

ICE-G: Image Conditional Editing of 3D Gaussian Splats

Vishnu Jaganathan¹, Hannah Hanyun Huang¹, Muhammad Zubair Irshad^{1,2},
 Varun Jampani³, Amit Raj⁴, Zsolt Kira¹

¹Georgia Institute of Technology, ²Toyota Research Institute, ³Stability AI, ⁴Google Research

{vjaganathan3, hhuang474, mirshad7, zkira}@gatech.edu,

varunjampani@gmail.com, amitraj93.github.io

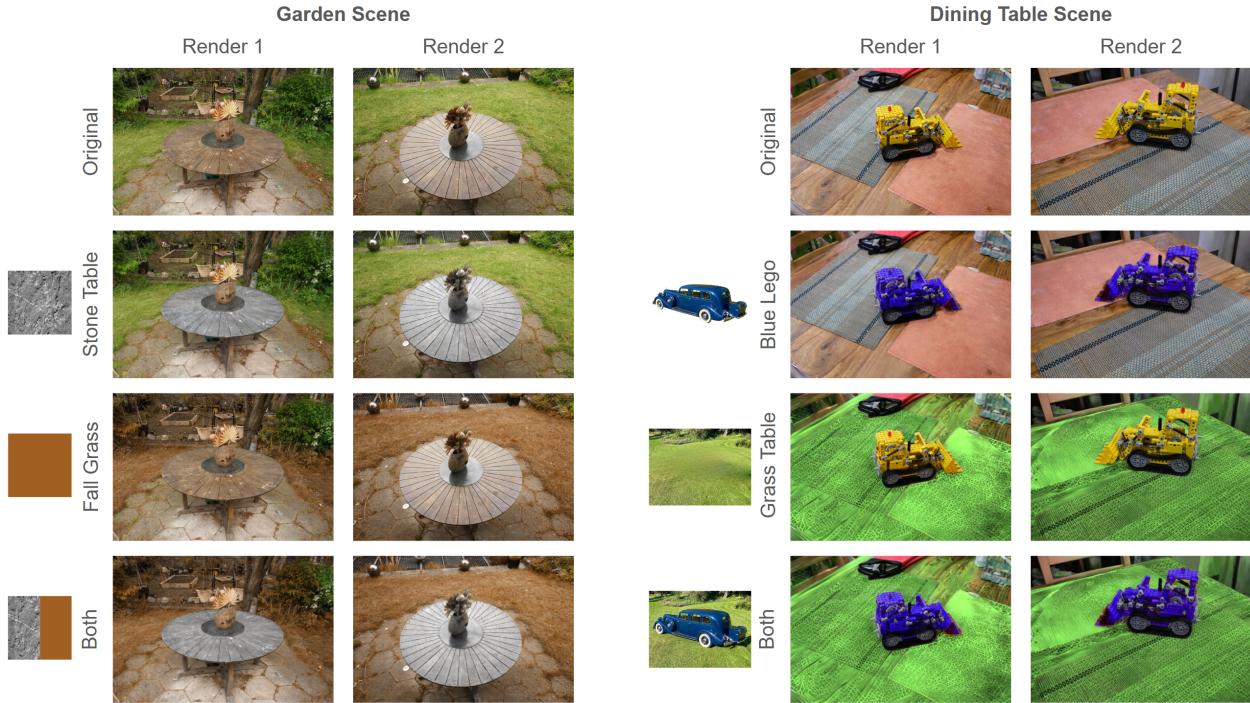


Figure 1. Our method, ICE-G, allows for quick color or texture edits to a 3D scene given a single style image, or mask selection on a single view. We show two rendered views of mask select editing for the Garden Scene where we apply stone texture to the table and fall colors to the grass (left). We also show two renders of correspondance based editing where we can transfer the color of the blue car to the lego and the texture of the grass to the table (right).

Abstract

Recently many techniques have emerged to create high quality 3D assets and scenes. When it comes to editing of these objects, however, existing approaches are either slow, compromise on quality, or do not provide enough customization. We introduce a novel approach to quickly edit a 3D model from a single reference view. Our technique first segments the edit image, and then matches semantically corresponding regions across chosen segmented dataset views using DINO features. A color or texture change from a particular region of the edit image can then be applied to other views automatically in a semantically sensible manner.

These edited views act as an updated dataset to further train and re-style the 3D scene. The end-result is therefore an edited 3D model. Our framework enables a wide variety of editing tasks such as manual local edits, correspondence based style transfer from any example image, and a combination of different styles from multiple example images. We use Gaussian Splats as our primary 3D representation due to their speed and ease of local editing, but our technique works for other methods such as NeRFs as well. We show through multiple examples that our method produces higher quality results while offering fine grained control of editing. Project page: ice-gaussian.github.io

1. Introduction

Editing of 3D scenes and models is an area of growing importance as applications like robotics simulation, video games, and virtual reality grow in popularity. Editable 3D representations can lead to dynamic and customizable environments in these applications, allowing artists, developers, and researchers alike to quickly iterate on projects and produce valuable content.

Recently, Gaussian Splat [12] have emerged as a powerful method to represent 3D objects and scenes, allowing for fast training and preservation of high-quality details. Prior to this, NeRFs (Neural Radiance Fields) [18] have been used extensively to create scenes, and many techniques have been introduced to edit the color and texture of NeRFs. However, such editing has thus far been slow and limited in the types of edits possible. Our work seeks to develop a general method that works on both Splat and NeRFs, supporting fast and high-quality style edits.

