Ouroboros3D: Image-to-3D Generation via 3D-aware Recursive Diffusion

Supplementary Material

1. Supplementary

1.1. Video Model Fine-tuning

Based on the approach outlined in SVD, the generation process employs the EDM framework. Let $p_{\text{data}}(\mathbf{x}_0)$ represent the video data distribution, and $p(\mathbf{x};\sigma)$ be the distribution obtained by adding Gaussian noise with variance σ^2 to the data. For sufficiently large σ_{max}, $p(x;\sigma^2_{\text{max}})$ approximates a normal distribution $\mathcal{N}(0,\sigma^2_{\text{max}})$. Diffusion models (DMs) leverage this property and begin with high variance Gaussian noise, $x_M \sim \mathcal{N}(0,\sigma^2_{\text{max}})$, and then iteratively denoise the data until reaching $\sigma_0 = 0$.

In practice, this iterative refinement process can be implemented through the numerical simulation of the Probability Flow ordinary differential equation (ODE):

$$d\mathbf{x} = -\dot{\sigma}(t)\sigma(t)\nabla_{\mathbf{x}} \log p(\mathbf{x};\sigma(t))\, dt \tag{1}$$

where $\nabla_{\mathbf{x}} \log p((\mathbf{x};\sigma)$ is called as score function.

DM training is to learn a model $s_\theta(\mathbf{x};\sigma)$ to approximate the score function $\nabla_{\mathbf{x}} \log p((\mathbf{x};\sigma)$. The model can be parameterized as:

$$\nabla_{\mathbf{x}} \log p((\mathbf{x};\sigma) \approx s_\theta((\mathbf{x};\sigma) = \frac{D_\theta(\mathbf{x};\sigma) - \mathbf{x}}{\sigma^2}, \tag{2}$$

where D_θ is a learnable denoiser that aims to predict ground truth $\mathbf{x}_0$.

The denoiser D_θ is trained via denoising score matching (DSM):

$$\mathbb{E}_{\mathbf{x}_0 \sim p_{\text{data}}(\mathbf{x}_0),(\sigma,n)\sim p(\sigma,n)} \left[\lambda_\sigma \| D_\theta(\mathbf{x}_0 + n;\sigma) - \mathbf{x}_0 \|_2^2\right], \tag{3}$$

where $p(\sigma, n) = p(\sigma)\mathcal{N}(n;0,\sigma^2)$, $p(\sigma)$ is a distribution over noise levels σ, λ_σ is a weighting function. The learnable denoiser D_θ is parameterized as:

$$D_\theta(\mathbf{x};\sigma) = c_{\text{skip}}(\sigma)\mathbf{x} + c_{\text{out}}(\sigma) F_\theta(c_{\text{in}}(\sigma)\mathbf{x}; c_{\text{noise}}(\sigma)), \tag{4}$$

where F_θ is the network to be trained.

We sample $\log\sigma \sim \mathcal{N}(P_{\text{mean}}, P^2_{\text{std}})$, with $P_{\text{mean}} = 1.0$ and $P_{\text{std}} = 1.6$. Then we obtain all the parameters as follows:

$$c_{\text{in}} = \frac{1}{\sqrt{\sigma^2+1}} \tag{5}$$

$$c_{\text{out}} = \frac{-\sigma}{\sqrt{\sigma^2+1}} \tag{6}$$

$$c_{\text{skip}}(\sigma) = \frac{1}{\sigma^2+1} \tag{7}$$

$$c_{\text{noise}}(\sigma) = 0.25 \log\sigma \tag{8}$$

$$\lambda(\sigma) = \frac{1+\sigma^2}{\sigma^2} \tag{9}$$

We fine-tune the network backbone F_θ on multi-view images of size 512×512. During training, for each instance in the dataset, we uniformly sample 8 views and choose the first view as the input view. view images of size 512×512.

1.2. Canonical Coordinates Map

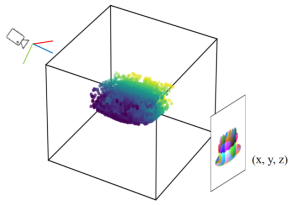

Figure 1. The projection process of coordinates map.

For control networks of image diffusion models, the conditional maps like depth maps need to be normalized to [0, 1], typically using the formula: $(p - p_{mean})/(p_{max} - p_{min})$. For multi-view generation, each view performs a normalize operation on itself, which results in a scale ambiguity. At the same time, the depth map is relative to a certain view, and the correlation between the depth values is not significant across views.

To avoid the above issues caused by self-normalization, we use canonical coordinate maps (CCM). Coordinate maps transform the depth value d to a common world coordinate system using the camera's intrinsic and extrinsic parameters, represented as (X, Y, Z). The transformation formula is:

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = K^{-1} \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \cdot d$$

where (u, v) are the pixel coordinates, d is the corresponding depth value, and K is the camera intrinsic matrix. Then the coordinate values of all views will be multiplied by a **global** scale and added an offset value to convert to the range of 0 to 1. This representation makes the correlation between different views more significant and is helpful for multi-view generation.

