CADCrafter: Generating Computer-Aided Design Models from Unconstrained Images

Supplementary Material

6. Supplementary Materials

We have prepared supplementary materials. The technical details of our implementation are discussed in Sec. 7 and Sec. 8. Moreover, we present additional examples and comparisons in Sec. 9 to demonstrate the performance of our method.

7. Technical Details

7.1. CAD Commands Encoding

We define the CAD command sequence following DeepCAD [37], focusing on the two commonly used categories: *sketch* and *extrusion*, where *sketch* includes commands `start`$\{\langle$`SOL`$\rangle\}$, `line`$\{$`L`$\}$, `arc`$\{$`A`$\}$, and `circle`$\{$`R`$\}$ and *extrusion* has a single command `E`, we also need an end command $\langle$`EOS`$\rangle$ for the entire command sequence. Each command is defined by a few parameters for their location, size, and orientation. The detailed definitions of the parameters are given in Table 4. For the i-th line of command $C_i = (s_i, p_i)$, where s_i is the command type and we stack all the parameters for all command types into a vector $p_i = [x, y, \alpha, f, r, \theta, \phi, \gamma, p_x, p_y, p_z, s, e_1, e_2, b, u]$, setting unused parameters to -1. We then pad the sequence to a fixed length, $N_c = 60$, using the empty command $\langle$`EOS`$\rangle$.

Commands	Parameters
$\langle$`SOL`$\rangle$	$\emptyset$
`L` (Line)	x, y : end-points of line
`A` (Arc)	x, y : end-points of arc α : sweep angle f : flag for counter-clockwise
`R` (Circle)	x, y : center of circle r : radius of circle
`E` (Extrude)	θ, ϕ, γ : orientation of sketch plane p_x, p_y, p_z : origin of sketch plane s : associated sketch profile scale e_1, e_2 : extrude distances toward both sides b : bool type, u : extrusion type
$\langle$`EOS`$\rangle$	$\emptyset$

Table 4. The CAD commands and parameters defined in DeepCAD [37] convention.

7.2. CAD Autoencoder

Our autoencoder architecture is similar to [37]. We formulate the task as a classification problem to simplify the learning process. We normalize all CAD models and quantize the continuous parameters into 256 levels represented as 8-bit integers. Therefore, each parameter $p_{i,j}$ where $j \in \{1 \cdots 16\}$ is represented by a one-hot embedding of dimension $256+1 = 257$ with an additional element reserved for unused parameters. We tokenize the commands by mapping them to embedding spaces with learnable matrices, the resulting embedding $e(C_i) = \boldsymbol{e}_i^{\text{cmd}} + \boldsymbol{e}_i^{\text{param}} + \boldsymbol{e}_i^{\text{pos}} \in \mathbb{R}^{d_{\text{E}}}$, where $\boldsymbol{e}_i^{\text{pos}}$ is a learnable positional embedding and $d_{\text{E}} = 256$ is the embedding dimension. The embedding is passed through four layers of transformer blocks and we take the averaged outputs as the latent vector z with the same dimension $d_{\text{E}} = 256$. Then, we reconstruct the CAD command sequence from the latent vector z through a decoder with the same structure as the encoder followed by two linear prediction heads for commands s_i and parameters p_i. The training objective of the autoencoder is to learn accurate predictions of CAD parameters and to regularize the latent space. The training loss is defined as a cross-entropy loss between the predicted $\hat{C}$ and ground-truth C.

7.3. Discussion on Regularization of Autoencoder.

In addition to the reconstruction loss mentioned above, to further regularize the generated latent space, we have also experimented with different regularization terms. For example, we use the KL divergence as a regularization term: $l_{\text{kl}} = D_{\text{KL}}(q(z|C_i) \parallel p(z))$. In this equation, D_{KL} represents the Kullback-Leibler divergence, $q(z|C_i)$ is the latent distribution conditioned on the input C_i, and $p(z)$ is the prior distribution of the latent space. This regularization term ensures that the encoded latent representation closely approximates the predefined prior distribution, which is set as a Gaussian distribution with zero mean and a standard deviation of 0.25. We also utilize a constant β to adjust the strength of the regularization, setting its value to 1×10^{-5}. The VAE reconstruction results are shown in Table 5, demonstrating that the model can reconstruct the sequence with high precision in both scenarios. The regularization terms have minimal impact on the results. Moreover, using the regularization term to train the diffusion model does not result in improvements, so our AE is only trained using the reconstruction loss. To obtain representations better suited for latent diffusion, future work could potentially increase the latent capacity, such as using a sequence of la-

