

A APPENDIX

Here we present additional details and evaluation. Note: we use LLMs as a Latex/Python programming reference and check for related works.

A.1 ADDITIONAL TECHNICAL DETAILS

To lift a depth map $D \in \mathbb{R}^{H \times W}$ to a 3D point cloud using the pinhole camera model, each pixel (u, v) with depth $d_{u,v}$ maps to a 3D point (X, Y, Z) as:

$$X = \frac{(u - c_x) \cdot d_{u,v}}{f_x}, \quad Y = \frac{(v - c_y) \cdot d_{u,v}}{f_y}, \quad Z = d_{u,v}$$

where (c_x, c_y) is the principal point (typically $W/2, H/2$), and (f_x, f_y) are the focal lengths along the image axes. DepthAnythingv2 supplies a metric depth module with reasonable camera assumptions. These 3D samples are required to supervise primitive fitting. At test-time, we can directly optimize primitive parameters using the training losses since these 3D samples are available.

Primitive fitting details. We use the standard ResNet-18 encoder (accepting RGBD input) followed by 3 fully-connected layers to predict the parameters of the primitives. We train different networks for different primitive counts $K \in \{4, 6, 8, 10, 12, 24, 36, 48, 60, 72\}$, and allow the user to select their desired level of abstraction. Alternatively, the ensembling method of Vavilala et al. (2025a) can automatically select the appropriate number of primitives. Depending on the primitive count, the training process takes between 40-100 mins on a single A40 GPU, and inference (including generating the initial primitive prediction, refinement, and rendering) can take 1-3 seconds per image. While traditional primitive-fitting to RGB images fits cuboids Kluger et al. (2021), we find that polytopes with more faces and without symmetry constraints yield more accurate fits. Thus, we use $F = 12$ face polytopes. We do not use a Manhattan World loss or Segmentation loss; the former helped on NYUv2 Silberman et al. (2012) but not on in-the-wild LAION images and the latter showed an approximately neutral effect in the original paper Vavilala & Forsyth (2023).

Table 2: AbsRel depth error metrics for varying numbers of 3D primitives (12-face polytopes). Lower values indicate better depth map approximation quality. While theory would predict $\text{AbsRel} \rightarrow 0$ as $K \rightarrow \infty$ (e.g. one primitive per pixel), in practice we run into bias-variance problems fitting more than 60 primitives. Generating primitives is efficient (approx. 1-3 seconds per image on the GPU including finetuning and rendering) so it is feasible for the user to select from a few candidates based on the desired level of abstraction. No other primitive-conditioned image synthesis method offers variable abstraction.

Number of Parts (K)	AbsRel Error↓
4	0.0376
6	0.0330
8	0.0295
10	0.0282
12	0.0265
24	0.0223
36	0.0203
48	0.0202
60	0.0194
72	0.0195

A.2 HYPERPARAMETER SELECTION

There are a number of hyperparameters associated with our procedure, and we perform a grid search on a held-out validation set to find the best ones. When generating correspondence maps between point clouds, we let `max_distance= 0.005`. In our confidence map, we dilate low-confidence pixels with a score less than $\tau = 0.01$ by 9 pixels, which tells the image model to synthesize new

texture near primitive boundaries that are often uncertain. We set (t_{start}, t_{end}) to (1000, 500) by default, though t_{end} can be tuned per test image by the user. Applying the hint for all time steps can reduce blending quality near primitive boundaries; not applying the hint for enough time steps could weaken texture consistency. Allowing some time steps to not follow the hint enables desirable super resolution behavior e.g. when bringing a primitive closer to the camera. The supplementary contains detailed algorithms for creating the hint and confidence mask.

Inpainting the hint image After warping the source image to the new view, we find it helpful to inpaint low-confidence regions of the hint $\mathbf{x}_{hint}$ before supplying it to the image model. We considered several possibilities, including `cv2_telea` and `cv2_ns` from the OpenCV package, as well as simply leaving them as black pixels. We find that Voronoi inpainting, a variation of nearest neighbor inpainting, works well. The `voronoi_inpainting` function performs image inpainting by filling in regions of low confidence in a hint image using colors from nearby high-confidence pixels, based on a Voronoi diagram approach.

Given a hint image I of shape $[H, W, 3]$ and a confidence mask C of shape $[H, W]$ (after resizing if necessary), we identify valid pixels where the confidence satisfies $C_{i,j} \geq \tau$, with τ being the threshold (default 0.01). For each pixel (i, j) in the image, we assign the color of the nearest valid pixel (k, l) , determined by Euclidean distance, effectively performing nearest-neighbor interpolation. Mathematically, the inpainted image I' is defined as:

$$I'_{i,j} = I_{k,l} \quad \text{where} \quad (k, l) = \arg \min_{(m,n) \in V} \sqrt{(i-m)^2 + (j-n)^2},$$

and $V = \{(m, n) \mid C_{m,n} \geq \tau\}$ represents the set of high-confidence pixel coordinates. This process leverages a KD-tree for efficient nearest-neighbor searches, ensuring that each pixel adopts the color of the closest reliable pixel, thus preserving local color consistency in the inpainted result.

For **FLUX image generation** we begin with the default settings from the diffusers FLUX controlled inpainting pipeline¹. We set the `strength` parameter (controlling starting noise strength) to 1.0 and `guidance` to 10. We use 30 `num_steps` for denoising. In comparative evaluation, we use the default settings from the authors.

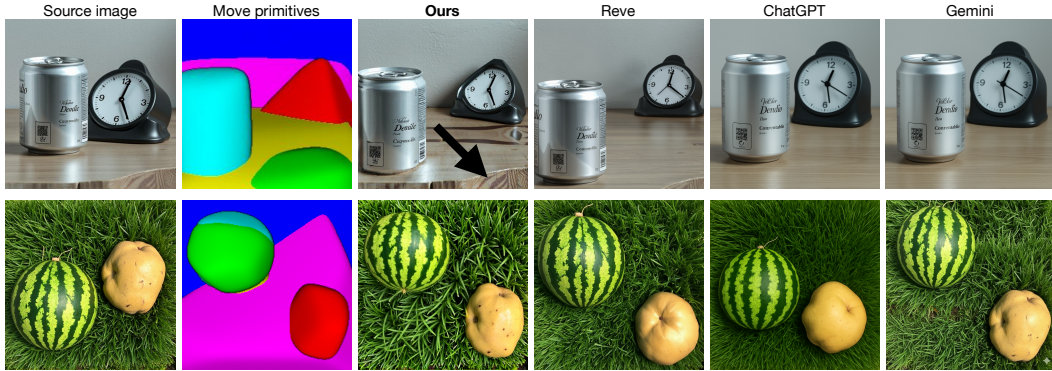


Figure 9: **Evaluation with production systems.** **First row, Left:** Source image. The next two columns show our primitive edits and the synthesized result. The arrow indicates a texture that our method reproduces faithfully, but others do not. The **fourth column** shows Reve (<https://app.reve.com/>), a commercial image generation system. We can prompt their model with 2D boxes to reposition objects, but we must manually estimate their size to take into account 3D perspective effects. With our 3D primitives, maintaining object scale is free. ChatGPT and Gemini do not have interaction mechanisms outside of text prompts and struggle to precisely move objects. Additionally, all 3 production methods added a “seconds hand” to the clock that wasn’t in the original. Those methods were also unable to generate precise camera moves that we can in this work. The **second row** shows another example. Our method can precisely move objects while maintaining texture. Reve changed the orientation of the potato. ChatGPT was unable to move the objects where requested (we tried variations of “move the watermelon to the top left, move the potato to the bottom right”). Gemini succeeded in this example.

