GENERATIVE BLOCKS WORLD: MOVING THINGS AROUND IN PICTURES

Anonymous authors
Paper under double-blind review

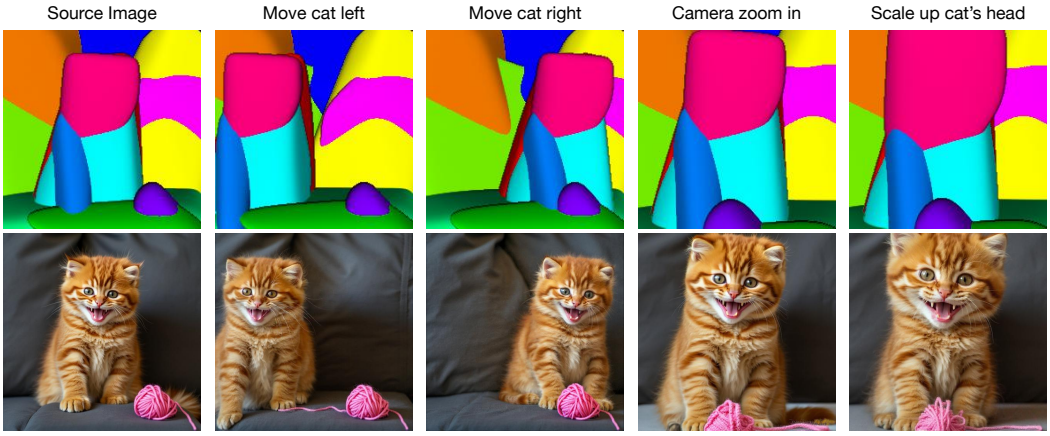


Figure 1: **Generative Blocks World.** Given an input image (bottom left), we extract a set of 3D convex primitives (top left) that provide an editable and controllable representation of the scene. These primitives are used to generate new images that respect geometry, texture, and the text prompt. The first column shows the original input and its primitive decomposition. Subsequent columns show sequential edits: translating the cat to the left (second column), translating it to the right (third column), moving the yarn in front of the cat and shifting the camera toward the scene center (fourth column), and scaling up the cat's head (burgundy primitive; fifth column). Our method enables 3D-aware semantic image editing through intuitive manipulation of these learned primitives.

ABSTRACT

We describe Generative Blocks World to interact with the scene of a generated image by manipulating simple geometric abstractions. Our method represents scenes as assemblies of convex 3D primitives, and the same scene can be represented by different numbers of primitives, allowing an editor to move either whole structures or small details. Once the scene geometry has been edited, the image is generated by a flow-based method, which is conditioned on depth and a texture hint. Our texture hint takes into account the modified 3D primitives, exceeding the texture-consistency provided by existing techniques. These texture hints (a) allow accurate object and camera moves and (b) preserve the identity of objects. Our experiments demonstrate that our approach outperforms prior works in visual fidelity, editability, and compositional generalization. Code will be released.

1 INTRODUCTION

There is a rich literature treating editing real and generated images using various image-centered interfaces like dragging features. Interaction paradigms that exploit explicit representations of 3D are much less common. This paper describes an image editor built on a full 3D interaction paradigm, using a representation that is both compact and accurate – a *generative blocks world*.

Any scene representation that supports a **camera move** has some form of 3D representation. An explicit 3D representation helps, Fig 1. Explicit 3D representations have other important advantages. First, they offer **shape constancy**. When an object is moved across a perspective view, it is seen from a new aspect because the location of the focal point moves in object coordinates. This means

that (a) the shape of the object may change, with a change that depends on the field of view of the camera and the shape of the object (e.g. alien head in Fig 7) and (b) some surface markings will become visible or invisible (e.g. bar code on soda can, Fig 3). When an object is moved toward or away from the camera, its image should expand or shrink (e.g. cat, Fig 1). If an editor does not preserve these properties correctly, the viewer may conclude that the shape or size of the object has changed. A properly constructed 3D representation will prevent this. Second, they offer **contact consistency**. A user who moves (say) a tin on a table generally expects the tin to remain in contact with the table. An explicit 3D representation allows the user to manage whether it does or not (e.g. dog in Fig 7; soda can, Fig 3). Third, they offer **shape completion**. Objects have backs that are not visible, but may have an effect when another object is moved in a scene. An explicit 3D representation can capture this effect.

It has been hard to build a 3D representation that: (a) represents the scene accurately enough that edited images are realistic and (b) is compact enough to support interactions. This paper uses modern fitting methods to represent scenes as small assemblies of meaningful parts or primitives (cf. *Blocks World* Roberts (1963) or *geons* Biederman (1987)). We call our method **Generative Blocks World**, *though our learned primitives are richer than cuboids*. Our method yields assemblies by decomposing an input image into a sparse set of convex polytopes (Vavilala et al., 2025a) that approximate the scene's depth map well enough to enable view-consistent texture projection. Further, our primitives respect object boundaries rather well. A user can reach into the primitive representation and move a primitive, with predictable results. Our highly reduced scene representations yield *hints* as to the appearance of the final image. These hints, together with the primitive depth map, are inputs to an off-the-shelf depth conditioned image generator, which renders very accurate images.

Good primitive decompositions have very attractive properties. They are *selectable*: individual primitives can be intuitively selected and manipulated (Fig. 1). They are *object-linked*: a segmentation by primitives is close to a segmentation by objects, meaning an editor is often able to move an object or part by moving a primitive (Figs 1; 3; 4). They are *accurate*: the depth map from a properly constructed primitive representation can be very close to the original depth map (Fig 3.1), which means primitives can be used to build texture hints (Section 3.2) that support accurate camera moves (Figs 2; 5). They have *variable scale*: one can represent the same scene with different numbers of primitives, allowing an editor to adjust big or small effects (Figs 7; 10; 13).

Contributions:

- We describe a pipeline that fuses convex primitive abstractions with a SOTA flow-based generator to yield a natural 3D interaction paradigm for image editing. Our pipeline uses a natural texture-hint procedure that supports accurate camera moves and edits at the object-level, while preserving identity.
- We provide extensive evaluation demonstrating superior geometric control, texture retention, and edit flexibility relative to recent state-of-the-art baselines.

2 Related Work

Primitive Decomposition: Early vision and graphics pursued parsimonious part-based descriptions, from Roberts' *Blocks World* Roberts (1963) and Binford's generalized cylinders Binford (1971) to Biederman's geons Biederman (1987). Efforts to apply similar reasoning to real-world imagery have been periodically revisited Gupta et al. (2010); Monnier et al. (2023); Bhattad et al. (2025) from various contexts and applications. Modern neural models revive this idea: BSP-Net Chen et al. (2020), CSG-Net Sharma et al. (2018), and CVXNet Deng et al. (2020) represent shapes as unions of convex polytopes, while Neural Parts Tulsiani et al. (2017), SPD Zou et al. (2018), and subsequent works Liu et al. (2022) learn adaptive primitive sets. Recent systems extend from objects to scenes: Convex Decomposition of Indoor Scenes (CDIS) Vavilala & Forsyth (2023) and its ensembling/Boolean refinement Vavilala et al. (2025a) fit CVXNet-like polytopes to RGB-D images, using a hybrid strategy. CubeDiff Kalischek et al. (2025) fits panoramas inside cuboids. Our work leverages CDIS as the backbone, but (i) improves robustness to in-the-wild depth/pose noise and (ii) couples the primitives to a Rectified Flow (RF) renderer, enabling controllable synthesis rather than analysis alone.