NeRF editing works can be categorized by their editing interfaces into text-based or image-based methods. Text-based approaches use text-image models for guidance, delivering results faithful to prompts but limited by the ambiguity of text descriptions for 3D scenes. This leads to uncertainties in conveying specific colors, styles, or textures, such as the exact shade of "light blue" or the precise pattern of a "sand texture."

Our image-based editing approach addresses the ambiguities of text-based methods, yet current techniques limit modifications to a single style image for the entire scene and lack the ability to transfer color or texture between different image parts or specify them per region. Additionally, 3D editing classifications based on changes—color, texture, shape, or their combinations—show that shape modifications often reduce image quality by converting 2D guidance into 3D using methods like Score Distillation Sampling (SDS) [19] or Iterative Dataset Update (IDU) [10], which generalize features at the expense of detail.

In this paper, we propose a method that aims to take the texture and/or color of different segmented regions from an editing image, and transfer them to corresponding segmented regions of a sampled set of 2D images from the original scene dataset in a 3D consistent manner. To generate high quality results, we restrict our method to editing color and texture while preserving shape. This editing image can either be a totally different object or an edited view from the original dataset.

To do this, given a 3D model (e.g. Splat or NeRF), we propose to sample and edit a subset of the original data as a preprocessing step. Specifically, we use the Segment Anything Model (SAM) [13] to find corresponding regions of both the editing image and the sampled views. For each region of each sampled view from the original dataset, we have to find the best corresponding region from the editing image

to transfer style from. To find these matches, we utilize a custom heuristic which minimizes the distance between these mask regions in an extracted DINO [4] feature space. We then copy over colors by changing the hue and copy over textures by refitting them with Texture Reformer [27]. To apply these updates onto the 3D model (e.g. Gaussian Splat), we then finetune it using L1 and SSIM losses for color, and a Nearest Neighbor Feature Matching (NNFM) loss [30] for texture.

We generate results for objects and scenes in the NeRF Synthetic [18], MipNeRF-360 [2], and RefNeRF [24] datasets and compare against color and texture editing baselines to show qualitative improvement of our method. Overall, our main contributions are: 1) We provide a flexible and expressive mode of specifying edits, leveraging SAM and using a DINO-based heuristic to match image regions to the editing image in a multiview consistent manner, and 2) We provide fine grained control of choosing colors and textures for each part of the segmented editing view.

2. Related Works

There are a few works that aim to edit 3D models, and they fall into a few categories. First are diffusion-based editing methods, which broadly try to lift inconsistent 2D image edits from text prompts into 3D via specialized losses. There are also local texture editing methods that are able to target regions and apply textures from a source image. Finally, since our method is capable of color editing as well, there are purely color edit methods we compare to that usually apply manually specified colors to images.

2.1. 2D Priors

InstructNeRF2NeRF [10] adapts the InstructPix2pix [3] 2D editing model to 3D, enabling edits in color, shape, and texture based on text prompts. Initially applied to select dataset images, these edits may lack multiview consistency but achieve 3D uniformity through Iterative Dataset Update (IDU), progressively refining more dataset examples. While this method supports extensive shape and color modifications, it tends to fall short in result quality and detailed texture rendering.

Vox-E [21] uses a voxel grid and diffusion model for updates, focusing on large feature edits. It processes views with text-guided noise predictions to align edits, but struggles with fine texture/color adjustments, often resulting in blocky textures or unintended area expansions.

Blended-NeRF [9] blends new objects or textures into scenes, guided by CLIP [20] losses to match text inputs within a chosen 3D scene region. It modifies the scene's MLP with CLIP loss and blends colors and densities for the edits. While it achieves realistic textures, as a text-based method, it faces challenges in accurately conveying complex textures or specific regions without image input.

2.2. Local Texture Editing

S2RF [15] introduces local texture editing for specific scene types, utilizing an object detection model along with SAM for precise region masking. This method applies NNFN loss from ARF for style/texture transfer onto masked areas, demonstrating the capability to apply varied textures to different scene parts.

Semantic-driven Image-based NeRF Editing (SINE) [1] offers a method for 3D texture editing, leveraging a prior-guided editing field combined with original views. It uses a ViT [6] extracted style features to adjust textures, enabling localized edits. While it supports seamless rendering by merging template NeRF with the editing field, the process demands 12 hours of training per scene and faces compatibility issues with Gaussian Splatting due to its unique rendering approach.

2.3. Color Editing

Decomposing NeRF for Editing via Feature Field Distillation [14] allows color editing of NeRFs using text prompts. It generates a feature field for selecting and altering colors in 3D regions. Utilizing CLIP-LSeg [17] and DINO [4] as 2D teacher networks, it learns an extra feature field integrated into the original NeRF, applying updates through photometric and feature loss functions. This approach enables soft 3D segmentation via a dot product between an encoded query and the feature field, facilitating text-specified color edits in 3D regions through modified rendering functions.

CLIP-NeRF [26] learns a conditional NeRF representation that aims to separate appearance and shape information. CLIP embeddings are passed through appearance and shape mappers which extract the respective information and additively combine them with the conditional NeRF. These mapping layers are trained along with the NeRF via a CLIP similarity loss iterating over randomly sampled NeRF views. This method primarily edits color, but also shows minor shape changes on objects like cars and chairs.

