Training-free Geometric Image Editing

Supplementary Material

A. Additional Results

We provide additional qualitative comparisons to state-of-the-art methods in Fig. 8, Fig. 9, and Fig. 10 for 2D editing tasks, Fig. 11 for 3D editing tasks, and Fig. 12, Fig. 13 for inpainting tasks.

B. Extended Applications

Beyond the core tasks presented in the main paper, FreeFine further supports **Partial Mask Editing**, **Appearance Transfer**, and **Cross-Image Composition** to address more complex editing scenarios. Qualitative results for these extended capabilities are shown in Fig. 1 and Fig. 2 , collectively demonstrating the versatility of FreeFine. We formalize the key implementation details as follows:

Appearance Transfer aims to preserve the shape of the source object while replacing its textural or color properties with those of a target object. This is achieved using our TCA module: specifically, by substituting the source image I_s and source mask M_s with the target appearance image and its corresponding object mask, while retaining M_t (the mask of the object to be edited). Additionally, we employ the LP module on M_t to facilitate appearance modification, and can optionally specify the new object category via our CG module to further enhance details.

Cross-Image Composition is a generalized geometric editing task that involves rearranging multiple objects within a shared canvas. This is implemented through our editing pipeline in three key steps: (1) Removing original objects from the source image to produce a clean canvas—this step is optional and only performed when existing objects need to be relocated or replaced; (2) Copying target objects into the canvas and adjusting their geometric properties (e.g., repositioning, reorienting, rescaling) to generate a coarse composite image $\hat{I}_c$; (3) Refining the target regions using the same refinement process described in the main paper.

For composing N objects, the required input parameters are formalized as a set $\mathcal{P}$:

$$\mathcal{P} = \left\{ \left(I_{s,i}, M_{s,i}, M_{t,i};\ L_i^*,\ M_{d,i}^* \right) \mid \forall i \in \{1, 2, ..., N\} \right\}$$

where $I_{s,i}$, $M_{s,i}$, and $M_{t,i}$ are mandatory parameters: $I_{s,i}$ denotes the source image of the i-th object, $M_{s,i}$ its source mask, and $M_{t,i}$ its target mask in the canvas; superscript * indicates optional parameters: L_i^* corresponds to the category label for the CG module, and $M_{d,i}^*$ denotes the draw mask of the i-th object for structure completion.

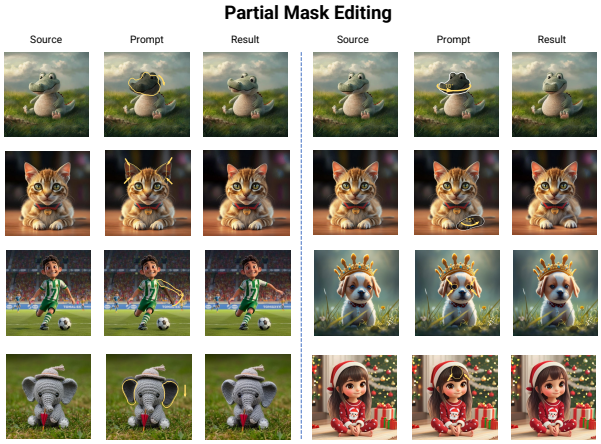


Figure 1. Qualitative result on Partial Mask editing tasks. Instead of editing a whole object, this task focuses on editing parts of an object.

C. Additional Ablation Studies

We further compare TCA with the timestep thresholding alternative mentioned in the main paper, which is equivalent to early stopping using MMSA after reaching a timestep threshold. As shown in Fig. 3, the early-stop strategy faces a trade-off: stopping too early introduces undesired changes, while stopping too late degrades structural completion. In contrast, TCA avoids this trade-off by smoothly transitioning between MMSA and self-attention, achieving both better completion and preservation of details.

D. Additional Implementation Details

D.1. Step 1: Object Transformation Details

As described in the main paper, Step 1 of our pipeline involves transforming the object in the source image I_s based on user instructions. Here, we provide additional details, organized from simple 2D edits to complex 3D transformations.

2D Transformations. For purely 2D edits, $\mathcal{T}_\theta$ represents an affine transformation, which can be expressed as a unified transformation matrix:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x \cdot \cos\phi & -s_y \cdot \sin\phi & t_x \\ s_x \cdot \sin\phi & s_y \cdot \cos\phi & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix},$$

where (x, y) are pixel coordinates in M_s, and (x', y') are the transformed coordinates. Here, s_x and s_y represent scaling factors along the x and y axes, ϕ is the rotation angle, and

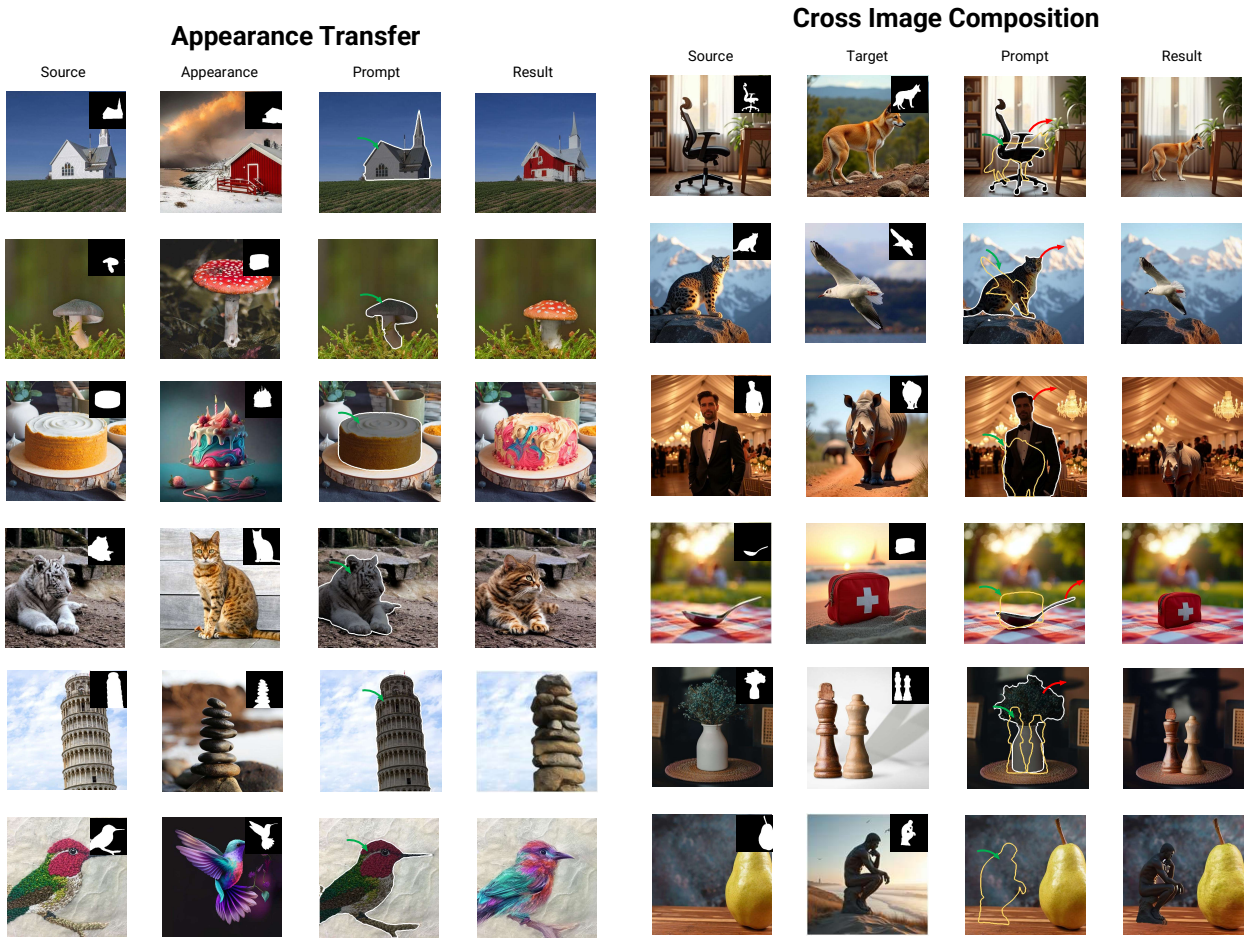


Figure 2. Qualitative results on Extended Applications

Figure 3. Comparisons between the proposed TCA and the straightforward early stop strategy with MMSA. Experiments are conducted while keeping LP and CG applied at the same scale.

