INLAY3D: PRECISE PART-LEVEL TEXTURE AND SHAPE MANIPULATION OF 3D ASSETS

Anonymous authors
Paper under double-blind review

ABSTRACT

In real-world 3D asset production pipelines, artists need to edit and refine the details of the local regions, yet this problem remains insufficiently explored. Such local edits should provide the following capabilities: 1) *Part-level spatial control*: identifying the complete 3D target part intended by the user while preserving non-target regions and boundaries; and 2) *Attribute-specific editing*: supporting texture editing, shape manipulation, or joint shape-and-texture manipulation within the target part. However, existing methods often define the editing region imprecisely, either with coarse 3D boxes or by implicit region selection. They lack an editing framework that enables decoupled texture and geometry editing. To address these issues, we propose Inlay3D, a novel framework for precise part localization and decoupled texture and shape editing. Specifically, Inlay3D first performs *Target Part Extraction*, selecting an explicit 3D target part from 3D part candidates guided by a lightweight 2D mask. It then applies *Two-Stage Editing* to the selected part. In *Isolated Mesh Editing*, we edit a part-isolated representation that contains only the selected region. As a result, the edit can focus on the target part itself, leading to high-quality results while reducing contextual coupling from non-edited regions. We further introduce *Boundary Refinement*, which refines a narrow boundary region to remove boundary artifacts while preserving the non-target region. Extensive experiments demonstrate that Inlay3D achieves state-of-the-art results in editing quality and non-target region preservation.

1 INTRODUCTION

With the rapid development of 2D and 3D generation techniques, generated digital assets have been increasingly used in game production, immersive content creation, digital entertainment, and AR/VR. Therefore, local editing and modification of 3D digital assets are becoming more and more important. When editing an existing 3D asset, users often need to preserve the global structure of the original asset while modifying a specific part.

During the process of creating 3D assets, artists first locate the local area to be edited and then perform modifications to the geometry, texture, or both. This procedure effectively preserves the non-edited regions and allows for the decoupling of texture and geometry editing. Based on these practical considerations, we consider that local 3D editing methods should provide two key capabilities: 1) *Part-level spatial control*: accurately identifying the 3D target part intended by the user, while preserving the geometry and texture of non-target regions. 2) *Attribute-specific editing*: applying texture-only, geometry-only, or joint shape-and-texture editing to the selected part.

In this paper, we construct an evaluation dataset and corresponding metrics to assess these two capabilities of existing methods. This dataset contains mesh assets from various sources, and the editing instructions include both texture and geometry editing commands. The dataset is shown in the right part of Fig. 1. However, we evaluate existing local 3D editing methods using this dataset and find that they struggle to provide these two capabilities. Existing approaches can be grouped into two families according to where the editing prior operates. The first family transfers 2D diffusion priors to 3D. Within this family, optimization-based methods, including Vox-E (Sella et al., 2023), DreamEditor (Zhuang et al., 2023), and GaussianEditor (Chen et al., 2024b), optimize a 3D representation using SDS guidance, but often produce low-quality or multi-view inconsistent results. Multi-view synthesis methods, such as Instant3dit (Barda et al., 2025) and MVEdit (Chen

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

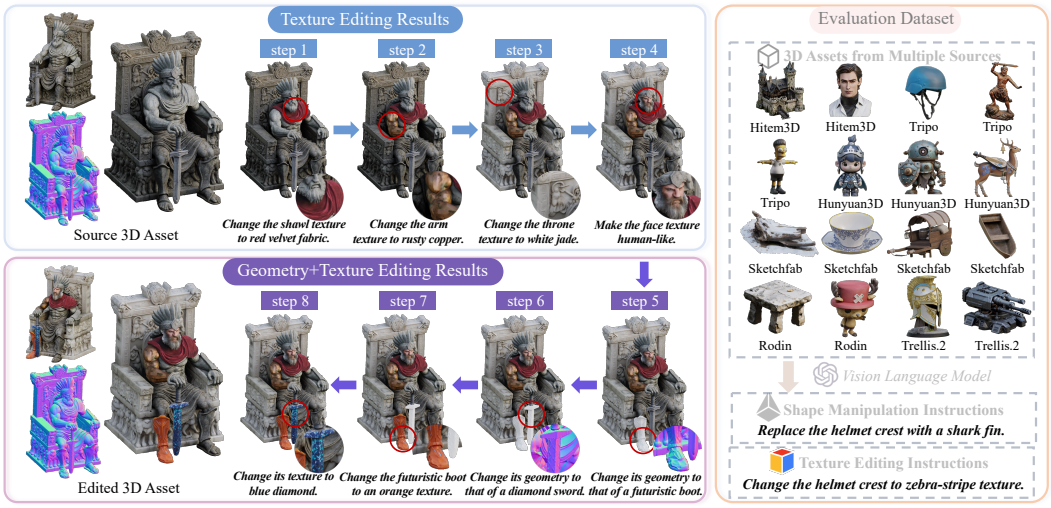


Figure 1: The left panel shows progressive local editing with Inlay3D, comprising four texture-only local edits (top) followed by two joint shape-and-texture local edits (bottom). The results accurately preserve previously edited parts and non-target regions. The right panel presents examples from our evaluation dataset, which combines mesh assets from various sources. Besides, we provide texture and geometry editing instructions to assess attribute-specific editing. Zoom in for a better view.

et al., 2024a), operate by editing rendered views and fusing the resulting edits back into 3D. However, since editing control is specified in selected views or through projected masks, occluded and back-facing regions may remain incomplete or inconsistent. The second family edits native 3D latent or structured representations. These methods differ mainly in how they specify the target region. VoxHammer (Li et al., 2026) enables explicit region control by replacing preserved-region features during denoising, but requires costly manual 3D masks. VecSet-Edit (Hsiao et al., 2026) localizes edits through mask-guided seeding and gating of spatial VecSet tokens from 2D image conditions. Mask-free methods infer the editable region more implicitly: Nano3D (Ye et al., 2026) adapts FlowEdit (Kulikov et al., 2025) to TRELLIS.2 (Xiang et al., 2026) and preserves content through voxel/slat merging, while 3DEditFormer (Xia et al., 2026) learns feature-based localization using dual-guidance attention and time-adaptive gating. More recent methods strengthen spatial control in different ways: ShapeUP (Gat et al., 2026) learns mask-free localization in native 3D latents, while Prox-E (Sella et al., 2026) derives spatial controls from a VLM-edited primitive proxy. Nevertheless, their edit regions remain latent- or proxy-defined rather than explicit subsets of the input mesh, limiting direct surface-level boundary control. Overall, existing methods do not jointly provide explicit part-level spatial control and attribute-specific editing on the input mesh.

In this paper, we propose Inlay3D, a novel training-free framework for local 3D editing of existing textured meshes. This framework follows the practical workflow of an artist editing a mesh: *Cut–Edit–Mend*. *Cut* explicitly extracts the target 3D part from the remaining asset, *Edit* manipulates the texture or shape within the isolated part, and *Mend* further refines the boundary region. To provide part-level spatial control, Inlay3D takes a single-view 2D mask as input and utilizes geometry-based 3D part segmentation methods (Liu et al., 2025; Ma et al., 2026) to explicitly extract the target region. To support attribute-specific editing, Inlay3D uses TRELLIS.2 (Xiang et al., 2026) for texture and shape manipulation, with joint edits performed by first modifying the shape and then editing its texture. Together, these components combine explicit part-level spatial control with flexible attribute editing while preserving the non-target regions.