¹https://huggingface.co/docs/diffusers/en/api/pipelines/control_flux_inpaint

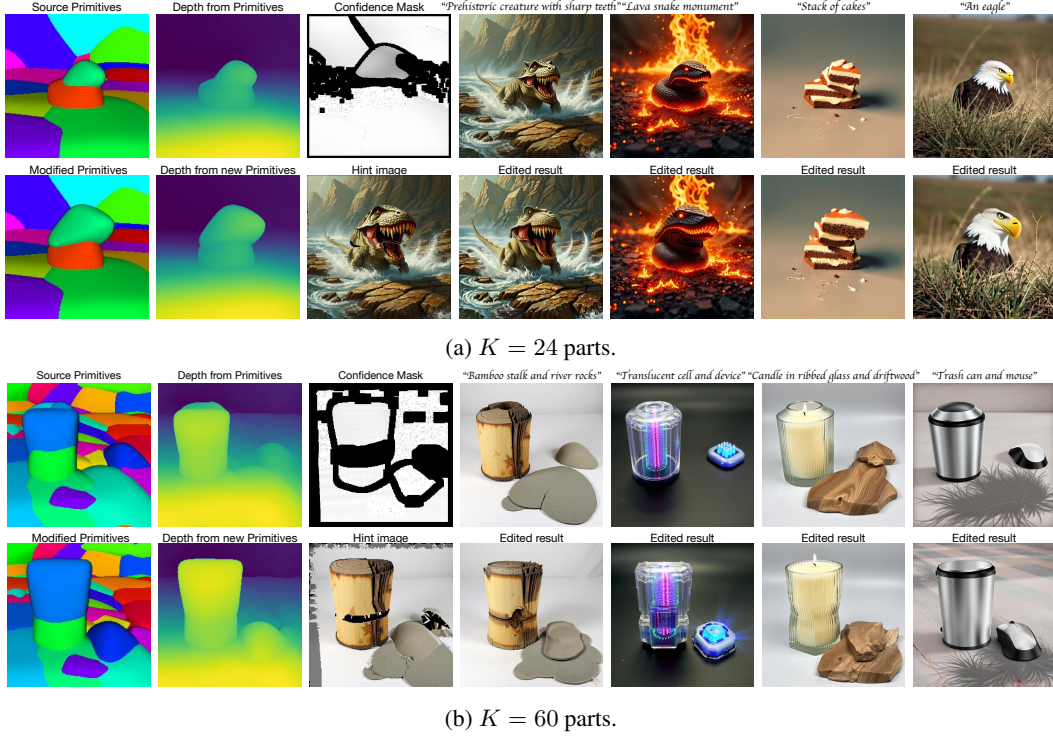


Figure 10: **Applying the same primitive edit for different text prompts at fine scale** ($K \in \{24, 60\}$ parts). Observe in the first two rows how all synthesized images respect the enlarged green primitive, while background texture is preserved. In the bottom two rows, we compose **several edits** using a large number of primitives ($K = 60$), enabling fine-scaled edits. We scale up the light blue primitive while scaling down the light green primitive on the left-hand side. We then translate the dark blue primitive on the right-hand side towards the bottom center of the image. We also slightly translate the camera upward. Observe how in the subsequent columns, the edited result respects the geometry specified by the primitives while following the high-level texture of the source image. However, notice how composing four edits challenges our procedure, as the texture preservation isn't as tight. For example, in the final column, a tiled pattern appears on the floor that wasn't in the source.

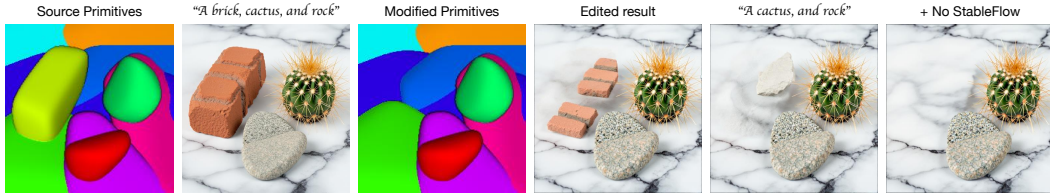


Figure 11: **Primitive edits can conflict with the text prompt.** Some geometric edits require changing the text prompt, for example, when removing an object. The **fourth column** mentions brick in the text prompt, but that primitive was removed, resulting in brick pieces in the inpainted region. In the **fifth column**, we remove the brick from the text prompt, which removes the brick pieces but it still leaves behind a white stone. In the **final column**, we use our texture hints but without StableFlow, getting a clean surface. The StableFlow key-value sharing approach placed brick and stone textures where we didn't want them. We conclude that our texture hints are critical, but combining them with StableFlow [Avrahami et al. \(2025\)](#) key-value sharing can help in some cases, hurt in others.



Figure 12: Our model is compatible with most depth-image synthesizers. While a pretrained FLUX works out of the box, LoRA weights on top of the base FLUX model are available (FLUX.1 Depth [dev] LoRA), exposing a new $lora_weight$ parameter (scaling the activations of the LoRA layers). This is intriguing in the context of our primitives, because they can either be used to coarsely model scene geometry (e.g. $lora_weight$ near 0.8, **second last column**), leaving details to the image synthesizer, or they can tightly control the result when $lora_weight$ is close to 1 (**final column**).