RecolorNeRF [8] aims to decompose the scene into a set of pure-colored layers, and editing that palette to change the color of the scene. This method achieves aesthetic results, but cannot distinguish between two different objects that have the same color in a scene. ProteusNeRF [25] is able to rapidly edit the color of a NeRF by selecting a masked region and change its color, propagating the change into 3D. ICE-NeRF [16] finetunes the NeRF with the desired color edits, introducing techniques to preserve multiview consistency and avoid unwanted color changes.

2.4. Concurrent Work in Gaussian Splat Editing

Recently many methods have emerged that show editing capabilities on Gaussian Splats. One such example is GaussianEditor: Editing 3D Gaussians Delicately with Text Instructions [7]. This pipeline feeds the user prompt and scene

description to an LLM to select regions of interest, and then applies a 2D diffusion prior to edit various views. Another similarly named paper GaussianEditor: Swift and Controllable 3D Editing with Gaussian Splatting [5] introduces Hierarchical Gaussian Splatting, a technique to allow more fine grained editing via 2D diffusion priors. The user must select points on the screen, and change visual aspects manually in 2D for the edit to be carried over to 3D.

Instruct-GS2GSEditing: Editing 3D Gaussian Splatting Scenes with Instructions [23] is based off InstructNerf2Nerf and inherits the same advantages and disadvantages of that method. The authors use the same IDU method, but tune some hyperparameters to suit Gaussian Splatting. Another paper TIP-Editor: An Accurate 3D Editor Following Both Text-Prompts And Image-Prompts [31] uses LoRA to personalize a diffusion model with the style of a reference image, and then uses this along with a user prompt to generate 2D edits. These edits are additionally bounded by a user specified region to contain edits.

Overall, these methods show some interesting results on editing using 2D Diffusion priors, but sometimes suffer the quality downgrade associated with diffusion models, and are not able to transfer style globally from a standalone 2D image.

3. Method

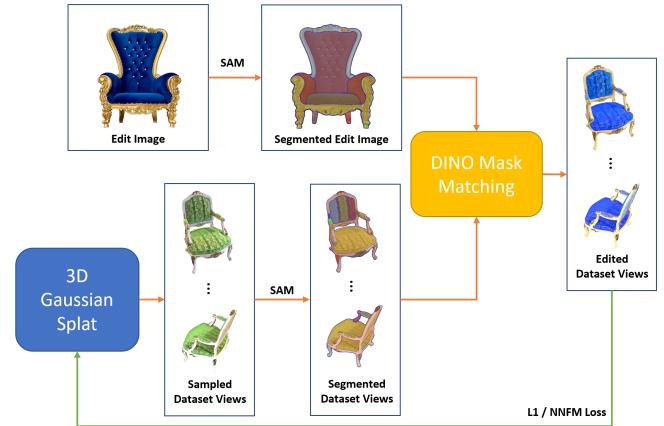


Figure 2. The user supplied style image is segmented and its masked regions are matched with masked regions of sampled dataset views via DINO correspondences. The color/texture is then transferred to those matching regions, and the splat is edited with this updated dataset.

Our method supports different types of 3D models, and we primarily demonstrate it on top of Gaussian Splatting [12] due to its favorable speed. We also implement our method on a regular NeRF framework [28] for time comparison. There are two main interfaces, one for manual texture/color editing and another for automatically transferring these attributes

from an example image as shown in Figure 2. The process differs only for creating the edit image, but uses the same segmentation, part matching, and texture/color losses across both. After making changes to the edit view, or choosing the conditional image, the algorithm is run on a number of sampled images where the style is transferred to these randomly sampled views. Color is naturally multiview consistent since only the hue is changed, and the underlying grayscale is preserved, so standard L1/SSIM loss is used to push the color updates. Since this is not the case for transferred texture updates, we employ the Nearest Neighbor Feature Matching (NNFM), originally proposed in ARF [30], to make the texture change 3D consistent. Texture changes are done with this NNFM loss in a first round of iterations, and then color is changed with L1/SSIM losses in a second round, since we find that more vivid color is transferred via standard loss functions than NNFM.

3.1. Preliminaries

3.1.1 Gaussian Splatting

Gaussian Splats [12] is a recent 3D scene representation technique that allow for faster training and rendering. The scene is represented as a collection of 3D Gaussians, which are defined by position, covariance, opacity, and color. A given view is rendered from a differentiable rasterizer, which returns any given 2D view of this set of gaussians given standard NeRF-style viewing parameters. Since the rasterizer is differentiable, edits to the returned 2D image are backpropagated, and make the appropriate changes to the underlying gaussian representation. This method has the benefit of quick training time and producing more realistic textures than NeRFs in many cases. We base our method off of Gaussian Splats, but the technique works for NeRFs as well.

3.1.2 SAM

The Segment Anything Model (SAM) [13] is a zero shot image segmentation model. It consists of a ViT encoder and mask decoder that produces the mask of each instance. This decoder is conditioned on either specific points, a box, or text to produce various masks. For the purpose of separating all parts of an object or scene, prompting with a grid of points is most effective. This produces distinct masked regions, which can be used as editing regions to apply new colors and textures.

3.1.3 DINO

Self-Distillation with No Labels (DINO) [4] is a self-supervised technique for training Vision Transformers (ViTs) [6] where a single ViT acts as both student and

teacher. The student model learns from input data using standard methods, while the teacher model updates its weights through an exponential moving average of the student’s weights, ensuring stable updates. This process encourages the student to learn generalizable and robust features by predicting the more stable teacher outputs. DINO facilitates effective ViT training without labeled data, leading to models that better focus on relevant image parts. The method’s ability to identify pixel-wise image correspondences is further demonstrated in [29].