1.3. Algorithm

Algorithm 1 Training

Input: x, cond_image, cameras, timestep
Output: loss
// Returns the loss on a training example x. Details about EDM are omitted here.
begin
  noise ← Sample from Normal Distribution
  noisy_x ← Add_Noise(x, noise, timestep)
  pred_x ← F(noisy_x, cond_image, timestep, cameras)
  pred_i ← VAE_Decoder(pred_x)
  self_cond ← $\mathcal{G}$(pred_i, cameras, timestep)
  if *Random_Uniform(0, 1) > 0.5* **then**
    pred_x ← F(noisy_x, cond_image, timestep, cameras, self_cond)
  end
  loss_mv ← MSE_Loss(pred_x, x)
  loss_recon ← MSE_Loss(self_cond, x) + LPIPS_Loss(self_cond, x)
  loss ← loss_mv + loss_recon
  return *loss*
end

Algorithm 2 Inference

Input: cond_image, cameras, timesteps
Output: images, 3d_model
// Generate multi-view images and 3D model from a condition image.
begin
  self_cond ← None
  x_t ← Sample from Normal Distribution
  foreach *timestep in timesteps* **do**
    pred_x ← F(x_t, cond_image, timestep, cameras, self_cond)
    pred_i ← VAE_Decoder(pred_x)
    self_cond ← $\mathcal{G}$(pred_i, cameras, timestep)
  end
  return *pred_i, self_cond*
end

1.4. 3D-aware Feedback

Fig. 2 and Tab. 1 provide a detailed illustration of the feedback injection netwrok. We use two networks to inject the coordinates map and RGB texture map feedback into the score function. Each network consists of four feature extraction blocks and three downsample blocks to adjust the feature resolution. The reconstruction coordinates map and RGB texture map initially have a resolution of 512×512. We employ the pixel unshuffle operation to downsample these maps to 64×64.

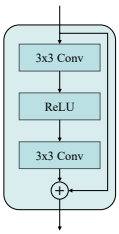

Figure 2. Architecture of the residual block used in the feedback stage.

Table 1. The detailed structure of all layers in the feedback injection network.

Input	inp $\in \mathbb{R}^{3\times512\times512}$
PixelUnshuffle	$192 \times 64 \times 64$
ResBlock $\times 3$	$320 \times 64 \times 64$
ResBlock $\times 3$	$640 \times 32 \times 32$
ResBlock $\times 3$	$1280 \times 16 \times 16$
ResBlock $\times 3$	$1280 \times 8 \times 8$

At each scale, three residual blocks are used to extract the multi-scale feedback features, denoted as $F_P = \{F_p^1, F_p^2, F_p^3, F_p^4\}$ and $F_T = \{F_t^1, F_t^2, F_t^3, F_t^4\}$ for the coordinates map and RGB texture map, respectively. These feedback features match the intermediate features $F_{\text{enc}} = \{F_{\text{enc}}^1, F_{\text{enc}}^2, F_{\text{enc}}^3, F_{\text{enc}}^4\}$ in the encoder of the UNet denoiser. The feedback features F_P and F_T are added to the intermediate features F_{enc} at each scale as described by the following equations:

$$\mathbf{F}_p = \mathcal{F}^0(P) \tag{10}$$

$$\mathbf{F}_t = \mathcal{F}^1(T) \tag{11}$$

$$\mathbf{F}_{\text{enc}}^i = \mathbf{F}_{\text{enc}}^i + \mathbf{F}_p^i + \mathbf{F}_t^i, \quad i \in \{1, 2, 3, 4\} \tag{12}$$

where P represents the coordinates map feedback input, and T represents the RGB texture feedback input. $\mathcal{F}^0$ and $\mathcal{F}^1$ denote the functions of the feedback inject network applied to the coordinates map and RGB texture map, respectively.

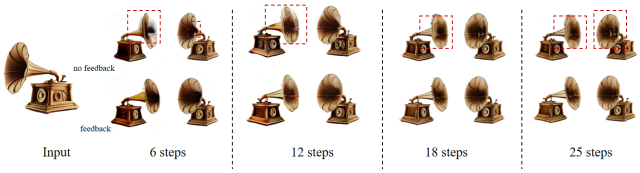

Figure 3. Visualization of the reconstruction results at different denoising steps. The process initially generates floaters and distorted geometries, but progressively refines them into cleaner representations. Through feedback mechanisms, the model optimizes shape and texture features in the early stages

1.5. visualization of denoising steps

We visualize the reconstruction process at different denoising steps in Fig. 3. Early stages show floating artifacts and distorted geometries due to multi-view inconsistency. As denoising progresses, our recursive diffusion method gradually refines both the geometric accuracy and material properties

of the reconstruction. By comparing the visual results, we observed that the 3D feedback mechanism achieved superior performance compared to the no-feedback condition.