FruitNinja: 3D Object Interior Texture Generation with Gaussian Splatting

Fangyu Wu   Yuhao Chen
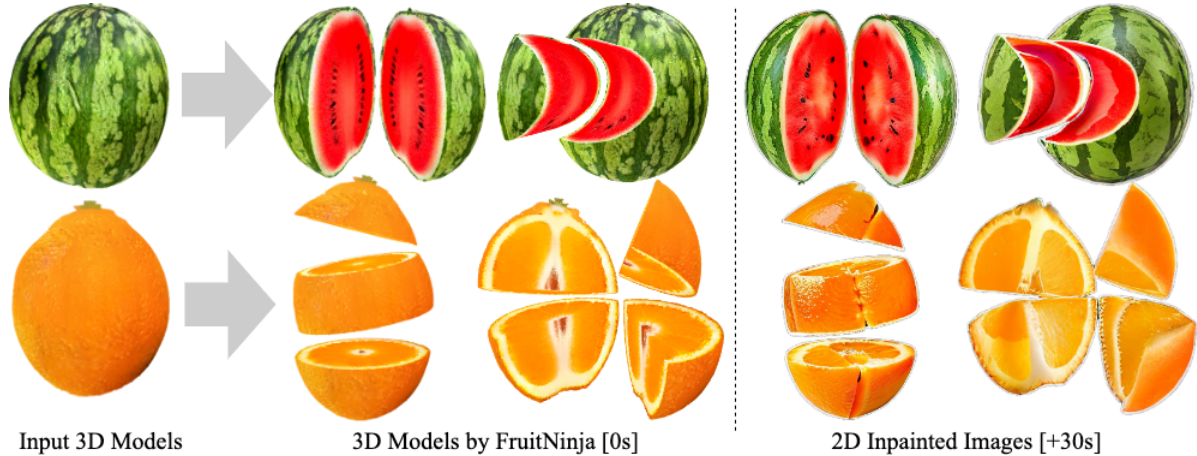
University of Waterloo

Figure 1. FruitNinja generates high-quality interior textures for 3DGS models, enabling real-time view rendering during arbitrary geometric transformations. In contrast, direct 2D inpainting requires additional optimization steps (∼30s), often misaligns edited geometry, and yields inconsistent results per edit.

Abstract

In the real world, objects reveal internal textures when sliced or cut, yet this behavior is not well-studied in 3D generation tasks today. For example, slicing a virtual 3D watermelon should reveal flesh and seeds. Given that no available dataset captures an object's full internal structure and collecting data from all slices is impractical, generative methods become the obvious approach. However, current 3D generation and inpainting methods often focus on visible appearance and overlook internal textures. To bridge this gap, we introduce FruitNinja, the first method to generate internal textures for 3D objects undergoing geometric and topological changes. Our approach produces objects via 3D Gaussian Splatting (3DGS) with both surface and interior textures synthesized, enabling real-time slicing and rendering without additional optimization. FruitNinja leverages a pre-trained diffusion model to progressively inpaint cross-sectional views and applies voxel-grid-based smoothing to achieve cohesive textures throughout the object. Our OpaqueAtom GS strategy overcomes 3DGS limitations by employing densely distributed opaque Gaussians, avoiding biases toward larger particles that destabilize training and sharp color transitions for fine-grained textures. Experimental results show that FruitNinja substantially outperforms existing approaches, showcasing unmatched visual quality in real-time rendered internal views across arbitrary geometry manipulations.

1. Introduction

Generating high-quality interactive 3D objects has diverse applications in fields such as augmented and virtual reality (AR/VR), digital gaming, and advertising. Recent advancements in 3D computer vision, particularly 3D Gaussian Splatting (3DGS) [18], have introduced efficient techniques for novel view synthesis. Leveraging its explicit point-cloud-based representation and suitability for post-editing tasks, researchers have further explored user-guided editing of 3DGS, including stylization [8, 20, 43], deformation[3, 13, 17, 39] , object removal [8, 41], inpainting [8, 23] and texture editing [25, 40]. In interactive 3D applications, it's common for users to perform customized or large-scale geometric modifications—such as cutting, tearing, or removing parts of an object's surface, which can reveal internal textures. If the internal Gaussians are not adequately trained, this may expose unrealistic internal struc-

tures, thereby compromising overall visual quality. Unfortunately, existing 3DGS editing frameworks primarily focus on manipulating an object's external appearance, while preserving the fidelity of object's internal textures remains under-explored.

Creating novel view synthesis of an object's internal structure, ensuring that the exposed internal texture appears realistic when sliced from any arbitrary angle, is particularly challenging due to scarcity of training data. Current 3D datasets predominantly emphasize the overall geometry and surface texture of objects [6, 9, 19, 36], often lacking details about their interiors. Acquiring data on internal structures typically requires specialized techniques such as X-ray scans, CT imaging, or compiling multiple cross-sectional images. Furthermore, when images are collected by disassembling objects (e.g., cutting objects in half), only partial internal structures are exposed. Reversing these manipulations to access other parts is often impractical, making it difficult to infer the complete internal structure.

Existing methods for texture generation in 3D objects [5, 7, 30] focus on generating surface textures for mesh UV maps, primarily addressing the object's external shell and therefore are not applicable for internal modeling in 3DGS. Some recent studies [13, 17] have employed ad-hoc inpainting on newly exposed regions after each editing step to mitigate visual artifacts. However, this per-edit approach can introduce inconsistencies across a series of transformations, and is unsuitable for real-time rendering. PhysGaussian introduced a mechanism designed to enhance the visual quality of 3D models under deformation by internally filling Gaussian particles[39]. This method discretizes the opacity field onto a 3D grid and uses ray casting to identify and fill void regions based on opacity thresholds, with each filled particle inheriting color and opacity from nearby surface Gaussian kernels. Nonetheless, this approach relies on the unrealistic assumption that internal textures resemble surface textures, while real-world objects often have different internal characteristics, causing inherited properties to fall short of capturing realistic details.

