FastText2ObjectStyle: Efficient 3D Object Stylization with Textual Input

Adam Barroso *, and Noé Artru *
University of Toronto
abarroso@cs.toronto.edu — noeartru@cs.toronto.edu
GitHub Repository

Abstract—With the rise of Neural Radiance Fields (NeRFs) enabling high-quality 3D scene representations as well as Gaussian Splatting for real-time rendering and efficient modification of 3D scenes, there is growing interest in interactive methods for 3D scene editing using Deep Learning. Such capabilities would allow artists to dynamically render and adjust 3D scenes with ease. In this work, we present a novel approach for stylizing individual objects within a 3D scene. Using our model, users can select an object, specify a desired style through textual input, and obtain a new Gaussian splatting representation of the scene in less than 2 minutes.

Index Terms—Gaussian Splatting, 3D Scene Stylization, Text-Based Stylization, 3D Object Segmentation, NeRF

Fig. 1: Example of stylized 3D scenes: (a) "Truck with flames" on truck object (b) "Fire floor" on floor

1 Introduction

Neural Radiance Fields (NeRFs) [1] have revolutionized the way we represent real-world environments as 3D virtual scenes by utilizing neural networks to encode the scene's information efficiently. Building on this research, Gaussian Splatting [2] has enabled real-time rendering of these 3D virtual scenes and has made the 3D rendering more human-interpretable by using Gaussians to form the objects of a scene.

Stylization of 2D images has also gained popularity, notably with the emergence of generative adversarial networks [3] [4] [5] and diffusion models [6] [7]. These models allow users to generate images, and also to stylize existing images (or objects within an image) according to their preferences, either through image or text. The intersection of these two research areas naturally leads to the task of stylizing 3D scenes. This is most commonly done using either an image or text to represent the target style. We will be focusing on textual input in this project, as it has the advantage of conveying a user's ideas more precisely and efficiently thanks to technologies such as speech-to-text [8], enabling real-time creative possibilities for 3D object stylization. A user could for example modify the floor of a 3D scene to resemble lava or a flowery field.

The goal of our project is to develop a semantically accurate 3D object stylizer that enables modifications to objects in a scene based on textual input from the user. Our project is also designed to run quickly enough to be viable for real-time scene modification.

2 Related Work

2.1 Scene Stylization with NeRFs

NeRFs allow us to represent 3D scenes using neural networks [1] to store the scene information in an incredibly dense manner. They map 3D spatial coordinates and viewing directions to RGB values and density. When using NeRF to stylize scene, their ability to synthesize photorealistic views consistently across all perspectives enables easy adjustments to lighting or materials while preserving scene coherence. However, their dense representation of the scene

*Both authors contributed equally to this work

using ”black-box” neural networks makes it particularly hard to stylize 3D transformations such as rotations or translations.

NeRF-Art [9] employs multiple NeRFs as the basis for text-based editing of a scene. A NeRF is trained on the input images to serve as a ground truth and ensure edits still preserve the structure of the original content. A second NeRF is trained to perform stylization. This is done using CLIP [10], which encodes text and images in the same feature space. This allows losses to be generated between the rendered images of different viewpoints of the scene and the input text features.

The paper achieves good results, however, some issues are prevalent with this method. Notably, due to the the compute power needed for training and stylizing NeRFs, the render and training time needed for inferences renders real-time applications impossible. For instance, simply rendering a pre-trained stylized NeRF using NeRF-Art on 100 viewpoints could take upwards of 10 minutes on a A6000 GPU. Given the time taken for inference and the number of adjustable parameters used in the paper, this also limits the amount of different scenes it can be applied to and how extensively a scene can be edited. This paper also did not focus on stylizing specific objects in a 3D scene, but rather the entire scene all at once.

2.2 Scene Stylization with Gaussian Splatting

Due to the aforementioned issues that stem from using NeRFs, Gaussian Splatting has become a preferred choice for most 3D scene representation and stylization applications. Gaussian Splatting involves representing scenes by projecting anisotropic 3D Gaussians onto a 2D plane for efficient, high-quality rendering and view synthesis [2]. As an extension to Gaussian Splatting, Gaussian Grouping [11] reconstruct the 3D scene while also segmenting objects in this 3D space, allowing Gaussian to be grouped according to their object. In particular, Gaussian Grouping relies on recent advancements with papers such as Segment Anything [12] and Decoupled Video Augmentations [13].