Methods	ACC_{cmd} ↑	ACC_{para} ↑	Med CD ↓	IR ↓
$AE_{w/o-L_{kl}}$	99.52	98.18	0.073	0.026
$AE_{w-L_{kl}}$	99.32	98.02	0.075	0.027

Table 5. Quantitative evaluation of different autoencoding strategies. The CD is multiplied by 10^2.

tent instead of a single latent.

7.4. Diffusion Transformer Network

Our diffusion transformer architecture follows DALLE-2 [25], comprising 12 blocks, each containing a self-attention layer and a fully connected layer. During testing, we start with a randomly sampled noise vector z_T drawn from a standard normal distribution $\mathcal{N}(0, I)$. Our diffusion model is then iteratively applied to this vector to progressively denoise it, resulting in the final output z_0. This process is described by:

$$z_0 = (f \circ \cdots \circ f)(z_T, T, f_m), \quad f(x_t, t) = \Omega(x_t, \gamma(t)|f_m) + \sigma_t \epsilon, \tag{5}$$

where σ_t represents the fixed standard deviation at each timestep t, and ϵ is sampled from $\mathcal{N}(0, I)$. We continue to denoise z_T through successive iterations until z_0 is achieved. The resulting latent vectors z_0 are then fed into the previously trained decoder to reconstruct the CAD sequence. We employ the DDPM solver [9]. Since our training objective function is to predict x_0, we can rearrange the equation of the forward diffusion process to compute ϵ from x_0. This allows us to predict the noise ϵ directly based on the predicted x_0.

8. Dataset Details

We render the compiled CAD models using Blender. To provide comprehensive multi-view information while accommodating our unconstrained testing scenario, for each model, we generate eight sets of four-view images. In each set, we sample four camera locations with mean azimuth angles separated by 90 degrees, applying a random perturbation within a 30-degree range to each azimuth. The four views share the same randomly chosen elevation angle and a radius sampled from 1.8 to 2.5 units. Additionally, for each set, the CAD object is randomly rotated within a range of -15 to 15 degrees along each axis.

While collecting our RealCAD dataset, the collector casually captured images of the object from approximately four different angles: front-left, front-right, back-left, and back-right. There were no specific requirements regarding the elevation and radius for these shots. The 3D-printed CAD models, featuring a variety of textures and colors, were photographed under standard indoor lighting conditions using iPhones.