Specifically, the *Cut* operation is implemented by our *Target Part Extraction* module. Unlike general part segmentation, this module uses the user mask as an editing constraint to guide candidate selection. The module segments the input mesh into candidate parts using 3D part segmentation methods (Ma et al., 2026; Liu et al., 2025). Furthermore, the candidate parts are inserted into a queue and processed by a breadth-first search algorithm. For each candidate, we project it to the mask view and compute the coverage with the 2D mask. Then each candidate is either accepted,

rejected, or further subsegmented. After iterative segmentation, the union of the accepted candidates forms the final target region, which is then used for subsequent local 3D editing.

We then introduce *Two-Stage Editing* to conduct the subsequent *Edit* and *Mend* operations. The first stage, *Isolated Mesh Editing*, isolates the target region and then edits it. The isolation operation removes the contextual effects of non-target regions and generates high-resolution results. The local editing result is then integrated back into the full asset, but direct integration may introduce boundary inconsistencies between the edited target part and the preserved mesh. To handle these inconsistencies, we propose the second stage, *Boundary Refinement*, to refine the boundary region. Through this two-stage editing module, Inlay3D produces high-quality editing results while preserving the non-target regions effectively. As shown in the left part of Fig. 1, Inlay3D can apply texture-only edits and joint shape-and-texture edits to different target parts. These results indicate that our method keeps previously edited parts and non-target regions stable after multiple edits.

In summary, our contributions are as follows:

- We introduce a training-free framework for explicit part-level editing of textured 3D assets. Given a single-view 2D mask, our framework identifies the corresponding 3D target part and enables local texture and shape editing while preserving non-target regions.
- We propose mask-guided Target Part Extraction, which iteratively accepts, rejects, and subdivides 3D part candidates to recover an explicit mesh-level editing region. We propose a two-stage editing strategy that combines isolated part editing with boundary refinement, using O-Voxel replacement to constrain changes outside the refinement band.
- We evaluate the framework on part localization and a local-editing benchmark, demonstrating improved edit fidelity and non-target preservation over the evaluated baselines.

2 Related Work

2.1 3D Asset Editing

Existing 3D editing methods either lift 2D priors or operate in native 3D representations. Optimization-based methods such as Vox-E (Sella et al., 2023), DreamEditor (Zhuang et al., 2023), and GaussianEditor (Chen et al., 2024b) are flexible but slow and may produce view-inconsistent results. Multi-view methods such as Instant3dit (Barda et al., 2025) and MVEdit (Chen et al., 2024a) edit rendered views and fuse them into 3D, but occluded regions can remain inconsistent. Native approaches improve 3D consistency: VoxHammer (Li et al., 2026) uses explicit 3D masks. VecSet-Edit (Hsiao et al., 2026) operates on spatial tokens. Nano3D (Ye et al., 2026) adapts FlowEdit (Kulikov et al., 2025) to structured latents. 3DEditFormer (Xia et al., 2026) learns feature-based localization. ShapeUP (Gat et al., 2026) and Prox-E (Sella et al., 2026) further introduce mask-free latent localization and VLM-edited primitive controls, respectively. However, these controls either require manual 3D annotation or remain latent/proxy-defined, rather than recovering explicit mesh parts from lightweight 2D guidance. In contrast, Inlay3D extracts the target part that directly constrains texture, geometry, or joint editing and supports stable non-target preservation.

2.2 3D Part Segmentation

Specifying where to edit is a central difficulty in local 3D editing. A manual 3D mask provides direct control, but it is costly to annotate and inconvenient for ordinary users. A coarse box or a mask projected from one view is easier to provide, but it may include neighboring structures, miss invisible surfaces, or fail to correspond to a semantic part. Recent 3D part segmentation methods offer a promising alternative. SAMPart3D (Yang et al., 2024) transfers 2D segmentation priors to 3D, PartField (Liu et al., 2025) learns a geometry-aware feature field for part discovery, and P3-SAM (Ma et al., 2026) provides native 3D point-prompted segmentation. PartSAM (Zhu et al., 2026) is trained natively on large-scale 3D data and supports both promptable part identification and automatic decomposition into surface and internal structures. These methods are designed for part discovery or segmentation, not for deciding the editable region of a particular user instruction. In local editing, the target region may require merging several candidates, rejecting visually overlapping non-target parts, or recursively refining an ambiguous candidate whose projection only partially matches the

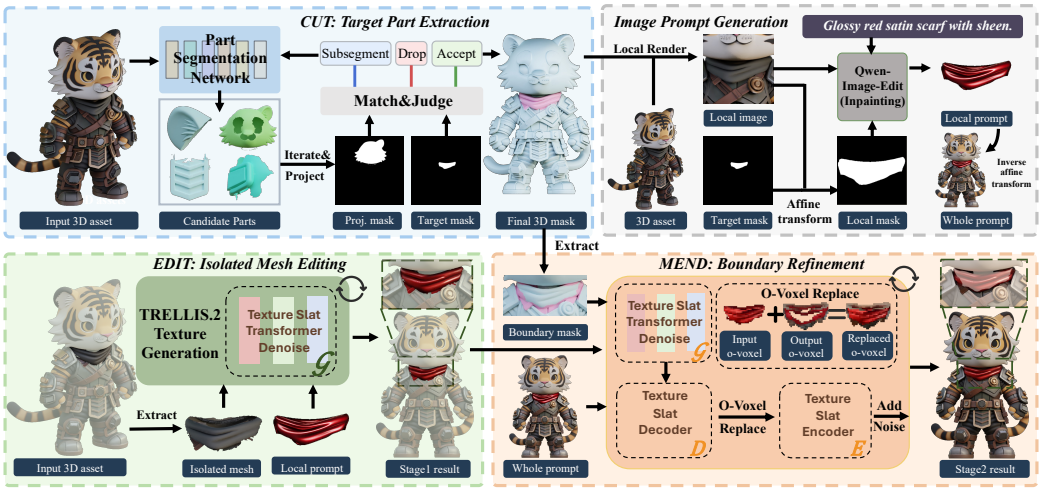


Figure 2: The figure illustrates the texture-editing workflow of Inlay3D, which also supports shape manipulation. In *Cut*, projected part candidates are recursively matched to a single-view mask to extract the 3D target part. In *Edit*, a local image prompt guides TRELLIS.2 texture generation on the isolated target mesh. In *Mend*, a whole-view prompt guides refinement within a narrow boundary band, while O-Voxel replacement preserves non-target regions during denoising.

user mask. Inlay3D builds on 3D part segmentation but reformulates it as editing-oriented target extraction: the 2D mask guides candidate acceptance, rejection, and subdivision, producing an explicit 3D target part for subsequent local texture or geometry editing.