Gaussian Grouping was employed in Style-Splat [14] to allow for both segmentation and stylization of multiple objects in an image separately. After training a Gaussian Grouping instance on a 3D scene and choosing a segmentation object, Style-Splat stylizes that object based on a target input image. It fine-tunes the Gaussians used to represent the scene by comparing renders from views in the 3D scene with the stylized image using a nearest neighbor feature matching loss (NNFM). This method is robust and is able to produce high quality stylizations of one or multitude of objects in a 3D scene. However, this method is limited by the need for an input image, which can be hard to produce or source, and can often be an inaccurate representation of the exact edit desired by a user.

2.3 Gaussian Based Scene Stylization

InstructPix2Pix [15] was a major milestone in the image stylization research scene. By creating and using a custom training dataset with GPT3 [16] and Stable Diffusion [17], it trained a conditional diffusion model for image stylization based on textual inputs. The results produced by this model are of very high quality thanks to the use of diffusion models, which have seen huge advancement in recent years.

This paper inspired InstructNeRF2NeRF [18], which allows for stylizing NeRFs using text. It runs by passing rendered views of a 3D scene through InstructPix2Pix and training its NeRF to match the generated images. This model performs very well, achieving better results than NeRF-Art [9] in adaptability, replicability, ease of use, quality and accuracy of edits. InstructGS2GS [19] then extended the same pipeline using Gaussian Splatting [2] instead of NeRFs, dramatically improving rendering times. However, because of the use of diffusion models in these models, training can quickly become both memory and time-intensive, and in particular, the model cannot handle large image sizes. In addition, the model’s overall segmentation process has room for improvements, even on more recent versions of InstructGS2GS. Our project looks to address these two issues, by achieving results without the use of diffusion models and by implementing precise object segmentation.

3 Proposed Method

After experimenting with multiple 3D stylization papers [1] [20], we decided to focus on taking the Style-Splat paper [14] as a basis, and expanding it to allow for both textual inputs and fast training and rendering performance.

3.1 CLIP Loss

Inspired by NeRF-Art’s use of CLIP to compare images and textual inputs, we decided to implement our own CLIP-based loss to train our Gaussians to stylize to certain textual inputs.

3.1.1 Initial Approach

Our initial approach for implementing a image-text loss directly compares rendered images from viewpoints in the 3D Gaussian Splatting scene with a text input. To do this, we evaluate the cosine similarity between the CLIP embeddings of the textual input and our rendered images. Our loss is opposite of that similarity. The algorithm used is described in further detail in the appendix 2.

Unfortunately, this approach had difficulty producing any modifications when training, regardless of training time or learning rate. We decided to investigate this issue by taking a closer look at cosine similarity values between an object in our scene, and a list of words.

The results shown in table 1 helped us realize that cosine similarities between positive and negative associations were very similar. If we wanted our object to stylize, we would have to make our loss more robust to the inherent uncertainty and noise in CLIP embeddings.

3.1.2 Finalized CLIP Loss

In order properly align our masked renderings with the textual input we desire, we decided to add negative text prompts unrelated to the initial textual input. This way, if the rendered image becomes similar to this unrelated text, it would negatively affect the CLIP loss. More precisely, we compute the cosine similarity between our rendered image and 10 different textual embeddings including the initial

TABLE 1: Cosine Similarity between a segmented truck object and different words

Word	Cosine Similarity
truck	0.3105
flower	0.1862
concrete	0.2412
car	0.2556
tree	0.2087
water	0.2194
red	0.1976
computer	0.2169
history	0.2070

prompt. We then compute the negative log likelihood of the softmax of these similarities as our loss. The algorithm used is described in the appendix 2. The negative texts can be chosen manually, but are randomly sampled from a 10000 word bank otherwise.

3.2 Regularization Loss

When we tested our model, we found that unfreezing the Gaussian weights related to opacity, scaling and rotation tended to reduce the fidelity to the original render of the updated scene. In order to balance this, we decided to implement a regularization loss. We implemented this by comparing ground truth renders to updated renders using a pre-trained VGG, a Deep Convolutional Networks specialized in evaluating the similarity of 2 images. We then added that loss to our CLIP loss, alongside a scaling factor to adjust its influence on the overall stylization result.

3.3 Diffusion Model Approach

In an effort to leverage the results of InstructGS2GS while improving its ability to perform object segmentation, we decided to combine it into the Style Splat pipeline. Instead of relying on CLIP loss, this version of our code leverages InstructPix2Pix to modify rendered scene viewpoints. We then compute loss between the original and modified images using LPIPS and L1 loss. LPIPS ensures perceptual similarity between the images, and L1 loss directly compares both images' pixels to capture finer details. Due to the diffusion model's high inference time, we had to make certain optimizations to our pipeline, which are discussed in the next section.

