Muses: Designing, Composing, Generating Nonexistent Fantasy 3D Creatures without Training

Supplementary Material

We provide additional details of our experimental setup in Sec. A and present further experimental results in Sec. B. Sec. C describes additional implementation details of our method. More qualitative results are included in the accompanying supplementary video.

A. More experimental details

A.1. Evaluation protocol

CLIPScore. For each 3D asset, we render eight images by placing the camera at radius of 2 with a pitch of 30° and yaw angles sampled every $45°$. We then extract CLIP features for each rendered image and its corresponding prompt and compute their cosine similarity. The average cosine similarity over all renders and assets is reported as the CLIPScore.

VQAScore. Because the widely used CLIPScore is known to be unreliable for complex, compositional prompts, we additionally report a VQAScore [4]. Text–image alignment is cast as a binary visual question answering task by converting each prompt into the question: "Does this figure show '{prompt}'? Please answer yes or no." The tokenized image and question are fed into an image–question encoder, and an answer decoder outputs the probability of the answer "Yes". We compute VQAScores using both the CLIP-FlanT5 [4] and ShareGPT4V [2] models and average the predicted probabilities over all rendered views and assets, using the same rendering settings as for CLIPScore.

A.2. User study

To assess how well different methods align with human judgments, we conducted a user study. In each trial, a participant was presented with a text prompt together with rotating video renderings of 3D assets produced by several competing methods, shown simultaneously. Participants were instructed to identify the model that they judged to best match the prompt in terms of visual fidelity and text alignment. In total, 63 participants took part in the study; each participant evaluated 10 distinct trials, and all choices were logged for subsequent statistical analysis. To minimize selection bias, no manual curation of candidate assets was performed. For every participant, the 10 trials were randomly sampled from a pool of 30 cases, and within each trial the display order of the methods was independently shuffled before presentation.

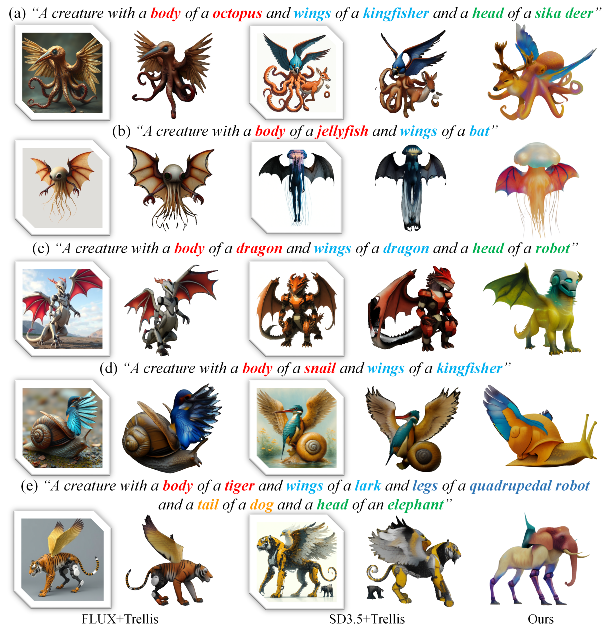

Figure 1. More comparisons with methods that synthesize creative 2D imagery and lift it to 3D, including FLUX.1-schnell [3] and Stable Diffusion 3.5 [1].

B. More results

B.1. Comparison with more 2D generators

As illustrated in Fig. 1, in addition to UNO [5], we also evaluate two additional 2D generative models, FLUX.1-schnell [3] and Stable Diffusion 3.5 [1], on 2D creative generation and compare their quantitative performance. Across these comparisons, our method synthesizes imaginative 3D creatures that remain well aligned with the input descriptions while achieving substantially higher visual fidelity, structural coherence, and improved part-wise consistency.

B.2. More qualitative results

Fig. 5 illustrates additional examples of creative 3D creatures generated by our method. Each 1×4 grid corresponds to a single generated asset shown from four different views.

