Dynamic Scene Editing via Semantically Trained 3D Guassians

by

Jordan Lam

S.B. Computer Science and Engineering, MIT, 2024

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2025

Authored by:	Jordan Lam Department of Electrical Engineering and Computer Science May 16, 2025
Certified by:	Gregory W. Wornell Professor of Electrical Engineering and Computer Science, Thesis Supervisor
Accepted by:	Katrina LaCurts Chair, Master of Engineering Theiss Committee

Dynamic Scene Editing via Semantically Trained 3D Guassians

by

Jordan Lam

Submitted to the Department of Electrical Engineering and Computer Science
on May 16, 2025 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

ABSTRACT

Image-based 3D scene reconstruction continues to be a challenge as it involves solving both the sufficient 3D representation problem and the 3D reconstruction itself. One approach to tackle the rendering problem is 3D Gaussian Splatting because of its potential to produce fast and realistic renders via 3D Gaussian representation. With many applications in the entertainment industry, there is motivation in using 3D Gaussian Splatting for not only reconstructing 3D dynamic scenes but also editing them. However, extending the problem to dynamic 3D scenes proves to be a challenging task as it involves discerning the correct representation of a 3D scene while maintaining the capability to render in real time. State-of-the-art methods have proposed methods that reconstruct dynamic scenes or edit static scenes, but the problem of editing dynamic scenes is still underexplored. This thesis analyzes the feasibility of editing semantically trained Gaussians for dynamic 3D scene editing. By training 3D Gaussians to represent the semantics across the time steps of a dynamic 3D scene, these primitives can be combined with an image editing pipeline to perform real-time, realistic 3D scene editing. Results show that editing segmented 3D Gaussians produces higher-quality and efficient renders as compared to editing without segmentation. However, when evaluated for mainstream applications, results show the impracticality of this pipeline and draw focus to memory and editing limitations that need to be further researched for future advances in 3D Gaussian Splatting.

Thesis supervisor: Gregory W. Wornell
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I thank my research advisor, Safa Medin, who not only helped me with writing this thesis and through research blockers but was also a precious friend to me in the past year. I could not have made all this progress without you. I also thank my supervisor, Gregory Wornell, for being patient and understanding of my research progress, and for all his expertise in writing this thesis. Finally, I thank all the friends I made in the SIA lab who continued to support me in my research endeavors.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Image-based 3D scene reconstruction aims to generate a digital 3D representation of a scene from multiple images, that can be computationally processed and manipulated. The problem of finding a sufficient 3D representation to model the complexity of real-world environments, and then solving the reconstruction itself, is fundamental for a wide range of applications such as robot navigation [1], autonomous driving [2], and 3D modeling and animation [3].

3D scene reconstruction's early endeavors began with traditional scene reconstruction and rendering, focusing on light fields, dense sampling, and structured capture [4–6]. The core problem was trying to balance the complexity of representing a scene while still being able to render it efficiently. For example, determining the sufficient number of spherical harmonic (SH) coefficients [7] to compute the appearance at a point induced more computational complexity, or the renderings of high-resolution images prompted a bigger demand for real-time rendering. With the emergence of structure-from-motion [8] and multi-view stereo algorithms [9], coupled with the surge of deep neural networks, a major success was found in 3D scene reconstruction. Neural Radiance Fields (NeRF) [10] was one such stride in producing high-quality results by representing scenes with a continuous, volumetric function, so a direct mapping can be made from spatial coordinates to color and density. However, NeRF-based methods are computationally intensive [11] and are resistant to modifications because a scene's geometry and appearance are implicitly defined.

This brings into context the emergence of 3D Gaussian Splatting (3DGS) [12] as a perspective shift in scene representation and reconstruction. Rather than representing a scene implicitly, as in NeRF, 3DGS uses an explicit scene representation of learnable 3D Gaussian primitives in space. 3DGS has not only found success in producing high-quality image synthesis but also efficient, parallelized rendering, outperforming NeRF-based methods.

Many advances [13] in 3DGS have found success in reconstructing dynamic scenes and editing static scenes for applications in the entertainment industry. Building on these advances, there lies the potential for a dynamic scene editor. One mainstream application with applications in dynamic scenes is video editing. Specifically, we set the setting as follows: a filmmaker has shot scenes, and in the process of splicing and editing the scenes together, they notice that a specific part of the scene is not to their standards. None of their available clips fit their standards, and they want to reschedule another shooting day to capture the clip they have in mind. However, this causes trouble in scheduling, incurs extra money costs, and the environment may not have the conditions (i.e., weather and lighting) that are needed

for the scene. A dynamic scene editor would allow filmmakers to avoid costly re-shooting while preserving the core visuals in their scenes. Other applications of dynamic scene editing include room simulators: simulating the various appearances of static objects as other objects in the scene interact with them, and virtual sandboxes: simulating the interactions of users with objects in a virtual environment.