Fortunately, many common objects possess symmetrical features, allowing their cross-sectional views at consistent angles to appear similar. For instance, slicing a watermelon horizontally at different levels consistently reveals similar patterns of skin, flesh, and seeds. Building on this observation, we propose FruitNinja, an effective method for generating 3D internal textures by using only a few cross-sectional views as references. Our method synthesizes the entire interior texture of an object without requiring additional optimization after geometry edits or topology changes. FruitNinja leverages a pre-trained diffusion model to guide the synthesis of cross-sectional views using Score Distillation Sampling (SDS)[27], thereby jointly training cross-section and surface views. To address inconsistent ar-

tifacts from varying generated cross-sectional views, Fruit-Ninja progressively optimizes reference views and applies voxel-grid-based smoothing to seamlessly blend the overall texture. As real-world objects are composed of millions of atoms and molecules, we adopt the OpaqueAtom GS settings inspired by AtomGS [22]. This overcomes two key limitations of the original 3DGS algorithm: (1) the tendency to optimize larger Gaussians, which limits the density of small Gaussian particles needed for stable training and introduces artifacts during editing; and (2) limitations in the GS ray-marching method, where blending front and back Gaussians compromises modeling abrupt color transitions (e.g., white flesh adjacent to green skin or red flesh in watermelon slices).

Our contributions are summarized as follows:

- We introduce the first method for generating textures for object interiors by progressively inpainting cross-sectional views and applying voxel smoothing, enabling real-time rendering of internal views.
- We propose an OpaqueAtom GS strategy for modeling 3D objects with realistic interior textures, allowing arbitrary slicing to reveal fine-grained details while overcoming limitations of 3DGS, including instability from large Gaussians and difficulty with sharp color transitions.
- We demonstrate that the proposed method effectively synthesizes internal textures across common objects, achieving superior view quality during various geometric transformations compared to existing methods, both qualitatively and quantitatively.

2. Related Work

2.1. 3D Representations

Various 3D representations—such as point clouds [33], meshes [4], 3D voxel grids [24], and signed distance functions (SDFs) [26]—each offers unique advantages for 3D tasks. Recently, Neural Radiance Fields (NeRF) introduced a breakthrough by implicitly encoding 3D scenes through deep networks and volumetric rendering, achieving high-quality reconstructions. Building on this innovation, some research has focused on enabling customized geometry deformations based on NeRF. Methods such as NeRF-Editing [42], Neural Impostor[21], and NeuPhysics[28] integrate explicit mesh structures with implicit neural representations to facilitate intuitive and physically accurate manipulations of 3D models. Alternatively, 3D Gaussian Splatting [18] employs an explicit, point-cloud-based approach using Gaussian kernels, offering faster and more easily editable representations. Techniques like GaussianEditor[8], GSDeformer[14], and Mani-GS[10] enable user-guided texture and geometry transformations by leveraging Gaussian semantic tracing, cage-based deformation, and mesh adaptation. Additionally, PhysGaussians[39] and VR-
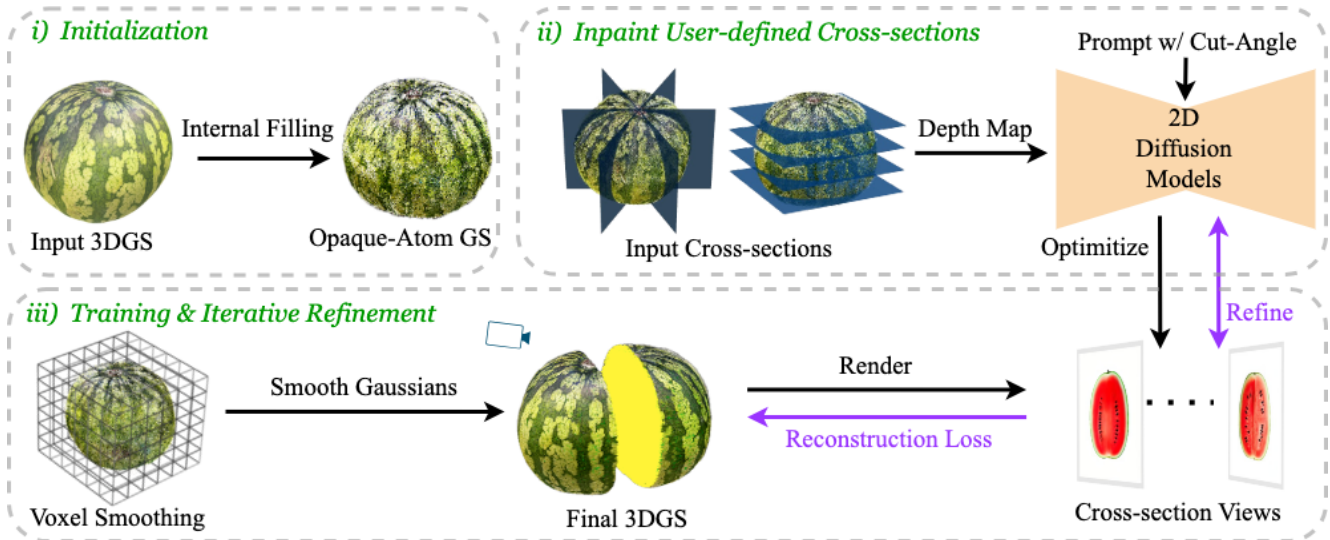
Figure 2. Method Overview. The input 3DGS is first transformed using opaque atomic Gaussians, and void regions within the object are filled with raw particles. For each user-defined cut angle, a reference cross-sectional view is generated via SDS. Subsequently, exterior Gaussians are iteratively masked out to render cross-sectional views, which are jointly trained with randomly selected external views. Each reference cross-sectional view is continuously refined by applying SDS to the existing rendering, while voxel smoothing is applied every N iterations to improve overall consistency.

GS[17] have introduced physics-based deformation methods based on 3DGS. AtomGS [22] enhances 3DGS by introducing Atomized Proliferation to densify small, uniform Gaussians in areas of fine detail, and Geometry-Guided Optimization to align Gaussians with scene geometry, reducing noise and sharpening edges. This approach enables high-fidelity rendering and precise geometry reconstruction, making it ideal for applications that require detailed textures. Inspired by these capabilities, we adopt an OpaqueAtom GS strategy to achieve high fidelity in fine-grained interior textures of 3D models.