(t_x, t_y) is the translation vector. This matrix supports combined operations such as scaling, rotation, and translation in a single transformation.

3D Transformations. For 3D edits, we explore two approaches: (1) depth estimation from a single image to establish a sparse 3D representation, lift the object to 3D space, apply transformations, and re-project it back to 2D; and (2) leveraging video models, particularly multi-view video generation from a single image [32]. Note that existing multi-view generation methods typically require clean foreground images with white backgrounds, and the transformed objects often suffer from detail loss and structural artifacts(e.g. incomplete or missing parts). These limitations highlight the importance of our Step 3 (target region refinement) for achieving high-quality results.

In the first approach, we estimate the scene depth using a depth estimator [34]. Given the depth map D_s of I_s, we lift the object in M_s to 3D space using the camera intrinsic matrix K:

$$P_s = K^{-1} \cdot (x, y, 1)^\top \cdot D_s(x, y),$$

where P_s represents the 3D coordinates of the object. We

then apply the transformation $\mathcal{T}_\theta$ in 3D space. For example, a rotation along the y-axis by ϕ degrees is represented as:

$$P_t = R_y(\phi) \cdot P_s,$$

where $R_y(\phi)$ is the rotation matrix. Finally, we re-project the 3D points back to 2D image space:

$$(x', y') = K \cdot P_t.$$

However, this method has several limitations: (1) the camera intrinsic matrix K is unknown and must be estimated or assumed, (2) the depth map from [34] provides only relative depth, and (3) the sparse 3D representation leads to artifacts during reprojection, requiring additional rasterization and refinement steps. Due to these limitations and its reliance on strong assumptions, this approach struggles to handle large-angle 3D transformations effectively.

To address the challenge of large-angle 3D transformations, we utilize video models such as [32] to generate a multi-view video of the target object. Following the approach of [32], we specify the elevation angle and generate an n-frame multi-view video by controlling the azimuth angles. By indexing the frames, we extract the corresponding transformed images, which are rendered on a white background. We then align the center of the transformed object with the source mask and composite it back into the original image. The key parameters include the azimuth angles (e.g., $[0°, 30°, \ldots, 330°]$) and elevation angles (e.g., $[-10°, 0°, 10°]$), which allow us to control the viewpoint and generate diverse transformations

Notably, both 3D rotation via video models (e.g., SV3D) and that via depth estimation (e.g., DepthAnything) suffer from imprecise angle control. In the main paper, we focus on fair quantitative comparisons under the same depth-based setup. As shown in Fig. 11, even with identical input angles, results from our SV3D-based approach and depth-based approach show deviations, yet the former yields more realistic editing outcomes.

Flexible Combination. Our pipeline supports arbitrary combinations of 2D and 3D transformations, allowing users to achieve both precise control and realistic results. For example, a 2D translation can be combined with a 3D rotation to create complex editing effects.

D.2. Local Perturbation

As described in the main paper, Local Perturbation (LP) introduces controlled stochasticity within a user-defined mask $\mathcal{M}$ to enable dramatic changes in editing scenarios. Here, we provide additional implementation details.

The original sampling process can be formulated as:

$$\hat{x}_0 = \frac{x_t - \sqrt{1-\alpha_t} \cdot \epsilon_\theta(x_t, t)}{\sqrt{\alpha_t}}, \tag{1}$$

$$\tilde{x}_{t-1} = \sqrt{\alpha_{t-1}} \cdot \tilde{x}_0 + \sqrt{1-\alpha_{t-1}-\sigma_t^2} \cdot \epsilon_\theta(x_t, t) + \sigma_t \cdot \epsilon, \tag{2}$$

where t is the timestep, x_t is the latent variable, $\epsilon_\theta(x_t, t)$ is the model-predicted noise, α_t is a noise scheduling parameter, $\epsilon \sim \mathcal{N}(0, I)$ is standard Gaussian noise, and σ_t is set to 0 for deterministic control.

Our key insight is to introduce additional stochasticity into local completion regions while maintaining deterministic behavior elsewhere. Specifically, we set:

$$\sigma_t = \begin{cases} \sqrt{\frac{1-\alpha_{t-1}}{1-\alpha_t} \cdot \left(1 - \frac{\alpha_t}{\alpha_{t-1}}\right)}, & \text{if } \mathcal{M} \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

This allows LP to selectively apply DDPM updates [7] within the mask and DDIM updates [25] elsewhere, balancing flexibility and control.

LP is versatile and can be applied to various tasks with different purposes. In Step 2 (source region inpainting), LP is applied to the source mask M_s to remove unwanted objects while preserving the background. In Step 3 (target region refinement), LP is applied to the target mask M_d to refine and complete structures. For general refinement tasks without structural completion demands (e.g., easy 2D transformations), we focus on the boundary between the object and the background to mitigate unnatural blending. Specifically, we define the boundary mask using morphological dilation [1]:

$$M_b = \text{Dilation}(M_s) - M_s,$$

and set $\mathcal{M} = M_b$ to guide the refinement process.

Since our framework's components can be seamlessly replaced by other tools, when undesired content is generated in the source region, we can further refine it in Step 3 by simply including the problematic region in $\mathcal{M}$ during LP.

E. Evaluation and Baselines

This section provides additional details on the evaluation metrics and the implementation of baseline methods used in our experiments.

E.1. Metrics

Details of how to calculate the metrics: (1) FID [6]. The implementation of pytorch-fid [23] is used, specifically by extracting feature vectors from Inception-v3 [28] and computing the Fréchet distance (FID) between the distributions

of the edited results and 2,000 randomly selected source images. Furthermore, we replace the Inception features with DINO features (DINO) and use kernel distance (KD) instead of the Fréchet distance to mitigate the bias inherent in the original FID, enabling a more comprehensive evaluation of image quality. (2) Subject Consistency (SC). It measures the consistency of the foreground subject between source image I_s and cgenetated image I_t. Given the source mask M_s and the target mask M_t, SC value is calculated as:

$$V_{SC} = \cos\left(\mathbf{F}_{\text{DINO}}[I_s \cdot M_s], \mathbf{F}_{\text{DINO}}[I_t \cdot M_t]\right),$$

where $\cos(\cdot)$ indicates the cosine distance and $\mathbf{F}_{\text{DINO}}$ represents the DINO [2] feature of a image. (3) Background Consistency (BC). It measures the consistency of the background and is calculated similarly to SC value but with the CLIP [19] feature $\mathbf{F}_{\text{CLIP}}$:

$$V_{BC} = \cos\left(\mathbf{F}_{\text{CLIP}}[I_s \cdot M_{bg}], \mathbf{F}_{\text{CLIP}}[I_t \cdot M_{bg}]\right),$$

where $M_{bg} = 1 - (M_s \cup M_t)$ represents the mask of background. The settings of using different features are from VBench [8]. (4) Warp Error (WE). We follow the implementation described in GeoDiffuser [22] to get the warped image I_w and compute the L1 error $L_1(\cdot)$:

$$V_{WE} = L_1\left(I_t \cdot M_t, I_w \cdot M_t\right).$$

(5) Mean Distance (MD). We first find interest points P_i on the source image using SiFT [15], which are transformed with the transform $\mathcal{F}$ to obtain target positions P_t and used to locate the corresponding points P_c in the edited image via DiFT [30]. The mean distance between P_t and P_c reflects the accuracy of the edit.