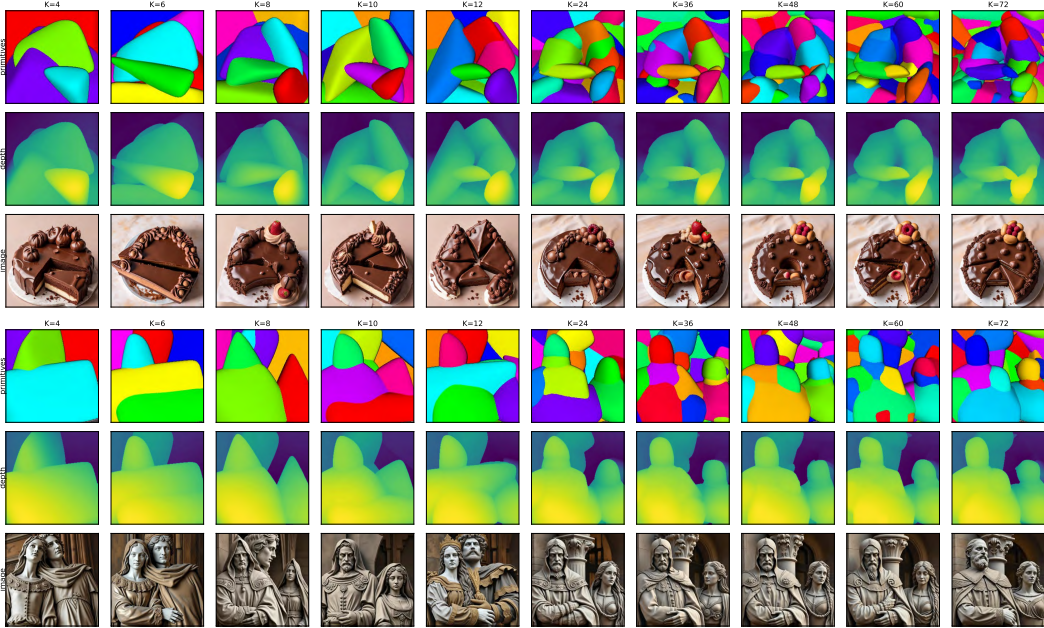


Figure 13: Given the same depth map, we extract primitives at variable resolution (from 4-72 parts). We show the depth maps in each second row, and synthesized result in each 3rd row. Observe how no matter the resolution, the FLUX-LoRA model (we use $lora_weight = 0.8$) gives an image that follows the primitive conditioning. We conclude that a wide array of primitive densities is tolerable to depth-to-image models, enabling meaningful artistic edits.

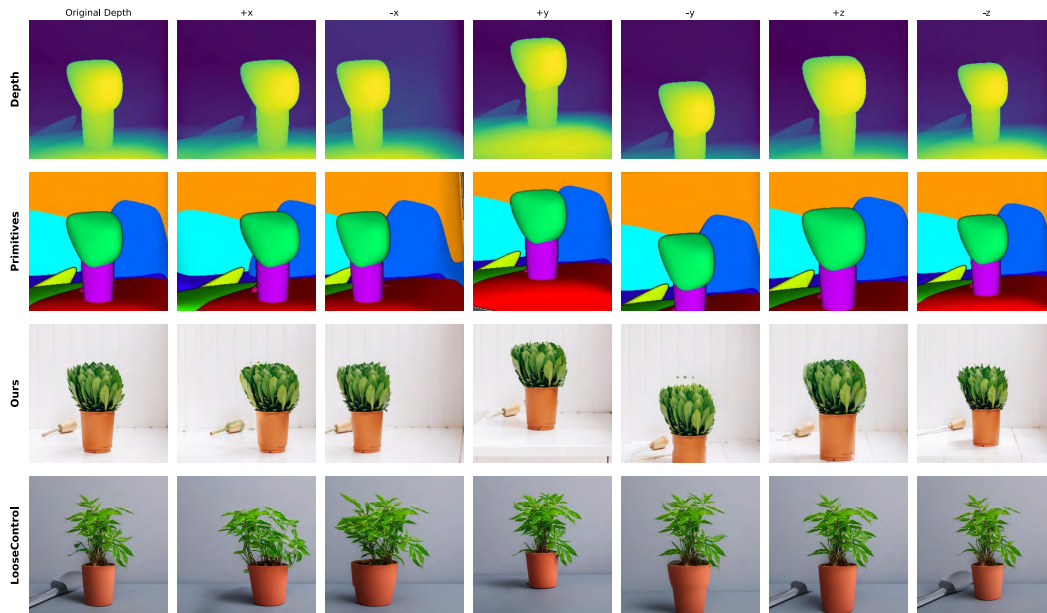


Figure 14: Additional move camera evaluations. Our method can simultaneously adhere to source texture and requested primitives.

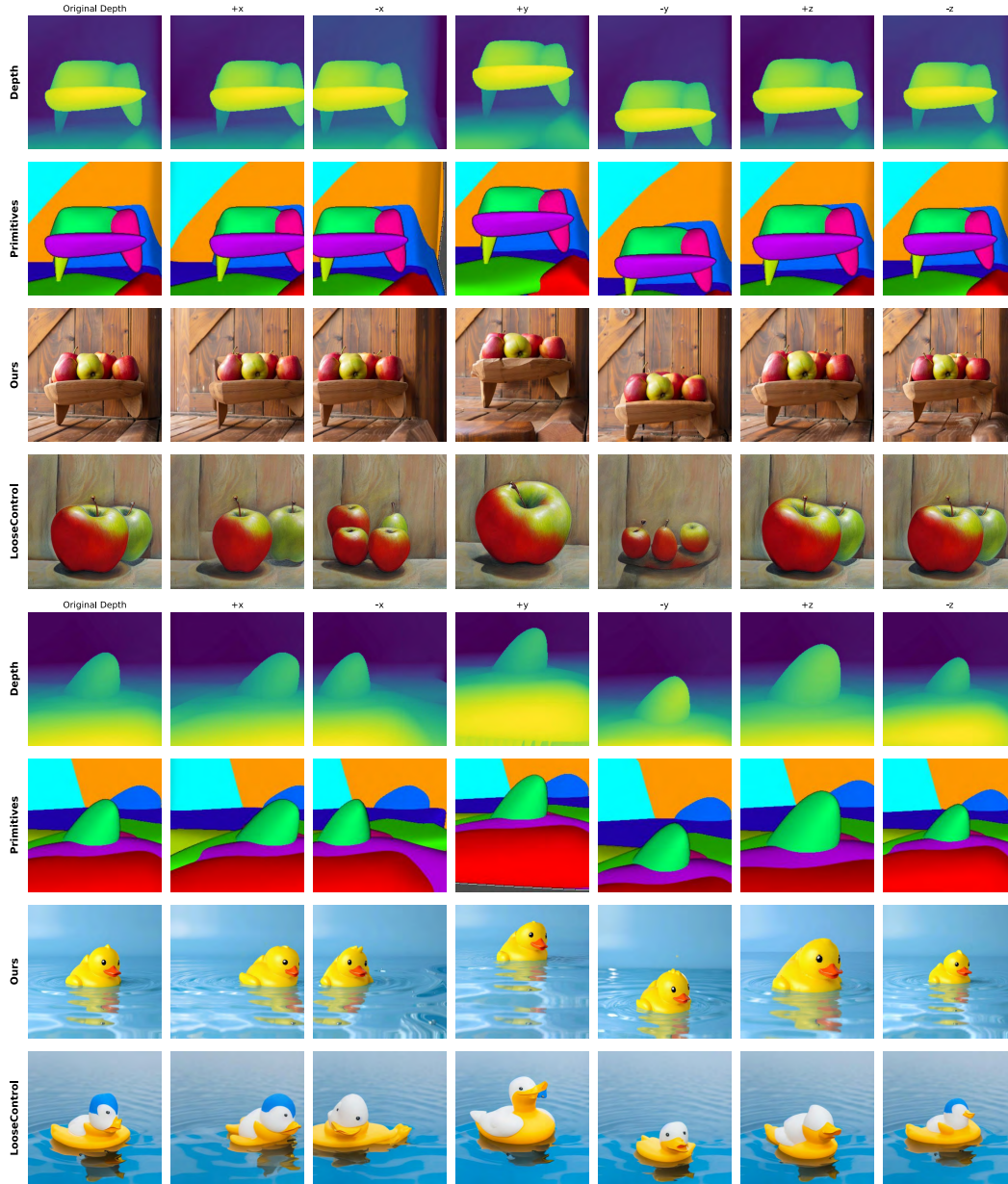


Figure 15: Additional move camera evaluations. Generative Blocks World can simultaneously adhere to source texture and requested primitives.