2.2. 3D Inpainting

3D inpainting addresses the challenge of filling missing or masked regions in 3D spaces by generating plausible geometry and textures. Early inpainting works predominantly focus on either geometry completion or texture synthesis, often treating these aspects separately. For instance, methods like [38, 44] concentrate on reconstructing the underlying geometric structures, while others methods like [5, 7, 16, 30, 34] target the generation of realistic textures for the completed regions. Recent advancements have enabled the simultaneous inpainting of both semantic content and geometric structures, effectively addressing the interplay between these two components. NeRF-based approaches have leveraged features from models like CLIP[29] to learn and incorporate 3D semantics into the inpainting process, enhancing the contextual relevance and realism of the completed regions[37]. Additionally, sev-

eral methods have explored inpainting techniques within the framework of 3DGS, benefiting from GS's rendering efficiency and high-quality reconstruction capabilities [8, 13, 23]. However, previous inpainting methods are mainly designed for static scenes, which limits their effectiveness in dynamic settings where untrained interior areas may be exposed. In this paper, we propose a method focused on inpainting internal textures to overcome these limitations.

3. Method

In this section, we outline our approach for generating internal textures for 3DGS objects to ensure visual coherence under arbitrary geometric modifications. Our method involves three steps: first, we populate the interior of the 3D object with raw Gaussian particles (Section 3.1.2). Next, we apply SDS optimization to the input cross-sectional views to produce an initial set of cross-sectional reference views (Section 3.2). Then, we use these reference views, along with surface views, to jointly train the 3DGS model with OpaqueAtom GS settings (Section 3.4), iteratively refining the reference views set and applying voxel-based smoothing to enhance texture consistency (Section 3.3).

3.1. Preliminary

3.1.1 3D Gaussian Splatting

3D Gaussian splatting represents a 3D scene as a set of N Gaussians $\mathcal{G} = \{g_1, g_2, \ldots, g_N\}$. Each Gaussian g_i is

characterized by its position $\mathbf{x}_i \in \mathbb{R}^3$, a covariance matrix $\Sigma_i \in \mathbb{R}^{3\times3}$ that describes its shape and orientation, color coefficients $\mathbf{c}_i \in \mathbb{R}^k$, and an opacity value $\alpha_i \in \mathbb{R}$. To render the scene from a new viewpoint, each Gaussian is projected onto the camera plane by applying a viewing transformation, which includes adjusting its covariance matrix Σ_i to account for perspective. For each pixel, the contributions from overlapping Gaussians are blended. The influence σ_i of Gaussian g_i on a pixel is computed based on its opacity α_i and the Gaussian function evaluated at that pixel. The final pixel color C is obtained by front-to-back compositing of each Gaussian's contribution:

$$C = \sum_{i=1}^{N} \mathbf{c}_i \, \sigma_i \prod_{j=1}^{i-1} (1 - \sigma_j). \quad (1)$$

3.1.2 Internal Gaussians Initialization

Following the internal filling mechanism from PhysGaussian, we fill the 3D object's internal regions with Gaussian primitives. To identify empty internal regions, we first construct a continuous 3D opacity field $d(\mathbf{x})$ by summing the contributions of all Gaussians:

$$d(\mathbf{x}) = \sum_p \sigma_p \exp\left(-\frac{1}{2} (\mathbf{x} - \mathbf{x}_p)^\top \mathbf{\Sigma}_p^{-1} (\mathbf{x} - \mathbf{x}_p) \right). \quad (2)$$

This field is discretized onto a 3D grid, and candidate voxels for internal filling are identified where the opacity is below a predefined threshold σ_{th}. For each voxel, we cast rays along the six principal axes to detect transitions from low- to high-opacity regions, marking these voxels for internal filling. Each identified voxel is then initialized with a predefined number of Gaussian primitives with uniform color and opacity. We set each new Gaussian's covariance matrix to be spherical with a randomly assigned scale capped by a maximum value (e.g., 10^{-3} times the object's overall size).

3.2. Conditioned Cross-section Inpainting

As discussed in Section 1, obtaining comprehensive cross-sectional data across arbitrary cutting angles is challenging. Fortunately, cross-sectional images of objects from canonical cut angles is easier to acquire. For example, people commonly slice an orange horizontally or vertically, making these views more accessible. Our approach leverages these canonical cross-sectional views to generate the complete internal texture efficiently. Specifically, our method relies on a set of user-specified cutting angles, defined by a cutting plane

$$ax + by + cz + d = 0 \quad (3)$$

where x, y, z are 3D coordinates, and a, b, c, d define the plane's orientation and position. These user-defined cross-sections are categorized based on the intrinsic characteristics of the 3D object as illustrated in the Figure 3, which

allows us to apply different text conditions for the diffusion prior. We denote the set of user-defined cross-sectional views as $\{V_1, V_2, \ldots, V_k\}$, where each V_i corresponds to a specific cutting plane.
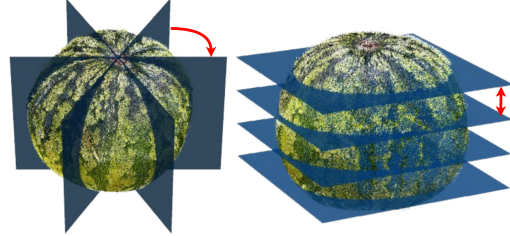

Figure 3. Example of user-defined cross-sections: Left: Vertical cross-sections through the watermelon center, spaced by a specified rotational angle. Right: Horizontal cross-sections, evenly spaced, slicing the watermelon at incremental depths. Note that these input cutting angles are required only during training; once trained, the 3DGS model can be sliced and rendered at arbitrary angles without additional optimization.