Conditioned Image Synthesis: Layout-to-image translation was pioneered in GANs Isola et al. (2017); Zhu et al. (2017); Park et al. (2019) and is now dominated by diffusion models such as Stable Diffusion Rombach et al. (2022), ControlNet Zhang et al. (2023), and T2I-Adapter Mou et al. (2024). These models can compose multiple spatial controls (Vavilala et al., 2024), perform color edits (Vavilala et al., 2025b) and relight scenes Xing et al. (2025). We utilize a pretrained depth-conditional FLUX model, conditioning it on depth maps derived from our 3D primitives.

Point-Based Interactive Manipulation: Methods like DragGAN (Pan et al., 2023) and its diffusion-based successors (Shi et al., 2024; Mou et al., 2023; Cui et al., 2024; Pandey et al., 2024) offer intuitive 2D control by dragging handle points. Some approaches extend this to 3D using NeRFs for multi-view consistency Guang et al. (2025) or leverage self-guidance for layout control (Epstein et al., 2023). Our work differs fundamentally by operating on selectable and editable 3D primitives, not 2D points or lines. This enables multi-resolution control and allows for camera movement while handling perspective, occlusion, and texture.

Object-Level and Scene-Level Editing: Many recent works embed 3D priors for editing, though often focusing on single objects Gu et al. (2022); Wang et al. (2023); Poole et al. (2023); Tang et al. (2023); Cheng et al. (2025) or using language to guide transformations Michel et al. (2023). Our Generative Blocks World generalizes to complex edits not easily described by text. Another paradigm, seen in Image Sculpting Yenphraphai et al. (2024) and OMG3D Zhao et al. (2025), reconstructs an explicit 3D mesh for manipulation before re-rendering. While precise, these multi-stage pipelines can be bottlenecked by reconstruction quality. Our method provides a more streamlined approach by operating on abstract primitives, achieving strong geometric control without the complexity of direct mesh manipulation.

Primitive-Based Scene Authoring: LooseControl (Bhat et al., 2024) enables control via box-like primitives by fine-tuning a diffusion model with LoRA weights. This training is necessary to bridge the domain gap between its coarse primitive-based depth and standard depth maps (Yang et al., 2024). In contrast, our underlying primitive representation is accurate enough to require no fine-tuning. Furthermore, by abstracting objects into single, monolithic boxes, LooseControl is limited to holistic transformations and cannot perform part-level edits. Our method uses structured geometry, decomposing objects into multiple convex polytopes at variable levels of detail for more granular control. A more recent work, Build-A-Scene (Eldesokey & Wonka, 2025), uses a similar pipeline to LooseControl and thus inherits its limitations. Our approach differs by: (i) decomposing objects into multiple convex polytopes for finer control, (ii) supporting camera movement, and (iii) allowing novel scenes to be authored from scratch via primitive assembly.

Table 1: 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 and optimization-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	✓	✓	✓	✓

3 METHOD

Generative Blocks World generates realistic images conditioned on a parsimonious and editable geometric representation of a scene: a set of convex primitives. The process consists of four main stages

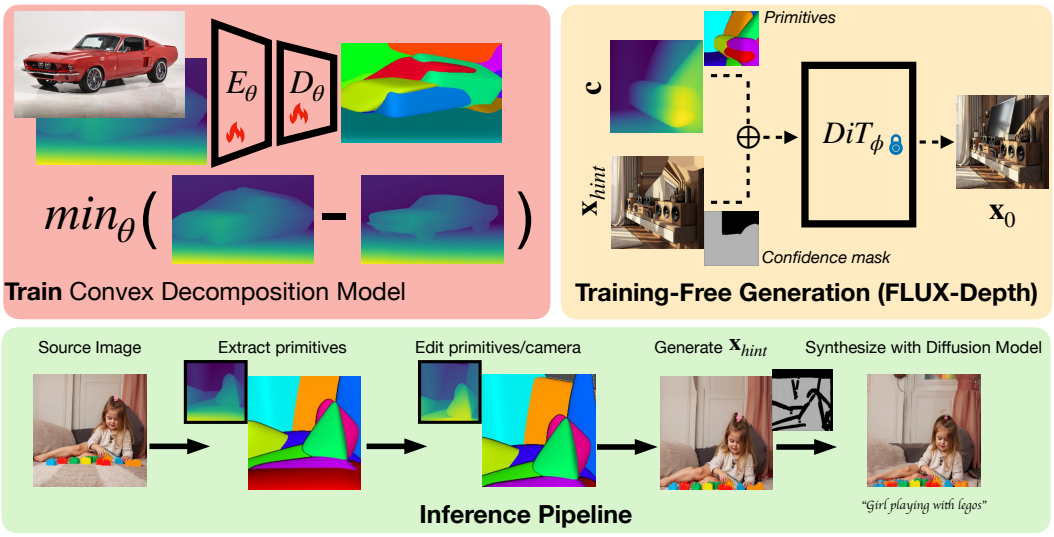


Figure 2: **Pipeline Overview. Top left:** We use convex decomposition models Vavilala et al. (2025a) to extract primitives from an input image at multiple scales. **Bottom:** Users can manipulate these primitives and the camera to define a new scene layout. We render the modified primitives into a depth map and generate a texture hint image. These serve as inputs to a pretrained depth-to-image model Labs (2024), which requires no fine-tuning (**Top right**). The generated image respects the modified geometry, preserves texture where possible, and remains aligned with the text prompt.

(Fig. 2): (i) primitive extraction from any image via convex decomposition (Sec. 3.1), (ii) generating an image conditioned on the primitives (and text prompt), (iii) user edits the primitives and/or camera, and (iv) generates a new image conditioned on the updated primitives, while preserving texture from the source image (Sec. 3.3). We describe each component in detail below.

3.1 Convex Decomposition for Primitive Extraction

Our primitive vocabulary is blended 3D convex polytopes as described in Deng et al. (2020). CVXnet represents the union of convex polytopes using indicator functions $O(x) \rightarrow [0, 1]$ that identify whether a query point $x \in \mathbb{R}^3$ is inside or outside the shape. Each convex polytope is defined by a collection of half-planes.

A half-plane $H_h(x) = n_h \cdot x + d_h$ provides the signed distance from point x to the h-th plane, where n_h is the normal vector and d_h is the offset parameter.

While the signed distance function (SDF) of any convex object can be computed as the maximum of the SDFs of its constituent planes, CVXnet uses a differentiable approximation. To facilitate gradient learning, instead of the hard maximum, the smooth LogSumExp function is employed to define the approximate SDF, $\Phi(x)$:

$$\Phi(x) = \text{LogSumExp}\{\delta H_h(x)\}$$

The signed distance function is then converted to an indicator function $C : \mathbb{R}^3 \rightarrow [0, 1]$ using: $C(x|\beta) = \text{Sigmoid}(-\sigma\Phi(x))$.

The collection of hyperplane parameters for a primitive is denoted as $h = \{(n_h, d_h)\}$, and the overall set of parameters for a convex as $\beta = [h, \sigma]$. While σ is treated as a hyperparameter, the remaining parameters are learnable. The parameter δ controls the smoothness of the generated convex polytope, while σ controls the sharpness of the indicator function transition. The soft classification boundary created by the sigmoid function facilitates training through differentiable optimization. For our primitive model we use ResNet-18 Encoder E_θ followed by 3 fully-connected layers that decode into the parameters of the primitives D_θ. While the model is lightweight, the SOTA of primitive prediction requires a different trained model for each primitive count K.