B.3. More application results

Fig. 2 illustrates the additional applications of our method in geometry and texture editing. Our method enables intuitive, disentangled part-level 3D editing without modifying

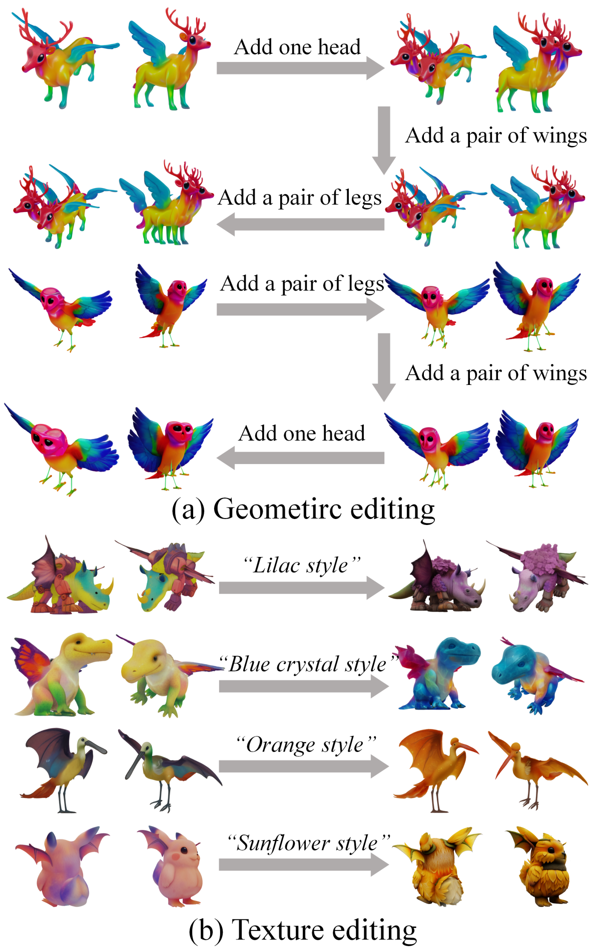

Figure 2. More application results of Muses.

other regions. It supports flexible control over the number of semantic parts, including heads, legs, wings, and tails. Furthermore, by leveraging our geometrically invariant texture editing scheme, we achieve structure-aligned appearance modifications that respect the underlying 3D shape.

C. More implementation details

C.1. Text-to-image

Fig. 3 illustrates the text-to-image process, including the prompts designed for Qwen-Plus [6]. We adopt a shared configuration of global scene settings, content constraints, appearance rules, and negative prompts for all objects to reduce blurriness and artifacts in the generated images. All animals are depicted in a side view, standing on a beach. For birds in particular, we require that their wings be fully spread, as occluded or folded wings frequently lead to errors in subsequent skeleton generation.

Algorithm 1: Graph-based Skeleton Classification

Input : 3D assets $\{\mathbf{X}_m\}_{m=1}^M$, skeletons $\{\mathbf{G}_m = (\mathbf{V}_m, \mathbf{E}_m)\}_{m=1}^M$, root joints $\{\mathbf{r}_m\}_{m=1}^M$

Output: $\{\mathbf{G}_m^{\text{body}}, \mathbf{G}_m^{\text{head}}, \mathbf{G}_m^{\text{leg}}, \mathbf{G}_m^{\text{wing}}, \mathbf{G}_m^{\text{tail}}\}_{m=1}^M$