3 METHOD

3.1 OVERVIEW

In this section, we present a training-free framework for local 3D mesh editing that follows a *Cut–Edit–Mend* workflow. Based on practical needs in 3D asset production, we argue that local 3D editing should provide two key capabilities: 1) *Part-level spatial control*: accurately identifying and editing the intended 3D target part while preserving non-target regions, and 2) *Attribute-specific editing*: supporting texture editing, shape manipulation, or joint shape-and-texture edits within the target part. Existing methods (Sella et al., 2023; Barda et al., 2025; Chen et al., 2024a; Hsiao et al., 2026; Ye et al., 2026; Xia et al., 2026) specify editing regions using either coarse bounding boxes or implicit latent features. In *Cut*, our *Target Part Extraction* module (Sec. 3.2) uses geometry-based 3D segmentation and a single-view 2D mask to iteratively select and merge candidate parts into an explicit 3D target region. Recent methods, e.g., VoxHammer Li et al. (2026), VecSet-Edit Hsiao et al. (2026), and Nano3D Ye et al. (2026), preserve non-target regions by replacing their representations with source features in latent space, which may introduce artifacts after decoding. In contrast, *Edit* and *Mend* constitute our *Two-Stage Editing* module (Sec. 3.3): *Edit* uses TRELLIS.2 (Xiang et al., 2026) to manipulate the texture, shape, or both in a part-isolated representation, while *Mend* refines a narrow boundary region after reinserting the edited part into the full asset. This design preserves non-target regions directly in 3D space. The overall framework is illustrated in Fig. 2.

3.2 TARGET PART EXTRACTION

In this subsection, we aim to use the capability of geometry-based 3D part segmentation methods to extract the 3D editing regions from a single-view 2D mask. Previous methods such as SAM-Part3D (Yang et al., 2024) use the mask to guide scale-conditioned 3D part segmentation, but we empirically find that the segmentation results are not reliable enough for edit-region extraction. Recent geometry-based 3D part segmentation methods such as P3-SAM (Ma et al., 2026) and PartField (Liu et al., 2025) can produce fine-grained 3D part candidates. A naive alternative is to match

these candidates' projections with the mask and select the one that matches best. However, these methods lack controllable segmentation granularity, leading to less accurate matching results.

To address this, our key insight is to use the 2D mask not only to match part candidates, but also to control the segmentation granularity. Based on this insight, we design a mask-guided *Iterative Part Selection* strategy to extract the target part. Specifically, let $\mathcal{M} = (V, F, \mathcal{U}, \mathcal{T})$ be the input textured mesh, where V denotes the mesh vertices, F denotes the mesh faces, $\mathcal{U}$ denotes the UV map, and $\mathcal{T}$ denotes the texture map. Let $\mathbf{I} = \mathcal{R}(\mathcal{M}, \pi)$ be its rendering under the user camera π. And $\mathbf{m}$ denotes the corresponding user mask under the same π. We first obtain candidate part groups $\mathcal{C} = \{C_i\}$ from a geometry-based 3D part segmentation method (Ma et al., 2026). The initial candidates are then inserted into a queue $\mathcal{Q}$ and processed with a queue-based breadth-first search. In the BFS algorithm, for each candidate C_i, we first render its visible projection:

$$\mathbf{p}_i = \mathcal{R}_{\text{mask}}(C_i, \pi), \tag{1}$$

where $\mathbf{p}_i$ is a binary mask image. We then compute the coverage score:

$$s_{\text{cover}}(C_i) = \frac{|\mathbf{p}_i \cap \mathbf{m}|}{|\mathbf{p}_i| + \epsilon}. \tag{2}$$

Here, $|\cdot|$ denotes the number of foreground pixels in a 2D mask, and ϵ is a tiny positive value. s_{cover} measures the fraction of the candidate projection overlapping with the user mask. Candidates are then judged to be accepted, rejected, or sub-segmented: a candidate is accepted if $s_{\text{cover}}(C_i) > \tau$, rejected if $|\mathbf{p}_i \cap \mathbf{m}| < \eta$, and sub-segmented into finer candidates if it is not accepted or rejected. Then the accepted candidates are added to a set $\mathcal{A}$, the rejected candidates are discarded, and the sub-parts of ambiguous candidates are appended to $\mathcal{Q}$ for further segmentation until a maximum recursion depth of three is reached. When $\mathcal{Q}$ is empty, the union of $\mathcal{A}$ forms the final target part, providing a precise 3D mask $\mathcal{P}$ for subsequent local 3D editing.

3.3 Two-Stage Editing

In this subsection, given $\mathcal{P}$, we aim to leverage TRELLIS.2 (Xiang et al., 2026) to edit the target region while keeping non-target regions unchanged. A *naive* solution is to adapt TRELLIS.2 into an inpainting pipeline by applying the 3D mask to the full asset. However, our experiments Fig. 3 show that target regions not visible in the reference image are often not edited, and visible regions exhibit weak editing quality. We attribute this to contamination from the surrounding non-target context.

Therefore, Inlay3D introduces *Two-Stage Editing*. We first propose *Isolated Mesh Editing*, which isolates the target region and edits it independently to remove non-target context. In the second stage, we propose *Boundary Refinement* to apply local inpainting only to the boundary region. In addition, unlike previous methods that preserve non-target regions through latent replacement (Li et al., 2026; Ye et al., 2026), we introduce an O-Voxel replacement strategy during boundary inpainting to preserve non-target regions directly in 3D space. Below, we mainly describe the two-stage procedure using texture editing. Shape manipulation follows the same framework and uses the extracted target part as the local editing domain. Its shape-specific details are provided in Appendix C.

We first generate image prompts using both the extracted target part $\mathcal{P}$ and the textual editing instruction T. We use a part-centric orthographic camera π_{loc}, with the same orientation as the user camera π, to render a local image $\mathbf{I}_{\text{loc}}$. Specifically, we recenter the camera on $\mathcal{P}$ and adjust its orthographic scale to frame the target part while keeping its orientation fixed. Since both cameras use orthographic projection, corresponding 3D points project to image coordinates that differ only by a uniform scaling and a 2D translation, defining an affine transformation $\mathcal{A}$ from the original view to the local view. We transform the mask $\mathbf{m}$ to the local view as $\mathbf{m}_{\text{loc}}$ using $\mathbf{m}_{\text{loc}} = \mathcal{A}(\mathbf{m})$. Given $\mathbf{m}_{\text{loc}}$, $\mathbf{I}_{\text{loc}}$, and T, we use the inpainting version of Qwen-Image-Edit-2511 (Wu et al., 2025) to generate a high-quality local-view image prompt $\hat{\mathbf{I}}_{\text{loc}}$. We then map $\hat{\mathbf{I}}_{\text{loc}}$ back to the original view with the inverse affine transformation $\hat{\mathbf{I}} = \mathcal{A}^{-1}(\hat{\mathbf{I}}_{\text{loc}})$, obtaining a whole-view image prompt $\hat{\mathbf{I}}$. Further implementation details of image prompt generation are provided in Appendix B. As a result, the two image prompts share the same target texture. Notably, $\hat{\mathbf{I}}$ additionally provides guidance for the boundary regions that is consistent with the original view. They serve as image prompts for the following two stages: $\hat{\mathbf{I}}_{\text{loc}}$ guides local attribute editing, and $\hat{\mathbf{I}}$ guides boundary refinement.