Recent work has adapted primitive decomposition to real-world scenes (as opposed to well-defined, isolated objects, such as those in ShapeNet Vavilala & Forsyth (2023)). These methods combine neural prediction with post-training refinement: an encoder-decoder network predicts an initial set

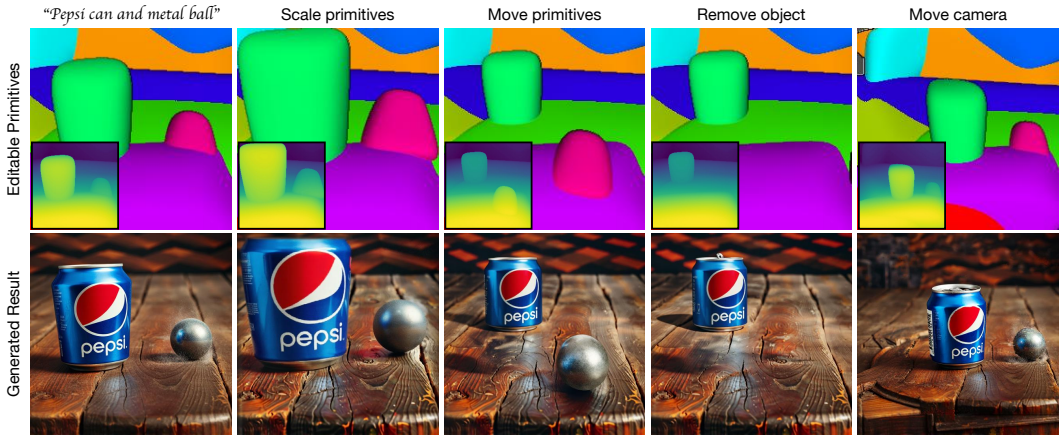


Figure 3: **Editable Primitives as a Structured Depth Prior for Generative Models.** Our method uses 3D convex primitives as an editable intermediate representation from which depth maps are derived. These depth maps (shown as insets in the top row) are used to condition a pretrained depth-to-image generative model. The top row shows primitive configurations after sequential edits—translation, scaling, deletion, and camera motion—alongside their corresponding derived depth maps. The bottom row shows the resulting synthesized images. Unlike direct depth editing, which is unintuitive and underconstrained, manipulating primitives offers a structured, interpretable, and geometry-aware interface for controllable image generation.

of convex polytopes, which is followed by gradient-based optimization to align the primitives closely to observed geometry. This approach is viable because the primary supervision for primitive fitting is a depth map (with heuristics that create 3D samples, and auxiliary losses to avoid degenerate solutions). Note that ground truth primitive parameters are not available (as they could be in many other computer vision settings e.g., segmentation Kirillov et al. (2023)). This is why the losses encourage the primitives to classify points near the depth map boundary correctly instead of directly predicting the parameters.

Rendering the primitives. We condition the RF model on the primitive representation via a depth map, obtained by ray-marching the SDF from the original viewpoint of the scene. Depth conditioning abstracts away potential 'chatter' in the primitive representation from e.g. over-segmentation, while simultaneously yielding flexibility in fine details (depth maps typically lack pixel-level high-frequency details). Depth-conditioned image synthesis models are well-established e.g. Zhang et al. (2023). Because **it's hard to edit a depth map, but easy to edit 3D primitives**, our work adds a new level of control to the existing image synthesis models. As we establish quantitatively in Table 3, our primitive generator is extremely accurate, and our evaluations show that we get very tight control over the synthesized image via our primitives. This means that whatever domain gap there is between depth from primitives and depth from SOTA depth estimation networks is not significant.

Scaling to in-the-wild scenes. We collect 1.8M images from LAION to train our primitive prediction models. To obtain ground truth depth supervision, we use DepthAnythingv2 Yang et al. (2024). We lift the depth map to a 3D point cloud using the pinhole camera model.

3.2 Depth-Conditioned Inpainting in Rectified Flow Transformers

Adding Spatial Conditions. We build upon the state-of-the-art Flux, a rectified flow model Esser et al. (2024); Labs (2024). Older ControlNet implementations Zhang et al. (2023) train an auxiliary encoder that adds information to decoder layers of a base frozen U-Net. Newer implementations, including models supplied by the Black Forest Labs developers, concatenate the latent $\mathbf{x_t}$ and condition (e.g., depth map) $\mathbf{c}$ as an input to the network, yielding tighter control. `FLUX.1 Depth [dev]` re-trains the RF model with the added conditioning; `FLUX.1 Depth [dev] LoRA` trains LoRA layers on top of a frozen base RF model. Both options give tight control and work well with our primitives, though LoRA exposes an added parameter $lora_{weight} \in [0, 1]$ tuning how tightly the depth map should influence synthesis. This is helpful when the primitive abstraction is too coarse relative to the geometric complexity of the desired scene (see Fig. 12).

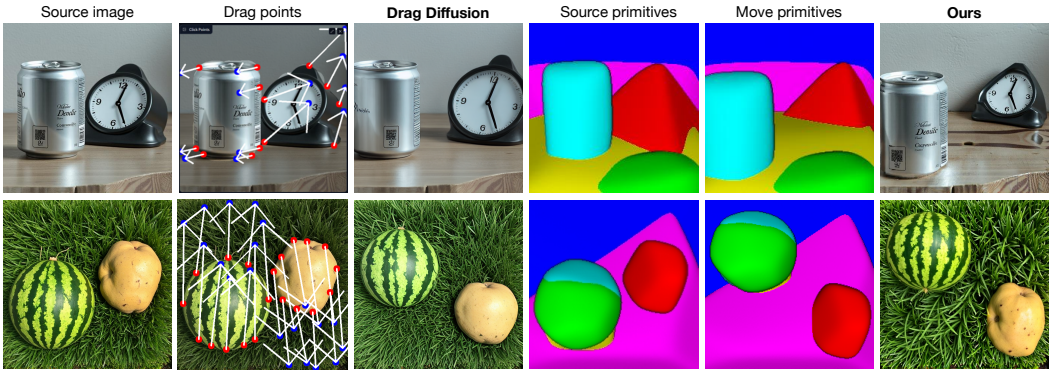


Figure 4: **Comparison with Drag Diffusion** (Shi et al., 2024). **First row:** Given a scene (first column), we attempt to reposition objects using a recent point-based image editing method by drawing drag handles (second column). However, drag points are ambiguous: it is unclear whether the intended operation is translation or scaling. As a result, the output lacks geometric consistency (third column). E.g., the clock changes shape, and pushing it deeper into the scene fails to reduce its size appropriately; fine details on the can are lost. In contrast, Generative Blocks World infers 3D primitives (fourth column) that can be explicitly manipulated (fifth column), producing a plausible image that respects object geometry, scale, positioning, and texture (last column). We also compare with proprietary models in supplement. **Second row** Drag Diffusion requires many arrows to place the objects. Notice how they are still not precisely where we want them, and there are shape and color mismatches on the rendered watermelon and potato. Our result respects both texture and geometry.

Role of Hint and Mask. A core contribution of this work is an algorithm to generate a "hint" image to guide the image generation process, as well as a confidence mask (see Sec 3.3). The hint and mask influence the generation within timesteps $t_{\text{end}} \leq t \leq t_{\text{start}}$, which are hyperparameters. The mask $\mathbf{m} \in [0, 1]$ specifies regions where the hint should guide the output. The hint is encoded into latents $\mathbf{x}_{\text{hint}}$ via the VAE. During denoising, the latents are updated as $\mathbf{x}_t = (1 - \mathbf{m}) \cdot \mathbf{x}_{\text{hint},t} + \mathbf{m} \cdot \mathbf{x}_t$, where $\mathbf{x}_{\text{hint},t}$ is the noised hint latent at timestep t: $\mathbf{x}_{\text{hint},t} = \text{SchedulerScaleNoise}(\mathbf{x}_{\text{hint}}, t, \boldsymbol{\epsilon})$. Thus, the hint image is *noised* to match the current timestep's noise level before incorporation, ensuring consistency with the denoising process. Outside $[t_{\text{end}}, t_{\text{start}}]$, the hint and mask are ignored.