For each cut angle V_i defined by Equation 3, we create a corresponding 3D mask to retain only the Gaussian primitives close to the cutting plane. During rendering, we mask out the exterior and reveal the slice plane. To account for the varying geometry of rendered cross-sections, we first render RGB images and generate estimated depth maps using a pre-trained depth estimator[1]. These depth maps serve as additional conditioning input to a Stable Diffusion model. We observe that directly optimizing 3DGS parameters for rendered V_i using SDS loss is inefficient in the beginning. During early iterations, many internal Gaussians are untrained, resulting in rendered views that lack distinctive features and hinder SDS convergence. To address this issue, we adopt a two-stage optimization process. In the first stage, we independently perform multiple SDS optimizations on each rendered cross-sectional views $V = \{V_i \mid i = 1, \ldots, k\}$. For each V_i, we employ the depth-conditioned Stable Diffusion model guided by cut-angle specific text prompts (e.g., *"the horizontal cross-sectional view of a watermelon"*). This tailored prompting guides the diffusion model to generate images I_{label}^p that align more closely with the orientation and characteristics of each cross-section. The SDS loss can be formulated as:

$$\mathcal{L}_{\text{SDS}} = \mathbb{E}_{t,\epsilon} \left[w(t) \, \| \epsilon - \epsilon_\theta \left(\mathbf{I}_{\text{label}}^p + \sigma_t \epsilon, t, e, d \right) \|^2 \right], \quad (4)$$

where $\mathbf{I}_{\text{label}}^p$ is the current reference image at viewpoint p, $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ is Gaussian noise added at timestep t, σ_t is the noise level corresponding to timestep t, ϵ_θ is the noise predicted by the diffusion model parameterized by θ, e is the text embedding of the cut-angle specific prompt, d is the depth map corresponding to the cross-sectional view, and $w(t)$ is a weighting function that balances the contributions from different timesteps.

In the second stage, we use these optimized cross-sectional reference images to update the 3DGS parameters by minimizing the reconstruction loss:

$$\mathcal{L}_{\text{recon}} = \alpha \, \mathcal{L}_{\text{MSE}} + (1 - \alpha) \, \mathcal{L}_{\text{SSIM}}, \tag{5}$$

where α is a weighting factor, $\mathcal{L}_{\text{MSE}}$ is the Mean Squared Error between the rendered images I_{RGB}^{p} and the reference images I_{REF}^{p} and $\mathcal{L}_{\text{SSIM}}$ is the Structural Similarity loss [15] computed between the same images. This combined loss leverages both pixel-wise differences and perceptual similarities. To preserve the object's external appearance during optimization on cross-sectional views, we also randomly select 10–20 surface views rendered from initial input 3DGS and jointly train them with V.
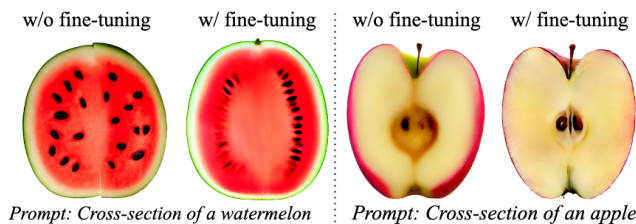

Figure 4. Images generated by stable-diffusion-2-depth[31]

Fine-tuning with DreamBooth We noticed that Stable Diffusion models often fail to generate high-quality cross-sectional images as shown in Figure 4, likely due to the scarcity of such images in the training data. Therefore, we optionally fine-tune the diffusion model using a small set (1–6) of cross-sectional images and the DreamBooth method [32]. Each image is paired with an angle-specific text prompt (e.g., *"A vertical cross-section of an object"*) to guide generation. As in DreamBooth, we apply a class-specific prior preservation loss to encourage diverse generation within each category.

3.3. Progressive Texture Refinement

Training 3DGS with reference cross-sectional views (see Section 3.2) may introduce spatial inconsistencies. For example, as shown in Figure 5, a vertical slice of a watermelon model might display a black seed at a specific location, while a horizontal slice through the same region shows only red flesh. To mitigate this, we employ an iterative refinement process that jointly optimizes rendering and generation, similar to the fine-tuning stage in Dream-Gaussians [35]. After each iteration, we render the current cross-sectional views $V = \{V_i \mid i = 1, \ldots, k\}$ from the trained 3DGS model, then we apply a few additional optimization steps for each reference cross-sectional view using SDS (Equation 4). We repeat this process until the reconstruction losses for all slices converge to below a predefined threshold ϵ. This iterative refinement ensures spatial consistency across the object's cross-sections by harmonizing
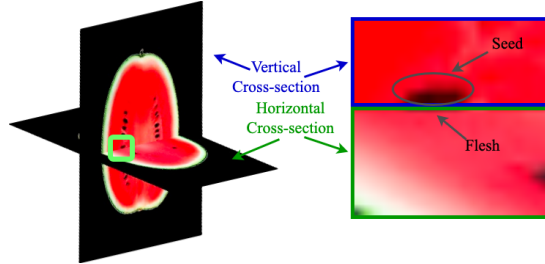

Figure 5. Within the intersected 3D region, the vertical cross-section (top) highlights a seed, while the horizontal cross-section (bottom) depicts the surrounding flesh, which introduces conflicting training signals.

textures from the trained 3DGS with features from the diffusion model. The algorithm converges when all slices are in agreement, effectively resolving inconsistencies introduced by independently optimized 2D references.

Voxel Smoothing Due to the discrete nature of input cross-sectional slices V, some Gaussian primitives are not covered by the generated masks in Section 3.2, leaving them untrained and potentially reducing visual fidelity when they are exposed. To avoid further optimization during rendering, we construct voxel grid over 3DGS and perform smoothing operation for each voxel cell during a predefined interval (e.g., 30–40 iterations). Specifically, untrained Gaussians are assigned colors using a distance-weighted average of nearby trained Gaussians:

$$C = \frac{\sum_i w_i \cdot C_i}{\sum_i w_i} \tag{6}$$

Here, C_i represents the color of each nearby trained Gaussian, and w_i is the inverse distance weight based on the Euclidean distance d_i between the untrained Gaussian and each trained Gaussian within the same voxel. The grid resolution is chosen empirically to maintain visual quality, ensuring that each voxel contains fewer than 1% of the total particles. This approach preserves fine texture details while promoting color consistency throughout the model.

3.4. Opaque Atomic Gaussian Particles

To ensure stable training and detailed texture representations, we employ the OpaqueAtom configuration, which incorporates two critical constraints: atomic clipping of Gaussian sizes and uniform high opacity across all Gaussians.