Isolated Mesh Editing. We first construct an isolated local mesh $\mathcal{M}_{\text{loc}}$ defined by $\mathcal{P}$:

$$\mathcal{M}_{\text{loc}} = (V_{\mathcal{P}}, F_{\mathcal{P}}, \mathcal{U}, \mathcal{T}_{\mathcal{P}}), F_{\mathcal{P}} = \{f \in F \mid f \in \mathcal{P}\}, V_{\mathcal{P}} = \{v \in V \mid \exists f \in F_{\mathcal{P}},\, v \in f\}. \tag{3}$$

Here, $\mathcal{T}_{\mathcal{P}}$ is the texture map of the target part. Similarly, we denote the remaining mesh $\mathcal{M}_{\text{rem}} = (V_{\text{rem}}, F_{\text{rem}}, \mathcal{U}, \mathcal{T}_{\text{rem}})$, where $F_{\text{rem}} = F \setminus F_{\mathcal{P}}$ and $V_{\text{rem}} = \{v \in V \mid \exists f \in F_{\text{rem}}, v \in f\}$. We then use $\hat{\mathbf{I}}_{\text{loc}}$ as the editing prompt, apply the TRELLIS.2 texture generation module to edit $\mathcal{M}_{\text{loc}}$, and obtain the edited local mesh $\hat{\mathcal{M}}_{\text{loc}}$ with its texture map $\hat{\mathcal{T}}_{\mathcal{P}}$. Notably, $\hat{\mathcal{T}}_{\mathcal{P}}$ is generated based on the extracted part rather than the full asset, preventing the generated texture from being affected by non-target context. Since $\hat{\mathcal{M}}_{\text{loc}}$, $\mathcal{M}_{\text{rem}}$, and $\mathcal{M}$ share the same UV map, we can easily integrate $\hat{\mathcal{T}}_{\mathcal{P}}$ into $\mathcal{M}$ and obtain the first-stage result $\mathcal{M}^{(1)}$ as follows:

$$\mathcal{M}^{(1)} = (V, F, \mathcal{U}, \mathcal{T}_{\text{rem}} \cup \hat{\mathcal{T}}_{\mathcal{P}}), \tag{4}$$

Boundary Refinement. In practice, $\mathcal{P}$ can be inaccurate around the boundary, so $\mathcal{M}^{(1)}$ may result in inconsistencies between the edited part and the non-edited regions. Therefore, we further refine the boundary region in the second stage. We define the boundary region as the mesh faces on both sides of the stitching interface between $\mathcal{M}_{\text{loc}}$ and $\mathcal{M}_{\text{rem}}$. We then dilate these boundary faces on the mesh graph to obtain a narrow boundary band $\mathcal{B}$. Implementation details of the dilation procedure are provided in Appendix A. Given the whole-view reference $\hat{\mathbf{I}}$, $\mathcal{B}$ is then inpainted using an O-Voxel replacement strategy. Specifically, during texture slat denoising, we decode the current slats to O-Voxels, replace non-boundary O-Voxels with those from $\mathcal{M}^{(1)}$, and re-encode them for the next denoising step. Finally, this stage generates $\mathcal{M}^{(2)}$ with a refined boundary band. By restoring O-Voxels outside the boundary band at each denoising step, our O-Voxel replacement strategy spatially constrains the refinement to the boundary neighborhood while preserving the remaining regions.

4 Experiments

4.1 Experimental Setup

Datasets. Target-part extraction is evaluated on Part-Objaverse-Tiny (Yang et al., 2024), an open-world 3D part segmentation benchmark curated from Objaverse (Deitke et al., 2023). Part-Objaverse-Tiny contains 200 complex meshes with fine-grained part annotations. To construct user guidance, we render each target part from a random view to obtain a 2D mask, which is then used as input to each method. Predicted results are evaluated against the dataset ground truth.

For local editing, we build a dataset to assess the local editing capabilities of current methods within a real 3D generation pipeline. In this pipeline, the meshes can be artist-generated or obtained through different 3D generation methods. Therefore, we collected 100 meshes from different sources. Specifically, the dataset includes 70 meshes obtained from recent 3D generation systems, including Hunyuan3D 2.5 (Tencent Hunyuan3D Team, 2025), Hitem3D (Hi3D, 2026), Hyper3D Rodin (Deemos Tech, 2026), Tripo (Tripo AI, 2026), and TRELLIS.2 (Xiang et al., 2026), and 30 textured assets collected from Sketchfab (Sketchfab, 2026). The assets cover diverse categories including realistic objects, characters, animals, and daily-use items. To evaluate attribute-specific editing, we provide editing instructions for each asset, including both texture-only editing and shape manipulation. We display some examples from the dataset in Fig.1.

Baselines. For target-part extraction, we compare our mask-guided *Iterative Part Selection* strategy with P3-SAM (Ma et al., 2026) under two settings, free segmentation with mask matching and point-prompted segmentation, and with SAMPart3D (Yang et al., 2024). Specifically, for P3-SAM free segmentation, we first obtain part candidates automatically and then select the best-matching candidates based on their projection overlap with the user mask. For point-prompted segmentation, we sample points from the visible target region and use them to prompt P3-SAM for part segmentation. We sample 10 prompt points for each part and measure the average metrics between the predicted masks for all prompts of all parts and their corresponding ground truth masks.

Texture editing is compared against representative 3D editing methods, including VoxHammer (Li et al., 2026), 3DEditFormer (Xia et al., 2026), Nano3D (Ye et al., 2026), and Instant3dit (Barda et al., 2025). In addition, we compare with the texture generation method Hunyuan3D 2.1 (Team Hunyuan3D et al., 2025) for texture editing. We compare shape manipulation against VoxHammer, VecSet-Edit (Hsiao et al., 2026), 3DEditFormer, Nano3D, and Instant3dit. Notably, since VoxHammer requires an explicit 3D mask, we provide the target region extracted by our target-part extraction module to ensure that the comparison focuses on the editing stage. In addition, VecSet-Edit uses tex-



Figure 3: Qualitative comparison on local editing. The top group shows texture-only edits, and the bottom group shows joint shape-and-texture edits. Each result is rendered from two views. Baselines often leak edits to non-target regions, under-edit the target part, or introduce global appearance and shape drift. Inlay3D produces more faithful target edits across views while better preserving the surrounding asset regions. Zoom in for a better view.

ture generation methods such as MV-Adapter (Huang et al., 2025) to generate textures for the edited mesh, so we do not include VecSet-Edit in the texture-editing comparison.

Metrics. Target-part extraction is measured by mIoU between predicted regions and ground-truth regions. Texture-editing quality is measured by CLIP-T and CLIP-Direction (Radford et al., 2021) on local renderings around the user camera. FID (Heusel et al., 2017) and KID (Bińkowski et al., 2018) quantify rendering distribution drift between edited and source assets. Non-target preservation is measured by SSIM (Wang et al., 2004), PSNR, and LPIPS (Zhang et al., 2018) on canonical-view renderings restricted to the non-target region. We evaluate non-target geometry preservation in shape manipulation using Chamfer Distance (CD) (Fan et al., 2017), F-score (Tatarchenko et al., 2019), and normal consistency, all computed on the non-target region. To measure editing fidelity, we provide GPT-5.6-sol (**?**) with four fixed views of the edited mesh rendered with a uniform white material and no textures, and ask it to assign a score from 0 to 10. The score rewards clear target-part geometry changes and multi-view consistency, and penalizes wrong edits and severe artifacts. The complete GPT evaluation details are provided in Appendix D.

