 0.0001 \) and \( \lambda_e = 0.005 \).

References