Atomic Clipping 3DGS often optimizes for larger Gaussian primitives during training. However, when these Gaussian particles grow excessively large, they may overlap multiple cross-sectional regions of the 3D object. Since cross-sectional views typically capture fine details in the object's texture, this overlap forces 3DGS to reconcile conflicting representations from various perspectives. Consequently, the training process becomes inefficient and may

Figure 6. Qualitative comparison with PhysGaussian and 2D inpainting using stable-diffusion-2-depth[31] (which requires ∼30s for 70 DDIM sampling steps per view). 3DGS from FruitNinja shows better texture consistency and visual quality without any optimization.

fail to converge. Besides, large Gaussians lack the spatial resolution needed to represent fine-grained textures accurately and limit precise geometry edits (e.g., slicing a single large Gaussian is not feasible). To counter these, we constrain Gaussian sizes following the approach inspired by AtomGS [22] during training. Specifically, we cap each Gaussian's scale at a minimal fraction (e.g., $1/3{,}000$) of the object's dimensions. This fine-grained control preserves texture details for stable training and allows more precise geometric edits.

Uniform Opacification 3DGS renders images by blending contributions from all Gaussians along each ray, including those behind the surface (see Section 3.1.1). While this approach efficiently models static 3D objects, it is not reliable in interactive scenarios where geometric changes or user edits alter the ordering and visibility of Gaussian particles dynamically. Particles in the back can unintentionally influence the foreground, leading to unrealistic color distributions. Thus, we assign full opacity to all Gaussian particles to maximize the contribution of front-most Gaussians to the rendered image. This adjustment accurately models opaque materials and ensures that each Gaussian particle faithfully reflects the real-world appearance of the material it represents during geometric deformation.

4. Experiment

4.1. Implementation Details

We build upon the original 3DGS implementation. For each object, we define one or two types of cross-sections (primarily horizontal and vertical) as shown in Figure 3. Horizontal slices are evenly spaced vertically, and vertical slices are arranged radially around the central axis at equal angular intervals. Each training iteration randomly

selects 20 surface appearance views rendered from the original 3DGS reconstruction. We use a $512 \times 512 \times 512$ 3D grid for voxel smoothing. For each reference view, we initially apply 20 SDS optimization steps for generation, followed by 3–4 refinement steps per iteration, as detailed in Section 3.3. Training typically required between 120 and 200 iterations. Optionally, we fine-tuned the stable diffusion model [31] using 1 to 4 collected cross-sectional views with DreamBooth[32], as specified in Section 3.2.

Dataset To evaluate our method, we collected a dataset comprising six common objects with internal textures distinct from their surface appearances. The dataset includes a watermelon, apple, orange, red velvet cake, loaf bread, and pomegranate. For each object, we captured 160–200 surface images from various angles for initial 3D reconstruction via 3DGS and 1-–4 cross-sectional images (horizontal and/or vertical) online which can be used for fine-tuning (as described in Section 3.2). We plan to release all datasets and code upon publication.
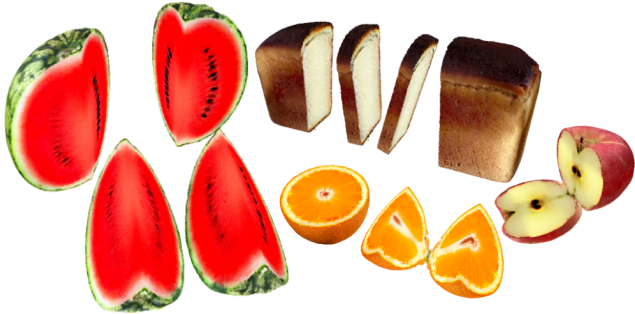

Figure 7. Extensive results. Each item is cut into multiple pieces arbitrarily, showcasing how the cross-sectional textures align consistently across slices.

4.2. Qualitative Comparisons

To the best of our knowledge, no prior work has specifically addressed the inpainting of an object's internal texture for interactive 3D editing use cases. Therefore, we compare our method to two baselines: (1) the internal filling logic from PhysGaussian and (2) a 2D inpainting approach using a depth-conditioned Stable Diffusion model [31], akin to the ad-hoc inpainting methods employed in VR-GS [17] and Infusion [23]. Specifically, we apply identical customized cuts to objects trained using both our method and the original PhysGaussian internal filling logic, then evaluate the exposed internal textures from two distinct camera angles. For the 2D inpainting baseline, we render views from these same camera angles without internal filling and inpaint the textures directly using the Stable Diffusion model with relevant text prompts, such as *'orange, partially cut showing internal structure and flesh'*.

As illustrated in Figure 1, the first column presents the input 3DGS, while the second column highlights the cut-off sections of each object. The results indicate that PhysGaussian's generated textures lack realism, appearing overly blurred and unnatural. Directly applying inpainting also struggles to produce faithful views for edited geometry structures, as aligning textures accurately with complex, user-edited shapes is challenging. For example, in the case of the apple, capturing the intricate details of the core, seeds, and uneven inner surface is difficult, especially with cuts made at arbitrary angles or depths. 2D inpainting can also lead to visual inconsistency, as seen with objects like a red velvet cake, where different angles reveal varying textures when inpainted. By comparison, our method generates textures that are highly aligned with the customized cut angles and preserve high visual fidelity. Additionally, 3D models generated by FruitNinja can be rendered in real-time while the ad-hoc inpainting method requires 70 diffusion sampling steps ($\sim$30s), which limits their use in interactive applications.

Figure 7 presents extensive results across additional customized geometry modifications, showing that our generated textures consistently maintain high visual fidelity under various cut angles.

4.3. Quantitative Evaluation

Table 1. Higher CLIP scores indicate better semantic alignment, while lower KID and FID scores signify superior texture fidelity.