E.2. DragonDiffusion [17]

DragonDiffusion supports object moving and resizing in its original implementation. The source mask M_s in our method is one of the inputs of DragonDiffusion to locate the original content position. Instead of the original way of representing edits with drag points, we directly translate the edit parameters into model inputs. By specifying the target mask M_t in our method as its target dragging position, we apply DragonDiffusion to the rotating task. As for 3D-edits, We implement it by setting the drag points in the content dragging demo. All other hyper parameters and optional inputs remain unchanged by default.

E.3. MotionGuidance [5]

The core input to MotionGuidance is the optical flow that represents the edit task. For 2D-edits, we can simply get the optical flow by geometric calculation with the source mask M_s for the object and the editing parameters. For 3D-edits, We use SV3D [32] to warp the source image and use RAFT [31] to estimate the optical flow between the warped image and the original. Once getting the source image and the target optical flow, we can implement MotionGuidance in different editing tasks. Notably, MotionGuidance has 500 denoising steps in the default settings and takes an average of 40 minutes per image. Considering the fairness and the cost of the comparison experiment, we set the same 50 steps as the others and use the editing results for the perceptual study. As for qualitative comparisons, the default settings are used.

E.4. RegionDrag [16]

RegionDrag requires the indication of source and target regions to perform editing tasks. This method supports 2D moving, scaling, and rotation; however, the 3D editability of RegionDrag is inherently limited due to the input form and optimization strategy. We use the official implementation released by the authors and pass in the source mask M_s and target mask M_t in GeoBench to indicate the source and target edit regions respectively. We use default values for other hyperparameters and optional inputs.

E.5. Self-Guidance [4]

Self-Guidance does not explicitly take in point-based or region-based input. Instead, this method needs textual input to indicate the subject that requires editing. We discover that the official demo code from the authors does not support real image editing; therefore, we follow the implementation described in GeoDiffuser [22] and use the code base from [35]. We first use DDIM Inversion on real images to obtain the starting noise latent. To indicate the desired 2D and 3D transformations, we use the geometric editing instructions from our dataset. Finally, we use Eq. 9 in Self-Guidance [4] to optimize both the object shape and appearance.

E.6. DragDiffusion [24]

DragDiffusion relies on the user input drag points for geometric image editing. This method gradually drags the source points to the target location by running an iterative optimization process on the image latent in a specific intermediate time step. We directly leverage the original implementation of DragDiffusion by feeding in the source mask M_s and editing parameters to automatically obtain the control points.

E.7. GeoDiffuser [22]

GeoDiffuser achieves both 2D and 3D edits by using shared reference and edit attention layers. The attention sharing mechanism will take in geometric transformations and create corresponding losses for image latent optimization. We directly use the original GeoDiffuser implementation for evaluation. We convert transformation parameters from GeoBench into transformation matrix to serve as the model

input. In 2D scenarios, we always use a constant depth map for the whole image following GeoDiffuser, while for 3D, we estimate the image depth using Depth Anything. [34]

E.8. Diffusion Handles [18]

Diffusion Handles performs geometric edits by leveraging depth maps and camera transformations. Its workflow flow proceeds as follows: first, it conducts null-text inversion using the depth-to-image Stable Diffusion model, then inpaints the foreground region of the target object with LaMa [27]. The inpainted image is subsequently used to estimate the scene's background depth, which is blended with the transformed foreground object. Finally, the edited image is generated using the transformed depth map combined with the transformed activations of the depth-to-image SD model, as detailed in [18]. Our implementation is built on the codebase of GeoDiffuser [22], which adopts the official implementation provided by the authors of Diffusion Handles [18]. For parameter compatibility, we adapt our editing parameters to fit the model's input format and use a constant depth setting for 2D editing.

E.9. DesignEdit [9]

DesignEdit treats geometric image editing as two sub-tasks: multi-layered image decomposition and multi-layered image fusion. This method segments out the foreground object and background in the latent space, then applies instruction guided latent fusion that pastes the multi-layered latent representations onto a canvas latent. We use the official implementation [9] released by the authors and additionally implement rotation editing on top of it.

E.10. Inpainting Methods

We conduct experiments using MAT [14], LaMa [27], BrushNet [10], and Stable Diffusion Inpainting [26]. As mask dilation is a common technique for object removal, we set a dilation factor of 30 on the source mask M_s for Step 2 across all methods, including ours, to minimize object remnants caused by imperfect masks. Additionally, we use the conditional prompt "empty scene" for guidance and fix the random seed in all comparative and ablation experiments to ensure fair and reproducible results.

For the structure completion task, we use manually drawn masks to guide region completion and employ object-specific text prompts from our GeoBench dataset. For general refinement tasks without structure completion, we use boundary masks, as described in the Local Perturbation (LP) section, to guide the refinement process.

F. Details of GeoBench

In the main paper, we introduced **GeoBench**, a benchmark for evaluating geometric editing methods. This section details its data generation pipeline, including mask generation, label extraction, editing instruction creation, and manual supervision, along with dataset statistics.

Mask Generation. Accurate object masks are essential for distinguishing foreground from background and enabling precise region editing. We use the segmentation tool SAM [12] to automatically segment datasets sourced from PIE-Bench [11] and Subjects200K [29], which consist of real and synthetic images featuring prominent objects suitable for geometric editing. However, SAM faces challenges in segmentation granularity (e.g., instance, panoptic, part-level) and may not align with our editing requirements. To overcome this, we use SemanticSAM [13], which allows manual adjustment of segmentation granularity, ensuring more precise and relevant masks. Post-processing first applies algorithm-based filtering: tiny masks are discarded, and images with excessive masks (over 50 per image, indicating crowded scenes with limited space for geometric edits) are excluded, where the former is defined as:

$$\texttt{tiny mask} \iff \frac{\sum M}{h \times w} < 0.001$$

where M denotes the object mask, h is the image height, and w is the image width. Remaining masks then undergo manual selection to retain the most prominent (e.g., texture-clear, non-blurry) and editable foreground objects (non-overlapping with other objects, not truncated by image boundaries).

Label Generation. We use ShareGPT4V [3] to generate detailed image descriptions and an algorithm based on CLIP similarity [19, 33] to extract object labels. By comparing the CLIP similarities before and after background inpainting, we select the top k most related labels and send them to human filtering for final verification. Here, we set $k = 5$ as the most appropriate label consistently appears in the top 5. An alternative approach, GroundingSAM [20], offers simultaneous mask and label generation but shares the same limitations regarding segmentation granularity and error accumulation, often resulting in irrelevant or overly coarse masks.

Editing Instruction Generation. To enable diverse and multi-level edits, we design a range of editing prompts and randomly generate instructions for each image, categorized into three difficulty levels: easy, medium, and hard, as detailed in Tab. 1. Instructions include transformations such as moving (eight directions: up, down, left, right, and diagonals), resizing (zooming in/out), and rotating (clockwise, counterclockwise, and 3D rotations). For 3D rotation instructions, we validate direct depth estimation-based transformations and exclude samples with unreliable depth estimates. We also exclude instructions that result in objects extending beyond the image boundaries. Additionally, we manually identify cases requiring structural completion and

Table 1. Parameter ranges for editing operations across three difficulty levels.