However, extending the principles of 3D reconstruction to editing dynamic scenes has proven to be difficult. State-of-the-art methods are often limited by their long training times and limited editing capabilities that are not performant in such applications. One method [14] uses image semantics to perform segmentation on 3D Gaussian primitives to track dynamic movement, but lacks an editing pipeline. Another method [15] performs edits on static scenes of 3D Gaussian primitives, but cannot perform edits on dynamic scenes. This thesis provides a research study that will demonstrate the feasibility of semantically edited Gaussian primitives to realize real-time dynamic 3D scene editing. It is a stride towards using 3D Gaussian Splatting for dynamic scene editing applications and highlighting bottlenecks and other limitations in this new domain.

To summarize, the main contributions of this paper include:

- We demonstrate the feasibility of semantically edited Gaussian primitives to realize real-time dynamic 3D scene editing
- We discuss the effectiveness of segmented editing compared to other image editors in the context of mainstream applications
- We introduce paths of further research that are needed in the recent growth of 3D Gaussian Splatting

Chapter 2

Related Work

In this chapter, we give a brief overview of traditional reconstruction and then discuss point-based rendering and radiance fields. We then follow with an overview of 3D Gaussian Splatting, which shares similarities to previous 3D reconstruction methods, and recent advances that influenced the direction of this thesis.

2.1 Traditional Scene Reconstruction and Rendering

The first approaches that pioneered novel-view synthesis were based on light fields that were densely sampled [4, 5], which was followed by unstructured capture [6]. The emergence of Structure-from-Motion (SfM) [8] paved the way for new research areas such that a collection of photos could be used for synthesizing novel views. SfM can be used to estimate a point cloud during camera calibration for simple visualization of 3D space. Building on SfM, subsequent multi-view stereo (MVS) was able to produce full 3D reconstruction algorithms [9], leading to the development of view synthesis algorithms [16–18]. These algorithms re-project and blend input images into the novel view camera, and guide the projection using the geometry. However, these methods generally suffer from under-reconstruction, are unable to recover parts of a scene, or over-reconstruction, when nonexistent geometry is generated from MVS. Recently, neural rendering approaches [19] reduce such artifacts and avoid the storage requirement of storing input images on GPUs.

2.2 Neural Rendering and Radiance Fields

Soft3D [20] is often viewed as a precursor to radiance fields because by introducing soft visibility and differentiable rendering, it set the stage for learning 3D representations without explicit 3D supervision. Combining deep-learning techniques with volumetric ray-marching, [21, 22] proposed a continuous differentiable density field to implicitly represent geometry. However, rendering with volumetric ray-marching incurs a high cost due to the high number of samples required to query the volume. Neural Radiance Fields (NeRF) [10] built on this approach by introducing importance sampling and positional encoding via a Multi-Layer Perceptron (MLP), but the large MLP contributed to slower rendering. Recent advances [23]

have augmented NeRF's approach to addressing quality and speed. Nerfies [24] proposed an additional continuous volumetric deformation field to warp each point into a canonical 5D NeRF, where input is given as a 3D spatial coordinate and 2D viewing direction. Ref-Nerf [25] improved NeRF's appearance rendering by using a representation of reflected radiance and structures. ENVIDR [26] improved the reconstruction of surfaces with challenging specular reflections by introducing a neural renderer, decoupled from scene representation, that learns the interactions between surface and environment lighting. Although these approaches produced high-quality renders, their render times remained non-ideal.

Other recent methods turned their attention to achieving faster training and rendering times with three main design choices, by 1) using spatial data structures to store neural features used during volumetric ray-marching, 2) employing various methods to achieve different encodings, and 3) optimizing MLP capacity. InstantNGP [27] tackled the slow render times by utilizing a hash grid and occupancy grid to accelerate computation. Plenoxels [28] avoided using neural networks and used a voxel grid to interpolate a continuous density field. Both methods utilize Spherical Harmonics: InstantNGP uses it to represent directional effects, and Plenoxels to encode inputs to the color network. However, both struggle to represent space effectively and are hindered by the need to query many samples for ray-marching.

2.3 Volumetric Rendering and NeRF

In an image formation model, a color C can be computed by volumetric rendering along a ray. Formally:

$$C = \sum_{i=1}^{N} c_i(1 - \exp(-\sigma_i\delta_i))T_i \text{ and } T_i = \exp(-\sum_{j=1}^{i-1} \sigma_j\delta_j)$$

where density σ, transmittance T, and color c are sampled along the ray with intervals δ_i. Rewriting this equation gives us:

$$C = \sum_{i=1}^{N} c_i\alpha_i T_i \text{ and } T_i = \prod_{j=1}^{i-1}(1 - \alpha_j)$$

$$\alpha_i = (1 - \exp(-\sigma_i\delta_i))$$

A neural point-based approach, as seen in NeRF [10], computes the color C of a pixel by blending P ordered points overlapping at that pixel as:

$$C = \sum_{i=1}^{P} c_i\alpha_i \prod_{j=1}^{i-1}(1 - \alpha_j)$$

where c_i is the color of a point.

The volumetric rendering in NeRF shares the same image formation model, but the rendering algorithm is different because expensive random sampling is required to find the

samples. Overall, NeRF can be summarized in Figure 2.1. Given an input where x is a point in space (x, y, z), (θ, ϕ) is a direction specified by spherical coordinates, and F_Θ is the MLP network that maps (x, θ, ϕ) to the output radiance information $RGB\sigma$ as color and density, which can be used for the volumetric rendering. Using the renderings, optimization can be done as the loss between the rendered and true pixel colors via total squared error.