Method	CLIP Score $\uparrow$	FID $\downarrow$	KID $\downarrow$ x10^{-3}
PhysGaussian	24.6	520.1	816.4
2D Inpainting (Fine-tuned)	32.3	**176.2**	**224.5**
2D Inpainting	25.1	314.2	536.3
Ours	**33.1**	209.2	323.7

Table 1 presents a quantitative comparison of our proposed method against PhysGaussian and a 2D ad-hoc inpainting approach, evaluated on only canonical cross-sections of 3D objects (e.g., horizontal and vertical slices for watermelon). For horizontal cross-sections, we apply random cuts aligned with the chosen orientation to extract the corresponding views. Vertical slices are arranged radially around the central axis at random angles (similar as Figure 3). We report the CLIP score[11] as evaluation metrics, which measures the compatibility between image-caption pairs using category-specific prompts, such as *'the vertical cross-section of a watermelon'*. Additionally, we use the collected real-world canonical cross-sections images mentioned in 4.1 to compute the average KID[2] and FID[12] scores for each rendered view per object. Our method achieves the highest CLIP scores, demonstrating superior semantic alignment with the intended cross-sectional prompts and more accurate texture generation. Moreover,

the KID and FID scores of our approach are approximately 60% better than those of PhysGaussian and are comparable to the 2D inpainting method (with fine-tuning).

Table 2. Higher cosine similarity indicates better consistency

Method	CLIP Score ↑	Cosine Similarity ↑
PhysGaussian	23.9	0.89
2D Inpainting	27.8	0.87
Ours	**29.1**	**0.96**

To evaluate the texture consistency, we conduct experiments by slicing 3D objects at 120 arbitrary, random angles. Then we compute the average pairwise cosine similarity of CLIP-encoded image features in Tab 2. Using object-specific prompts, such as 'the cross-section of a ⟨ object ⟩', we also report the average CLIP score. Results indicate that textures generated by FruitNinja achieve the highest CLIP scores and cosine similarity, demonstrating superior consistency while maintaining fidelity.
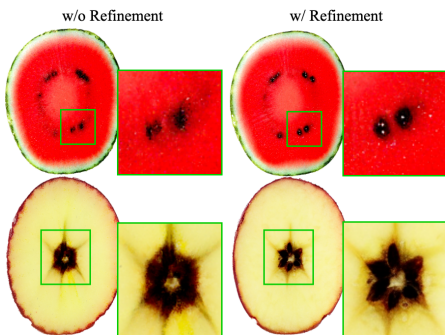
5. Ablation

Figure 8. Without continuous texture refinement, reference cross-sectional views often exhibit spatial conflicts as described in Figure 5, leading to noisy and unrealistic textures in the trained 3DGS models as highlighted in the left column.

Progressive Texture Refinement To assess the necessity of progressive texture refinement (as described in Section 3.3), we performed an ablation study. Initially, the 3D model was trained using reference 2D views from predefined cross-sectional angles. Texture refinement was then applied 60 iterations on the two selected objects (a watermelon and an apple). Figure 8 illustrates horizontal cross-sectional views of both objects with (right) and without texture refinement (left). Without refinement, the rendered views exhibit unrealistic textures, characterized by noisy pixels and blurred edges around the seeds, which obscures their shapes and introduces artifacts. These imperfections degrade the visual fidelity of the seeds, making them appear indistinct and unnatural. In contrast, rendered views after

refinement show improved visual realism, reduced noise, and enhanced multi-view consistency across cross-sectional perspectives.
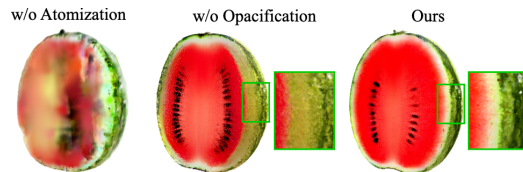

Figure 9. OpaqueAtomGS (right-most) achieves more stable convergence and sharper texture transitions.

Opaque-Atomic Gaussians To validate the effectiveness of the OpaqueAtomGS strategy in Section 3.4, we conducted ablation studies under three configurations: (1) without atomic clipping, (2) without uniform and high opacity, and (3) with full OpaqueAtomGS setting. Each configuration's 3D model was trained for the same number of iterations. As illustrated in Figure 9, without atomic clipping, the 3D model struggled to converge and failed to generate realistic textures that align with the references. Without high opacity, the model could not accurately represent the abrupt color transitions at the interface between the white flesh and green surface in the sliced watermelon.
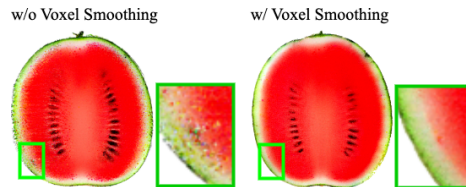

Figure 10. Without voxel smoothing (left), Gaussians that are not exposed at the forefront of the cross-sectional plane exhibit random and noisy colors, which reduces overall visual fidelity.

Voxel Smoothing Figure 10 shows that voxel smoothing significantly reduces color distortion in untrained Gaussians, enhancing fidelity for internal textures in cross-sectional views outside the training set.

6. Conclusion

We present FruitNinja, a novel method for generating realistic internal textures for 3DGS objects, enabling real-time rendering of interior views during unconstrained geometric and topological changes. Experimental results show that FruitNinja significantly improves texture realism and coherence, establishing its effectiveness for immersive 3D applications.

References

[1] Reiner Birkl, Diana Wofk, and Matthias Müller. Midas v3.1 – a model zoo for robust monocular relative depth estimation. *arXiv preprint arXiv:2307.14460*, 2023. 4

[2] Mikołaj Bińkowski, Dougal J. Sutherland, Michael Arbel, and Arthur Gretton. Demystifying MMD GANs. In *International Conference on Learning Representations (ICLR)*, 2018. 7

[3] Piotr Borycki, Weronika Smolak, Joanna Waczyńska, Marcin Mazur, Sławomir Tadeja, and Przemysław Spurek. Gasp: Gaussian splatting for physic-based simulations. *arXiv preprint arXiv:2409.05819*, 2024. 1

[4] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. *Polygon Mesh Processing*. AK Peters/CRC Press, 2010. 2

[5] Tianshi Cao, Karsten Kreis, Sanja Fidler, Nicholas Sharp, and Kangxue Yin. Texfusion: Synthesizing 3d textures with text-guided image diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4169–4181. IEEE/CVF, 2023. 2, 3

[6] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 2

[7] Dave Zhenyu Chen, Yawar Siddiqui, Hsin-Ying Lee, Sergey Tulyakov, and Matthias Nießner. Text2tex: Text-driven texture synthesis via diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 18558–18568, 2023. 2, 3