$\deg(\mathbf{v}) \leftarrow$ degree of $\mathbf{v}$;
$\mathbf{P}(\mathbf{u} \rightsquigarrow \mathbf{v}) \leftarrow$ the shortest path from $\mathbf{u}$ to $\mathbf{v}$;
Stage 1: Preprocessing & orientation calculation;
for $m \leftarrow 1$ **to** M **do**
  $\widetilde{\mathbf{G}}_m \leftarrow \Phi_{\text{clean}}(\mathbf{G}_m)$;
  $\boldsymbol{\delta}_m \leftarrow \Phi_{\text{orient}}(\mathbf{X}_m, \widetilde{\mathbf{G}}_m)$;
  if $\Phi_{\text{fish}}(\widetilde{\mathbf{G}}_m, \boldsymbol{\delta}_m) = 1$ **then**
    $\mathbf{G}_m^{\text{body}} = \widetilde{\mathbf{G}}_m$;
    continue;
  end
  Stage 2: Leg and tail processing;
  $\mathbf{b}_m \leftarrow \Phi_b(\widetilde{\mathbf{G}}_m, \mathbf{r}_m)$; ▷ see Eq. (1)
  $\mathbf{V}_m^{\text{leaf}} = \{\mathbf{v} \in \mathbf{V}_m : \deg(\mathbf{v}) = 1\}$;
  $\mathbf{V}_m^{\text{low}} = \{\mathbf{v} \in \mathbf{V}_m^{\text{leaf}} \mid \mathbf{v}_y < \mathbf{b}_{m_y}\}$;
  $\mathbf{P}_m \leftarrow \{\mathbf{P}(\mathbf{b}_m \rightsquigarrow \mathbf{v}) \mid \mathbf{v} \in \mathbf{V}_m^{\text{low}}\}$; ▷ candidate paths
  $(\mathbf{G}_m^{\text{leg}}, \mathbf{G}_m^{\text{tail}}) \leftarrow \Phi_{\text{lt}}(\mathbf{P}_m, \mathbf{b}_m, \boldsymbol{\delta}_m)$;
  Stage 3: Body, wing and head processing;
  $\mathbf{d}_m \leftarrow \Phi_d(\widetilde{\mathbf{G}}_m, \boldsymbol{\delta}_m, \mathbf{b}_m)$; ▷ see Eq. (2)
  $\mathbf{G}_m^{\text{body}} \leftarrow \mathbf{P}(\mathbf{b}_m \rightsquigarrow \mathbf{d}_m)$;
  $\mathbf{P}'_m \leftarrow \{\mathbf{P}(\mathbf{d}_m \rightsquigarrow \mathbf{v}) \mid \mathbf{v} \in \mathbf{V}_m^{\text{leaf}}\}$;
  $(\mathbf{G}_m^{\text{wing}}, \mathbf{G}_m^{\text{leg}}, \mathbf{G}_m^{\text{head}}) \leftarrow \Phi_{\text{trunk}}(\mathbf{P}'_m, \mathbf{d}_m, \boldsymbol{\delta}_m)$;
  $(\mathbf{G}_m^{\text{front leg}}, \mathbf{G}_m^{\text{hind leg}}) \leftarrow \Phi_{\text{fh}}(\mathbf{G}_m^{\text{leg}}, \mathbf{b}_m, \mathbf{d}_m)$; ▷ split legs into front / hind
end
return $\{\mathbf{G}_m^{\text{body}}, \mathbf{G}_m^{\text{head}}, \mathbf{G}_m^{\text{leg}}, \mathbf{G}_m^{\text{wing}}, \mathbf{G}_m^{\text{tail}}\}_{m=1}^M$;

C.2. Skeleton-guided concept design

Algorithm 1 presents the pseudocode of our algorithm for structurally classifying skeletons into five semantic parts: body, wings, legs, head, and tail. Fig. 6 provides a detailed illustration of the interaction between the user and the LLM (Qwen-Plus [6]) during LLM-guided assembly reasoning.

C.3. Geometrically invariant texture editing

We use FLUX.1 Kontext [3] for image editing, focusing on texture generation while preserving geometric alignment. As illustrated in Fig. 4, the "mythic and dazzling" style can be replaced with arbitrary target styles.

References

[1] Stability AI. Stable diffusion 3.5. https://github.com/Stability-AI/sd3.5, 2024. Accessed: 2024-11-14. 1

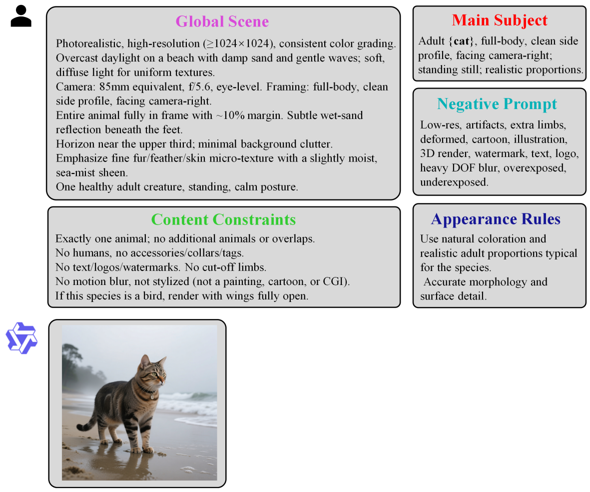

Figure 3. The text-to-image process.