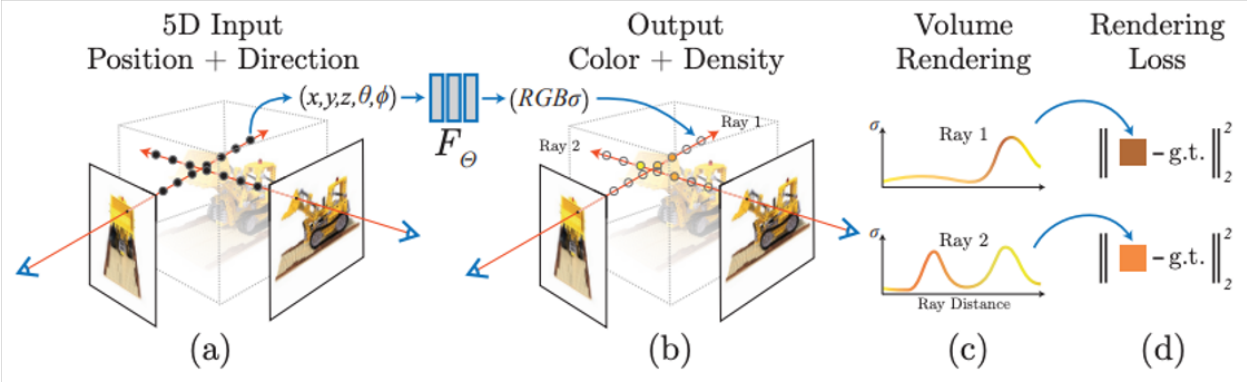

Figure 2.1: An overview of NeRF [10], starting with the querying of a point and viewing direction (a), the output radiance information (b), volumetric rendering (c), and ending with the optimization of the network (d).

2.4 3D Gaussian Splatting

At its core, point sample rendering [29] rasterizes an unstructured set of points, for which it utilizes graphics APIs [30] or parallel software rasterization on the GPU [31, 32]. Although point sample rendering faces issues with holes in the renders, aliasing, and is strictly discontinuous, by "splatting" point primitives larger than a pixel, such as circular or elliptic discs, ellipsoids, or surfels [33–36], these issues can be addressed.

Differentiable point-based rendering techniques [37, 38] have also been proposed, and by augmenting the points with neural features and rendering with a CNN [39, 40], they can result in fast and even real-time view synthesis.

3D Gaussian Splatting (3DGS) [12] combines explicit, unstructured data storage of classical rendering techniques with modern optimization *without* neural components. By representing a scene with GPU-friendly 3D Gaussian primitives, which we will refer to as 3D Gaussians in the rest of this thesis, coupled with a real-time differentiable renderer, 3DGS achieves better quality and faster rendering speeds than previous methods. Advances in 3DGS [41] have succeeded in both high-quality renders and fast rendering times by augmenting 3DGS. PixelSplat [42] efficiently reconstructs 3D radiance fields from pairs of images by overcoming local minima via sampling from a dense probability distribution over 3D. GS-IR [43] proposed an optimization scheme that incorporated regularization for normal estimation and baking-based occlusion to model indirect lighting. In this section, we give a brief overview of 3DGS.

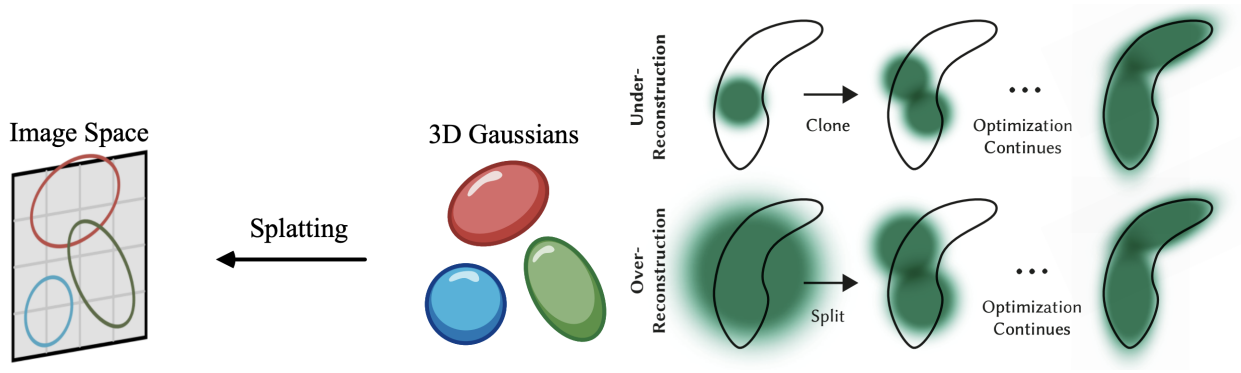

Figure 2.2: An overview of 3D Gaussian Splatting, starting with the projection of 3D Gaussians into image space [41] followed by the adaptive densification scheme [12].