3.1.4 Texture Reformer

Texture Reformer [27] introduces View-Specific Texture Reformation (VSTR) for transferring textures between image regions. By utilizing source and target semantic masks along with VGG feature extraction [22], it overlays textures from one area to another, adjusting to the new shape’s contours. The technique employs patch grids and convolution for texture application, with statistical refinements ensuring realistic integration within the targeted masked regions.

3.1.5 NNFM Loss

Artistic Radiance Fields (ARF) [30] offers a method for infusing 3D NeRF scenes with style elements from 2D images. By processing 2D scene views alongside style images through a VGG-16 encoder, it applies a novel NNFM loss to match local features between the two, diverging from traditional Gram matrix losses that blend style details globally. This local matching technique ensures the preservation of texture specifics, marking a notable advancement over previous approaches.

3.2. 2D Editing

There are two options for editing. Firstly, we can take a different conditional image(s), and copy styles from all parts onto the target objects views. In this approach, we use the texture-reformer module to bring all source textures onto a square array, so they can be cropped to the size of the target masks as necessary. We also store which colors correspond with which mask ids. Secondly, there is manual editing, where we start with any arbitrary view of the target object, and assign different styles to different regions. In both cases, we seek to generate a mapping of mask id to color/texture to find and copy these styles to the appropriate regions in the next steps. When editing, we can specify whether we want to copy only the color or the texture as well. This will determine whether we use the Texture Reformer module to extract textures, and whether we use NNFM loss or L1 loss alone in the downstream style applying steps, as opposed to sequentially.

3.3. Segmentation

The Segment Anything Model (SAM) [13] is an encoder-decoder model that can be prompted with several grid points to make masks of most identifiable parts of an image. We use SAM to segment both the edit image and sampled views into their component parts, since it is the state of the art at this task and runs fairly quickly. Since we provide an option to manually specify which masks to edit, in our mask processing step, we allow users to specify a limit of N masks for simplicity. We choose the largest N-1 masks and group the rest of the image into the Nth mask. This basically enables the user to separate the editing view into any number of parts to have control over fine grained features. We first segment the editing image and store those masks, and segment each dataset view as we iterate over it.

3.4. DINO Mask Matching

We use DINO features to find which is the best editing image region to copy style from for each region of each of the sampled dataset views. Extracted DINO features have been shown to find corresponding pixels between two images [4]. We use a similar feature extraction technique, but create a custom heuristic to measure the distance between two masks in the DINO feature space. First, we extract the DINO feature vector for each of the masks in the editing image, and store this information as it does not change. When iterating over a given dataset image, we extract DINO features after segmentation, and find the best matching region with the following heuristic:

$$M_P = \underset{E}{\operatorname{argmin}} \frac{1}{N_E} \sum_{i \in P} (D(i) - D(E))^2 \quad (1)$$

To find the best match M for a given part P of a sampled dataset view, we find the editing image part E which is closest in the DINO feature space D.

3.5. Texture Reformer

We use the texture reformer [27] module to copy textures from the editing image. Since we will be obtaining masked regions (see next section), we can use that for our source semantic map, and the editing image itself for our source texture. For our target semantic mask, we can use the entire blank image of the same size. When applying textures to various different regions of the different dataset views, we can simply crop this full sized texture to shape. The reason we do this, rather than mapping the texture to each individual semantic mask for each view, is because we find empirically it does not matter, and time is saved by just doing this once. Running Texture Reformer per view does not lead to any more naturally view consistent results without NNFM loss. This is done after the edit image is segmented in Figure 2.

3.6. Applying Edits

3.6.1 Applying Color

Applying a color change to a region of a view image is done in the HSV representation. In this representation the image is split up into the three channels of Hue, Saturation, and Value, rather than the standard RGB. The hue controls what color is expressed, the saturation controls how strong the color is, and the value controls how light or dark it is. The grayscale of an image, which contains the texture of the original view is the value. Therefore, to edit the color in a given target region, we copy over the average hue and saturation values of the source region, while leaving the value alone. If the user wants to brighten or darken a view overall, that can also be achieved by shifting the value field by a specified constant.

Once this edit is made on the views, we use this as the data for training the edited 3D model. The loss function used is standard Structural Similarity Index (SSIM) and L1 interpolation used to train Gaussian Splatting:

$$\mathcal{L}_{GS} = \lambda \mathcal{L}_1 + (1 - \lambda) \mathcal{L}_{SSIM} \quad (2)$$

3.6.2 Applying Texture

For texture, we either have manually specified a texture from a pattern image, or we automatically extracted texture from a matching region and expanded it as a pattern image with texture reformer. In either case we can crop this image sized texture to fit the mask region and add it. The texture will have the same pattern cropped to different viewpoints, and so is not 3D consistent. However, we again train a 3D model using this data and the NNFM loss, and over several iterations this will blend the image to be so. We find that using NNFM alone causes degradation in image quality and artifacts, and so we regularize it with the original Gaussian Splat training loss:

$$\mathcal{L}_{texture} = \mathcal{L}_{NNFM} + \alpha \mathcal{L}_{GS} \quad (3)$$

This texture transferring often imprints the correct pattern on the Gaussian Splat, but leads to color appearing washed out. Thus, we follow up with some iterations of the color applying stage, copying over the average hue and saturation of the texture image.

4. Results

4.1. Experiment Details

We discovered in our ablation study in Figure 3, that sampling around 20% of the images in the dataset for editing is sufficient for a good quality result. The color editing stage is run for around 2000 iterations, and the texture editing takes 3000 iterations to fully stylize the Gaussian Splat like the

editing inputs. For the L1+SSIM portion of the loss, we use the Gaussian Splat implementation default interpolation. For texture loss, we find that adding 50% of the original loss as a regularizer to the NNFm loss works best.