[8] Yiwen Chen, Zilong Chen, Chi Zhang, Feng Wang, Xiaofeng Yang, Yizhou Wang, Zhaowen Cai, Liang Yang, Hong 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 (CVPR)*, 2024. 1, 2, 3

[9] 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 (CVPR)*, pages 13142–13153, 2023. 2

[10] Xiangjun Gao, Xiaoyu Li, Yiyu Zhuang, Qi Zhang, Wenbo Hu, Chaopeng Zhang, Yao Yao, Ying Shan, and Long Quan. Mani-gs: Gaussian splatting manipulation with triangular mesh. *arXiv preprint arXiv:2405.17811*, 2024. 2

[11] Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. CLIPScore: A reference-free evaluation metric for image captioning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7514–7528, 2021. 7

[12] 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 (NeurIPS)*, 2017. 7

[13] Jiajun Huang and Hongchuan Yu. Point'n move: Interactive scene object manipulation on gaussian splatting radiance fields. *arXiv preprint arXiv:2311.16737*, 2023. 1, 2, 3

[14] Jiajun Huang and Hongchuan Yu. Gsdeformer: Direct cage-based deformation for 3d gaussian splatting. *arXiv preprint arXiv:2405.15491*, 2024. 2

[15] Youyou Huang, Rencheng Song, Kuiwen Xu, Xiuzhu Ye, Chang Li, and Xun Chen. Deep learning-based inverse scattering with structural similarity loss functions. *IEEE Sensors Journal*, 21(4):4900–4907, 2020. 5

[16] Yi-Hua Huang, Yan-Pei Cao, Yu-Kun Lai, Ying Shan, and Lin Gao. Nerf-texture: Texture synthesis with neural radiance fields. In *ACM SIGGRAPH 2023 Conference Proceedings*, pages 1–10, 2023. 3

[17] Ying Jiang, Chang Yu, Tianyi Xie, Xuan Li, Yutao Feng, Huamin Wang, Minchen Li, Henry Lau, Feng Gao, Yin Yang, and Chenfanfu Jiang. Vr-gs: A physical dynamics-aware interactive gaussian splatting system in virtual reality. In *ACM SIGGRAPH 2024 Conference Papers*, 2024. 1, 2, 3, 7

[18] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuhler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (ToG)*, 42(4):1–14, 2023. 1, 2

[19] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. Abc: A big cad model dataset for geometric deep learning. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9593–9603, 2019. 2

[20] Kunhao Liu, Fangneng Zhan, Muyu Xu, Christian Theobalt, Ling Shao, and Shijian Lu. Stylegaussian: Instant 3d style transfer with gaussian splatting. *arXiv preprint arXiv:2403.07807*, 2024. 1

[21] Ruiyang Liu, Jinxu Xiang, Bowen Zhao, Ran Zhang, Jingyi Yu, and Changxi Zheng. Neural impostor: Editing neural radiance fields with explicit shape manipulation. *Computer Graphics Forum*, 42(7), 2023. 2

[22] Rong Liu, Rui Xu, Yue Hu, Meida Chen, and Andrew Feng. Atomgs: Atomizing gaussian splatting for high-fidelity radiance field. *arXiv preprint arXiv:2405.12369*, 2024. 2, 3, 6

[23] Zhiheng Liu, Hao Ouyang, Qiuyu Wang, Ka Leong Cheng, Jie Xiao, Kai Zhu, Nan Xue, Yu Liu, Yujun Shen, and Yang Cao. Infusion: Inpainting 3d gaussians via learning depth completion from diffusion prior. *arXiv preprint arXiv:2404.11613*, 2024. 1, 3, 7

[24] Matthias Niessner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3d reconstruction at scale using voxel hashing. In *ACM Transactions on Graphics (TOG)*, page 169, 2013. 2

[25] Francesco Palandra, Andrea Sanchietti, Daniele Baieri, and Emanuele Rodolà. Gsedit: Efficient text-guided editing of 3d objects via gaussian splatting. *arXiv preprint arXiv:2403.05154*, 2024. 1

[26] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In

Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 165–174, 2019. 2

[27] Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. In *International Conference on Learning Representations (ICLR)*, 2023. 2

[28] Yi-Ling Qiao, Alexander Gao, and Ming C. Lin. Neuphysics: Editable neural geometry and physics from monocular videos. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2022. 2

[29] 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 International Conference on Machine Learning (ICML)*, pages 8748–8763, 2021. 3

[30] Elad Richardson, Gal Metzer, Yuval Alaluf, Raja Giryes, and Daniel Cohen-Or. Texture: Text-guided texturing of 3d shapes. *arXiv preprint arXiv:2302.01721*, 2023. 2, 3

[31] 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)*, pages 10684–10695, 2022. 5, 6, 7

[32] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 5, 7

[33] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–4, 2011. 2

[34] Yawar Siddiqui, Justus Thies, Fangchang Ma, Qi Shan, Matthias Nießner, and Angela Dai. Texturify: Generating textures on 3d shape surfaces. In *European Conference on Computer Vision (ECCV)*, 2022. 3

[35] Jiaxiang Tang, Jiawei Ren, Hang Zhou, Ziwei Liu, and Gang Zeng. Dreamgaussian: Generative gaussian splatting for efficient 3d content creation. In *International Conference on Learning Representations (ICLR)*, 2024. 5

[36] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Duc Thanh Nguyen, and Sai-Kit Yeung. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In *International Conference on Computer Vision (ICCV)*, 2019. 2

[37] Can Wang, Menglei Chai, Mingming He, Dongdong Chen, and Jing Liao. Clip-nerf: Text-and-image driven manipulation of neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3835–3844, 2022. 3

[38] Weiyue Wang, Qiangui Huang, Suya You, Chao Yang, and Ulrich Neumann. Shape inpainting using 3d generative adversarial network and recurrent convolutional networks.