Difficulty	2D Rotate (°)	Move (Frac. of Img Size)	Resize (Enlarge)	Resize (Shrink)	3D Rotate (°)
Easy	5–10	0.05–0.1	1.1–1.3	0.8–0.9	5–10
Medium	10–20	0.1–0.2	1.3–1.5	0.6–0.8	15–20
Hard	20–40	0.2–0.4	1.5–3.0	0.4–0.6	25–40

create a dedicated subset for these tasks, along with manually drawn completion masks for each image.

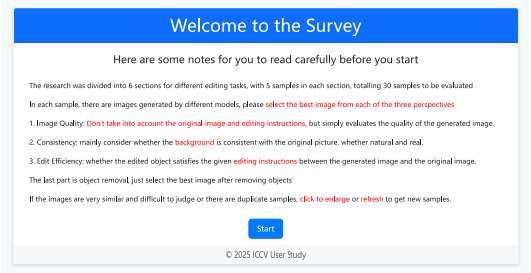
Dataset Statistics. GeoBench comprises 811 source images and 5,988 editing instructions, including 2,267 easy, 2,075 medium, and 1,646 hard edits. The dataset is further divided into three subsets: (1) general 2D edits (5,677 instructions), (2) 3D edits (190 instructions), and (3) a manually annotated subset for structural completion tasks (121 instructions).

G. User Study Details

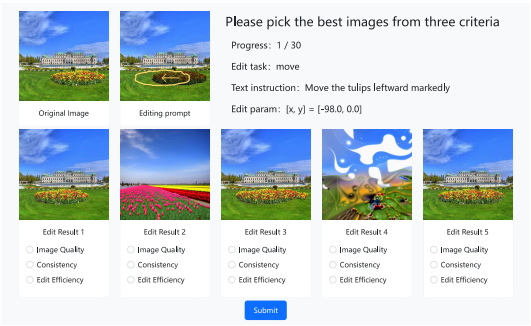
A website was built for the user study and we recruited 35 participants with diverse backgrounds in computer vision to vote online. The home page (Fig. 4 (a)) explains the voting task and provides guidelines for participants. The survey includes 6 sections (Move, Rotate, Resize, 3D-Edits, Region Refinement, Region Inpainting), with 5 samples per section, totaling 30 evaluations and the first three sections are all 2D-edits. The Start button redirects users to the vote page (Fig. 4 (b)), which presents a random editing sample, including the original image, the visualisation of editing prompt, anonymous editing results of our method and comparative models in a random order and the textual information about the sample. Figure shows visualization results of specific tasks (Move, Rotate, Resize) in 2D-edits.

We compared (1) editing models (DragonDiffusion [17], RegionDrag [16], Self-Guidance [4], MotionGuidance [5]) in 2D-edits and 3D-edits, (2) inpainting models (BrushNet [10], SD-inpainting-v1.5 [21], LaMa [27], MAT [14]) in the region inpainting task, (3) BrushNet [10], SD-inpainting-v1.5 [21] and editing models in the Region Refinement task. 35 participants picked the best image from three criteria and submitted it, generating 2,622 valid votes (Tab. 3). In the final section, participants selected one best image (instead of three criteria) in the region inpainting task, with the voting number as one-third of the other sections. Fig. 5 shows voting statistics in the 2D-edits and 3-edits from different criteria, only the editing models are counted.

We also validate the alignment between the metrics used in the main paper and user preferences across three key dimensions: Image Quality, Consistency, and Editing Effectiveness, as seen in the Fig. 6 with data from Tab. 2. The



(a) Home page



(b) Vote page

Figure 4. Screen shots of the website for user study.

Table 2. Voting statistics in the 2D-edits and 3-edits from different criteria, only the editing models are counted.

Method	Image Quality	Consistency	Editing Effectiveness	Total
Ours	**475**	**473**	**543**	**1491**
DragonDiffusion [17]	139	149	87	375
RegionDrag [16]	31	40	29	100
Self-Guidance [4]	23	6	9	38
MotionGuidance [5]	0	0	0	0
Total	668	668	668	2004

seven metrics generally correlate strongly with human preference. However, the WE metric under Editing Effectiveness shows some degree of misalignment, as evidenced by the human preference ratings for RegionDrag and DragonDiffusion. Developing more robust quantitative metrics to better measure editing effectiveness remains an important direction for future work.

H. Failure Cases and Limitations

While our method achieves strong performance across a variety of geometric editing tasks, it still faces certain failure cases and limitations, which are detailed below.

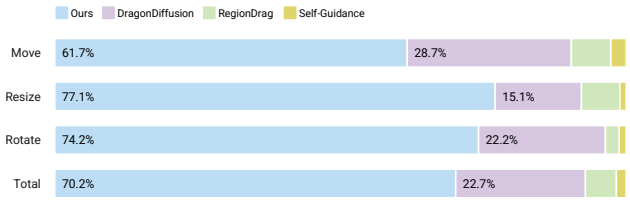


Figure 5. Visualization results of perceptual study in 2D-edits (Move, Rotate, Resize).

Table 3. Voting statistics in different editing tasks. The blank cells indicate that the model was not compared in the task.

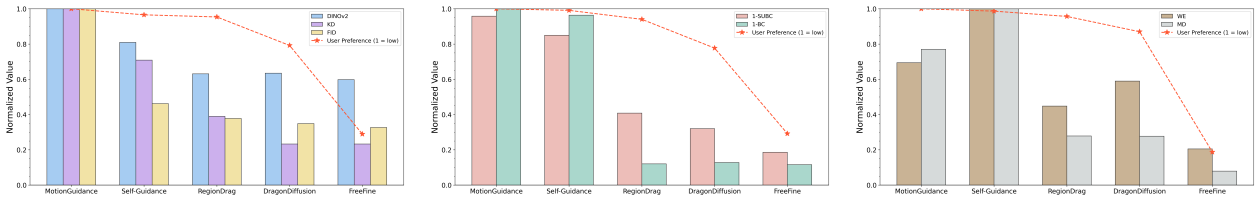
Method	2D-Edits				3D-Edits	Region Refinement	Region Inpainting	Total
	Move	Resize	Rotate	Total				
Ours	**374**	**347**	**365**	**1086**	**405**	**258**	37	1786
DragonDiffusion[17]	174	68	109	351	24	50		425
RegionDrag[16]	42	30	12	84	16	12		112
Self-Guidance[4]	16	5	6	27	11	3		41
MotionGuidance[5]	0	0	0	0	0	0		0
BrushNet[10]						89	22	111
SD-inpaint-v1.5[21]						56	21	77
LaMa[27]							**52**	52
MAT[14]							18	18
Total	606	450	492	1548	456	468	150	2622

H.1. Failure Cases

Failure cases are illustrated in Fig. 7: (1) Unintended color changes may occasionally occur in the background during editing. This likely stems from the need for early denoising steps in Step 2 (implemented to avoid interference from the source region's context), which can cause information loss and hinder color consistency. (2) Fine details such as tiny text or guitar strings may appear blurry, as preserving high-frequency features remains a technical challenge. (3) Despite supporting larger 3D rotations via SV3D, our method still struggles with large-angle 3D rotations for many objects. This is due to suboptimal coarse edits from lifting models, creating significant barriers to effective refinement. We hope, however, that FreeFine's inherent flexibility will help mitigate these issues through the integration of stronger foundation models in future work. Addressing these failure cases is a key direction for our future work.

H.2. Limitations

While our method shows strong performance, it still has several limitations. First, it relies on user input in two aspects: (1) it requires user-provided masks for editing guidance, and (2) structure completion tasks depend on manual intervention to guide the process. Developing automated pipelines for these steps would reduce manual effort. Second, for complex 3D edits, our method depends on depth estimates or other 3D models. Removing this requirement, if possible, would make the system more broadly applicable. Third, similar to most diffusion-based methods, our approach has relatively high computational costs compared to feed-forward models like GANs. Adopting more efficient sampling strategies could lower inference costs.