4.2 Qualitative Comparison

Figure 3 provides qualitative comparisons. For texture editing, Hunyuan3D 2.1-Paint and Nano3D lack precise local control and obtain incomplete results. For example, the blue roof and red armor targets largely retain their original appearance. 3DEditFormer causes noticeable geometry and appearance drift. In the roof sample, the pillars appear blurred, and the roof bricks have numerous artifacts. VoxHammer tends to under-edit the target texture, leaving the roof tiles and forearm armor less faithful to the prompt. All baselines struggle to preserve non-target regions. As seen in the lower left corner of the first sample, the red and white striped fabric texture is corrupted in these methods. For joint shape-and-texture editing, all baselines keep the original target shape, as shown by the wheel and bag-handle examples. They also produce unstable materials or distort nearby structures.

Table 1: Target-part extraction results on Part-Objaverse-Tiny. We report mIoU (%) between extracted mesh faces and ground-truth target-part faces.

Category	**Ours**	P3-SAM Free Seg	P3-SAM Point Prompt	SAMPart3D
Human-Shape	**80.81**	45.05	21.53	46.65
Animals	**74.68**	49.04	14.15	32.98
Daily-Used	**74.49**	43.70	17.29	33.47
Buildings & Outdoor	**69.92**	43.39	17.30	39.26
Transportations	**70.07**	43.76	8.48	34.59
Plants	**66.33**	42.73	21.02	34.90
Food	**76.95**	43.79	29.03	43.38
Electronics	**81.13**	50.64	20.56	41.40
Average	**74.30**	45.26	18.67	38.33

Table 2: Texture editing comparison. We compare against Hunyuan3D 2.1 and representative 3D editing baselines. CLIP scores are multiplied by 100, and KID is multiplied by 1000.

Metric	**Ours**	Full Mesh Inpainting	Hunyuan3D-2.1	VoxHammer	3DEditFormer	Nano3D	Instant3dit
CLIP-T ↑	**23.20**	22.84	22.80	21.60	21.40	21.00	19.00
CLIP-Dir. ↑	**6.55**	5.27	5.16	2.34	1.94	1.22	0.00
FID ↓	**54.15**	58.09	65.01	96.82	115.76	86.25	174.44
KID ↓	**0.05**	0.158	1.02	5.91	12.41	4.86	53.68
SSIM ↑	**0.96**	0.93	0.92	0.84	0.81	0.82	0.81
PSNR ↑	**28.30**	24.75	24.00	18.47	16.52	17.40	14.59
LPIPS ↓	**0.040**	0.068	0.075	0.116	0.160	0.152	0.206

Table 3: Shape manipulation comparison. CD, F-score, and normal consistency measure non-target geometry preservation, while the GPT-based Editing Score measures target-edit fidelity. VoxHammer is provided with our extracted 3D mask. CD is multiplied by 1000. F-score and normal consistency are reported as percentages.

Metric	**Ours**	VoxHammer	VecSet-Edit	3DEditFormer	Nano3D	Instant3dit
CD $\times 1000$ ↓	**4.25**	4.82	8.92	9.95	12.03	110.00
F-score ↑	**93.57**	93.24	82.34	77.19	11.97	10.52
Normal Consistency ↑	**90.80**	88.70	82.16	82.70	85.70	51.08
GPT Editing Score ↑	**4.80**	3.61	3.85	4.15	3.54	2.30

In comparison, Inlay3D applies the requested roof, armor, wheel, and handle edits more faithfully across views while better preserving the rest of the asset.

4.3 Quantitative Comparison

As shown in Table 1, Inlay3D achieves an average mIoU of 74.30%, outperforming the strongest baseline on average, P3-SAM Free Segmentation, by 29.04 percentage points, corresponding to a 64.2% relative improvement. The advantage is consistent across all eight object categories: compared with the strongest competing result in each category, the improvement ranges from 23.60 percentage points on Plants to 34.16 percentage points on Human-Shape. These results indicate that recursively refining and merging part candidates is more reliable than selecting from a fixed segmentation or using independently sampled point prompts.

Table 2 shows that Inlay3D improves both target-edit fidelity and non-target preservation. Compared with Hunyuan3D 2.1, Inlay3D improves CLIP-T from 22.80 to 23.20 and CLIP-Direction from 5.16 to 6.55, with the latter corresponding to a 26.9% relative improvement. At the same time, it reduces FID from 65.01 to 54.15 (16.7%) and KID from 1.02 to 0.05 on the reported scale. For non-target regions, Inlay3D improves SSIM from 0.92 to 0.96 and PSNR by 4.30 dB, while reducing LPIPS from 0.075 to 0.040, a 46.7% relative reduction. Although the gain in CLIP-T is relatively modest, the larger improvement in CLIP-Direction together with the preservation gains shows that Inlay3D strengthens local editing without introducing additional appearance drift in surrounding regions.

As reported in Table 3, VoxHammer is the strongest competing method in terms of non-target geometry preservation. Compared with VoxHammer, Inlay3D reduces CD from 4.82 to 4.25 (11.8%), increases F-score by 0.33 percentage points, and improves normal consistency by 2.10 percentage points. More importantly, its GPT Editing Score increases from 3.61 to 4.80, an absolute gain of 1.19 points, indicating that the improved preservation does not come from suppressing the requested shape change. Compared with 3DEditFormer, which obtains the second-highest GPT Editing Score

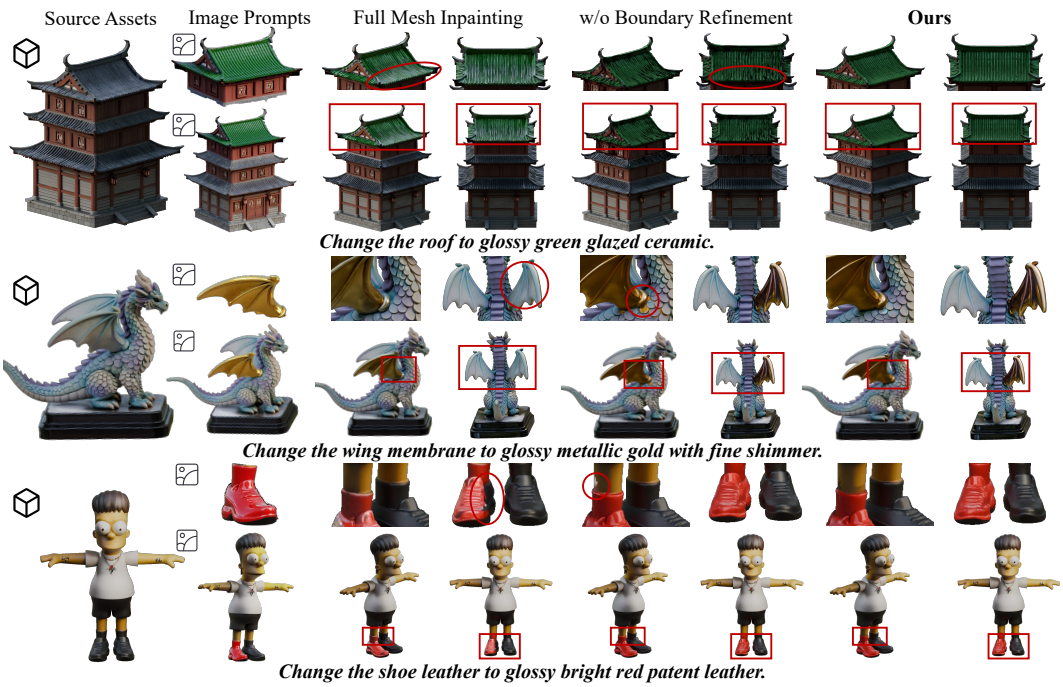


Figure 4: Qualitative ablation of two-stage texture editing, comparing three variants from two views with enlarged boundary regions. Full-mesh inpainting produces incomplete or view-inconsistent edits. Isolated Mesh Editing without Boundary Refinement leaves interface seams. The complete Inlay3D produces coherent target textures and clean transitions while preserving non-target regions. Red boxes mark enlarged boundary regions, and red ellipses highlight artifacts.

of 4.15, Inlay3D improves the editing score by 0.65 points while reducing CD by 57.3% and increasing F-score and normal consistency by 16.38 and 8.10 percentage points, respectively. However, the absolute GPT Editing Score of 4.80 out of 10 indicates that geometric editing quality remains limited. Some fine-grained editing results are barely noticeable.