3.3 Texture Hint Generation for Camera and Object Edits

A number of methods have been proposed to preserve texture/object identity upon editing an image. A common and simple technique is to copy the keys and values from a style image into the newly generated image (dubbed "style preserving edits"). For older U-Net-based systems, this is done in the bottleneck layers Bhat et al. (2024). For newer DiTs, this is done at selected "vital" layers Avrahami et al. (2025). In our testing, key-value copying methods are insufficient for camera/primitive moves (see Fig. 6). Further, because of our primitives, we have a geometric representation of the scene. Here we demonstrate a routine to obtain a source "hint" image $\mathbf{x}_{hint}$ as well as a confidence mask $\mathbf{m}$ that can be incorporated in the diffusion process. The hint image is a rough approximation of what the synthesized image should look like using known spatial correspondences between primitives in the first view and the second. The confidence mask indicates where we can and cannot trust the hint, commonly occurring near depth discontinuities. We rely on the diffusion machinery to essentially clean up the hint, filling gaps and refining blurry projected textures so it looks like a real image. The result of our process is an image that respects the text prompt, source texture, and newly edited primitives/camera.

Creating point cloud correspondences We develop a method that accepts point clouds at the ray-primitive intersection points, a *convex_map* integer array indicating which primitive was hit at each pixel, a list of per-primitive transforms (such as scale, rotate, translate), and a hyperparameter $max\_distance$ for discarding correspondences. This procedure also robustly handles camera moves because the input point clouds are representations of the same scene in world space.

Creating a texture hint Given a correspondence map of each 3D point in the new view relative to the original view, we can apply this correspondence to generate a hint image that essentially projects pixels in the old view onto the new view. This is the $\mathbf{x}_{\mathbf{hint}}$ supplied to the image generation model,

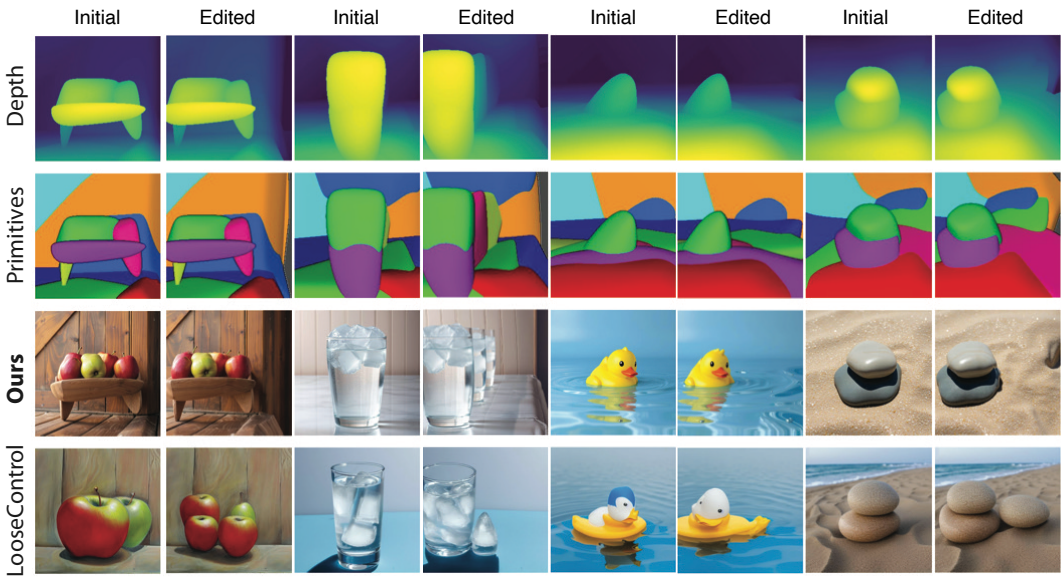


Figure 5: **Comparison with LooseControl** Bhat et al. (2024). Existing work struggles with camera moves. Four scenes (**left** side of each pair), synthesized from the depth maps shown. In each case, the camera is moved to the right (**right** side of each pair), and the image is resynthesized. Note how, for LooseControl, the number of apples changes (first pair); the level of water in the glass changes and there is an extra ice cube (second pair); the duck changes (third pair); and an extra rock appears (fourth pair). In each case, our method shows the same scene from a different view, because the texture hint image is derived from the underlying geometry, and strongly constrains any change.

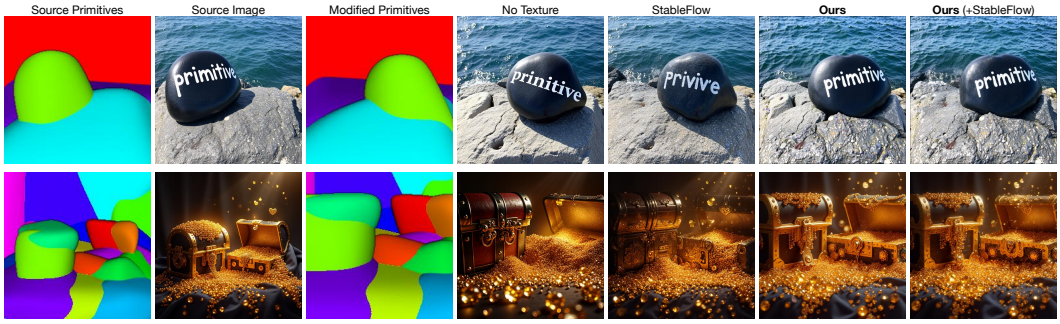


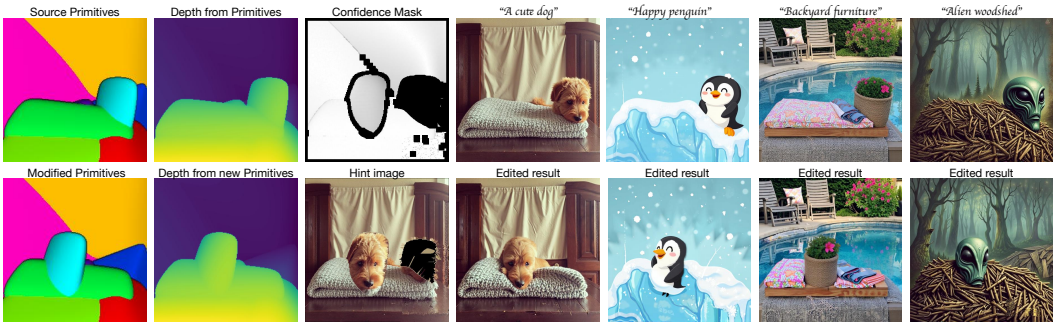
Figure 6: **Projection-Based Texture Hints Preserve Object Identity After Edits.** This figure compares our projection-based texture hints against StableFlow Avrahami et al. (2025), which uses vital-layer key-value injection. **First two columns:** input primitives and image. **Third:** edited primitives. **Fourth:** synthesis from original depth, revealing consistent geometry but altered texture. **Fifth:** StableFlow's approach often changes texture or object identity. **Sixth:** our projection-based hints maintain texture fidelity despite edits. **Seventh:** combining both approaches sometimes improves fine detail recovery (e.g., the treasure chest).

taking into account both camera moves and primitive edits like rotation, translation, and scaling. The point cloud correspondence ensures that if a primitive moves, its texture moves with it. In practice, this hint is essential for good texture preservation (see Fig. 6). Correspondence and hint generation take about 1-2 seconds per image; 30 denoising steps of FLUX at 512 resolution take about 3 seconds on an H100 GPU.