(a) Image Quality (b) Consistency (c) Editing Effeteness

Figure 6. Assessment of the Alignment between Metrics from the Main Paper and User Preferences across Three Dimensions.

(a) Unintended Color Change (a) Loss of High-Frequency Details (C) Large Angle 3D Rotation Problem

Figure 7. Visualization of failure cases.

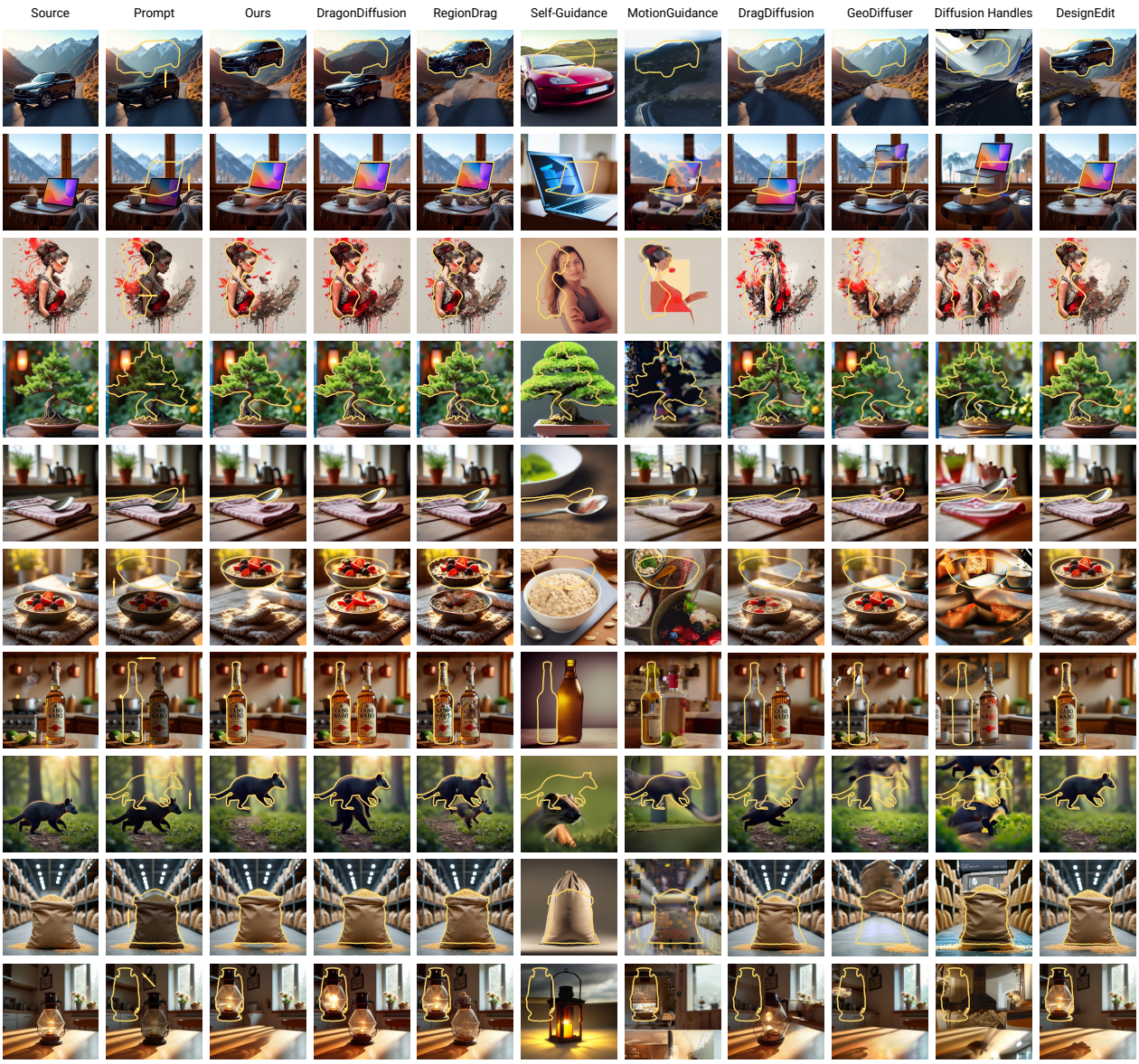


Figure 8. Qualitative comparison with state-of-the-art editing methods in moving operations.

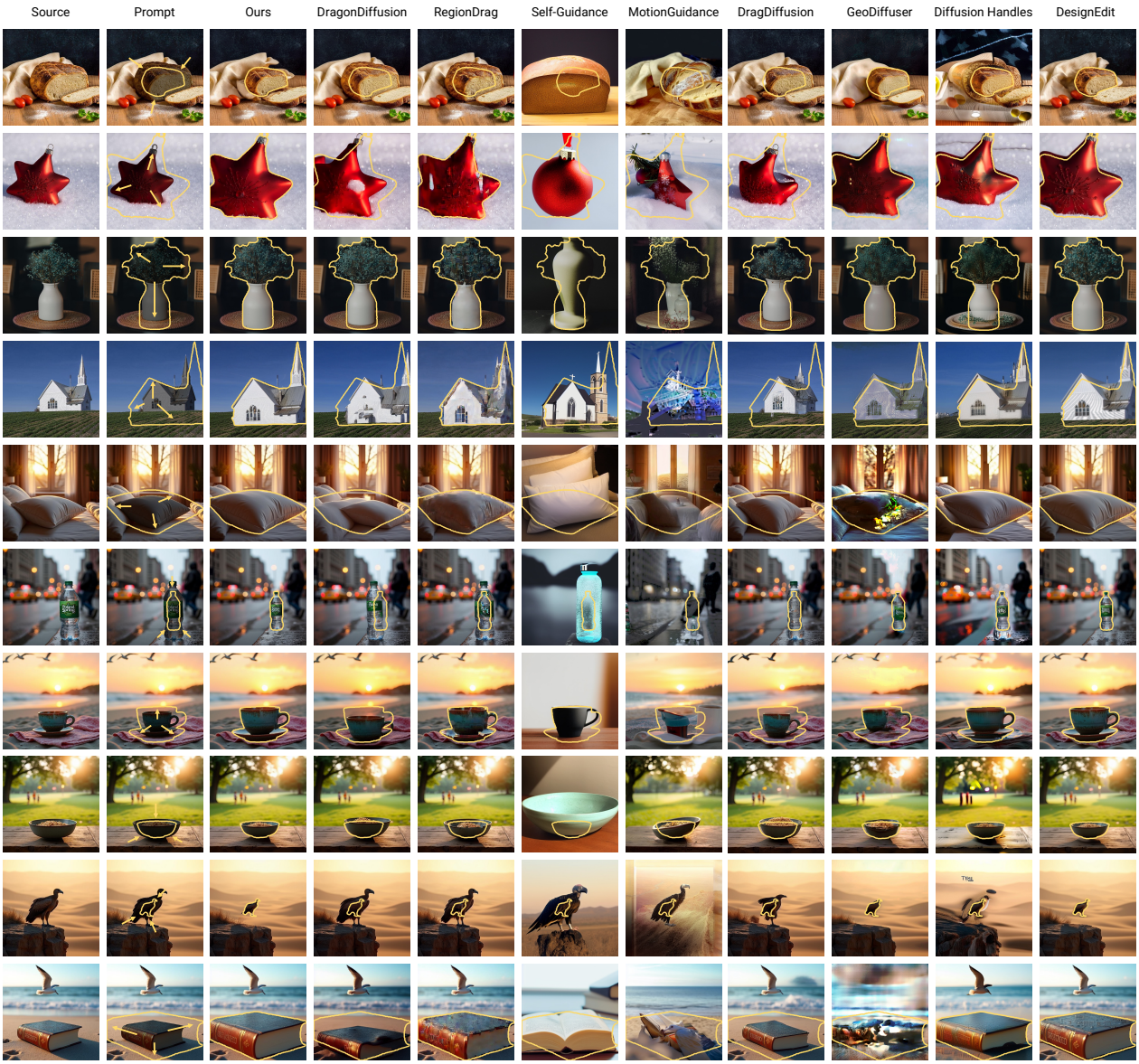


Figure 9. Qualitative comparison with state-of-the-art editing methods in scaling operation.