Figure 3. Comparing different dataset sampling rates for turning the road to a river. Sampling 5% or 10% of images from a dataset to edit results in numerous artifacts and other degradations, and quality peaks at around 20% sampling.



Figure 4. Adding texture to the garden table from mip-NeRF360 (left). Using paintings to texture the table (right).

4.2. Texture Editing

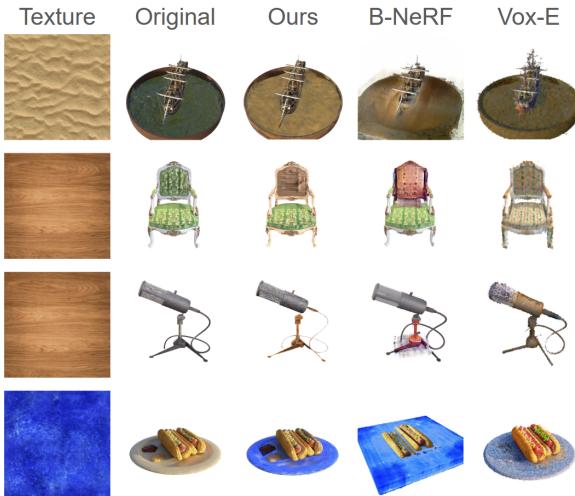


Figure 5. Comparison of our method in local texture editing of ours and baselines.

In our method: For the ship, we selected the mask that corresponds to the water and indicate that the sand texture should be applied there. For the chair, we select the back and armrests. For the mic we select the stand and for the hotdog we select the plate.

In BlendedNeRF: To edit, a 3D box region and a corresponding prompt are required. For the ship, the box covers

the xy plane, extending upwards to the ship’s start, with the prompt ‘dunes of sand’. The chair’s box is above the seat cushion, including the armrests, using ‘a wooden chair’. The mic’s stand is boxed with ‘a wooden stand’. For the hotdog, the box spans the xy plane, extending vertically to the hotdogs’ start.

In Vox-E: We should specify our prompt as what we want the final image to be, as this is the input to the diffusion guidance. For the ship we use ‘a ship in sand’. For the chair we use ‘a chair with a wooden back’. For the mic we use ‘a microphone with a wooden stand’, and for the hotdog we use ‘a hotdog on a blue granite plate’.

4.2.1 Analysis

Vox-E allows users to use text prompts for editing object voxel grids but has notable limitations. The text prompt can’t focus edits on specific areas, leading to unwanted changes, such as unnecessary coloring in the chair and microphone examples as in Figure 5. It also struggles with texture representation, producing rough, pixelated textures that don’t match the intended edits, as seen in the plate and ship examples. Additionally, while Vox-E can change shapes, this sometimes results in unintended alterations.

BlendedNeRF can produce high-quality visuals but suffers from unintended artifacts and shape distortions due to its edit region being box-shaped, making precise edits difficult in intertwined areas. This issue is evident in examples like the ship, where sand spills out improperly, the chair with misplaced wooden panels, a mic with fuzzy artifacts, and a hotdog plate turned square. Unlike box-based edits, our mask-based approach allows for more precise region modifications. Additionally, BlendedNeRF struggles with texture definition, failing to produce detailed contours in sand or realistic wood grain, as highlighted in the ship and wood examples.

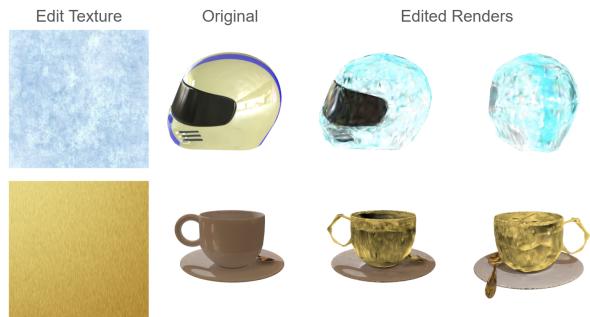


Figure 6. Making an ice helmet and a gold foil coffee cup from RefNeRF.



Figure 7. Applying a snow and ice texture to the scene (left), and turning the stump scene to fall (right)

4.3. Color Editing



Figure 8. Showing local and correspondance based color editing across ours and baselines.

In our method: The purple color is applied to the water region of the image for the ship. For the chair, drums, and plant, appropriate color regions are automatically extracted with mask matching and applied onto the dataset views.

In D.F. Fields: This method uses a text phrase for the new color and a filter phrase for object selection, where simply naming the object works best due to D.F. Fields’ difficulty in selecting object subparts. Adding ‘background’ to the filter phrase improves performance. Color prompts are specific, like ‘purple water’ for the ship, ‘brown drums’, ‘golden plant’, and ‘blue and gold chair’. Excessively detailed prompts tend to reduce effectiveness.

In CLIP-NeRF: In CLIP-NeRF, we specify a prompt as a sentence of what we want to see in the result sentence. For the ship, this is ‘a ship in purple water’. For the chair, it is ‘a chair with blue cushions and a gold frame’. For the drums is ‘brown drumset with bronze symbols’, and for the ficus it is ‘plant with fall-colored leaves’.