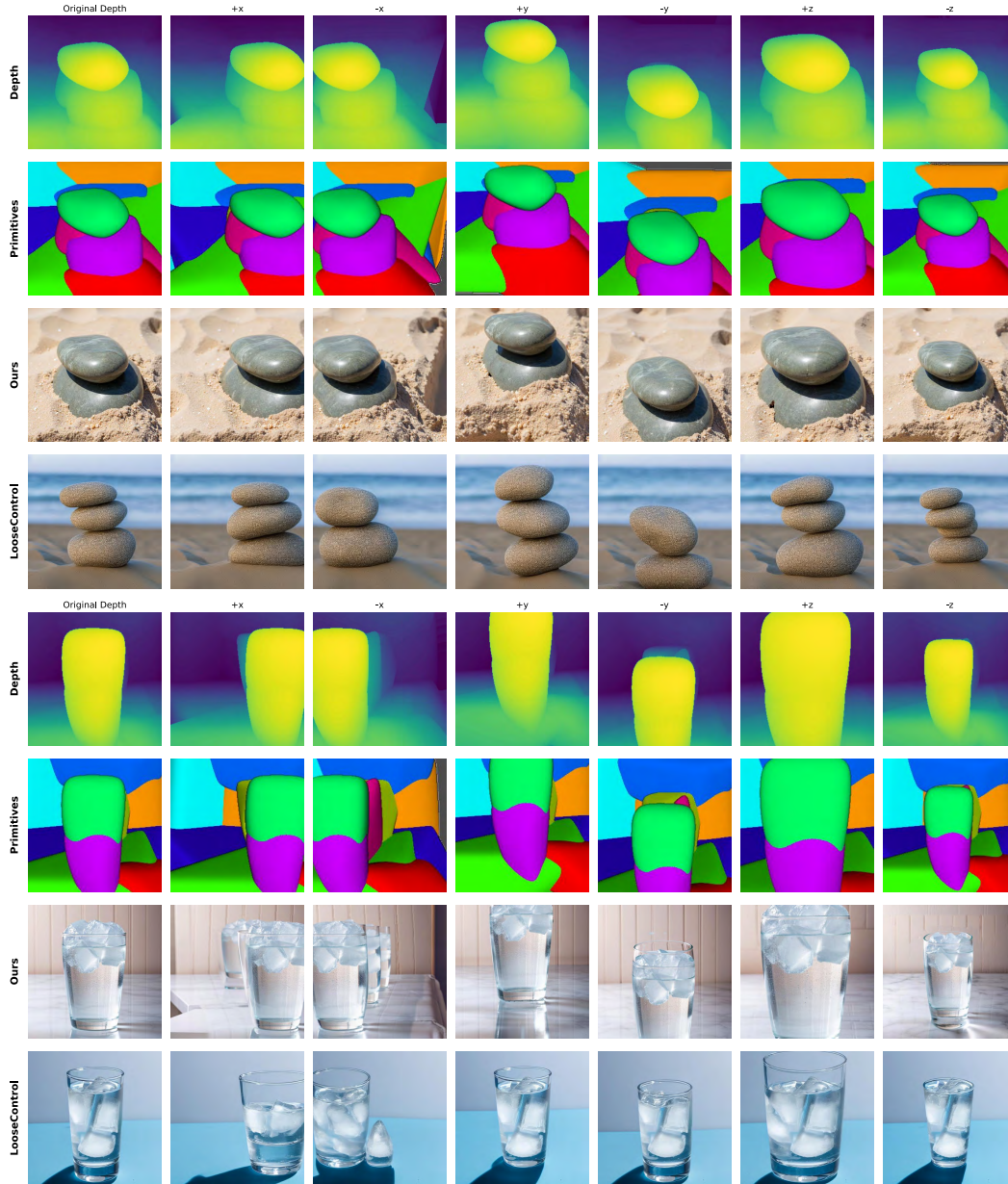


Figure 16: Additional move camera evaluations. Our method can simultaneously adhere to source texture and requested primitives.



Figure 17: We repeat the analysis of StableFlow [Avrahami et al. \(2025\)](#), which applies U-Net based key-value transfer of older-generation Diffusion models to newer Diffusion Transformers. Specifically, their work analyzes FLUX.1 [dev]; given that our work uses depth maps to communicate geometric information to our image generation model, we analyze Vital Layers in FLUX.1 Depth [dev] and FLUX.1 Depth [dev] LoRA, finding the top 5 multimodal and single modal layers to be essentially identical. We try using the vital layers we identified for texture transfer, finding this method to be inadequate (see Fig. 6).

Algorithm 1: Point Cloud Correspondence Generation**Input:** $\mathcal{P}_1, \mathcal{P}_2$: point clouds; $\mathcal{M}_1, \mathcal{M}_2$: convex maps; $\mathcal{T}$: primitive transforms; $\mathcal{C}$: centers; $d_{\max} = 0.005$: max distance threshold**Output:** $\mathcal{R}$: correspondence map; $\mathcal{W}$: confidence map**Function** ApplyTransform($\mathbf{p}, \mathbf{c}, \mathbf{T}$):

```

 $\mathbf{p}' \leftarrow \mathbf{p} - \mathbf{c};$  // Center the point
if  $\mathbf{T}$  contains translation then
   $\mathbf{p}' \leftarrow \mathbf{p}' - \mathbf{T}_{\text{trans}};$ 
end
if  $\mathbf{T}$  contains rotation angle  $\theta$  then
   $c, s \leftarrow \cos(-\theta), \sin(-\theta);$ 
   $x', z' \leftarrow x' \cdot c - z' \cdot s, x' \cdot s + z' \cdot c;$  // Y-axis rotation
end
if  $\mathbf{T}$  contains scaling factor  $scale$  then
   $\mathbf{p}' \leftarrow \mathbf{p}' / scale;$ 
end
return  $\mathbf{p}' + \mathbf{c};$ 

```

 $\mathcal{R} \leftarrow \mathbf{0}_{H \times W \times 2};$ // Initialize correspondence map $\mathcal{W} \leftarrow \mathbf{0}_{H \times W};$ // Initialize confidence map**for** $p \in \text{unique}(\mathcal{M}_1)$ **do**

```

if  $p < 0$  or  $p \geq |\mathcal{C}|$  or  $p \notin \mathcal{M}_1$  or  $p \notin \mathcal{M}_2$  then
  continue;
end