2017 IEEE International Conference on Computer Vision (ICCV), pages 2317–2325, 2017. 3

[39] Tianyi Xie, Zeshun Zong, Yuxing Qiu, Xuan Li, Yutao Feng, Yin Yang, and Chenfanfu Jiang. Physgaussian: Physics-integrated 3d gaussians for generative dynamics. *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4389–4398, 2023. 1, 2

[40] Tian-Xing Xu, Wenbo Hu, Yu-Kun Lai, Ying Shan, and Song-Hai Zhang. Texture-gs: Disentangling the geometry and texture for 3d gaussian splatting editing. In *European Conference on Computer Vision (ECCV)*, 2024. 1

[41] Mingqiao Ye, Martin Danelljan, Fisher Yu, and Lei Ke. Gaussian grouping: Segment and edit anything in 3d scenes. In *European Conference on Computer Vision (ECCV)*, 2024. 1

[42] Wentao Yuan, Lingjie Gao, Chen Zheng, and Yebin Liu. Nerf-editing: Geometry editing of neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 2

[43] Dingxi Zhang, Yu-Jie Yuan, Zhuoxun Chen, Fang-Lue Zhang, Zhenliang He, Shiguang Shan, and Lin Gao. Stylizedgs: Controllable stylization for 3d gaussian splatting. *arXiv preprint arXiv:2404.05220*, 2024. 1

[44] Junzhe Zhang, Xinyi Chen, Zhongang Cai, Liang Pan, Haiyu Zhao, Shuai Yi, Chai Kiat Yeo, Bo Dai, and Chen Change Loy. Unsupervised 3d shape completion through GAN inversion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 3

Appendix

A. Input Cross-Section Specifications

In Section 3.2, we introduced the use of user-defined cross-sections for generating internal textures in Figure 3. Here, we present the detailed cross-sectional angles used for training the six objects in our experiments. We used two cross-section types for objects in Fig. 11 due to their more complex internal structures, whereas the objects in Fig. 12 used only a single type. Additional visual results for objects after arbitrary geometric transformations demonstrate texture coherence, even when cuts are misaligned with the trained angles.
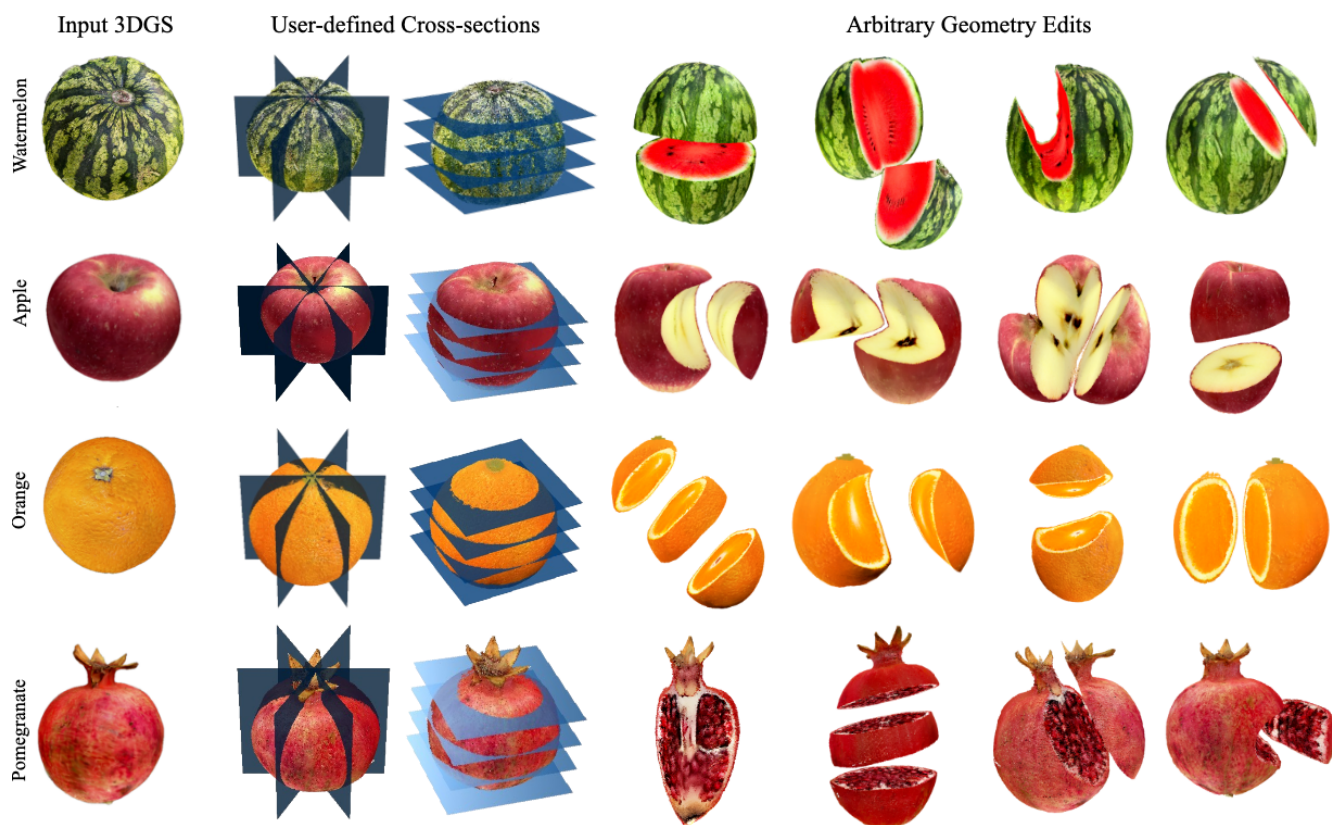
Figure 11. For the four objects shown above (watermelon, apple, orange, and pomegranate), we use two types of input cross-sections: (1) **Vertical Cross-sections**: 30 slices evenly spaced in angle, spanning a full rotation around the central axis of the object (as illustrated on the second column). (2) **Horizontal Cross-sections**: We use 40 horizontal slices (as shown in the third column), evenly spaced along the vertical axis to cover the entire object.

Figure 12. For bread and red velvet cake, we use only vertical cross-sections: (1) **Bread**: 60 evenly spaced vertical cross-sections (second column) covering the object. (2) **Red Velvet Cake**: 30 vertical cross-sections, evenly distributed and radially arranged around the central axis, spanning a full rotation of the object.