3.4 Evaluation

We seek error metrics to establish (1) geometric consistency between the primitives requested vs. the image that was synthesized and (2) texture consistency between the source and edited image. For (1) we compute the AbsRel between the depth map supplied to the depth-to-image model (obtained

(a) $K = 6$ parts.



(b) $K = 8$ parts.

Figure 7: **Applying same primitive edit for different text prompts at coarse scale ($K \in \{6, 8\}$ parts)**. First row in each subplot contains source primitives and depth (first two columns); the confidence mask for hint generation, followed by four source RGB images. Second row shows the modified primitives and depth, followed by the hint image x_{hint}, followed by the four corresponding edited images. At coarse scales, moving a primitive can move a lot of texture at once. Observe how our hint generation procedure automatically yields confidence masks and hints, assigning low confidence to boundaries of primitives that moved (e.g., the dog's hair) and reveals holes when moving objects. The image model cleans up the low-confidence regions and even handles blurry/aliased texture in the hint when $t_{end} > 0$, meaning that the hint is not used for some denoising steps.

by rendering the primitives) and the estimated depth of the synthesized image (we use the hypersim metric depth module from Yang et al. (2024) to get linear depth). Consistent with standard practice in depth estimation, we use least squares to fit scale and shift parameters onto the depth from RGB (letting the primitive depth supplied to the DM be GT).

To evaluate texture consistency, we apply ideas from the novel view synthesis literature and our existing point cloud correspondence pipeline. Given the source RGB image and the synthesized RGB image (conditioned on the texture hint), we warp the second image back into the first image's frame using our point cloud correspondence algorithm. If we were to synthesize an image in the first render's viewpoint using the second render, this is the texture hint we would use. In error metric calculation, the first RGB image is considered ground truth, the warped RGB image from the edited synthesized image is the prediction, and the confidence mask filters out pixels that are not visible in view 1, given view 2. This evaluation procedure falls in the category of cycle consistency/photometric losses that estimate reprojection error Fang et al. (2024); Jeong et al. (2024); Li et al. (2025); Qin et al. (2025).

4 Results

Fig. 4 shows how users can manipulate depth map inputs to depth-to-image synthesizers; Fig 5 shows camera moves. We have precise control over synthesized geometry while respecting texture. The evaluation in Table 2, demonstrates we hit both goals conclusively. Existing texture preservation based on key-value transfer do not preserve details very well, only high-level semantics and style. We ablate the advantage of our texture preservation approach in Fig. 6. When there are few

Table 2: Comparison of image reconstruction and generation metrics between our method and LooseControl. **AbsRel**$_{\text{src}}$ and **AbsRel**$_{\text{dst}}$ are absolute relative errors evaluating how well the generated images adhere to the requested primitive geometry (source and modified, respectively). PSNR and SSIM are evaluated by reprojecting the second synthesized image back to the original camera viewpoint (see Sec 3.4) and measuring texture consistency with the source. Observe how our procedure simultaneously offers tight geometric adherence to the primitives while preserving the source texture. Results obtained by averaging 48 test images with random camera moves. Because Bhat et al. (2024) does not offer primitive extraction code, we supply our own primitives to both methods for evaluation. We use $K = 10$ parts for this evaluation.

Method	**AbsRel**$_{\text{src}}$ ↓	**AbsRel**$_{\text{dst}}$ ↓	**PSNR** ↑	**SSIM** ↑
Ours	**0.072**	**0.076**	**18.7**	**0.874**
LooseControl Bhat et al. (2024)	0.143	0.146	6.65	0.670

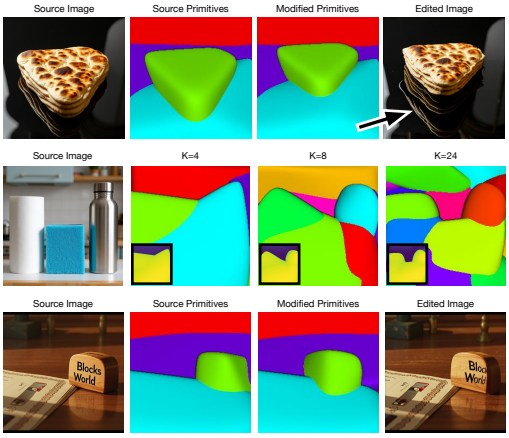


Figure 8: **Failure cases. Top: Illumination misalignments.** Our pixel-space texture hints fail to model lighting (e.g., reflections, shadows) outside primitive boundaries. Consequently, moving an object like the bread stack does not update its static reflection. **Middle: Poor decomposition.** In cluttered scenes or near image edges, sparse depth can cause primitive fitting to fail, incorrectly merging adjacent objects (bottle and paper towel) and resulting in poor control. **Bottom: Rotation artifacts.** Large object rotations (50 degrees) disrupt geometry and texture consistency, causing distortions or hallucinated content (warped text), likely due to a distribution shift in the texture hints.

primitives, moving one primitive affects a big part of the scene; when there are a lot of primitives, we can make fine-scale edits. We show several such examples in Figs. 7, 10.

5 Discussion

3D primitives offer precise geometric control over image generation model outputs, and preserve high-level textures more effectively than key-value transfer methods. This works because primitive decompositions offer several useful properties: they are selectable; they are object-linked; they are compact; they allow edits at coarse and fine grain; and they are accurate enough to yield depth maps that support high-quality texture projection. Our pipeline is designed to allow users to choose between coarse and fine control by adjusting the number of primitives to suit the editing task and scene context.

Our methods have difficulty with some non-convex shapes (e.g. underside of a chair or handle of a coffee mug); additional segmentation and masking, more primitives, or more types of primitive might help. Depth-of-field blurring/bokeh may not be resolved or sharpened when bringing out-of-focus objects into focus. Significant object rotations may also fail (see Fig. 8). In an interactive workflow, manually expanding the confidence mask to include problematic regions e.g., unwanted reflections that don't move with a primitive, can fix some issues. Future work that applies our point correspondences within the network layers themselves (e.g., in vital layers) may yield more robust solutions. Our method does not yet account for view-dependent lighting effects and does not enforce temporal consistency across frames for video synthesis.

Our work highlights the delicacy of the links between the text prompt, hint image, initial noise tensor, and depth map. Current inverters do not support our editing model, apparently because edited images should start from the same noise tensor and prompt as the source image to achieve good results. Certain edits that are at odds with the text prompt are likely to cause problems (e.g., if the prompt mentions an object is on the right, but a user manipulates the primitives to move the object to the left). Changing the text prompt could work in some circumstances (Fig. 11).

REFERENCES

Omri Avrahami, Or Patashnik, Ohad Fried, Egor Nemchinov, Kfir Aberman, Dani Lischinski, and Daniel Cohen-Or. Stable flow: Vital layers for training-free image editing. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR)*, pp. 7877–7888, June 2025.

Shariq Farooq Bhat, Niloy Mitra, and Peter Wonka. Loosecontrol: Lifting controlnet for generalized depth conditioning. In *ACM SIGGRAPH 2024 Conference Papers*, SIGGRAPH '24, New York, NY, USA, 2024. Association for Computing Machinery. doi: 10.1145/3641519.3657525. URL https://doi.org/10.1145/3641519.3657525.

Anand Bhattad, Konpat Preechakul, and Alexei A. Efros. Visual jenga: Discovering object dependencies via counterfactual inpainting, 2025. URL https://arxiv.org/abs/2503.21770.

I Biederman. Recognition by components : A theory of human image understanding. *Psychological Review*, (94):115–147, 1987.

TO Binford. Visual perception by computer. In *IEEE Conf. on Systems and Controls*, 1971.

Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. Bsp-net: Generating compact meshes via binary space partitioning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Yen-Chi Cheng, Krishna Kumar Singh, Jae Shin Yoon, Alexander Schwing, Liangyan Gui, Matheus Gadelha, Paul Guerrero, and Nanxuan Zhao. 3D-Fixup: Advancing Photo Editing with 3D Priors. In *Proceedings of the SIGGRAPH Conference Papers*. ACM, 2025. doi: 10.1145/3721238.3730695.

Yutao Cui, Xiaotong Zhao, Guozhen Zhang, Shengming Cao, Kai Ma, and Limin Wang. Stabledrag: Stable dragging for point-based image editing. In *European Conference on Computer Vision*, pp. 340–356. Springer, 2024.

Boyang Deng, Kyle Genova, Soroosh Yazdani, Sofien Bouaziz, Geoffrey Hinton, and Andrea Tagliasacchi. Cvxnet: Learnable convex decomposition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Abdelrahman Eldesokey and Peter Wonka. Build-a-scene: Interactive 3d layout control for diffusion-based image generation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=gg6dPtdC1C.

Dave Epstein, Allan Jabri, Ben Poole, Alexei A. Efros, and Aleksander Holynski. Diffusion self-guidance for controllable image generation. 2023.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first International Conference on Machine Learning*, 2024.

Qihang Fang, Yafei Song, Keqiang Li, Li Shen, Huaiyu Wu, Gang Xiong, and Liefeng Bo. Evaluate geometry of radiance fields with low-frequency color prior. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'24/IAAI'24/EAAI'24. AAAI Press, 2024. doi: 10.1609/aaai.v38i2.27938. URL https://doi.org/10.1609/aaai.v38i2.27938.

Jiatao Gu, Lingjie Liu, Peng Wang, and Christian Theobalt. Stylenerf: A style-based 3d aware generator for high-resolution image synthesis. In *International Conference on Learning Representations*, 2022.

Weiran Guang, Xiaoguang Gu, Mengqi Huang, and Zhendong Mao. Dragin3d: Image editing by dragging in 3d space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21502–21512, June 2025.

Abhinav Gupta, Alexei A. Efros, and Martial Hebert. Blocks world revisited: Image understanding using qualitative geometry and mechanics. In *European Conference on Computer Vision (ECCV)*, 2010.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017.

Yoonwoo Jeong, Jinwoo Lee, Seokyeong Lee, Doyup Lee, and Minhyuk Sung. NVS-Adapter: Plug-and-play novel view synthesis from a single image. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2024.

Nikolai Kalischek, Michael Oechsle, Fabian Manhardt, Philipp Henzler, Konrad Schindler, and Federico Tombari. Cubediff: Repurposing diffusion-based image models for panorama generation, 2025. URL https://arxiv.org/abs/2501.17162.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 3992–4003, 2023. doi: 10.1109/ICCV51070.2023.00371.

Florian Kluger, Hanno Ackermann, Eric Brachmann, Michael Ying Yang, and Bodo Rosenhahn. Cuboids revisited: Learning robust 3d shape fitting to single rgb images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Gwanhyeong Koo, Sunjae Yoon, Younghwan Lee, Ji Woo Hong, and Chang D. Yoo. Flowdrag: 3d-aware drag-based image editing with mesh-guided deformation vector flow fields. In *Proceedings of the 42nd International Conference on Machine Learning (ICML 2025)*, June 2025. URL https://icml.cc/virtual/2025/poster/43848. Poster (Spotlight Poster).

Black Forest Labs. Flux. https://github.com/black-forest-labs/flux, 2024.

Feifei Li, Qi Song, Chi Zhang, Hui Shuai, and Rui Huang. PoI: Pixel of interest for novel view synthesis assisted scene coordinate regression. *arXiv preprint arXiv:2502.04843*, 2025.

Haolin Liu, Yujian Zheng, Guanying Chen, Shuguang Cui, and Xiaoguang Han. Towards high-fidelity single-view holistic reconstruction of indoor scenes. In *European Conference on Computer Vision*, pp. 429–446. Springer, 2022.

Oscar Michel, Anand Bhattad, Eli VanderBilt, Ranjay Krishna, Aniruddha Kembhavi, and Tanmay Gupta. Object 3dit: Language-guided 3d-aware image editing. *Advances in Neural Information Processing Systems*, 36:3497–3516, 2023.

Tom Monnier, Jake Austin, Angjoo Kanazawa, Alexei A. Efros, and Mathieu Aubry. Differentiable Blocks World: Qualitative 3D Decomposition by Rendering Primitives. In *NeurIPS*, 2023.

Chong Mou, Xintao Wang, Jiechong Song, Ying Shan, and Jian Zhang. Dragondiffusion: Enabling drag-style manipulation on diffusion models. *arXiv preprint arXiv:2307.02421*, 2023.

Chong Mou, Xintao Wang, Liangbin Xie, Yanze Wu, Jian Zhang, Zhongang Qi, and Ying Shan. T2i-adapter: Learning adapters to dig out more controllable ability for text-to-image diffusion models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(5):4296–4304, Mar. 2024. doi: 10.1609/aaai.v38i5.28226. URL https://ojs.aaai.org/index.php/AAAI/article/view/28226.

Jiteng Mu, Michaël Gharbi, Richard Zhang, Eli Shechtman, Nuno Vasconcelos, Xiaolong Wang, and Taesung Park. Editable image elements for controllable synthesis. In *Proceedings of the European Conference on Computer Vision (ECCV)*, volume 15059 of *Lecture Notes in Computer Science*, pp. 616–635. Springer, 2024. doi: 10.1007/978-3-031-72627-9_3.

Xingang Pan, Ayush Tewari, Thomas Leimkühler, Lingjie Liu, Abhimitra Meka, and Christian Theobalt. Drag your gan: Interactive point-based manipulation on the generative image manifold. In *ACM SIGGRAPH 2023 conference proceedings*, pp. 1–11, 2023.

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 *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7695–7704, 2024. doi: 10.1109/CVPR52733.2024.00735.

Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. In *Proceedings of the International Conference on Learning Representations (ICLR)*. OpenReview.net, 2023. URL https://openreview.net/forum?id=FjNys5c7VyY.

Yuxin Qin, Xinlin Li, Linan Zu, and Ming Liang Jin. Novel view synthesis with depth priors using neural radiance fields and cyclegan with attention transformer. *Symmetry*, 17(1), 2025. ISSN 2073-8994. doi: 10.3390/sym17010059. URL https://www.mdpi.com/2073-8994/17/1/59.

L. G. Roberts. *Machine Perception of Three-Dimensional Solids*. PhD thesis, MIT, 1963.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10674–10684, 2022. doi: 10.1109/CVPR52688.2022.01042.

Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. Csgnet: Neural shape parser for constructive solid geometry. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Yujun Shi, Chuhui Xue, Jun Hao Liew, Jiachun Pan, Hanshu Yan, Wenqing Zhang, Vincent YF Tan, and Song Bai. Dragdiffusion: Harnessing diffusion models for interactive point-based image editing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8839–8849, 2024.

Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *Proceedings of the 12th European Conference on Computer Vision - Volume Part V*, ECCV'12, pp. 746–760, Berlin, Heidelberg, 2012. Springer-Verlag. doi: 10.1007/978-3-642-33715-4_54. URL https://doi.org/10.1007/978-3-642-33715-4_54.

Junshu Tang, Tengfei Wang, Bo Zhang, Ting Zhang, Ran Yi, Lizhuang Ma, and Dong Chen. Make-it-3d: High-fidelity 3d creation from a single image with diffusion prior. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 22819–22829, October 2023.