2.4.1 Rendering

Using 3D Gaussians as the primitives to represent a scene rather than points, they can be easily projected to 2D splats, allowing for fast α-blending. Each 3D Gaussian is characterized by its center position μ and 3D covariance matrix Σ [36] at a voxel location x:

$$G(x) = \exp(-\frac{1}{2}(x-\mu_i)^T\Sigma_i^{-1}(x-\mu_i))$$

Since the 3D Gaussian is in 3D space, it needs to be projected into 2D for rendering. Given a viewing transformation matrix W, the covariance matrix $\mathbf{\Sigma}$ to do this projection [33] for a 3D Gaussian can be computed as:

$$\mathbf{\Sigma_i} = JW\Sigma_i W^T J^T$$

where J is the Jacobian of the affine approximation of the projective transformation. After projection, the 3D Gaussian can be evaluated as a 2D Gaussian G' at a pixel location x':

$$G'(x') = \exp(-\frac{1}{2}(x'-\mu_i')^T\Sigma_i'^{-1}(x'-\mu_i')$$

where the 2D Gaussian is evaluated with the covariance matrix Σ' [38], and μ' is a point in the projected space. Volumetric rendering can then follow as usual with α_i defined as multiplying the learned opacity α' by the 2D Gaussian during the blending process:

$$\alpha_i = \alpha_i' \times \exp(-\frac{1}{2}(x'-\mu_i')^T\Sigma_i'^{-1}(x'-\mu_i')$$

2.4.2 Optimization

Once an image is constructed via rendering, the difference between the rendered image and the ground truth image can be measured. All the learnable parameters can be optimized using stochastic gradient descent and the L_1 and SSIM [44] loss functions:

$$L = (1-\lambda)L_1 + \lambda L_{SSIM}$$

Optimizing the 3D Gaussians also includes the "splatting" component of 3D Gaussian Splatting. This begins with the initialization of these primitives from a set of sparse points via Structure from Motion (SfM) [45] or random initialization. A good initialization is important for convergence and reconstruction quality [46].

As a final step, optimization of the 3D Gaussians involves density control. Since the scene representation is modeled as a set of individual primitives lying in 3D space, the density of a subset of primitives can be adaptively increased to better capture the details of a scene. As depicted in Figure 2.2, the densification procedure involves cloning small primitives in under-reconstructed areas, splitting large primitives in over-reconstructed regions, or pruning less impactful primitives.

2.5 Dynamic Scene Reconstruction

Recently, a new challenge has been added to the 3D reconstruction problem. That is, the reconstruction of a given scene where objects are interacting with each other, a *dynamic* scene. Recent advances in 3D Gaussian Splatting [13] have demonstrated various dynamic scene reconstruction capabilities with 3D Gaussians. GaussianAvatar [47] proposes a dynamic appearance network coupled with an optimizable feature tensor to learn motion-to-appearance mapping and a differentiable motion condition for motion estimation. Neural Parametric Gaussians (NPGs) [48] proposes a two-stage approach by first using a low-rank neural deformation model to learn an object's deformations, and second, optimizing a local 3D Gaussian representation for non-rigid reconstruction. [49] proposes binding 3D Gaussians over an explicit mesh to enable interactive deformation.

On the other hand, D-NeRF [3] tackles dynamic 3D scene reconstruction via a combination of implicit radiance fields with deformation fields, producing exceptional renders of dynamic objects. Applying deformation fields to 3D Gaussian Splatting, one can decouple a scene into a static representation and a separate deformation network that operates on the primitives [50]. Dynamic 3D Gaussians Distillation (DGD) [14] utilizes this observation and distills semantic features to each primitive, which can be used to track and optimize primitive changes over time. DGD uses score distillation sampling (SDS) to generate 3D content from given text prompts [51] by distilling information from 2D images produced by a pre-trained text-to-image diffusion model into a differentiable 3D representation. Therefore, the semantic features of a 3D scene can be generated and used as a loss within a generic continuous optimization problem.

2.6 Static Scene Editing

In a similar field, many works propose static scene editing. Instruct-GS2GS [52] builds on Instruct-NeRF2NeRF [53] by proposing an iterative dataset update method to make consistent edits on 3D Gaussians. GaussianEditor [54] proposes Gaussian semantic tracing to trace objects, so editing in the form of object removal and integration can be performed. [55] proposes editing content on a 2D image plane to drive the 3D scene editing of 3D Gaussians. In all of these methods, fast renderings are achieved by taking advantage of the primitive

nature of 3D Gaussians. In other words, given a desired edit, it is not necessary to train all 3D Gaussians in the scene, but only the 3D Gaussians that are affected by the edit, and all other 3D Gaussians can be rendered as they usually would. This holds an advantage over neural network scene editors, such as NeRF-Editing [56], since rendering edits involves retraining and querying the neural network, hindering the capabilities of editing. Direct modifications to the neural network's weights do not directly relate to changes in a scene's geometry or appearance.

Direct Gaussian 3D Editing (DGE) [15] utilizes the advantage of editing 3D Gaussians, and trains 3D Gaussians via the denoising diffusion model InstructPix2Pix (IP2P) [57]. IP2P has seen exceptional performance in high-quality renders in a matter of seconds. Combining the performance of IP2P with the editing capabilities of 3D Gaussians, DGE updates 3D Gaussians directly to obtain multi-view consistent edits.