4.3.1 Analysis

DF Fields effectively changes colors across broad areas but struggles with precision in smaller regions. This model can-

not finely detail or recolor small parts of images, exemplified by unchanged colors in the instruments 8 and difficulty in differentiating the ship from its surrounding water. Additionally, trying to specify exact RGB colors through text often leads to discrepancies between intended and actual colors.

CLIP-NeRF produces attractive results but can deviate from precise prompts. For instance, a request for a ship in purple water resulted in both the ship and a bucket being colored purple instead of just the water. This indicates a challenge with the precision of text-based editing compared to direct mask and color adjustments. Other examples include unintended additions like a gold pattern on a chair meant to be simple, minimal changes to drum colors, and an unexpectedly colored pot on a plant, showcasing the limitations in accurately reflecting user intentions through text/vector embedding.

4.4. Inherited Limitations

We inherit a few limitations from pretrained components utilized in our method. SAM, which performs segmentation on selected views can sometimes fail to generate fine grained masks from certain angles, lumping together two parts of an object. When this happens, an edit that would have normally been constrained to one area in that particular view can sometimes bleed into other areas. This is rare in most scenes, but can occur in complicated scenes, or object sections with ill-defined boundaries. Also, using SAM to select masked regions means that our method cannot perform edits to the 3D geometry of the object.

The NNFMloss function is great at copying texture and overall style from a source area to destination area, but makes the result unreflective. This is seen in Figure 6, where the original surface of objects was reflective, and applying a new texture unintentionally overwrote those effects. Likewise, if the edit image’s texture contains any such light scattering effects, these are not carried over onto the Gaussian Splat.

4.5. Data

We use publically available synthetic datasets from NeRF [18] and RefNeRF [24], as well as real scenes from Mip-NeRF [2] and NeRDS360 [11]. We use internet images under cc-license for the style conditioning.

4.6. Computation Time

We find that our method performs much faster when implemented on Gaussian Splats, showing that color and style losses can be applied faster on this representation. In our experiments, running our method on top of standard NeRFs took more iterations to transfer style, and each iteration also ran slower, showing that it is easier to change color and texture on a Gaussian Splat. We include timings for the other baselines we tested in Table 3, along with the timing for SINE from that paper.

	Avg Time (Mins)
Vox-E	52
DF Fields	33
CLIP-NeRF	35
BlendedNeRF	118
SINE	720
Ours (NeRF)	40
Ours (GS)	21

Table 1. Average runtimes we observed for obtaining quality results for each method on a single NVIDIA A40 GPU.

4.7. User Study

In the user study, we seek to understand how users perceive our method as compared with leading baselines. Since the text prompts we chose for each of these baselines detailed in Sections 4.2 and 4.3 are a faithful representation of the edit we intend to express with the conditional image we use for our method, we can compare against these baselines accurately. We solicit feedback on the user preferences from 38 people, and asked about their expertise with generative models. The ratings were requested via a Google Form. *Ten were familiar with generative computer vision and twenty-eight were not.*

4.7.1 Texture

As displayed in Figure 5 in the paper, we test on the baselines of Vox-E and BlendedNeRF. For each of the texture editing instructions, we ask the user to choose the result that best transfers the texture shown onto the specified area of the image, and specify the following instructions:

- Turning the water into sand
- Turning the chair back and frame into wood
- Turning the plate blue granite
- Turn the mic stand wood

Object	Ours	BlendedNeRF	Vox-E
Ship	86.8%	7.9%	5.3%
Chair	63.2%	26.3%	10.5%
Mic	68.4%	13.2%	18.4%
Plate	73.7%	5.3%	21.1%

Table 2. Percent of users who preferred each method for texture editing.

In all cases, our method was favored by most users. For the ship example, it received high preference due to BlendedNeRF’s sand spilling out of bounds and Vox-E’s unrepresentative grainy texture. In the chair scenario, 63.2% preferred our method, noting it provided a reasonable texture, whereas BlendedNeRF was a close second. Vox-E’s inaccuracies, such as miscoloring parts of the mic, were noted by attentive

users. Our plate design also won majority preference, with BlendedNeRF’s version turning square and Vox-E erasing condiments. Similarly, our method was the top choice for the mic, as BlendedNeRF’s edits introduced unwanted artifacts.

4.7.2 Color

Here we test against Distilled Feature Fields and CLIP-NeRF, with three global style transfer examples from conditional images, and one local color editing example as in Figure 8. For the global color transfer, we explain the concept of correspondence in simple English, by asking the user to select the result which takes on the color scheme of the edit image applied onto the original. For the local color transfer on the ship example, we mention that the goal is to turn the water in the image purple.

Object	Ours	DF Fields	CLIP-NeRF
Ship	84.2%	7.9%	7.9%
Chair	73.7%	5.3%	21.1%
Drums	73.7%	10.5%	15.8%
Plant	65.8%	2.6%	31.6%

Table 3. Percent of users who preferred each method for color editing.

In color editing, our method again won over 60% of user preference in each scenario. For the ship, our recolor was favored as Distilled Feature Fields partially recolored the tray border and CLIP-NeRF mistakenly colored the ship, not the water. Our chair was preferred for its accurate gold frame and blue cushion, matching the throne, though CLIP-NeRF also attracted 21.1% of users with its intriguing, albeit unintended, result. Both DF Fields and CLIP-NeRF struggled with coloring the drums correctly, leading to low preference. For the plant, DF Fields failed to alter its green color, while CLIP-NeRF’s reddish fall colors caught some interest, but overall, our method was seen as most accurately reflecting the intended edits.