3.4 Optimizing Code

3.4.1 Batching

As discovered in 3.3 an issue with the initial implementation was the time overhead for our training loop, as well as the over-editing of images. As our current toolkit is not designed for rendering multiple views at once, we designed a custom Dataloader to mask and render viewpoints in batches. Our Dataload also grabs all ground-truth images once before the training loop.

The dataset class is then also used to store a dictionary of edited images, this way edits can be made periodically and previous edits can be trained against for multiple epochs. This algorithm is broken down in the appendix 4.

3.4.2 Image sizes

We implemented a more efficient CLIP pipeline by resizing the rendered output directly to 224x224 resolution, matching the input size expected by the CLIP model. This adjustment eliminated the need for post-processing interpolation, which previously added computational overhead. Interestingly, we observed that resizing the image at the render stage, rather than during interpolation, resulted in less noise in the final output, though the exact cause remains unclear. Additionally, this resizing strategy provided a significant reduction in backpropagation time, as we were now training with 224x224 images instead of larger 500+ pixel images. This reduction in backpropagation time proved to be a key benefit, dramatically improving the efficiency of our pipeline without worsening results.

4 Analysis and Evaluation

4.1 Shape

We decided to unfreeze certain Gaussian parameters such as scale, opacity and rotation when running stylization in order to test the feasibility of multi-dimensional edits, a feature previously not implemented in Style-Splat:

Results in figure 2 showed us that unfreezing parameters could help significantly improve results for modifying 3D attributes of an object. In particular, unfreezing rotation was found to have little impact. Unfreezing opacity showed major artifacts appearing beyond the objects boundaries because of invisible Gaussians slowly appearing, and unfreezing scaling was shown to have the most beneficial impact, allowing for a successful "Spiky Truck" result.

4.2 Regularization loss

We conducted experiments on the influence of the regularization loss by scaling it with different factors relative to the CLIP loss. As seen in figure 3, the regularization loss has a quite significant impact, helping reduce overstretched or out of position Gaussians. However, it either doesn't completely negate them on smaller scaling values, or leads to blurring on higher scaling values. We hypothesize that this is due to our algorithm not being able to further subdivide Gaussians during stylization. This leads to the regularization loss applying blurring of Gaussians as the most efficient solution, which provides an unsatisfying local minimum.

4.3 Diffusion Model results

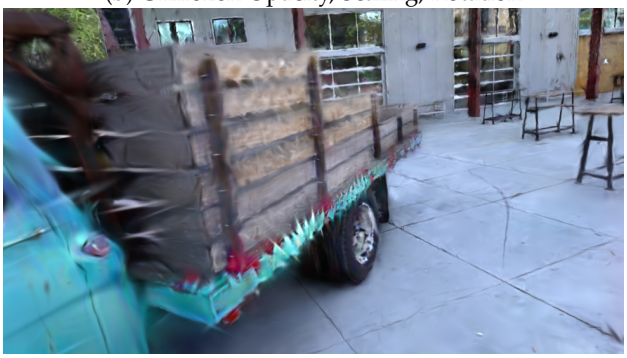
After as tuning hyper-parameters and unfreezing the correct Gaussian parameters, we were able to obtain results such as in figure 4. It was found that training for more time resulted in a reduction in overall quality caused by unnaturally stretched Gaussians. This is also why 4 total edits were used, to prevent over fitting leading to Gaussian expansion. The edits are noticeable when looking at the loss graph in figure 4. The training was performed on an RTXA6000 using the maximum batch size of 64, and took around 10 minutes in total for the edit.

We initially encountered memory limitations when increasing batch size in our training loop. To remedy this we had to downsize the image rendering to 128x128. However,

(a) Frozen Opacity, Scaling, Rotation (Just Color)

(b) Unfrozen Opacity, Scaling, Rotation

(c) Unfrozen Opacity, Rotation



(d) Unfrozen Scaling, Rotation

Fig. 2: Various combinations of Gaussian parameter unfreezing for the prompt "Spiky Truck"

this modification could be the cause of reduced image quality we obtain. In particular, interpolating the images back to their original scale could be causing smearing and Gaussian noise.

4.4 Speedup Comparison

In order to measure the effects of the optimization steps outlined in 3.4, training time comparisons were done over a series of tests using the same prompt with results as shown in table 2.