A significant limitation of DGE is its application to only static scenes. By avoiding the capturing of objects across time frames, DGE is unable to represent the features of a dynamic scene. Although DGE can edit each image frame in a video separately, these edits are done independently from each other, so scene information is lost over time. Dynamic semantics, however, can be integrated into the editing process of DGE, which would make it applicable for real-time dynamic 3D scene editing. Our research proposes a method to perform dynamic scene editing via semantically trained 3D Gaussians.

Chapter 3

Methodology

This chapter explains the method taken to research the use of semantically trained 3D Gaussians for dynamic scene editing. At a high level, we research the feasibility of combining the segmentation pipeline of DGD [14] with the editing pipeline of DGE [15]. In other words, the two major components of performing dynamic scene editing can be described as: one, segmenting a dynamic scene of trained 3D Gaussians, and two, using the segmented 3D Gaussians in conjunction with an image editor to perform selective scene editing. Specifically, using a 2D feature extractor, 3D Gaussians are trained to learn a semantic feature parameter, to extract a mask of 3D Gaussians of similar semantic feature value for use in an image editing pipeline for 3D Gaussians.

3.1 Semantically Training 3D Gaussians

Editing dynamic scenes begins with acquiring 3D Gaussians, which are trained on a dynamic scene. The method used in DGD was adapted to train 3D Gaussians on data in the Hypernerf [58] dataset. In a later section, further discussion is provided on the selection of this dataset. In this section, we elaborate on the design of the segmentation pipeline in DGD and describe the two main steps of training semantic feature values and segmentation. We also note design changes made to have it work with an editing pipeline.

3.1.1 Training Semantic Value

The semantic features of a scene can be distilled into 3D Gaussians by first attaching a learnable feature value f_n, making each 3D Gaussian characterized with parameters:

$$G(\mu_n, \Sigma_n, c_n, \alpha_n, f_n)$$

where the learnable parameters are μ_n as the center position, Σ_n as the covariance matrix, c_n as the color, and α_n as the opacity. Given a video, we can sample N equally spaced frames to obtain the frames $\{t_0, ...t_{N-1}\}$, the pretrained 2D feature extractor DINOv2 [59] associates each frame with a feature value F_D which is used to train the learnable parameter f_n that is

rendered to the value F:

$$F = \sum_{i=1}^{N} f_i \alpha_i \prod_{j=1}^{i-1} (1 - \alpha_j)$$

By designing the feature value to be modeled similarly to the color value of a 3D Gaussian, it allows the feature value to be optimized and rendered in the same process as color. Therefore, it follows that we can run the same process of rendering and optimization of 3D Gaussians as before, leaving the principal concept of the training pipeline unchanged.

In addition, we train with a deformation field D, parameterized as a multilayer perceptron (MLP). Specifically, given a time frame t and a 3D Gaussian $G(\mu_n, \Sigma_n, c_n, \alpha_n, f_n)$, the deformation field D can output the changes $\delta\mu_n, \delta\Sigma_n$, resulting in the translated 3D Gaussian $G(\mu_n + \delta x_n, \Sigma_n + \delta\Sigma_n, c_n, \alpha_n, f_n)$. That is:

$$\delta\mu_n, \delta\Sigma_n = D(\gamma(\mu_n), \gamma(t))$$

where γ is frequency based encoding of [60]. This allows us to adaptively move 3D Gaussians based on the time frame, and remove the requirement to save a point cloud of 3D Gaussians for each time frame.

3.1.2 Segmentation

Given the trained 3D Gaussians from the previous section, segmentation can be performed. Specifically, a mask of 3D Gaussians M can be obtained by filtering 3D Gaussians with a semantic feature f_n that falls within a cosine similarity threshold of a desired semantic feature's rendered value. More concretely:

$$M = \cos sim(\frac{1}{p} \sum_{i=1}^{p} F(x_i, y_i), f_n) > \theta$$

where p is the number of points in a patch of coordinates (x_i, y_i) and θ is a threshold in the range [0.0, 1.0]. The cosine similarity between two vectors $\vec{a}$ and $\vec{b}$ is used as a similarity metric between two vectors, and is defined as:

$$\cos sim(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|}$$

The mask M is a boolean vector where, given a list of 3D Gaussians, M has a value of 1 at the same index as a 3D Gaussian that will be segmented. Thus, by obtaining the mask M, segmentation can be performed by rendering only the masked 3D Gaussians with a uniform color and rendering unmasked 3D Gaussians regularly. The selection of coordinates (x_i, y_i) and threshold θ is left as user input, and discussed further in a later section.

3.1.3 Dynamic Ratio Design

A crucial design change made to the segmentation pipeline is the implementation of allowing various image ratios to be used during the training. Recall that the semantic feature value f_n is optimized via the pre-trained 2D feature extractor and other 3D Gaussian parameters are optimized separately. We observe that the 3D representation of our scenes is dependent on both the image sizes fed into the pre-trained 2D feature extractor and the image sizes used to optimize the 3D Gaussians. Therefore, depending on the image sizes used at these specific sections of the segmentation pipeline, the optimized semantic feature value and Gaussian parameter values will vary.