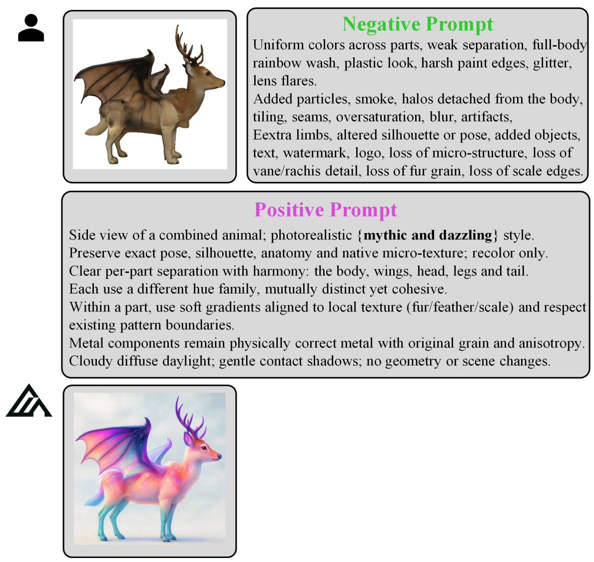

Figure 4. The geometrically invariant texture editing process.

[2] Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Zehui Chen, Haodong Duan, Jiaqi Wang, Yu Qiao, Dahua Lin, et al. Are we on the right way for evaluating large vision-language models? *Advances in Neural Information Processing Systems*, 37:27056–27087, 2024. 1

[3] Black Forest Labs, Stephen Batifol, Andreas Blattmann, Frederic Boesel, Saksham Consul, Cyril Diagne, Tim Dockhorn, Jack English, Zion English, Patrick Esser, et al. Flux. 1 kontext: Flow matching for in-context image generation and editing in latent space. *arXiv preprint arXiv:2506.15742*, 2025. 1, 2

[4] Zhiqiu Lin, Deepak Pathak, Baiqi Li, Jiayao Li, Xide Xia, Graham Neubig, Pengchuan Zhang, and Deva Ramanan. Evaluating text-to-visual generation with image-to-text generation. In *European Conference on Computer Vision*, pages 366–384. Springer, 2024. 1

[5] Shaojin Wu, Mengqi Huang, Wenxu Wu, Yufeng Cheng, Fei Ding, and Qian He. Less-to-more generalization: Unlocking more controllability by in-context generation. *arXiv preprint arXiv:2504.02160*, 2025. 1

[6] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. 2

Figure 5. More results generated by Muses.

System Prompt

You are an experienced layout designer responsible for assembling a 3D skeleton from various parts and using the provided operators to control 3D skeleton parts (body, head, wings, legs, tail, etc.) in a scene. Given various skeleton parts, you should use the provided operators to plan and achieve a coherent composite animal skeleton.
All coordinates, centroids, and sizes in the skeleton configuration are in the same length unit.
Angles are defined in degrees. The dimension sequence is x,y,z. We define +x/−x as the right/left direction, +y/−y as the up/down direction, and +z/−z as the front/back direction.
Positive angles are counterclockwise, and negative angles are clockwise, following the right-hand rule around the part's predefined orientation axis. All transforms are relative to the body's coordinate frame.
Primitive editing operators:
Rotate a part: [Rotate, Target Part Key, Angle (degrees)]
Translate a part: [Translate, Target Part Key, Direction (x/y/z), Distance (same units as coordinates)]
Scale a part: [Scale, Target Part Key, Scale Factor]
Matters needing attention:
1. The body defines the reference frame. All positions, rotations, and translations are interpreted in the body"s coordinate frame.
2. You will be given structured information for each part: key, category (e.g., body, head, wing, tail, leg, front leg, hind leg), orientation, and geometric data (vertices, bounding box, etc.). Use all these attributes to understand how parts should connect.
3. When planning transforms, think in the order Scale → Rotate → Translate. Consider potential collisions or self-intersections between parts and avoid obviously impossible or tangled configurations.
4. When you scale a part, always scale uniformly with a single float Scale Factor. Do not propose anisotropic scaling.
5. Your goal is to attach each non-body part to the body in an anatomically reasonable way (e.g., heads near the neck region, wings near the shoulders, legs under the body, tail at the rear), while maintaining a balanced and realistic overall skeleton layout.
6. If there are multiple parts of the same category (e.g., front/hind legs), reason about which one to move using their locations relative to the body (left/right, front/back, up/down). In the operator calls, always use the precise Target Part Key.
7. Try to use the minimum number of commands needed to satisfy the assembly request and produce a coherent, collision-free configuration.
8. All operators must be convertible to a list of strings and numbers that can be directly processed by json.loads() (no nested tuples or custom objects).
Examples:
1. If you want to rotate the head by 30 degrees and there is only one head in the scene with key "head_0", you should use:["Rotate", "head_0", 30]
2. If you want to move the wing (key "wing_0") slightly upward along +y by 0.25 units, you should use:["Translate", "wing_0", "y", 0.25]
3. If you want to make the tail (key "tail_0") a bit longer but keep its proportions, you should use:["Scale", "tail_0", 1.1]
Think about the skeleton layout step by step:
Which parts need to be attached or adjusted? How should their sizes compare to the body? What rotations make the pose natural? What translations bring each part into correct contact with the body without collisions? Then express your final plan only as a sequence of operator calls.
Final output format:
Return a single JSON array of operations. Each operation must be one of the primitive editing operators. No extra text, no explanations, no comments, and no markdown.