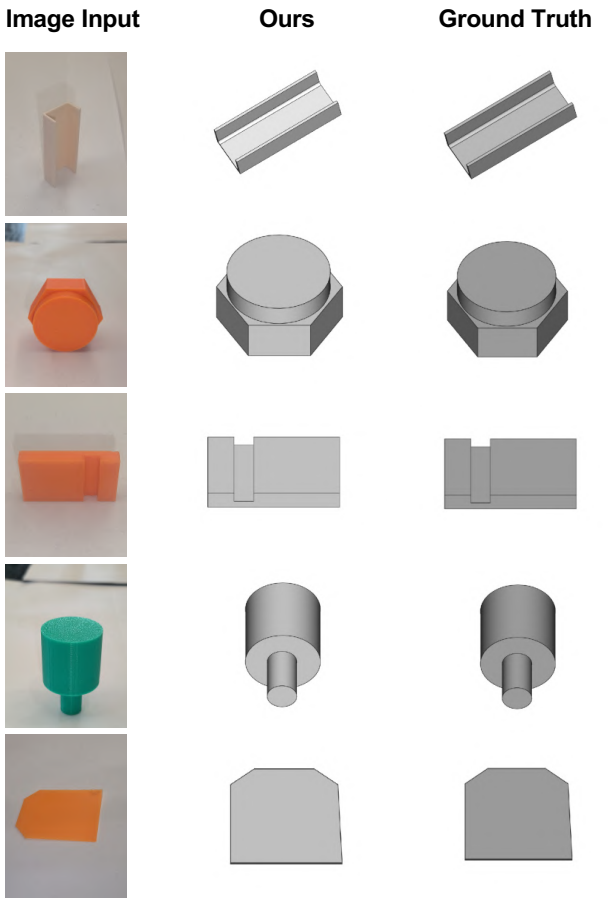

Figure 8. More generated results on RealCAD dataset by our method, the real images are shown on the left.

9. More Results

9.1. Multi-View Reconstruction Diversity

In Figure 7 of the main text, we showcase the diverse results generated using a single view as input. In the single-view setting, our model can produce results with varying levels of complexity for the unseen parts of the object. This is because, with only one view, the model infers the hidden regions, leading to diversity in the generated outputs.

When we switch to the multi-view setting, the multiple perspectives provide comprehensive information about the object. Consequently, the generated results typically present a complete reconstruction of the object's shape, differing mainly in size. As shown in the upper part of Figure 9, we provide examples generated using multi-view inputs. Across different sampling runs, our model consistently recovers the object's shape. However, due to the inherent ambiguity in the image data regarding object scale, the generated results exhibit variations in size. Additionally, our method can generate various CAD design sequences for the

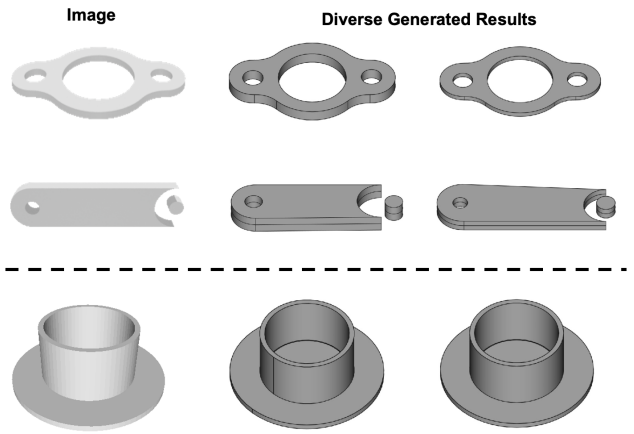

Figure 9. Diverse generated results with multi-view input. To simplify, we use a single image to represent multi-view inputs. Our model reliably captures geometric details, with occasional size variations (upper part). It also generates diverse designs, such as representing a circle as either a full circle or two semi-circular curves (lower part).

Methods	ACC_{cmd} ↑	ACC_{para} ↑	**Med CD** ↓	**IR** ↓
$\text{CADCrafter}_{zero123}$	63.89	42.98	0.201	0.466
CADCrafter	84.62	73.31	0.026	0.036

Table 6. Performance comparisons of the multi-view diffusion model on the DeepCAD dataset.

same model. As shown in the lower part of Figure 9, the generated circle may be represented as either a full circle or two semi-circular curves.

9.2. Discussion on Multi-View Diffusion

In our architecture, we employ a distillation loss to enable our single-view geometry encoder to learn from multi-view knowledge. We have also explored an alternative approach where a multi-view diffusion model is directly employed to generate images from different views using a single-view input. For this experiment, we fine-tune the Zero-1-to-3 model [18] using our rendered CAD image dataset. Despite this effort, the multi-view diffusion model struggled to accurately capture geometry across different views, introducing noise during the conditioning process and ultimately degrading overall performance. We quantitatively evaluate this method on DeepCAD, and the results shown in Table 6 further underscore the necessity of our designs.

9.3. More Results on RealCAD

Here, we showcase more generated results on the RealCAD dataset by our method in Figure 8. It can be observed that our model handles different object poses and sizes effectively. For instance, in the last row, even for very thin objects, the parameters are generated correctly.