Specifically, given that the images of an input video are of size $1x$, the segmentation pipeline has been adjusted to use images of sizes $1x, 0.5x, 0.25x, 0.125x$ for both the 2D feature extractor and 3D Gaussian optimizer, and can be tuned to achieve desired parameter values. To find a default ratio, images of size $0.5x$ are used for the 2D feature extractor, and images of size $0.25x$ for the 3D Gaussian optimization.

3.2 Editing Semantically Trained 3D Gaussians

Following the training and segmentation of 3D Gaussians, the method in [15] is adapted to perform scene editing. We propagate the selected Hypernerf dataset from the segmentation pipeline into this editing pipeline. In this section, we elaborate on its design and describe the three main steps as adapting the segmented mask M, performing multi-viewing editing, and 3D reconstruction of those edits. We also detail the design changes made to achieve compatibility with the previous segmentation.

3.2.1 Adapting Segmented Mask M

One way to adapt the segmented mask M into the editing pipeline is to compute a 3D Gaussian mask M for every frame and apply an image editor to every frame's view. In other words, given N frames $t_0, ...t_{N-1}$, we could produce a mask M_t for each frame. We disregard this method for two reasons and opt for an optimized approach.

One, we find the computation of a new mask for each frame to be unnecessary because the same mask at an initial frame t_0 can be used for $t \in \{t_0...t_{N-1}\}$. Since the mask M is used to segment a subset of 3D Gaussians, and recalling that the new position of a 3D Gaussian in this subset can be computed via the deformation model, then given a time t, the same mask M will select the newly positioned 3D Gaussian. Therefore, we simplify the process by computing only the mask at the first time frame t_0 and using it for all time frames. Supposing that the same semantic feature values and thresholds were chosen at each time frame, recomputing the mask would produce the same mask M and introduce redundant computation. Allowing new user input for each time frame is left for a future study.

Two, this method was disregarded because of the adaptive nature of optimizing 3D Gaussians. When optimizing 3D Gaussians, the number of 3D Gaussians in our 3D representation periodically shifts using cloning and pruning, so the mask M must change in conjunction with the number of 3D Gaussians that change. Computing a new mask from a view at a time

t would require rendering the view from the new 3D Gaussians. However, we observe that the cloning and pruning from [12] is done by comparing the gradients of the 3D Gaussians' parameters against a threshold, and does not involve rendering. Therefore, since the semantic feature is optimized in the same way as the color feature, we can clone and prune the semantic feature with the same gradient checking and avoid the need to render the view. In this manner, the mask at a time t would adapt to the change in the semantic feature values.

To reiterate, a mask M_t is used in the editing pipeline by first computing the mask M_{t_0} and then using that same mask at a time frame t. At every instance that the density of the 3D Gaussians has changed, we similarly update the density of the mask.

3.2.2 Multi-View Editing

Now that a mask M_t has been implemented into the editing pipeline, we focus on the editing process itself and describe how to obtain an edited view. As described in DGE, an image can be edited via a diffusion-based image editor that utilizes self-attention. Given a frame t, we can render multiple views n to generate a video of the static scene at t. Therefore, each view in a frame t can attend to every other view $i \in \{i_0, ...i_{n-1}\}$ in that same frame t, and we can treat the dynamic scene editing problem as if we are editing N static scenes.

We further extend DGE such that given a video with N frames $t_0, ...t_{N-1}$, we apply the mask M_t to the 3D Gaussians to render a masked image I_t as input for the image editor. Given a time frame t and a text prompt T, we can compute an edited image E_t:

$$E_t = P(I_t, T, \text{STAttn}(Q, K, i))$$

where P is a diffusion-based image editor such as IP2P [57], and STAttn is the self-attention score computed in the same manner as [15]:

$$\text{STAttn}(Q, K, i) = \text{Softmax}(\frac{Q_i \cdot [K_1, ..., K_n]}{\sqrt{d}})$$

where the Q_i, K_i, and V_i are defined as the queries $\{Q_i\}_{i=1}^n$, keys $\{K_i\}_{i=1}^n$, and values $\{V_i\}_{i=1}^n$ of the generated view i of views n, and d is the embedding dimension of those keys and queries. The queries are computed by projecting rendered views i, via a learned linear transformation as described in [61]. The keys and values are transformed the same way, except using renders of the original input views pre-editing.

3.2.3 3D Reconstruction of Editing

Once an edited image E_t is obtained, reconstruction of the 3D scene can be done by optimizing the 3D Gaussians as usual. Although the editing pipeline in DGE outputs n edited views as well as the edited image E_t, we claim that this section of the original editing pipeline can be removed, and the usage of an image E_t stops once it is used for the optimization of the 3D Gaussians. In other words, the image E_t is used for the sole purpose as a guide to the optimization of the point cloud of 3D Gaussians. Once we acquire a trained point cloud of 3D Gaussians, we can simply render the time frames of the dynamic scene during evaluation, without needing the edited images. At a high level, this means we change the input-output

model of the editing pipeline, such that instead of outputting edited *images* from an input point cloud, we output an edited *point cloud* from an input point cloud.