```

```

 $\mathcal{I}_1 \leftarrow \{(y, x) : \mathcal{M}_1[y, x] = p\};$  // Pixel indices for primitive  $p$  in map 1

```

```

 $\mathcal{I}_2 \leftarrow \{(y, x) : \mathcal{M}_2[y, x] = p\};$  // Pixel indices for primitive  $p$  in map 2

```

```

 $\mathcal{Q}_1 \leftarrow \{\mathcal{P}_1[y, x] : (y, x) \in \mathcal{I}_1\};$  // 3D points for primitive  $p$ 

```

for $(y_2, x_2) \in \mathcal{I}_2$ **do**

```

 $\mathbf{q} \leftarrow \mathcal{P}_2[y_2, x_2];$  // Query point from second cloud

```

if $p \in \mathcal{T}$ **then**

```

   $\mathbf{q} \leftarrow \text{ApplyTransform}(\mathbf{q}, \mathcal{C}[p], \mathcal{T}[p]);$  // Apply transformation

```

end

```

 $\mathbf{d} \leftarrow \|\mathcal{Q}_1 - \mathbf{q}\|_2;$  // Compute distances to all points

```

```

 $i^* \leftarrow \arg \min_i \mathbf{d}[i];$  // Find nearest neighbor

```

 $d_{\min} \leftarrow \mathbf{d}[i^*];$ **if** $d_{\min} \leq d_{\max}$ **then**

```

   $(y_1^*, x_1^*) \leftarrow \mathcal{I}_1[i^*];$  // Get corresponding pixel coordinates

```

```

   $\mathcal{R}[y_2, x_2] \leftarrow [x_1^*, y_1^*];$ 

```

```

   $\mathcal{W}[y_2, x_2] \leftarrow 1 - \min(d_{\min}/d_{\max}, 1);$  // Confidence score

```

end**end****end****return** $\mathcal{R}, \mathcal{W};$

Algorithm 2: Hint Generation from Correspondence Maps**Input:** $\mathbf{I}_{\text{src}} \in \mathbb{R}^{C \times H_s \times W_s}$: source image; $\mathcal{R} \in \mathbb{R}^{H_r \times W_r \times 2}$: correspondence map; $\mathcal{W} \in \mathbb{R}^{H_r \times W_r}$: confidence map; $\mathcal{M}_{\text{hit}} \in \{0, 1\}^{H_r \times W_r}$: hit mask**Output:** $\mathcal{H} \in \mathbb{R}^{C \times H_s \times W_s}$: generated hint image**Function** BilinearSample($\mathbf{I}$, y , x):

```

     $C, H, W \leftarrow \text{shape}(\mathbf{I});$ 
     $x \leftarrow \text{clip}(x, 0, W - 1.001), y \leftarrow \text{clip}(y, 0, H - 1.001);$ 

     $x_0, y_0 \leftarrow \lfloor x \rfloor, \lfloor y \rfloor;$  // Floor coordinates
     $x_1, y_1 \leftarrow \min(x_0 + 1, W - 1), \min(y_0 + 1, H - 1);$ 
     $w_x, w_y \leftarrow x - x_0, y - y_0;$  // Interpolation weights

     $\mathbf{v}_{\text{top}} \leftarrow \mathbf{I}[:, y_0, x_0] \cdot (1 - w_x) + \mathbf{I}[:, y_0, x_1] \cdot w_x;$ 
     $\mathbf{v}_{\text{bot}} \leftarrow \mathbf{I}[:, y_1, x_0] \cdot (1 - w_x) + \mathbf{I}[:, y_1, x_1] \cdot w_x;$ 
    return  $\mathbf{v}_{\text{top}} \cdot (1 - w_y) + \mathbf{v}_{\text{bot}} \cdot w_y;$ 

 $\lambda_h \leftarrow H_s / H_r, \lambda_w \leftarrow W_s / W_r;$  // Scale factors
 $\mathcal{H} \leftarrow \mathbf{0}_{C \times H_s \times W_s};$  // Initialize hint image

for  $y \in [0, H_r)$  do
    for  $x \in [0, W_r)$  do
        if  $\mathcal{M}_{\text{hit}}[y, x] = 1$  then
            continue; // Skip hit pixels
        end

         $(x_c, y_c) \leftarrow \mathcal{R}[y, x];$  // Get correspondence
         $w \leftarrow \mathcal{W}[y, x];$  // Get confidence
        if  $w < 0.1$  then
            continue; // Skip low-confidence correspondences
        end

         $y_{\text{src}} \leftarrow y_c \cdot \lambda_h, x_{\text{src}} \leftarrow x_c \cdot \lambda_w;$  // Scale to source resolution
         $y_{\text{start}} \leftarrow \lfloor y \cdot \lambda_h \rfloor, y_{\text{end}} \leftarrow \lfloor (y + 1) \cdot \lambda_h \rfloor;$ 
         $x_{\text{start}} \leftarrow \lfloor x \cdot \lambda_w \rfloor, x_{\text{end}} \leftarrow \lfloor (x + 1) \cdot \lambda_w \rfloor;$ 

        for  $y_s \in [y_{\text{start}}, y_{\text{end}})$  do
            for  $x_s \in [x_{\text{start}}, x_{\text{end}})$  do
                if  $y_s \notin [0, H_s)$  or  $x_s \notin [0, W_s)$  then
                    continue; // Boundary check
                end

                 $\alpha_y \leftarrow \frac{y_s - y_{\text{start}}}{\max(y_{\text{end}} - y_{\text{start}}, 1)};$  // Normalized offset
                 $\alpha_x \leftarrow \frac{x_s - x_{\text{start}}}{\max(x_{\text{end}} - x_{\text{start}}, 1)};$ 
                 $y_{\text{sample}} \leftarrow y_{\text{src}} + \alpha_y \cdot \lambda_h;$ 
                 $x_{\text{sample}} \leftarrow x_{\text{src}} + \alpha_x \cdot \lambda_w;$ 
                 $\mathcal{H}[:, y_s, x_s] \leftarrow \text{BilinearSample}(\mathbf{I}_{\text{src}}, y_{\text{sample}}, x_{\text{sample}});$ 
            end
        end
    end
end

return  $\mathcal{H};$ 

```

A.3 RELATED WORK MATRIX

Table 3: Related work comparison summary. While all methods can move objects, ours is the only one that uses 3D primitives (at varying density) in a training-free pipeline, while also supporting scene-level camera moves. Previous drag-based works use 2D arrows in pixel space to prompt the edit (combined with spatial masks); in contrast, our approach is prompted via selecting and moving 3D primitives. Loose Control is the closest approach to ours that uses 3D boxes, but it requires training and cannot support detailed, variable-density primitives. Here, Variable LoD refers to support for variable primitive count, to enable both coarse and fine edits. Note that while our image generation process is training-free, our primitives are learned.