TABLE 2: Comparison of Training Pipelines: With and Without Code Optimizations

Metric	Without	With
Prompt	Truck with Flames	
Hardware	RTX A6000	
Editing Epochs	20 (5020 Equivalent Iterations)	
Batch Size	N/A (1)	64
Average Training Time (seconds)	398 ± 1.5	101 ± 0.5

The resulting renders were then compared with no noticeable degradation in image quality despite a fourfold speed increase. The results are shown in figure 5.

4.5 Final Results

After all modifications to our training losses, tweaking learning rates for all the different Gaussians parameters, and adjusting global hyper-parameters, we obtained a few interesting results which we show below in the appendix 6.

5 Future Work

One of the key possible improvements we identified is the integration of Gaussian densification during the stylization process. Currently, we do not remove or add Gaussians during the stylization process, which leads to Gaussians spreading and stretching to try and match the edit. This in turn causes artifacts and blurriness to appear in the image, which might correspond to local minima in our loss because of the lack of densification. Adding densification could address these current limitations, leading to more refined and visually appealing results.

Another key improvement would be adding automatic object selection. By enabling the method to automatically identify and select objects within a scene, we can significantly improve both the efficiency and user-friendliness of the system, reducing the need for manual intervention.

Optimizing rendering speed and reducing overall run time are also priorities for future work on this paper. This could be done through more efficient storage or better parallelization processes. Furthermore, work could be done to enable coherent post- processing of the images after rendering to reduce artifacts without the need of future training.

6 Discussion and Conclusion

Our project explored the current state of 3D scene stylization research. By re-implementing foundational papers in 3D scene stylization, we identified a central research question: how can we achieve efficient, close to real-time, text-based stylization of objects within a 3D scene.

(a) Scale Factor: 0.01

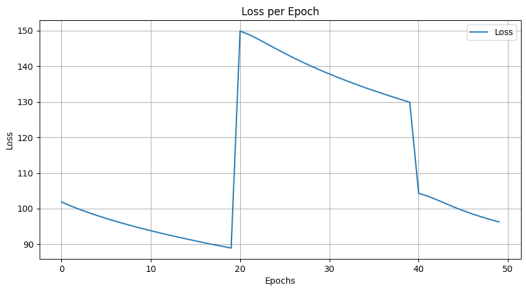
(b) Scale Factor: 0.1

(c) Scale Factor: 0.4

(d) Scale Factor: 1

Fig. 3: Various scaling factors for Regularization Loss on ”Van Gogh” prompt

(a) ”Truck with flames” prompt



(b) Loss plot

Fig. 4: Diffusion Model pipeline results on truck stylization prompt

One of our main contributions was modifying the Style-Splat paper to incorporate a CLIP-based loss for text-based stylization of 3D object. We also tested various modifications to enhance the original paper's performance, including selectively freezing or unfreezing specific Gaussian parameters and introducing a regularization loss using a VGG model.

We also experimented with integrating a pixel-to-pixel approach similar to GS2GS into our pipeline. However, through this process, we concluded that relying on diffusion models for stylization would require too much computational cost for use real-time applications, despite the significant quality improvements.

This motivated us to optimize the Style-Splat pipeline, by implementing batching and adjusting the image resizing process. This helped us achieving a 4 times improvement in model speed without sacrificing results.

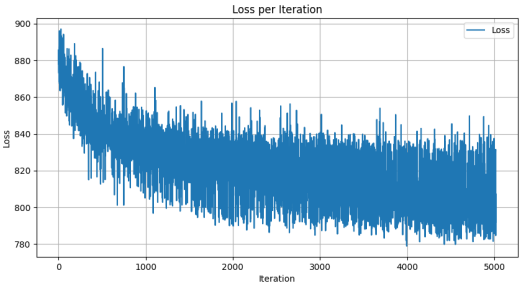
Despite not having quite achieved real-time editing speed of a few seconds, our model can produce encouraging results in less than a minute, with highly accurate object segmentation. Potential improvements to our training pipeline could allow for more fine-detailed edits and run times even closer to real-time editting.