User Input

[Skeleton configurations]:
Part 0: {"key": "body_0", "vertices": [[-0.78, 6.34, 2.48], …], "edges": [[0, 4], …], "animal_name": "dog", "bounding_box": {"min": [-0.78, 6.34, -18.13], "max": [-0.78, 7.13, 21.50], "center": [-0.78, 6.66, 2.00]}, "orientation": "z"}
Part 1: {"key": "wing_0", "vertices": [[-16.73, 14.78, 3.55], …], "edges": [[0, 1], …], "animal_name": "seagull", "bounding_box": {"min": [-16.73, -4.26, 3.55], "max": [15.79, 14.78, 12.27], "center": [-0.55, 5.26, 6.25]}, "orientation": "z"}
Part 2: {"key": "leg_0", "vertices": [[-7.53, 2.51, 33.13], …], "edges": [[0, 4], …], "animal_name": "cat", "bounding_box": {"min": [-7.53, -27.57, 26.79], "max": [6.72, 8.84, 33.13], "center": [-0.23, -11.92, 28.55]}, "orientation": "z"}
Part 3: {"key": "tail_0", "vertices": [[-0.78, -11.89, -45.08], …], "edges": [[0, 1], …], "animal_name": "fox", "bounding_box": {"min": [-0.78, -11.89, -45.08], "max": [-0.78, 8.72, -18.13], "center": [-0.78, -1.25, -32.74]}, "orientation": "x"}
Part 4: {"key": "head_0", "vertices": [[-0.05, -0.91, 10.79], …], "edges": [[0, 1], …], "animal_name": "sika deer", "bounding_box": {"min": [-7.97, -0.91, 10.79], "max": [8.65, 24.43, 32.17], "center": [0.21, 15.19, 26.18]}, "orientation": "z"}
[Command]:
Assemble all given skeleton parts into a coherent composite animal skeleton by attaching the head near the front, the tail near the rear, placing the legs under the body, and positioning the wings near the body center, using realistic, collision-free rotations, translations, and uniform scales relative to the body frame.

LLM Response

[["Scale", "head_0", 0.9], ["Rotate", "head_0", 10], ["Translate", "head_0", "x", -0.78], ["Translate", "head_0", "y", -6.34], ["Translate", "head_0", "z", -3.50], ["Translate", "leg_0", "x", -0.93], ["Translate", "leg_0", "y", 9.12], ["Translate", "leg_0", "z", -20.38], ["Scale", "wing_0", 0.8], ["Rotate", "wing_0", 15], ["Translate", "wing_0", "x", -0.18], ["Translate", "wing_0", "y", 1.25], ["Translate", "wing_0", "z", -0.37], ["Scale", "tail_0", 0.9], ["Rotate", "tail_0",135], ["Translate", "tail_0", "x", 0.26], ["Translate", "tail_0", "y", 8.81], ["Translate", "tail_0", "z", -0.94]]

Figure 6. An example of our LLM-guided assembly reasoning process.