Method	Interaction	Training-free	Move Objects	Variable LoD	Camera Move
Diff. Self-Guid. Epstein et al. (2023)	2D guidance	✓	✓	✗	✗
Diffusion Handles Pandey et al. (2024)	3D handles	✓	✓	✗	✗
Edit. Image Elements Mu et al. (2024)	2D elements	✗	✓	✗	✗
DragDiffusion Shi et al. (2024)	2D points	✗	✓	✗	✗
GoodDrag Zhang et al. (2025)	2D points	✓	✓	✗	✗
FlowDrag Koo et al. (2025)	2D points	✓	✓	✗	✗
DragIn3D Guang et al. (2025)	3D points	✗	✓	✓	✗
LooseControl Bhat et al. (2024)	3D boxes	✗	✓	✗	✓
Ours	3D primitives	✓	✓	✓	✓

B ADDITIONAL EVALUATION

Table 4: **Additional ablations evaluating geometric consistency and texture consistency on the open DragBench dataset.** Similar to Table 1, we evaluate novel-view consistency. $\text{AbsRel}_{\text{src}}$ and $\text{AbsRel}_{\text{dst}}$ measure depth consistency for source and destination views respectively. Best values are shown in **bold**. $\uparrow$ indicates higher is better; $\downarrow$ indicates lower is better. This experiment uses $K = 10$ parts, and on average 78.1% of pixels were visible in both views. Notice how our texture-hint mechanism is essential for good results; StableFlow Avrahami et al. (2025) is occasionally helpful. See Fig. 6 for qualitative examples.

DragBench dataset: <https://huggingface.co/datasets/LuJingyi/DragBench>

Method	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$	$\text{AbsRel}_{\text{src}} \downarrow$	$\text{AbsRel}_{\text{dst}} \downarrow$
LooseControl Bhat et al. (2024)	7.36	0.501	0.335	0.148	0.151
Ours FLUX (No Hint)	8.42	0.465	0.373	0.099	0.107
Ours FLUX + StableFlow	9.11	0.477	0.331	0.099	0.107
Ours FLUX + Hint	14.24	0.658	0.165	0.099	0.102
Ours FLUX + Hint + StableFlow	14.46	0.654	0.169	0.099	0.113

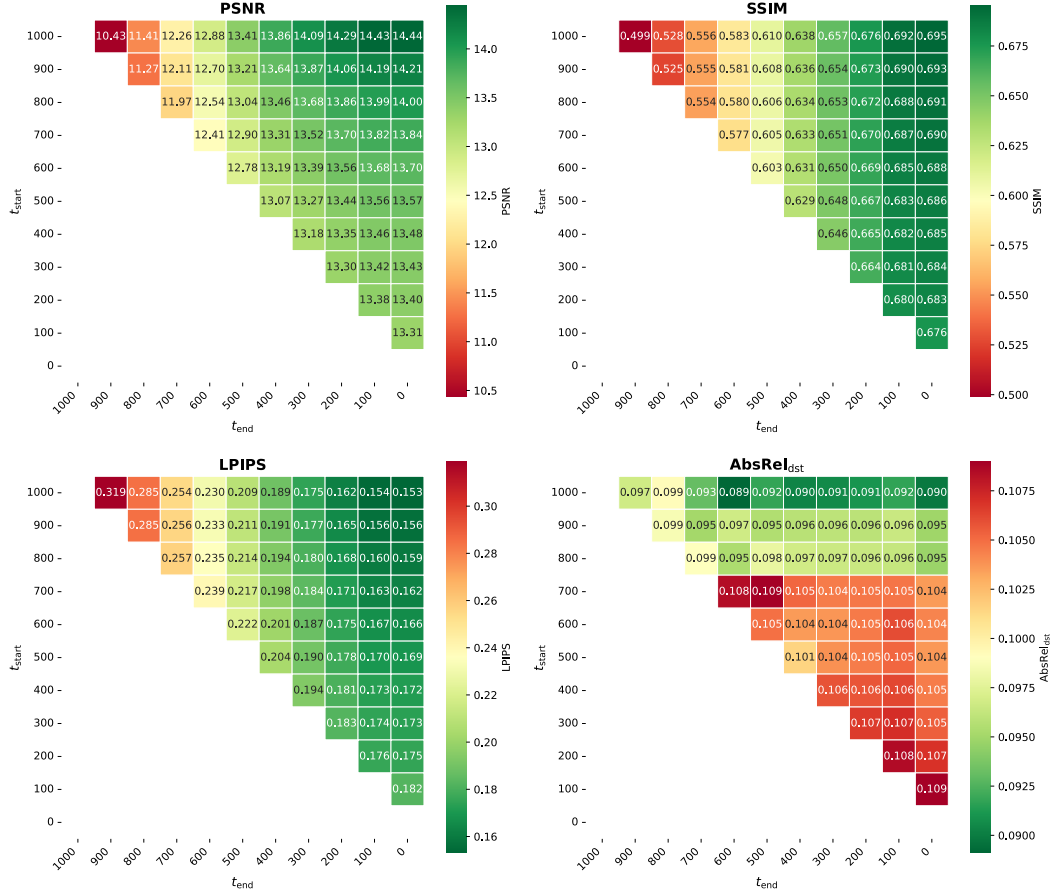


Figure 18: **Grid search over hint-timestep parameters t_{start} and t_{end} .** The hint image steers the generation only within these timesteps, and only for pixels in the confidence mask. This evaluation suggests that $t_{\text{start}} \approx 1000$ and $t_{\text{end}} \approx 0$ obtain the best overall texture-consistency metrics in regions shared between source and destination views (as expected, since the hint is applied at every step). However, we observed unwanted artifacts when applying the hint throughout the full trajectory, as warped pixels fail to harmonize with inpainted regions. Through qualitative inspection (see Fig. 19), we find $t_{\text{start}} = 1000$, $t_{\text{end}} = 300$ works well (and is user-tunable at test time); this quantitative evaluation confirms we sacrifice little in texture consistency or depth accuracy.