References

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, “Nerf: Representing scenes as neural radiance fields for view synthesis,” 2020. [Online]. Available: https://arxiv.org/abs/2003.08934

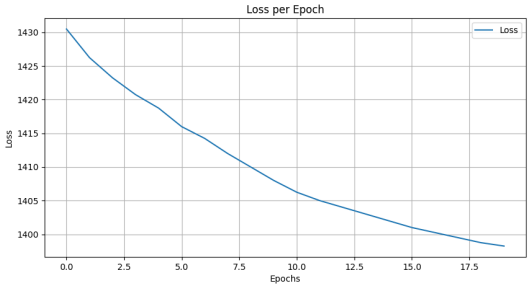
[2] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, “3d gaussian splatting for real-time radiance field rendering,” *ACM Transactions on Graphics*, vol. 42, no. 4, July 2023. [Online]. Available: https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

(a) Original Pipeline "Truck with Fire"



(b) Original Pipeline Loss vs Iterations

(c) Optimized Pipeline "Truck with Fire"



(d) Optimized Pipeline Loss vs Epochs

Fig. 5: Comparison between original and optimized pipeline on "Truck with Flames" prompt

[3] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014. [Online]. Available: https://arxiv.org/abs/1406.2661

[4] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," 2019. [Online]. Available: https://arxiv.org/abs/1812.04948

[5] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," 2020. [Online]. Available: https://arxiv.org/abs/1703.10593

[6] S. Li, "Diffstyler: Diffusion-based localized image style transfer," 2024. [Online]. Available: https://arxiv.org/abs/2403.18461

[7] Z. Wang, L. Zhao, and W. Xing, "Stylediffusion: Controllable disentangled style transfer via diffusion models," 2023. [Online]. Available: https://arxiv.org/abs/2308.07863

[8] N. Sethiya and C. K. Maurya, "End-to-end speech-to-text translation: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2312.01053

[9] C. Wang, R. Jiang, M. Chai, M. He, D. Chen, and J. Liao, "Nerf-art: Text-driven neural radiance fields stylization," *arXiv preprint arXiv:2212.08070*, 2022.

[10] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," 2021. [Online]. Available: https://arxiv.org/abs/2103.00020

[11] M. Ye, M. Danelljan, F. Yu, and L. Ke, "Gaussian grouping: Segment and edit anything in 3d scenes," 2024. [Online]. Available: https://arxiv.org/abs/2312.00732

[12] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, "Segment anything," 2023. [Online]. Available: https://arxiv.org/abs/2304.02643

[13] H. K. Cheng, S. W. Oh, B. Price, A. Schwing, and J.-Y. Lee, "Tracking anything with decoupled video segmentation," 2023. [Online]. Available: https://arxiv.org/abs/2309.03903

[14] S. Jain, A. Kuthiala, P. S. Sethi, and P. Saxena, "Stylesplat: 3d object style transfer with gaussian splatting," 2024. [Online]. Available: https://arxiv.org/abs/2407.09473

[15] T. Brooks, A. Holynski, and A. A. Efros, "Instructpix2pix: Learning to follow image editing instructions," in *CVPR*, 2023.

[16] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 1877–1901, 2020.

[17] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 10 684–10 695.

[18] A. Haque, M. Tancik, A. Efros, A. Holynski, and A. Kanazawa, "Instruct-nerf2nerf: Editing 3d scenes with instructions," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

[19] C. Vachha and A. Haque, "Instruct-gs2gs: Editing 3d gaussian splats with instructions," 2024. [Online]. Available: https://instruct-gs2gs.github.io/

[20] K. Liu, F. Zhan, M. Xu, C. Theobalt, L. Shao, and S. Lu, "Stylegaussian: Instant 3d style transfer with gaussian splatting," *arXiv preprint arXiv:2403.07807*, 2024.

APPENDIX
VARIOUS RESULTS

(a) ”black hole” on manhole cover

(b) ”Truck with flames” on truck

(c) ”Fire floor” on floor

(d) ”Galaxy” on truck

Fig. 6: Various result using our model (FastText2Object Style)

ALGORITHMS IMPLEMENTED FOR OUR PROJECT

Algorithm 1 Updated Pipeline for CLIP Loss

Initialize scene v, textual input T.
Compute textual CLIP embedding: $E_t \leftarrow CLIP(T)$
for each iteration **do**
  Render the current view: $I_r \leftarrow \text{render}(v)$.
  Generate object mask: $M \leftarrow \text{generate\_object\_mask}(v)$.
  Mask image: $I_r^M \leftarrow I_r \odot M$.
  Compute Image CLIP embedding: $E_i \leftarrow CLIP(I_r)$
  Compute loss using cosine similarity:
    $\mathcal{L}_{CLIP\_TEST} \leftarrow -COS\_SIM(E_t, E_i)$.
  Backpropagate against $\mathcal{L}_{CLIP\_TEST}$, ensuring updates only affect masked object features:
    backpropagate($\mathcal{L}_{CLIP}, M$).
end for
Finalize training.