A crucial design change made to the ending of the editing pipeline is the implementation of rendering without segmentation. With the first renderer performing segmentation on a dynamic scene, this made it incompatible with rendering views *after editing* since we do not want to segment our edited renders. Rather, we implement a variation of the renderer that skips the segmentation process. It is analogous to selecting a threshold $\theta = 1.0$ during segmentation, such that we render views from the edited point cloud and perform no segmentation since no 3D Gaussians were selected for computing a mask M.

3.2.4 Dataset Consistency

To keep the ratio of image sizes consistent between the segmentation pipeline and editing pipeline, the editing process was updated to use the same input image size as the segmentation.

In addition, we observed that the editing pipeline in DGE is not configured for the Hypernerf dataset. Parsing camera and scene data was configured in the editing pipeline to work with Hypernerf data. However, by changing parts of the system to work with the new dataset, the perceptual loss metric [62] used to optimize the 3D Gaussians was no longer compatible. Thus, this loss metric was removed from the optimization scheme as we claim that results will not be significantly impacted, as further discussed in a later section.

Chapter 4

Results

In this chapter, we evaluate the feasibility of dynamic scene editing via semantically trained 3D Gaussians. We recall the main goal of the research as alluded to in the beginning, that we hope to achieve real-time, high-quality dynamic scene editing for common scene applications. Specifically, we want to evaluate the effectiveness of such a pipeline in mainstream applications based on the results produced.

Before deciding on the effectiveness of the dynamic scene editor in mainstream applications, we evaluate its effectiveness within its context qualitatively. The segmentation of the semantically trained 3D Gaussians is first studied to show evidence that it is feasible to train 3D Gaussians on a dynamic scene. We follow the segmentation results with editing results to show evidence that it is feasible to edit 3D Gaussians in a dynamic scene. Then, quantitative studies are performed to highlight limitations evident in the results. Finally, we conclude this chapter by evaluating the dynamic scene editor within the context of mainstream applications.

The main scene candidate is the Hypernerf breaking of a cookie dataset. Two additional scenes in the Hypernerf dataset, the opening and closing of a hand, and the torching of chocolate, are experimented on. All experiments were performed on a single NVIDIA RTX A6000.

4.1 Qualitative Studies

Throughout this section, the quality of the rendered scenes will be measured qualitatively, by visual inspection of whether a rendered scene has artifacts, objects, or appearances comparable to the real world.

4.1.1 Segmentation

We showcase the results tuned to parameters that best fit a segmentation of the cookie scene, where the segmented 3D Gaussians are rendered with the color green. Table 4.1 details the parameters used.

At first glance, the segmentation of the cookie scene is visually appealing in Figure 4.1. Looking at the first frame $t = 0$, the input point $(135, 185)$ corresponds to the middle of the cookie, which means we expect only the cookie itself to be segmented as the dynamic scene

Parameter	Cookie
Resolution	1072x1920
DINOv2 Input Size	1072x1920
Training Input Size	268x480
Frames	223
# Iterations	40,000
Points	[(135, 185)]
θ	0.55
# 3D Gaussians	1,243,539
Training Time	12 hours
Render Time	3 min
Scale Learning Rate	0.0002

Table 4.1: Summary of Segmentation Parameters Used for Cookie Scene

continues through the rest of the time frames. Following the time frames, this is exactly what we see: a majority of the segmented 3D Gaussians correspond to the cookie and not any other object. Although a couple of segmented 3D Gaussians correspond to parts of the model's hands and the space between the broken cookies, the segmentation is quite effective.

Parameter	Hand	Torch
Resolution	1072x1920	1072x1920
DINOv2 Input Size	536x960	536x960
Training Input Size	268x480	268x480
Frames	223	223
# Iterations	40,000	40,000
Points	[(110, 150)]	[(30, 130)]
θ	0.70	0.55
# 3D Gaussians	1,186,889	971,413
Training Time	13 hours	12 hours
Render Time	3 min	3 min
Scale Learning Rate	0.0002	**0.0020**

Table 4.2: Summary of Segmentation Parameters Used

We turn our attention to other segmentation results and evaluate the segmentation effectiveness in different scenes. In Figure 4.2 we observe similar results to the cookie scene. The selected point $(110, 150)$ corresponds to desiring 3D Gaussians similar to a hand, which is what the results yielded since the 3D Gaussians close to the hand are segmented as the time frames of the scene continue. We also observe the segmentation in Figure 4.3 yielding similar results. **We will disregard the abnormal quality of the rendered torch scene to focus on segmentation and editing effectiveness, but we will return to it in a later discussion section.**