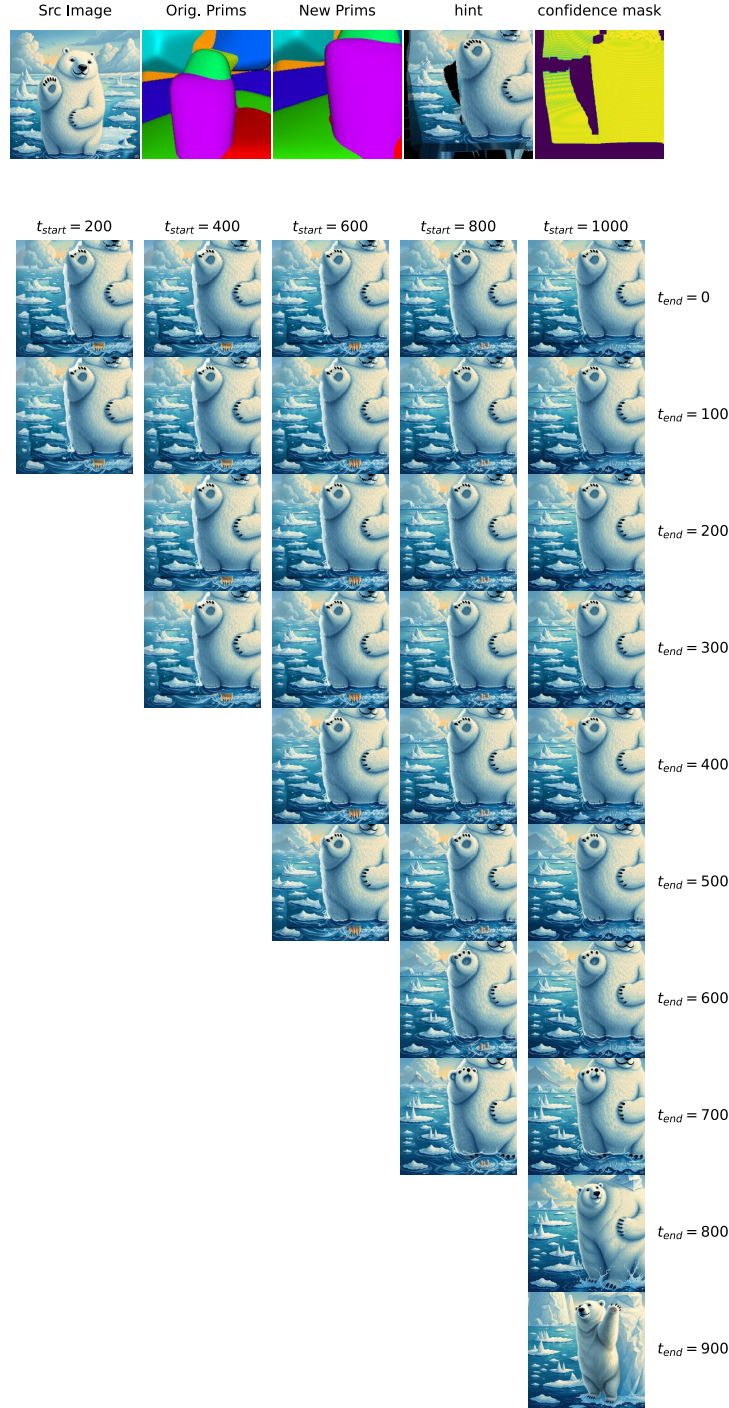


Figure 19: **Qualitative ablation over hint-timestep parameters t_{start} and t_{end} .** When $t \in [t_{\text{start}}, t_{\text{end}}]$, the hint image influences synthesis. The hint image is essential for steering the model toward texture-consistent solutions, as shown in Table 4. However, it introduces warping and aliasing artifacts that the diffusion process can correct. For example, when $t_{\text{end}} < 200$, aliasing appears near the bear’s mouth. If t_{end} is too large (> 700), the model diverges from the hint (e.g., the bear’s head pose changes). Similarly, if the hint begins too late ($t_{\text{start}} < 800$), unwanted content is synthesized (e.g., the orange paw in the ocean). Both quantitative metrics and visual results favor $(t_{\text{start}}, t_{\text{end}}) \approx (1000, 300)$.

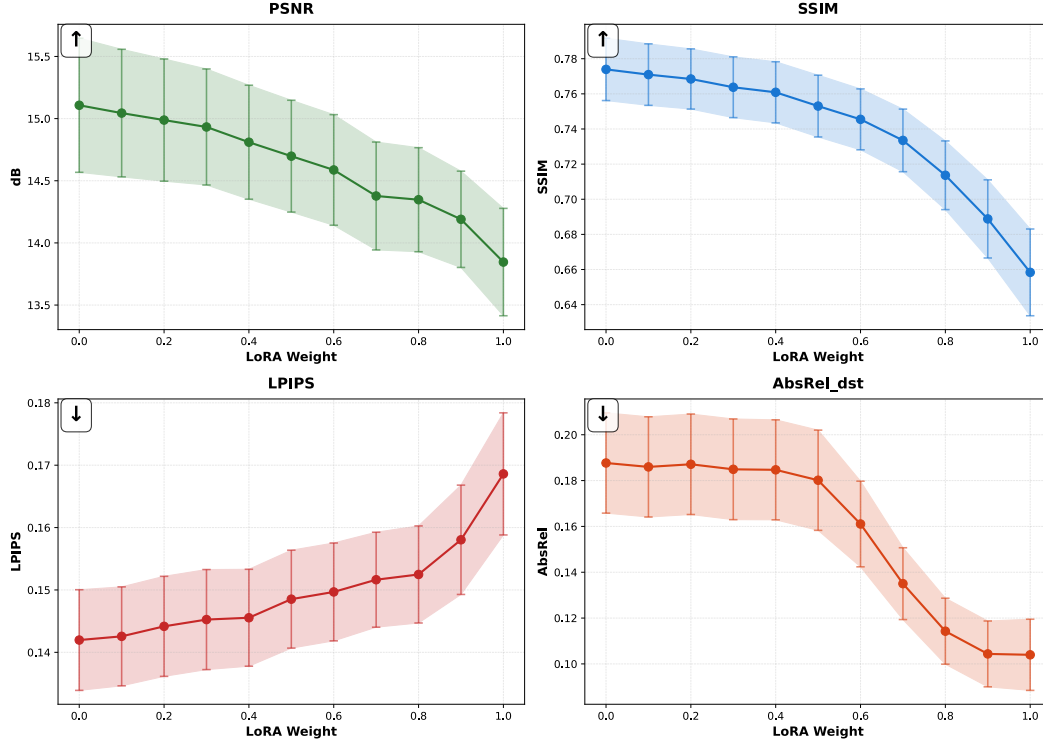
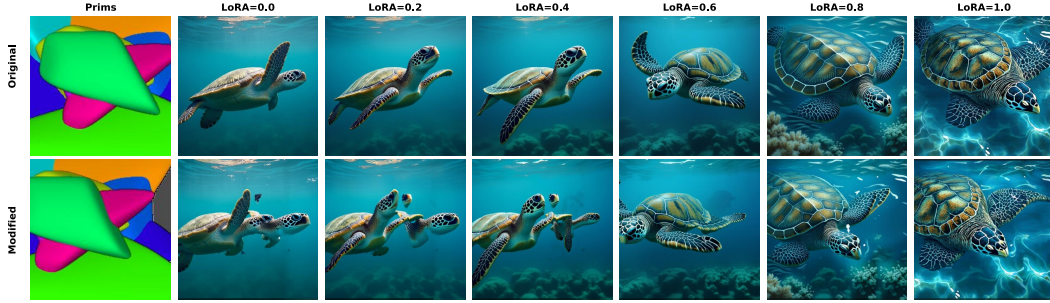
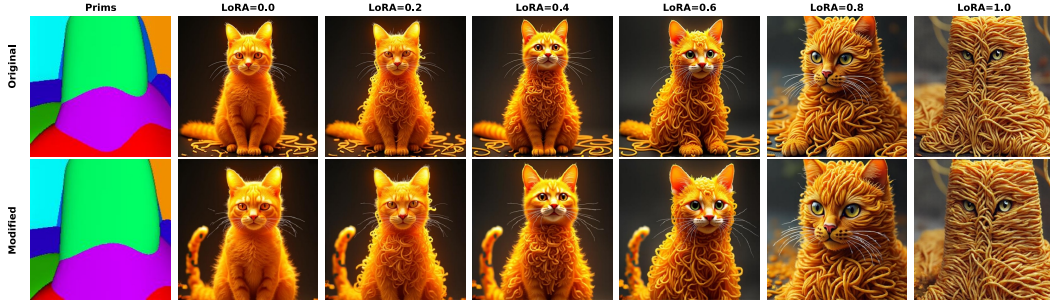