Algorithm 2 Pipeline for CLIP Loss

Initialize scene v, textual input T, negative texts $N_0, ..., N_9$, temperature t.
Compute textual CLIP embeddings: $E_t \leftarrow CLIP(T), E_{N_0} \leftarrow CLIP(N_0), ...$
for each iteration **do**
  Render the current view: $I_r \leftarrow \text{render}(v)$.
  Generate object mask: $M \leftarrow \text{generate\_object\_mask}(v)$.
  Mask image: $I_r^M \leftarrow I_r \odot M$.
  Compute Image CLIP embedding: $E_i \leftarrow CLIP(I_r)$
  Compute similarity weights: $W_t = \exp(\text{CS}(E_t, E_i)/t)$, $W_0 = \exp(\text{CS}(E_{N_0}, E_i)/t), ...$
  Compute negative log likely-hood loss:
    $\mathcal{L}_{CLIP} \leftarrow -\log\left(\frac{W_t}{W_t + \sum_{i=0}^{9} W_i}\right)$
  Backpropagate against $\mathcal{L}_{CLIP}$, ensuring updates only affect masked object features:
    backpropagate($\mathcal{L}_{CLIP}, M$).
end for
Finalize training.

Algorithm 3 Pipeline for Object Segmentation and Editing using Style Splat and InstructGS2GS

Initialize pipeline parameters.
for each iteration **do**
  Render the current view: $I_r \leftarrow \text{render}(v)$.
  Retrieve ground truth image for the view: $I_g \leftarrow \text{get\_ground\_truth}(v)$.
  Generate object mask: $M \leftarrow \text{generate\_object\_mask}(v)$.
  Mask image: $I_g^M \leftarrow I_g \odot M$, $I_r^M \leftarrow I_r \odot M$.
  Generate edited image using InstructPix2Pix:
    $I_e \leftarrow \text{instruct\_pix2pix}(I_r, I_g^M, \text{text\_input})$.
  Compute losses:
    $\mathcal{L}_{LPIPS} \leftarrow \text{compute\_lpips\_loss}(I_e, I_r)$.
    $\mathcal{L}_{L1} \leftarrow \text{compute\_l1\_loss}(I_e, I_r)$.
    $\mathcal{L}_{RGB} \leftarrow \text{compute\_rgb\_loss}(I_e, I_r)$.
  Combine losses: $\mathcal{L}_{total} \leftarrow \mathcal{L}_{LPIPS} + \mathcal{L}_{L1} + \mathcal{L}_{RGB}$.
  Backpropagate against $\mathcal{L}_{total}$, ensuring updates only affect masked object features:
    backpropagate($\mathcal{L}_{total}, M$).
end for
Finalize training.

Algorithm 4 Diffusion Object Style Transfer Batching Training Loop

Initialize training parameters: num_epochs, edit_freq, dataloader, previously_edited_images.
for epoch = 1 **to** num_epochs **do**
  for batch **in** dataloader **do**
    Load batch of rendered images $I_r \leftarrow$ load_batch(v), ground truth images $I_g \leftarrow$.
    if epoch % edit_freq == 0 **then**
      Edit batch images using current model:
        $I_e \leftarrow$ edit_images(I_r, I_g).
      Update edited images dictionary:
        edited_images[epoch] $\leftarrow$ I_e.
      Append indices of processed batch images to seen indices: seen_indices.append(indices).
    else
      Load previously edited batch images from data:
      $I_e \leftarrow$ previously_edited_images$[indices]$.
    end if
    Compute losses:
    LPIPS loss:
    $\mathcal{L}_{LPIPS} \leftarrow$ compute_lpips_loss(I_e, I_r).
    texttt{L1 loss:
    $\mathcal{L}_{L1} \leftarrow$ compute_l1_loss(I_e, I_r).
    RGB loss:
    $\mathcal{L}_{RGB} \leftarrow$ compute_rgb_loss(I_e, I_r).
    Total loss: $\mathcal{L}_{total} \leftarrow \mathcal{L}_{LPIPS} + \mathcal{L}_{L1} + \mathcal{L}_{RGB}$.
    Back-propagate loss for batch: back-propagate$(\mathcal{L}_{total})$ and update model weights.
  end for
end for
Finalize training.