Shubham Tulsiani, Hao Su, Leonidas J. Guibas, Alexei A. Efros, and Jitendra Malik. Learning shape abstractions by assembling volumetric primitives. In *Computer Vision and Pattern Recognition (CVPR)*, 2017.

Vaibhav Vavilala and David Forsyth. Convex decomposition of indoor scenes. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9142–9152, 2023. doi: 10.1109/ICCV51070.2023.00842.

Vaibhav Vavilala, Rahul Vasanth, and David Forsyth. Denoising monte carlo renders with diffusion models, 2024. URL https://arxiv.org/abs/2404.00491.

Vaibhav Vavilala, Florian Kluger, Seemandhar Jain, Bodo Rosenhahn, Anand Bhattad, and David Forsyth. Improved convex decomposition with ensembling and boolean primitives, 2025a. URL https://arxiv.org/abs/2405.19569.

Vaibhav Vavilala, Faaris Shaik, and David Forsyth. Dequantization and color transfer with diffusion models. In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 9630–9639, 2025b. doi: 10.1109/WACV61041.2025.00932.

Haochen Wang, Xiaodan Du, Jiahao Li, Raymond A. Yeh, and Greg Shakhnarovich. Score jacobian chaining: Lifting pretrained 2d diffusion models for 3d generation. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12619–12629, 2023. doi: 10.1109/CVPR52729.2023.01214.

Xiaoyan Xing, Konrad Groh, Sezer Karaoglu, Theo Gevers, and Anand Bhattad. Luminet: Latent intrinsics meets diffusion models for indoor scene relighting. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 442–452, 2025.

Lihe Yang, Bingyi Kang, Zilong Huang, Zhen Zhao, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything v2. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 21875–21911. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/26cfdcd8fe6fd75cc53e92963a656c58-Paper-Conference.pdf.

Jiraphon Yenphraphai, Xichen Pan, Sainan Liu, Daniele Panozzo, and Saining Xie. Image sculpting: Precise object editing with 3d geometry control. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4241–4251, 2024.

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 3813–3824, 2023. doi: 10.1109/ICCV51070.2023.00355.

Zewei Zhang, Huan Liu, Jun Chen, and Xiangyu Xu. Gooddrag: Towards good practices for drag editing with diffusion models. In *The Thirteenth International Conference on Learning Representations*, 2025.

Ruisi Zhao, Zechuan Zhang, Zongxin Yang, and Yi Yang. 3d object manipulation in a single image using generative models, 2025. URL https://arxiv.org/abs/2501.12935.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, 2017.

Chuhang Zou, Alex Colburn, Qi Shan, and Derek Hoiem. Layoutnet: Reconstructing the 3d room layout from a single rgb image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

A APPENDIX

Here we present additional details and evaluation. Note: we use LLMs to get LaTeX syntax and cross-check for missing related works. We also use it to create UI for the generation of our results.

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 3: 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 AbsRel $\to 0$ as $K \to \infty$ (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**$\downarrow$
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, worked 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 pipeline [1]. 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.

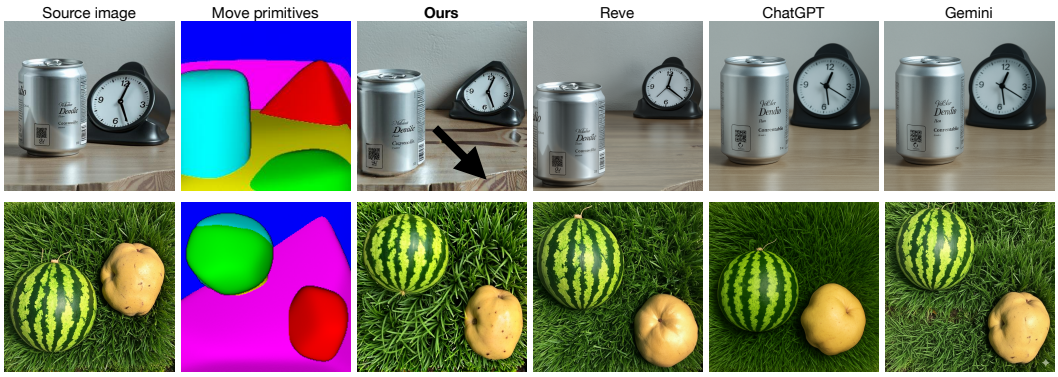
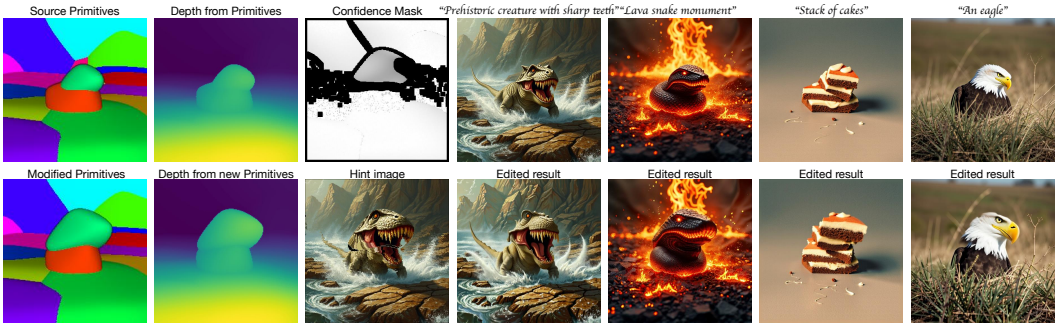
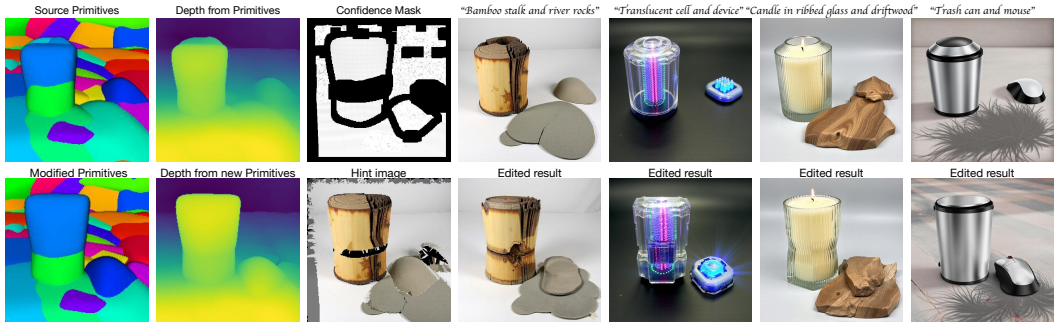


Figure 9: **Evaluation with production systems. First row, Left:** Source image. The next two columns show our primitive edits and the synthesized result. The arrow indicates a texture that our method reproduces faithfully, but others do not. The **fourth column** shows Reve (https://app.reve.com/), 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.

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



(a) $K = 24$ parts.



(b) $K = 60$ parts.

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.

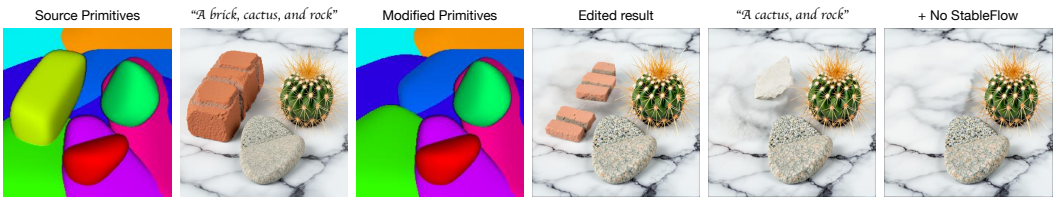


Figure 11: **Primitive edits can conflict with the text prompt.** Some geometric edits require changing the text prompt, for example, when removing an object. The **fourth column** mentions brick in the text prompt, but that primitive was removed, resulting in brick pieces in the inpainted region. In 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**).