4.4 Ablation Studies

Full Mesh Inpainting. This variant applies TRELLIS.2 inpainting directly to the full asset. As shown in Table 2, the complete Inlay3D improves CLIP-Direction from 5.27 to 6.55 and non-target PSNR from 24.75 to 28.30. These improvements indicate better target-edit fidelity and non-target preservation with our two-stage design. Figure 4 further shows that full-mesh inpainting produces incomplete or inconsistent textures on the roof tiles, dragon wing, and boot.

Boundary Refinement. We then remove Boundary Refinement while retaining Isolated Mesh Editing. The current metrics do not reliably capture localized seam artifacts or their correction, so we report only qualitative results. As shown in Figure 4, omitting refinement leaves visible artifacts along the roof edge, wing root, and boot cuff. Boundary Refinement removes these artifacts and improves transitions between the edited part and the surrounding asset.

5 Conclusion

We presented Inlay3D, a training-free framework for part-level texture and shape editing of textured meshes. *Target Part Extraction* converts a single-view 2D mask into an explicit 3D target region through recursive part selection. *Isolated Mesh Editing* reduces interference from non-target regions, while *Boundary Refinement* repairs interface artifacts and preserves the surrounding asset. Experiments demonstrate accurate target localization, high-quality edits, and strong non-target preservation, supporting controllable editing of 3D assets.

6 AI USE STATEMENT

We used generative AI to generate synthetic 3D assets and editing examples, assist with mathematical claims and proofs, implement parts of the method, and translate, draft, and edit manuscript text. We did not use it to develop theoretical or conceptual frameworks, propose or refine hypotheses, design or critique methods or experiments, clean or reformat datasets, conduct qualitative or thematic analysis, or interpret results. We reviewed all AI-assisted content by inspecting generated assets, independently checking mathematical derivations, reviewing and testing code, and verifying text against the original meaning and experimental evidence. The authors take full responsibility for the final text, claims, code, data, and other artifacts.

REFERENCES

Amir Barda, Matheus Gadelha, Vladimir G. Kim, Noam Aigerman, Amit H. Bermano, and Thibault Groueix. Instant3dit: Multiview inpainting for fast editing of 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16273–16282, 2025.

Mikołaj Bińkowski, Danica J. Sutherland, Michael Arbel, and Arthur Gretton. Demystifying mmd gans. In *International Conference on Learning Representations*, 2018.

Hansheng Chen, Ruoxi Shi, Yulin Liu, Bokui Shen, Jiayuan Gu, Gordon Wetzstein, Hao Su, and Leonidas Guibas. Mvedit: Generic 3d diffusion adapter using controlled multi-view editing. *arXiv preprint arXiv:2403.12032*, 2024a.

Yiwen Chen, Zilong Chen, Chi Zhang, Feng Wang, Xiaofeng Yang, Yikai Wang, Zhongang Cai, Lei Yang, Huaping Liu, and Guosheng Lin. Gaussianeditor: Swift and controllable 3d editing with gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024b.

Deemos Tech. Hyper3d rodin: Ai 3d model generator, 2026. URL `https://hyper3d.ai/`. Accessed: 2026-07-25.

Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

Haoqiang Fan, Hao Su, and Leonidas J. Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 605–613, 2017.

Inbar Gat, Dana Cohen-Bar, Guy Levy, Elad Richardson, and Daniel Cohen-Or. ShapeUP: Scalable image-conditioned 3d editing. In *ACM SIGGRAPH 2026 Conference Papers*, pp. 12:1–12:11, 2026. doi: 10.1145/3799902.3811073.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, 2017.

Hi3D. Hi3d: Ai-powered 3d model generator, 2026. URL `https://www.hi3d.ai/`. Accessed: 2026-07-25.

Teng-Fang Hsiao, Bo-Kai Ruan, Yu-Lun Liu, and Hong-Han Shuai. Vecset-edit: Unleashing pre-trained lrm for mesh editing from single image. In *ACM SIGGRAPH 2026 Conference Papers*, pp. 10:1–10:12, 2026.

Zehuan Huang, Yuan-Chen Guo, Haoran Wang, Ran Yi, Lizhuang Ma, Yan-Pei Cao, and Lu Sheng. Mv-adapter: Multi-view consistent image generation made easy. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 16377–16387, 2025.

Vladimir Kulikov, Matan Kleiner, Inbar Huberman-Spiegelglas, and Tomer Michaeli. Flowedit: Inversion-free text-based editing using pre-trained flow models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 19721–19730, 2025.

Lin Li, Zehuan Huang, Haoran Feng, Gengxiong Zhuang, Rui Chen, Chunchao Guo, and Lu Sheng. Voxhammer: Training-free precise and coherent 3d editing in native 3d space. In *Proceedings of the International Conference on 3D Vision (3DV)*, pp. 1281–1292, 2026. doi: 10.1109/3DV69130.2026.00126.

Minghua Liu, Mikaela Angelina Uy, Donglai Xiang, Hao Su, Sanja Fidler, Nicholas Sharp, and Jun Gao. Partfield: Learning 3d feature fields for part segmentation and beyond. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9704–9715, 2025.

Changfeng Ma, Yang Li, Xinhao Yan, Jiachen Xu, Yunhan Yang, Chunshi Wang, Zibo Zhao, Yanwen Guo, Zhuo Chen, and Chunchao Guo. P3-sam: Native 3d part segmentation. In *Proceedings of the European Conference on Computer Vision*, 2026.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *Proceedings of the International Conference on Machine Learning*, 2021.

Etai Sella, Gal Fiebelman, Peter Hedman, and Hadar Averbuch-Elor. Vox-e: Text-guided voxel editing of 3d objects. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

Etai Sella, Hao Phung, Nitay Amiel, Or Litany, Or Patashnik, and Hadar Averbuch-Elor. Prox-E: Fine-grained 3d shape editing via primitive-based abstractions. In *ACM SIGGRAPH 2026 Conference Papers*, 2026. doi: 10.1145/3799902.3811141.

Sketchfab. Sketchfab, 2026. URL `https://sketchfab.com/`. Accessed: 2026-07-25.

Maxim Tatarchenko, Stephan R. Richter, Rene Ranftl, Zhuwen Li, Vladlen Koltun, and Thomas Brox. What do single-view 3d reconstruction networks learn? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3405–3414, 2019.