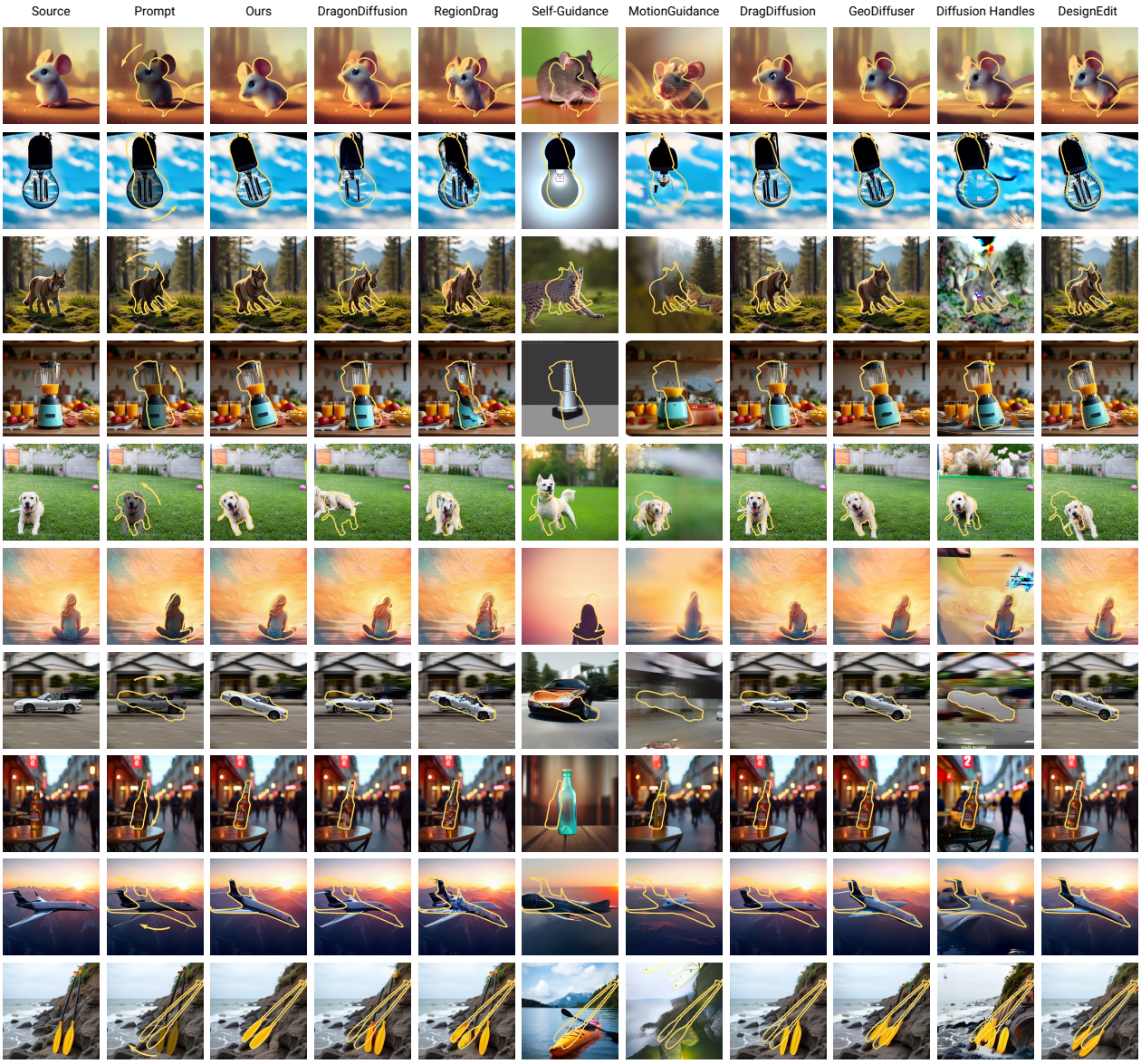


Figure 10. Qualitative comparison with state-of-the-art editing methods in rotation operation.

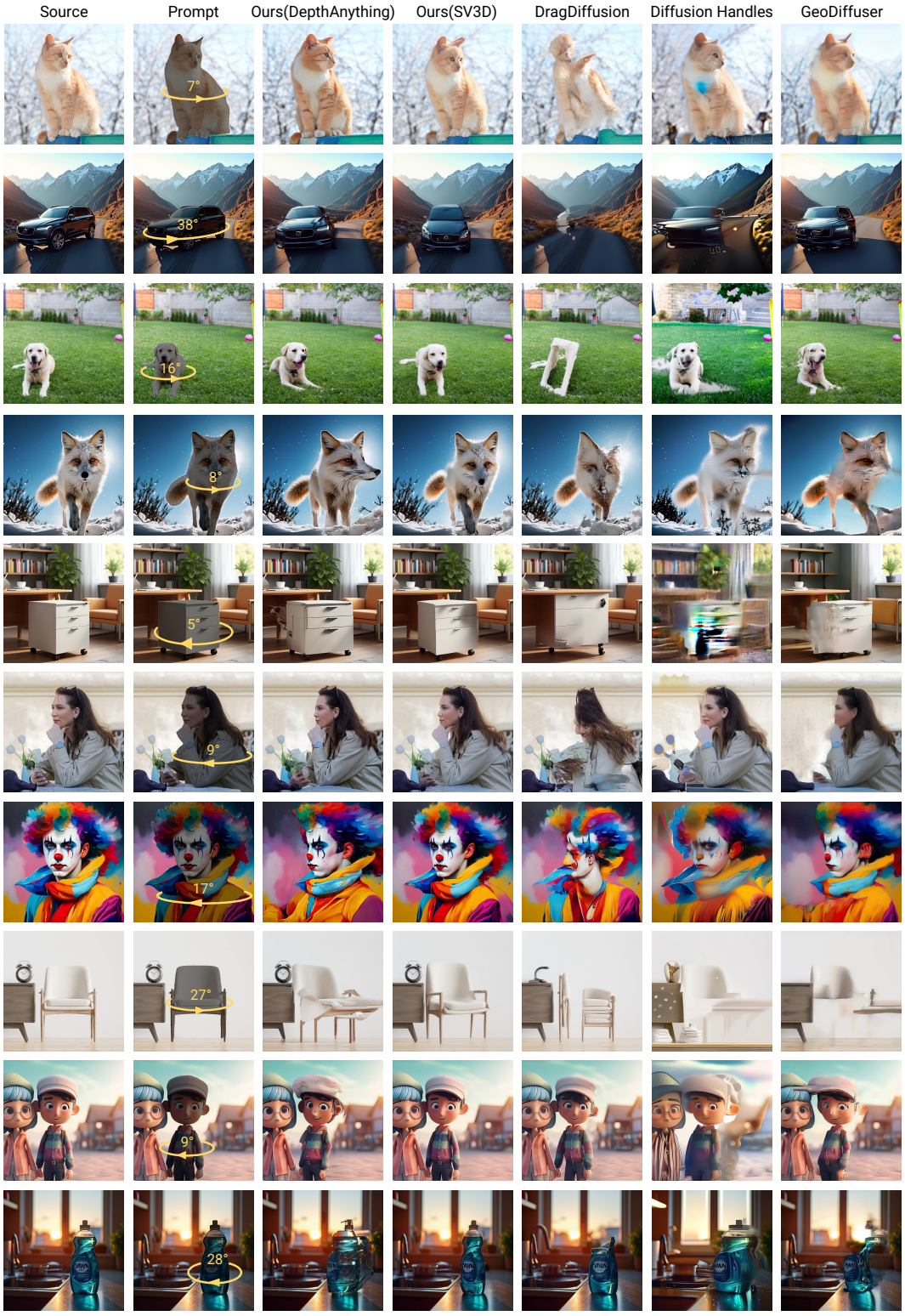


Figure 11. Qualitative comparison with state-of-the-art inpainting methods in 3D-editing scenarios. Two variants of our method are distinguished by suffixes, indicating the different underlying lifting models.