Figure 20: **Ablation of LoRA depth-conditioning weight on texture and geometry.** Error bars show 95% confidence intervals. As the LoRA weight increases, texture similarity to the hint image steadily degrades across LPIPS, SSIM, and PSNR, revealing the inherent difficulty of achieving strong texture consistency and strong depth consistency simultaneously. Increasing LoRA weight strengthens depth-specific pathways in the diffusion model, improving geometric faithfulness—evidenced by the monotonic reduction in AbsRel up to weight = 0.9—but at the cost of reduced adherence to the hint image. This highlights an intrinsic tradeoff between geometric accuracy and texture preservation, and the LoRA weight provides a simple, interpretable knob to tune this balance. See Fig. 21 for qualitative examples.



"Turtle is swimming"



"Cat made of spaghetti"



"Girl wearing pearl earrings"

Figure 21: **Ablating the effect of $lora_{weight}$ on image editing.** The **first row** in each set shows source primitives followed by the source (generated) image for $lora_{weight} \in [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]$. As the weight increases, the model more strongly adheres to the depth conditioning. The **second row** shows edited primitives (first column) followed by the edited results (using our texture-hint mechanism). With weak depth adherence (low LoRA weights), textures do not follow the primitive geometry. Stronger weights (0.8, 1.0) simultaneously preserve both primitives and texture hints.

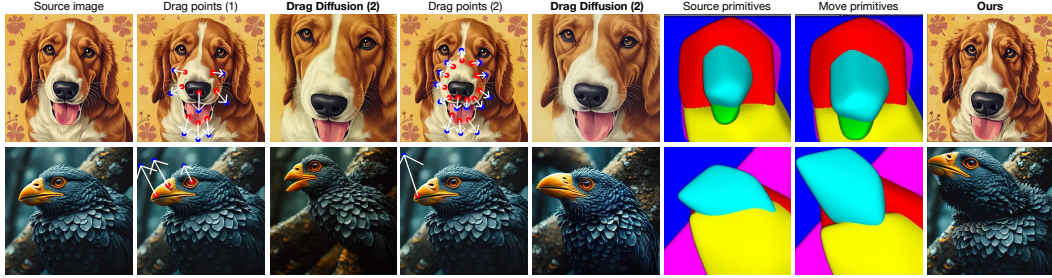


Figure 22: In addition to Fig. 4 of the main paper, we provide more comparison with DragDiffusion Shi et al. (2024). The first column shows the source image. The next two columns show the prompt and result from DragDiffusion. For 3D-aware edits such as enlarging a dog’s nose or moving a bird’s head, DragDiffusion’s arrow-based interaction makes precise control difficult; users must effectively draw dense optical-flow vectors. The next two columns show another attempt. DragDiffusion fails to consistently move the bird’s beak to the intended location and alters the texture undesirably. Notice DragDiffusion changes the beak size in both attempts; ours keeps it consistent. It also removes background patterns in the dog example and unintentionally changes the head shape. The final three columns show our approach, which closely follows the primitives while preserving source textures. Our primitive-based interface is more intuitive and produces reasonable part segmentations as a by-product (e.g., red primitive for the dog’s head, blue for the snout, green for the tongue).

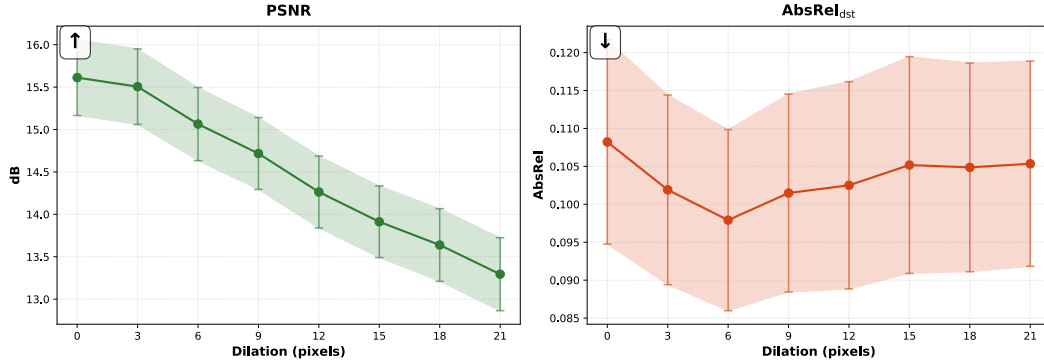


Figure 23: **Quantitative ablation on confidence-mask dilation.** As discussed in Sec. A.2, dilating the confidence mask by a few pixels reduces edge artifacts and improves harmonization of moved objects with the scene. While dilation slightly hurts texture-consistency metrics, it can improve depth accuracy and reduce visible artifacts. Dilating removes warped pixels near primitive boundaries, which often contain misaligned texture due to geometric approximation error. In future work, integrating recent segmentation methods Kirillov et al. (2023) may reduce the need for dilation while maximizing useful hint information. See Fig. 24 for qualitative examples.



Figure 24: **Qualitative ablation on confidence-mask dilation.** The first column in each pair shows the primitives and their edits; subsequent columns show source and edited images across dilation parameters. Dilation expands the unconfident regions of the mask, letting the diffusion model freely synthesize more pixels instead of following the hint. In each pair, the black arrow (bottom left) highlights edge artifacts near hint boundaries. Allowing moderate dilation (typically 9–12 pixels) is sufficient to eliminate these artifacts.

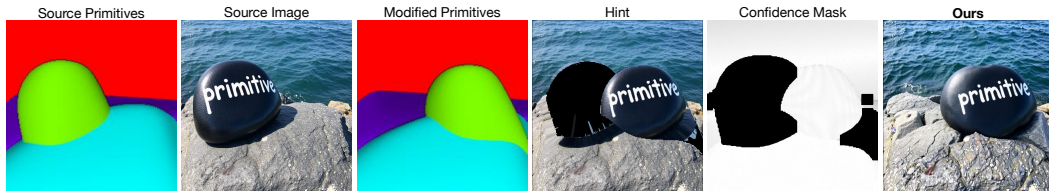


Figure 25: **Move object with hint and confidence mask shown.**