Team Hunyuan3D, Shuhui Yang, Mingxin Yang, Yifei Feng, Xin Huang, Sheng Zhang, Zebin He, Di Luo, Haolin Liu, Yunfei Zhao, et al. Hunyuan3d 2.1: From images to high-fidelity 3d assets with production-ready pbr material. *arXiv preprint arXiv:2506.15442*, 2025.

Tencent Hunyuan3D Team. Hunyuan3d 2.5: Towards high-fidelity 3d assets generation with ultimate details. *arXiv preprint arXiv:2506.16504*, 2025. URL `https://arxiv.org/abs/2506.16504`.

Tripo AI. Tripo 3d: Ai 3d model generator, 2026. URL `https://www.tripo3d.ai/`. Accessed: 2026-07-25.

Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4): 600–612, 2004.

Chenfei Wu, Jiahao Li, Jingren Zhou, Junyang Lin, Kaiyuan Gao, Kun Yan, Shengming Yin, Shuai Bai, Xiao Xu, Yilei Chen, et al. Qwen-image technical report. *arXiv preprint arXiv:2508.02324*, 2025.

Ruihao Xia, Yang Tang, and Pan Zhou. Towards scalable and consistent 3d editing. In *Proceedings of the International Conference on Machine Learning*, 2026.

Jianfeng Xiang, Xiaoxue Chen, Sicheng Xu, Ruicheng Wang, Zelong Lv, Yu Deng, Hongyuan Zhu, Yue Dong, Hao Zhao, Nicholas Jing Yuan, and Jiaolong Yang. Native and compact structured latents for 3d generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14419–14429, 2026.

Yunhan Yang, Yukun Huang, Yuan-Chen Guo, Liangjun Lu, Xiaoyang Wu, Edmund Y. Lam, Yan-Pei Cao, and Xihui Liu. Sampart3d: Segment any part in 3d objects. *arXiv preprint arXiv:2411.07184*, 2024.

Junliang Ye, Shenghao Xie, Ruowen Zhao, Zhengyi Wang, Hongyu Yan, Wenqiang Zu, Lei Ma, and Jun Zhu. Nano3d: A training-free approach for efficient 3d editing without masks. In *International Conference on Learning Representations*, 2026.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Zhe Zhu, Le Wan, Rui Xu, Yiheng Zhang, Honghua Chen, Zhiyang Dou, Cheng Lin, Yuan Liu, and Mingqiang Wei. PartSAM: A scalable promptable part segmentation model trained on native 3d data. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=y8sZUQPYXC.

Jingyu Zhuang, Chen Wang, Liang Lin, Lingjie Liu, and Guanbin Li. Dreameditor: Text-driven 3d scene editing with neural fields. In *SIGGRAPH Asia 2023 Conference Papers*, pp. 26:1–26:10, 2023. doi: 10.1145/3610548.3618190.

APPENDIX

Algorithm 1 Target Part Extraction

Require: Textured mesh $\mathcal{M} = (V, F, \mathcal{U}, \mathcal{T})$, user camera π, binary mask $\mathbf{m}$, 3D part segmenter Φ, maximum depth D
Ensure: Target part $\mathcal{P}$

Render $\mathbf{I} = \mathcal{R}(\mathcal{M}, \pi)$ and store the face-id map $\mathbf{F}_\pi$
Obtain initial part groups $\mathcal{C} = \{C_i\} \leftarrow \Phi(\mathcal{M})$
$\mathcal{Q} \leftarrow \{(C_i, 0) \mid C_i \in \mathcal{C}\}, \quad \mathcal{A} \leftarrow \emptyset$
while $\mathcal{Q}$ is not empty **do**
    Pop a candidate (C_i, d) from $\mathcal{Q}$
    Render its visible projection $\mathbf{p}_i = \mathcal{R}_{\text{mask}}(C_i, \pi)$ using $\mathbf{F}_\pi$
    Compute $s_{\text{cover}}(C_i) = \frac{|\mathbf{p}_i \cap \mathbf{m}|}{|\mathbf{p}_i| + \epsilon}$
    if $|\mathbf{p}_i \cap \mathbf{m}| < \eta$ **then**
        Reject C_i
    else if $s_{\text{cover}}(C_i) \geq \tau_{\text{cover}}$ **then**
        $\mathcal{A} \leftarrow \mathcal{A} \cup \{C_i\}$
    else if $d < D$ **then**
        Sub-segment the candidate into finer parts: $\mathcal{C}_i = \{C_{i,j}\} \leftarrow \Phi(\mathcal{M}|_{C_i})$
        Map each $C_{i,j}$ back to the face indices of $\mathcal{M}$
        if the split produces valid finer parts **then**
            for all $C_{i,j}$ with $|\mathcal{R}_{\text{mask}}(C_{i,j}, \pi) \cap \mathbf{m}| \geq \eta$ **do**
                Push $(C_{i,j}, d+1)$ into $\mathcal{Q}$
            end for
        end if
    end if
end while
return $\mathcal{P} = \bigcup_{C_i \in \mathcal{A}} C_i$

A DETAILS OF TARGET PART EXTRACTION AND BOUNDARY EXTRACTION

Target Part Extraction. The complete Target Part Extraction procedure is provided in Algorithm 1. We use P3-SAM (Ma et al., 2026) as the part segmenter and the resolution of the rendered face-id map is 1024x1024. In practice, we set $\eta = 10$, $\tau_{\text{cover}} = 75\%$, and $D = 3$.

Boundary Extraction. The extracted target mask $\mathcal{P}$ also defines the interface from which we construct the mask for *Boundary Refinement*. Following the notation in the main text, we partition the input mesh faces into $F_{\mathcal{P}}$ and $F_{\text{rem}} = F \setminus F_{\mathcal{P}}$. We first identify edges shared by a target face and a remaining face, and collect the incident faces on both sides as the initial boundary region. Expanding this region yields the boundary band $\mathcal{B}$ used for refinement.

For distance-based expansion, we center the input mesh at its bounding-box center and normalize its longest side length to one. We then include any face with at least one vertex less than τ_{b} away from a vertex of the initial boundary region in Euclidean distance. The default threshold is $\tau_{\text{b}} = 0.02$, corresponding to 2% of the input mesh's longest bounding-box side. Alternatively, we expand the initial region by k face-adjacency steps using breadth-first search. Disabling expansion retains only the initial boundary faces.

The expansion radius τ_{b} (or depth k) controls the extent of refinement. After inspecting the first-stage editing result $\mathcal{M}^{(1)}$, users can adjust this parameter to enlarge $\mathcal{B}$ until it covers all visible artifacts around the stitching interface. We recommend using the smallest band that covers these artifacts to limit changes to otherwise satisfactory regions during the second stage.

B DETAILS OF LOCAL REFERENCE GENERATION

As described in the main paper, local reference generation requires determining the part-centric camera π_{loc} and the affine transformation $\mathcal{A}$ between the original and local views. We detail these two steps below.