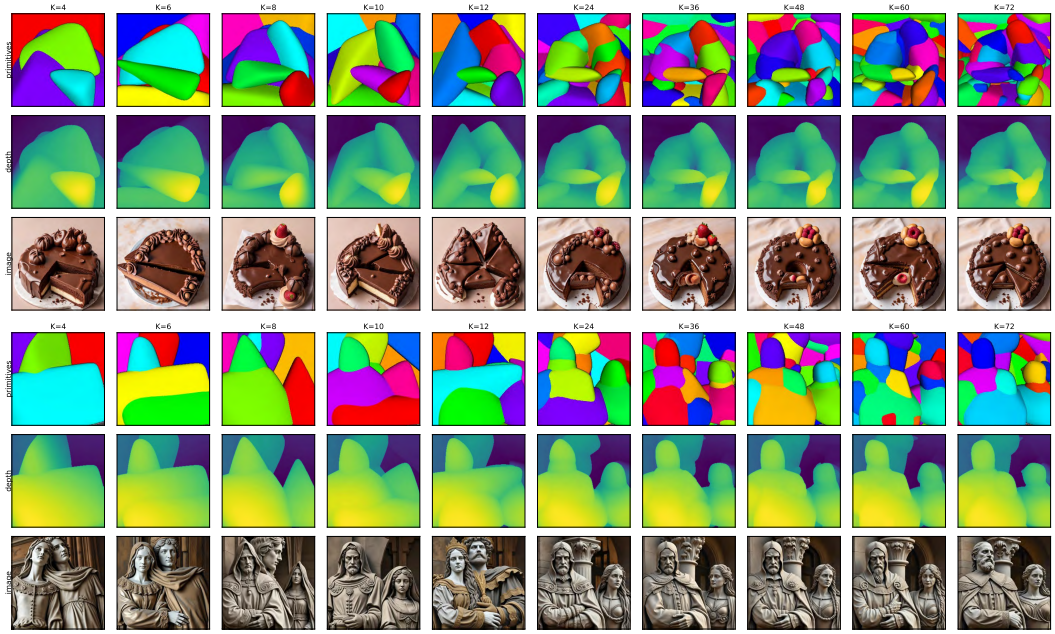


Figure 13: Given the same depth map, we extract primitives at variable resolution (from 4-72 parts). We show the depth maps in each second row, and synthesized result in each 3rd row. Observe how no matter the resolution, the FLUX-LoRA model (we use $lora_{weight} = 0.8$) gives an image that follows 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.

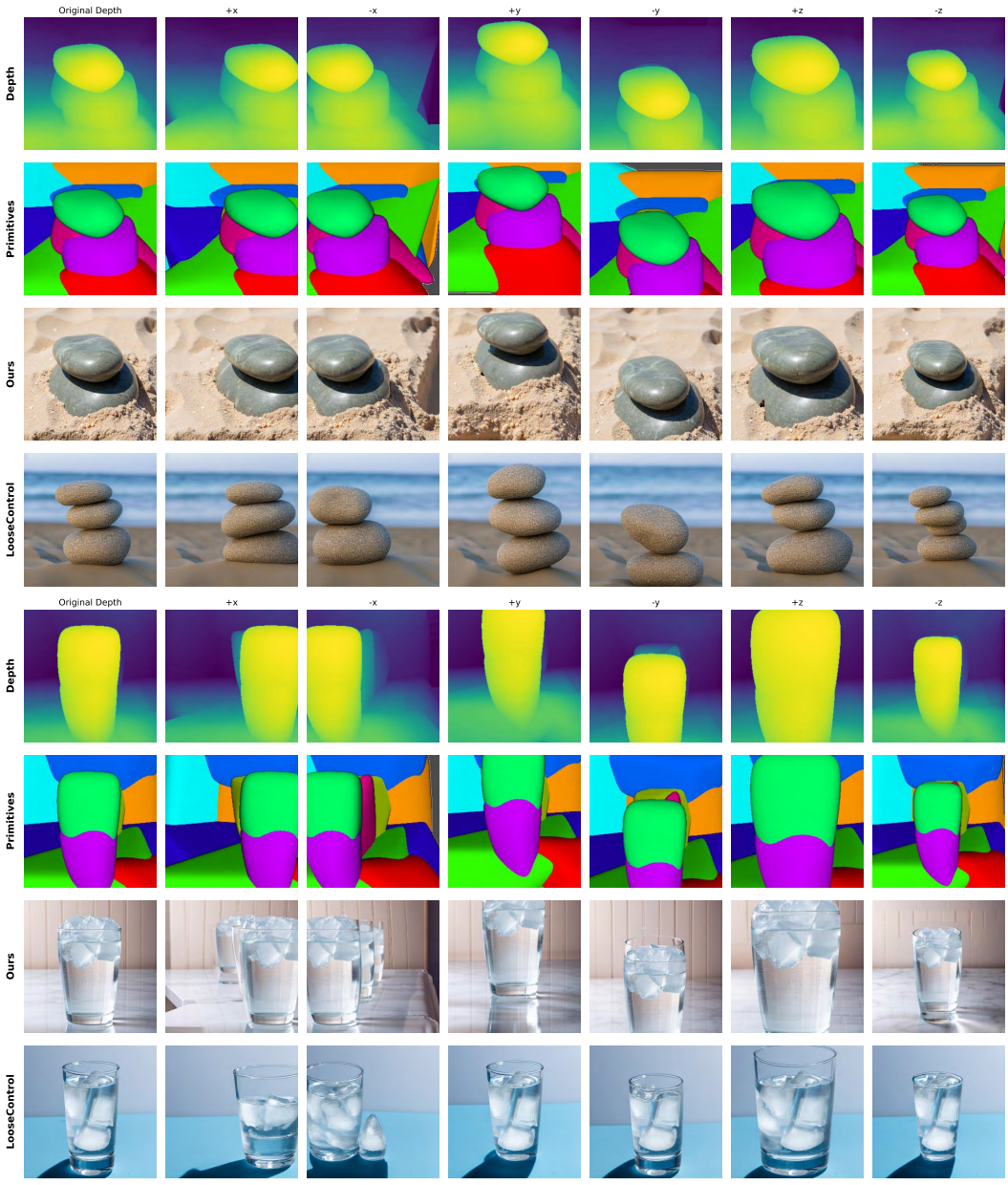


Figure 16: Additional move camera evaluations. Our method can simultaneously adhere to source texture and requested primitives.

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* θ **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 $p \in$ *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**
    - $\mathbf{q} \leftarrow$ `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 $\mathcal{M}_{hit}[y, x] = 1$ **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 $y_s \in [y_{start}, y_{end})$ **do**
      for $x_s \in [x_{start}, x_{end})$ **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`
        $\alpha_x \leftarrow \frac{x_s - x_{\text{start}}}{\max(x_{\text{end}} - x_{\text{start}}, 1)}$;
        $y_{\text{sample}} \leftarrow y_{\text{src}} + \alpha_y \cdot \lambda_h$;
        $x_{\text{sample}} \leftarrow x_{\text{src}} + \alpha_x \cdot \lambda_w$;
        $\mathcal{H}[:, y_s, x_s] \leftarrow$ `BilinearSample`$(\mathbf{I}_{\text{src}}, y_{\text{sample}}, x_{\text{sample}})$;
      end
    end
  end
end
return $\mathcal{H}$;