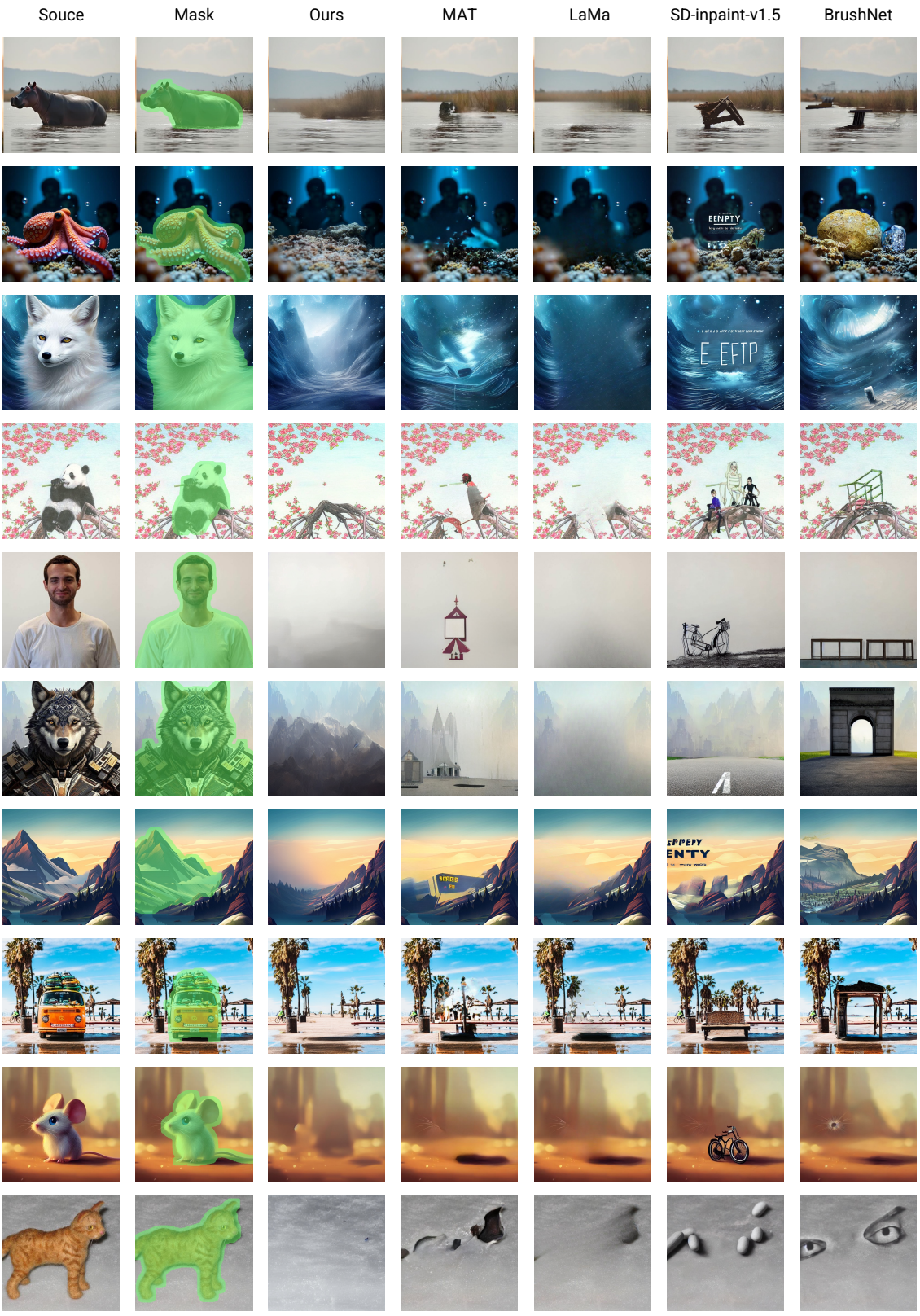


Figure 12. Qualitative comparison with state-of-the-art inpainting methods in source region inpainting.

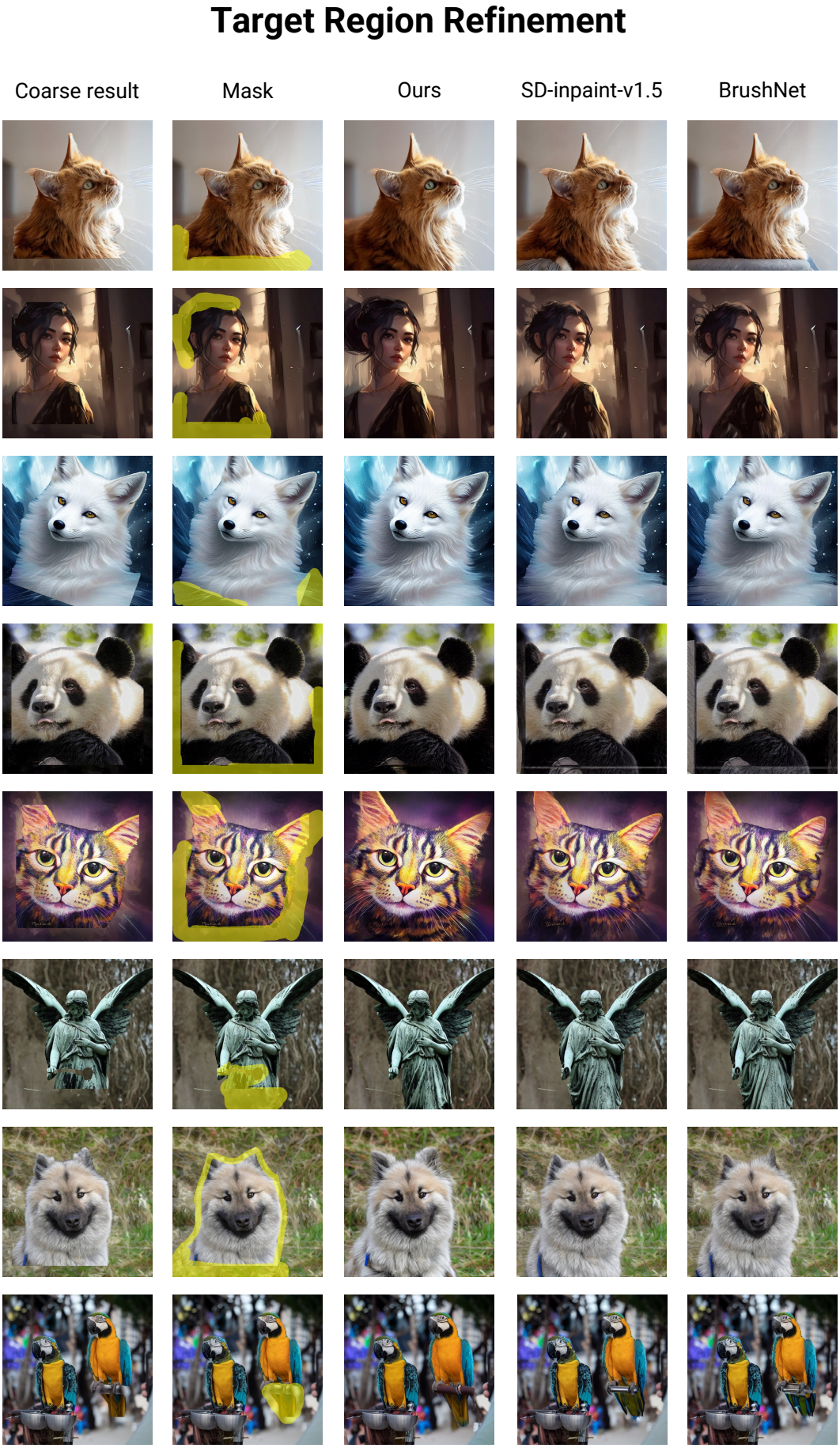


Figure 13. Qualitative comparison with state-of-the-art inpainting methods in target region refinement.