5. Conclusion

In this work, we have introduced a robust and flexible method for editing color and texture of 3D images and scenes. We provide interfaces to copy style from an edit image or manually specify changes, enabling creative appearance editing for a variety of applications. Our key innovation, DINO-based mask matching, runs quickly and contains edits to discrete regions, leading to higher quality than other methods. Future work could explore how to make 3D consistent shape changes to these discrete regions in addition to color and texture, without compromising on resulting 3D scene quality like most other current methods do. Overall, we showcase our method’s unique input expressivity and resulting 3D model quality on a variety of objects and scenes, proving it is well suited for creative applications.

References

- [1] Chong Bao, Yinda Zhang, Bangbang Yang, Tianxing Fan, Zesong Yang, Hujun Bao, Guofeng Zhang, and Zhaopeng Cui. Sine: Semantic-driven image-based nerf editing with prior-guided editing field. In *The IEEE/CVF Computer Vision and Pattern Recognition Conference (CVPR)*, 2023. 3
- [2] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *CVPR*, 2022. 2, 7
- [3] Tim Brooks, Aleksander Holynski, and Alexei A. Efros. Instructpix2pix: Learning to follow image editing instructions. In *CVPR*, 2023. 2
- [4] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021. 2, 3, 4, 5
- [5] 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, 2023. 3
- [6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021. 3, 4
- [7] Jiemin Fang, Junjie Wang, Xiaopeng Zhang, Lingxi Xie, and Qi Tian. Gaussianeditor: Editing 3d gaussians delicately with text instructions. In *CVPR*, 2024. 3
- [8] Bingchen Gong, Yuehao Wang, Xiaoguang Han, and Qi Dou. Recolornerf: Layer decomposed radiance fields for efficient color editing of 3d scenes. *arXiv preprint arXiv:2301.07958*, 2023. 3
- [9] Ori Gordon, Omri Avrahami, and Dani Lischinski. Blended-nerf: Zero-shot object generation and blending in existing neural radiance fields. *arXiv preprint arXiv:2306.12760*, 2023. 2
- [10] Ayaan Haque, Matthew Tancik, Alexei Efros, Aleksander Holynski, and Angjoo Kanazawa. Instruct-nerf2nerf: Editing 3d scenes with instructions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023. 2
- [11] Muhammad Zubair Irshad, Sergey Zakharov, Katherine Liu, Vitor Guizilini, Thomas Kollar, Adrien Gaidon, Zsolt Kira, and Rares Ambrus. Neo 360: Neural fields for sparse view synthesis of outdoor scenes. 2023. 7
- [12] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), 2023. 2, 3, 4
- [13] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything, 2023. 2, 4, 5
- [14] Sosuke Kobayashi, Eiichi Matsumoto, and Vincent Sitzmann. Decomposing nerf for editing via feature field distillation. In *Advances in Neural Information Processing Systems*, 2022. 3
- [15] Dishani Lahiri, Neeraj Panse, and Moneish Kumar. S2rf: Semantically stylized radiance fields, 2023. 3
- [16] Jae-Hyeok Lee and Dae-Shik Kim. Ice-nerf: Interactive color editing of nerfs via decomposition-aware weight optimization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3491–3501, 2023. 3
- [17] Boyi Li, Kilian Q Weinberger, Serge Belongie, Vladlen Koltun, and Rene Ranftl. Language-driven semantic segmentation. In *International Conference on Learning Representations*, 2022. 3
- [18] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 2, 7
- [19] Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv*, 2022. 2
- [20] 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, 2021. 2
- [21] Etaï Sella, Gal Fiebelman, Peter Hedman, and Hadar Averbuch-Elor. Vox-e: Text-guided voxel editing of 3d objects, 2023. 2
- [22] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015. 4
- [23] Cyrus Vachha and Ayaan Haque. Instruct-gs2gs: Editing 3d gaussian splats with instructions, 2024. 3
- [24] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T. Barron, and Pratul P. Srinivasan. Ref-NeRF: Structured view-dependent appearance for neural radiance fields. *CVPR*, 2022. 2, 7
- [25] Binglun Wang, Niladri Shekhar Dutt, and Niloy J Mitra. Proteusnerf: Fast lightweight nerf editing using 3d-aware image context. *arXiv preprint arXiv:2310.09965*, 2023. 3
- [26] Can Wang, Menglei Chai, Mingming He, Dongdong Chen, and Jing Liao. Clip-nerf: Text-and-image driven manipulation of neural radiance fields. *arXiv preprint arXiv:2112.05139*, 2021. 3
- [27] Zhizhong Wang, Lei Zhao, Haibo Chen, Ailin Li, Zhiwen Zuo, Wei Xing, and Dongming Lu. Texture reformer: towards fast and universal interactive texture transfer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 2624–2632, 2022. 2, 4, 5
- [28] Lin Yen-Chen. Nerf-pytorch. <https://github.com/yenchenlin/nerf-pytorch/>, 2020. 3
- [29] Junyi Zhang, Charles Herrmann, Junhwa Hur, Luisa Polania Cabrera, Varun Jampani, Deqing Sun, and Ming-Hsuan Yang. A tale of two features: Stable diffusion complements dino for zero-shot semantic correspondence. 2023. 4
- [30] Kai Zhang, Nick Kolkin, Sai Bi, Fujun Luan, Zexiang Xu, Eli Shechtman, and Noah Snavely. Arf: Artistic radiance fields, 2022. 2, 4
- [31] Jingyu Zhuang, Di Kang, Yan-Pei Cao, Guanbin Li, Liang Lin, and Ying Shan. Tip-editor: An accurate 3d editor following both text-prompts and image-prompts, 2024. 3