Local Camera. Let $\mathbf{c}_{\mathcal{M}}$ be the center of the input mesh and $\mathbf{c}_{\mathcal{P}}$ be the bounding-box center of $\mathcal{P}$. We write the user camera as $\pi = (\mathbf{e}, \mathbf{c}_{\mathcal{M}}, \mathbf{r}, \mathbf{u}, \mathbf{v}, s)$, where $\mathbf{r}$, $\mathbf{u}$, and $\mathbf{v}$ denote its right, up, and viewing directions, respectively, and s is its orthographic scale. The local camera is placed to look at $\mathbf{c}_{\mathcal{P}}$ while preserving the orientation of π:

$$\pi_{\mathrm{loc}} = (\mathbf{c}_{\mathcal{P}} - \rho_{\mathrm{loc}}\mathbf{v}, \mathbf{c}_{\mathcal{P}}, \mathbf{r}, \mathbf{u}, \mathbf{v}, s_{\mathrm{loc}}) \,. \tag{5}$$

Here, ρ_{loc} only specifies the camera distance and does not affect an orthographic projection. Let $V_{\mathcal{P}}$ denote the vertices of $\mathcal{P}$. Its local orthographic scale is determined by its extent in the camera plane:

$$s_{\mathrm{loc}} = \max_{\mathbf{k}\in\{\mathbf{r},\mathbf{u}\}} \left(\max_{\mathbf{x}\in V_{\mathcal{P}}} \langle \mathbf{x}, \mathbf{k} \rangle - \min_{\mathbf{x}\in V_{\mathcal{P}}} \langle \mathbf{x}, \mathbf{k} \rangle \right). \tag{6}$$

We then use the local camera to render the local view of the source mesh as $\mathbf{I}_{\mathrm{loc}} = \mathcal{R}(\mathcal{M}, \pi_{\mathrm{loc}})$.

Affine Transformation. Since π and π_{loc} share the same orientation, their image coordinates differ only by a uniform scaling and an image-plane translation. For simplicity, assume that both views have the same square resolution $W \times W$, and let $\mathbf{z}_0 = (W/2, W/2)^\top$ be the image center. The camera displacement in the image plane is

$$\Delta_r = \langle \mathbf{c}_{\mathcal{P}} - \mathbf{c}_{\mathcal{M}}, \mathbf{r} \rangle, \qquad \Delta_u = \langle \mathbf{c}_{\mathcal{P}} - \mathbf{c}_{\mathcal{M}}, \mathbf{u} \rangle. \tag{7}$$

Thus, a pixel $\mathbf{z}$ in the original view is mapped to the local view by

$$\mathcal{A}(\mathbf{z}) = \frac{s}{s_{\mathrm{loc}}}(\mathbf{z} - \mathbf{z}_0) + \mathbf{z}_0 + \frac{W}{s_{\mathrm{loc}}} \begin{bmatrix} -\Delta_r \\ \Delta_u \end{bmatrix}. \tag{8}$$

The signs in the translation term account for the downward-pointing vertical pixel axis. We use this transformation to obtain $\mathbf{m}_{\mathrm{loc}} = \mathcal{A}(\mathbf{m})$, and use $\mathcal{A}^{-1}$ to map the edited local image prompts back to the original view, as described in the main paper.

C Details of Two-Stage Shape Manipulation

Shape manipulation follows the same local-reference generation and two-stage editing framework as texture editing. We only describe the shape-specific changes here.

Isolated Geometry Generation. Unlike texture editing, which keeps the geometry of $\mathcal{M}_{\mathrm{loc}}$ fixed, shape manipulation regenerates the local geometry with the TRELLIS.2 shape model. However, directly regenerating the geometry can lead to slight deformation. This is because the occupied o-voxels consist of only a single surface layer, which limits the generation of large deformations. To mitigate this, we apply an inward dilation to the occupied o-voxels. This enables TRELLIS.2 to generate geometry in a thicker space. The edited local mesh is formulated as $\hat{\mathcal{M}}_{\mathrm{loc}} = (\hat{V}_{\mathcal{P}}, \hat{F}_{\mathcal{P}}, \hat{\mathcal{U}}_{\mathcal{P}}, \hat{\mathcal{T}}_{\mathcal{P}})$. It is inserted back into the source scene using the transformation defined by the bounding box of $\mathcal{M}_{\mathrm{loc}}$, and is then combined with $\mathcal{M}_{\mathrm{rem}}$:

$$\mathcal{M}^{(1)} = (V_{\mathrm{rem}} + \hat{V}_{\mathcal{P}}, F_{\mathrm{rem}} + \hat{F}_{\mathcal{P}}, (\mathcal{U}, \hat{\mathcal{U}}_{\mathcal{P}}), (\mathcal{T}_{\mathrm{rem}}, \hat{\mathcal{T}}_{\mathcal{P}})). \tag{9}$$

Here, since the UV map of the generated geometry may not be consistent with that of the original mesh, we directly use two UV maps and two texture maps to represent the edited mesh.

Geometry Boundary Refinement. For shape manipulation, the purpose of this stage is to blend the interface region between $\mathcal{M}^{(1)}$ and $\mathcal{M}_{\mathrm{rem}}$ from the first stage. Before isolated geometry generation, we determine the boundary region $\mathcal{B}$ from the input mesh using the same stitching-interface strategy described in the main paper. We reuse this precomputed boundary region, and execute the same O-Voxel replacement strategy as texture editing. The strategy ensures that generated geometry is retained only inside $\mathcal{B}$ and that non-boundary geometry is restored from $\mathcal{M}^{(1)}$. To blend the UV maps and obtain a single texture map, we conduct UV unwrapping and resample a new texture map. Notably, the texture attributes of the non-edited regions are unchanged during this two-stage editing. As a result, we achieve precise manipulation of the mesh shape, while preserving the geometry and texture of the non-edited region.

D Details of the GPT-Based Editing Score

Evaluation purpose. Open-ended shape-editing instructions generally do not have a unique ground-truth geometry. We therefore use GPT-5.6-sol (**?**) to evaluate whether the requested edit

is successfully reflected in the target part. This score measures target-edit fidelity and multi-view consistency. For each edited asset, we render the mesh with a uniform white material and no textures from the same four fixed camera views for every method. These white-mesh renderings focus the evaluation on geometry rather than texture or color changes. We arrange them in a 2×2 contact sheet. Each cell is 512×512 pixels, resulting in a 1024×1024 input image. The evaluator is instructed to judge only whether the rendered asset follows the given editing instruction. It rewards a clear and correct geometric or semantic change in the target part and consistency of that change across views. It penalizes missing or weak edits, modification of the wrong part, cross-view inconsistency, severe geometric artifacts, and failed renderings. We use the following fixed rubric for all methods and examples:

> **10**: The edit is clearly and correctly applied to the target region across views.
> **7**: The edit is mostly correct, with only minor artifacts or incomplete details.
> **5**: The result partially follows the instruction, but the edit is weak, ambiguous, or inconsistent.
> **2**: The result barely matches the instruction or severely damages the object.
> **0**: The result is unrelated to the instruction, the object is missing, or the rendering fails.

We compute the final Editing Score as the mean over all valid examples. The same views, prompt, rubric, temperature, and response parsing are used for every compared method.

E LIMITATIONS

Target Part Extraction relies on a single-view mask, leaving back-facing surfaces without direct supervision. Segmentation errors on these surfaces can cause incomplete edits or unintended changes to non-target regions. Additional views or visibility-aware uncertainty handling could improve extraction accuracy.

Boundary Refinement is sensitive to the expansion extent of $\mathcal{B}$. Even with an accurate interface, excessive expansion can cover most or all of a small or thin target part, altering satisfactory regions of $\mathcal{M}^{(1)}$. Users can adjust the expansion parameter, but balancing artifact coverage against preservation requires manual tuning. Adapting the band to local geometry and artifact extent could reduce this sensitivity.