References

[1] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000. 3

[2] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *ICCV*, pages 9630–9640, 2021. 4

[3] Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Conghui He, Jiaqi Wang, Feng Zhao, and Dahua Lin. Sharegpt4v: Improving large multi-modal models with better captions. In *ECCV*, pages 370–387, 2024. 5

[4] Dave Epstein, Allan Jabri, Ben Poole, Alexei A. Efros, and Aleksander Holynski. Diffusion self-guidance for controllable image generation. In *NeurIPS*, 2023. 4, 6, 7

[5] Daniel Geng and Andrew Owens. Motion guidance: Diffusion-based image editing with differentiable motion estimators. In *ICLR*, 2024. 4, 6, 7

[6] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *NeurIPS*, pages 6626–6637, 2017. 3

[7] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020. 3

[8] Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, Yaohui Wang, Xinyuan Chen, Limin Wang, Dahua Lin, Yu Qiao, and Ziwei Liu. VBench: Comprehensive benchmark suite for video generative models. In *CVPR*, 2024. 4

[9] Yueru Jia, Yuhui Yuan, Aosong Cheng, Chuke Wang, Ji Li, Huizhu Jia, and Shanghang Zhang. Designedit: Multi-layered latent decomposition and fusion for unified & accurate image editing. *CoRR*, abs/2403.14487, 2024. 5

[10] Xuan Ju, Xian Liu, Xintao Wang, Yuxuan Bian, Ying Shan, and Qiang Xu. Brushnet: A plug-and-play image inpainting model with decomposed dual-branch diffusion. In *ECCV*, pages 150–168, 2024. 5, 6, 7

[11] Xuan Ju, Ailing Zeng, Yuxuan Bian, Shaoteng Liu, and Qiang Xu. Pnp inversion: Boosting diffusion-based editing with 3 lines of code. *International Conference on Learning Representations (ICLR)*, 2024. 5

[12] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloé Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross B. Girshick. Segment anything. In *ICCV*, pages 3992–4003, 2023. 5

[13] Feng Li, Hao Zhang, Peize Sun, Xueyan Zou, Shilong Liu, Jianwei Yang, Chunyuan Li, Lei Zhang, and Jianfeng Gao. Semantic-sam: Segment and recognize anything at any granularity. *arXiv preprint arXiv:2307.04767*, 2023. 5

[14] Wenbo Li, Zhe Lin, Kun Zhou, Lu Qi, Yi Wang, and Jiaya Jia. MAT: mask-aware transformer for large hole image inpainting. In *CVPR*, pages 10748–10758, 2022. 5, 6, 7

[15] G Lowe. Sift-the scale invariant feature transform. *Int. j*, 2004. 4

[16] Jingyi Lu, Xinghui Li, and Kai Han. Regiondrag: Fast region-based image editing with diffusion models. In *ECCV*, pages 231–246, 2024. 4, 6, 7

[17] Chong Mou, Xintao Wang, Jiechong Song, Ying Shan, and Jian Zhang. Dragondiffusion: Enabling drag-style manipulation on diffusion models. In *ICLR*, 2024. 4, 6, 7

[18] Karran Pandey, Paul Guerrero, Matheus Gadelha, Yannick Hold-Geoffroy, Karan Singh, and Niloy J. Mitra. Diffusion handles enabling 3d edits for diffusion models by lifting activations to 3d. In *CVPR*, pages 7695–7704. IEEE, 2024. 5

[19] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, pages 8748–8763, 2021. 4, 5

[20] Tianhe Ren, Shilong Liu, Ailing Zeng, Jing Lin, Kunchang Li, He Cao, Jiayu Chen, Xinyu Huang, Yukang Chen, Feng Yan, Zhaoyang Zeng, Hao Zhang, Feng Li, Jie Yang, Hongyang Li, Qing Jiang, and Lei Zhang. Grounded SAM: assembling open-world models for diverse visual tasks. *arXiv preprint arXiv:2401.14159*, abs/2401.14159, 2024. 5

[21] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, pages 10674–10685, 2022. 6, 7

[22] Rahul Sajnani, Jeroen van Baar, Jie Min, Kapil Katyal, and Srinath Sridhar. Geodiffuser: Geometry-based image editing with diffusion models. In *WACV*, pages 472–482, 2025. 4, 5

[23] Maximilian Seitzer. pytorch-fid: FID Score for PyTorch. https://github.com/mseitzer/pytorch-fid, 2020. Version 0.3.0. 3

[24] Yujun Shi, Chuhui Xue, Jun Hao Liew, Jiachun Pan, Hanshu Yan, Wenqing Zhang, Vincent Y. F. Tan, and Song Bai. Dragdiffusion: Harnessing diffusion models for interactive point-based image editing. In *CVPR*, 2024. 4

[25] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *ICLR*, 2021. 3

[26] Lorenzo Stacchio. Train stable diffusion for inpainting, 2023. 5

[27] Roman Suvorov, Elizaveta Logacheva, Anton Mashikhin, Anastasia Remizova, Arsenii Ashukha, Aleksei Silvestrov, Naejin Kong, Harshith Goka, Kiwoong Park, and Victor Lempitsky. Resolution-robust large mask inpainting with fourier convolutions. In *WACV*, pages 3172–3182, 2022. 5, 6, 7

[28] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016. 3

[29] Zhenxiong Tan, Songhua Liu, Xingyi Yang, Qiaochu Xue, and Xinchao Wang. Ominicontrol: Minimal and universal control for diffusion transformer. *CoRR*, abs/2411.15098, 2024. 5

[30] Luming Tang, Menglin Jia, Qianqian Wang, Cheng Perng Phoo, and Bharath Hariharan. Emergent correspondence from image diffusion. In *NeurIPS*, 2023. 4

[31] Zachary Teed and Jia Deng. RAFT: recurrent all-pairs field transforms for optical flow. In *ICCV*, pages 402–419, 2020. 4

[32] Vikram Voleti, Chun-Han Yao, Mark Boss, Adam Letts, David Pankratz, Dmitry Tochilkin, Christian Laforte, Robin Rombach, and Varun Jampani. SV3D: novel multi-view synthesis and 3d generation from a single image using latent video diffusion. In *ECCV*, pages 439–457, 2024. 2, 3, 4

[33] Feng Wang, Jieru Mei, and Alan L. Yuille. SCLIP: rethinking self-attention for dense vision-language inference. In *ECCV*, pages 315–332, 2024. 5

[34] Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything: Unleashing the power of large-scale unlabeled data. In *CVPR*, pages 10371–10381, 2024. 2, 3, 5

[35] Shengzhe Zhou. Diffusion self guidance implementation. 4