6. Appendix

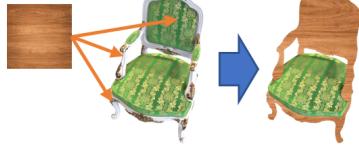
6.1. User Workflow

1. Image Conditional Editing

Choose different image
to auto map style

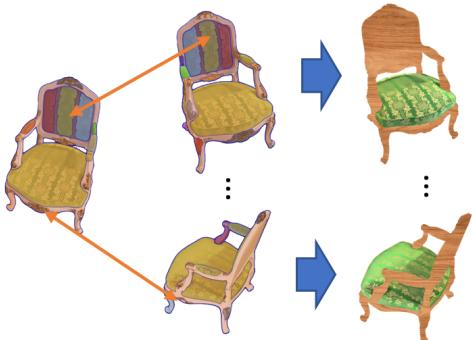


Or edit an
existing view



2. Make Multiple Views

DINO heuristic matches masks between
edit view and sampled views, and copies
changes to appropriate regions in 2D



3. Apply Views onto 3D Gaussian Splat

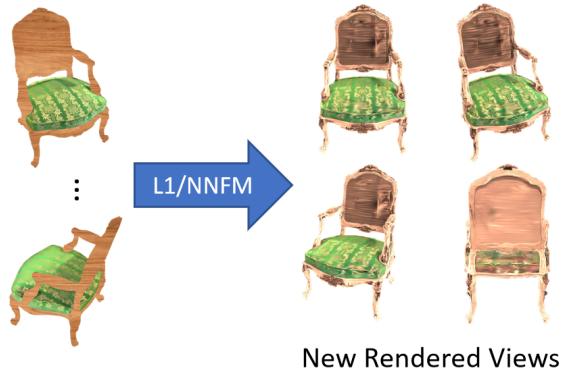


Figure 9. User initially chooses an image to automatically extract correspondences from, or makes an edit to an existing view.

6.2. Additional Editing Results

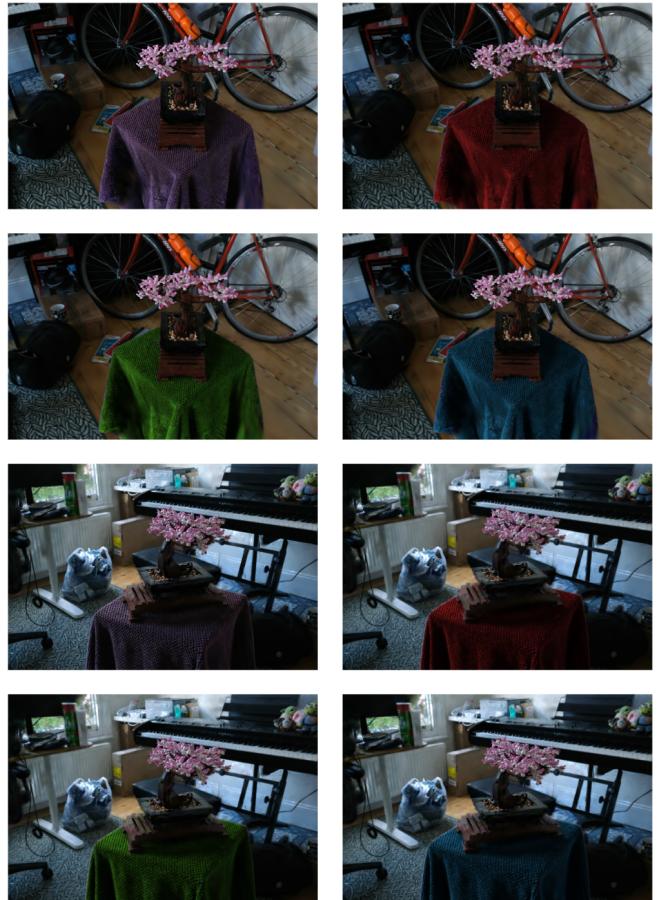


Figure 10. Our method is able to perform bounded and accurate color changes across views, even in cases of a cluttered background with numerous object masks.



Figure 11. Toasting the toast without affecting toaster.



Figure 12. Turning the sidewalk light blue without affecting the street.

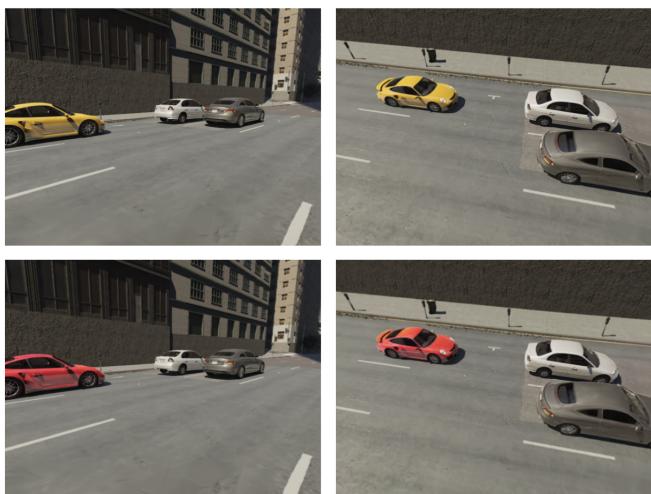


Figure 13. Turning just the yellow car red.



Figure 14. One limitation of our method is that it struggles to accurately copy textures from layered objects like liquid behind a glass.