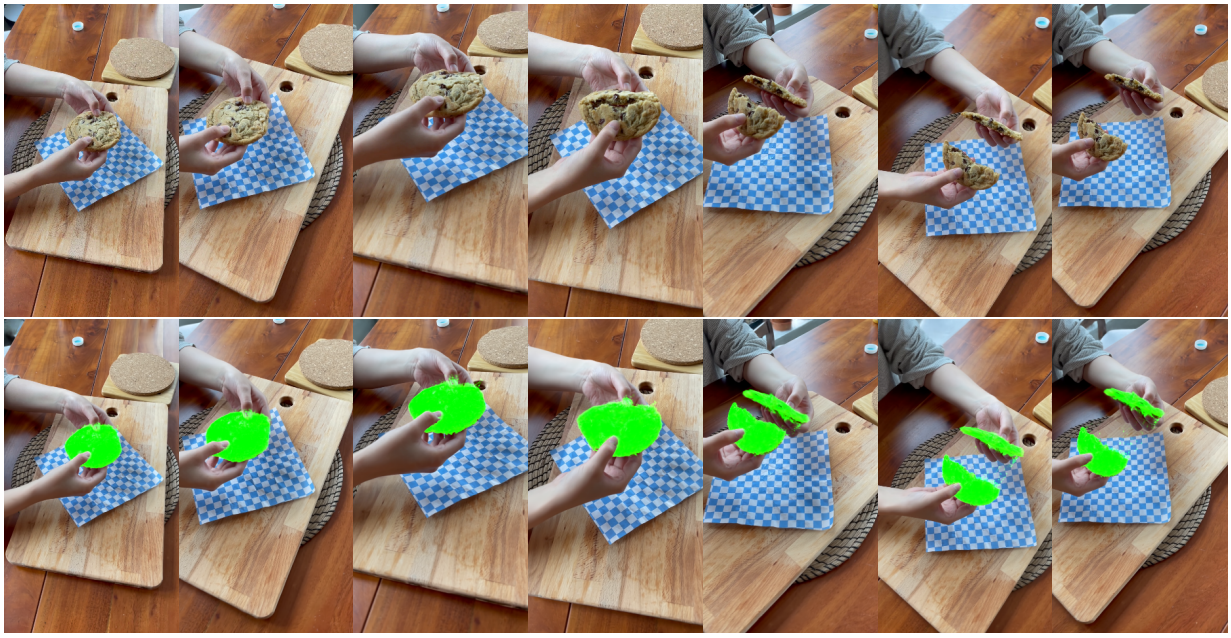

Figure 4.1: Frames 0, 39, 79, 119, 159, 199, and 222 of the input cookie video and its segmentation.

Figure 4.2: Frames 0, 39, 79, 119, 159, 199, and 222 of the input hand video and its segmentation.

Figure 4.3: Frames 0, 39, 79, 119, 159, 199, and 222 of the input torch video and its segmentation.

4.1.2 Edits

Continuing the segmentation of the cookie Hypernerf dataset, we perform various edits on this scene. In these results, we showcase the differences between running edits without our segmentation pipeline, edits with our segmentation pipeline, and edits using various text prompts.

Parameter	Cookie Base	Cookie	Cookie Chocolate	Cookie Red
Resolution	268x480	268x480	268x480	268x480
Training Input Size	268x480	268x480	268x480	268x480
Frames	223	223	223	223
# Iterations	500	500	500	500
Points	[(135, 185)]	[(135, 185)]	[(135, 185)]	[(135, 185)]
θ	0.55	0.55	0.55	0.55
# 3D Gaussians	936,525	1,127,158	1,123,778	1,123,214
Training Time	11 min	11 min	11 min	11 min
Render Time	3 min	3 min	3 min	3 min

Table 4.3: Summary of Editing Parameters Used for Cookie Scene where "Base" indicates editing without segmentation pipeline

To evaluate the performance of the dynamic scene editor, we compare the editing results of editing without our segmentation pipeline to edits with it. For simplicity of analysis, we label the editing without our segmentation pipeline as the baseline. We take a look at Figure 4.4 and notice two major differences between the two. One, we notice that objects we do not expect to change have been edited. Specifically, the top right object in the scene is not a cookie, yet it is edited into a pizza object as indicated by the text prompt. Two, with frames $t = 0$ and $t = 39$ showing a clearer indication of this, small volumes of 3D Gaussians are edited in the background of the scene, even though we do not expect them to change. For example, 3D Gaussians near the left forearm and the corners of the table have been edited to try to be a pizza. The tonal color of the baseline scene appears to be less saturated than the input video or our resulting edit. This prompts us to claim that our editor proves to be better in this scene because of its capability to selectively edit objects of choice without interfering with objects in the background.

We further evaluate the editing performed on the object in focus, the cookie itself. Keeping in mind the evaluation of the segmentation performed on the cookie in the previous section, we expect the editing process to be smooth, following the segmented 3D Gaussians. **This is not the case** starting frame 159 in Figure 4.4. It is around this frame number that the broken half of the cookie lying in the model's left-hand does not have any of its 3D Gaussians being edited. This editing outcome continues to be the case for the rest of the frames.

This is a major flaw of the editing pipeline. We suspect that there is a lack of camera data to capture details of the cookie when it is in a state of being angled away from the scene's cameras. In other words, the moment the broken cookie starts to angle away from the camera's view, the editing pipeline no longer recognizes it as a "cookie" object, so it does not perform any edits on it. We view these results to be consistent with various text prompts as seen in Figure 4.5. When presented with a dynamic scene such that the dynamics reduce

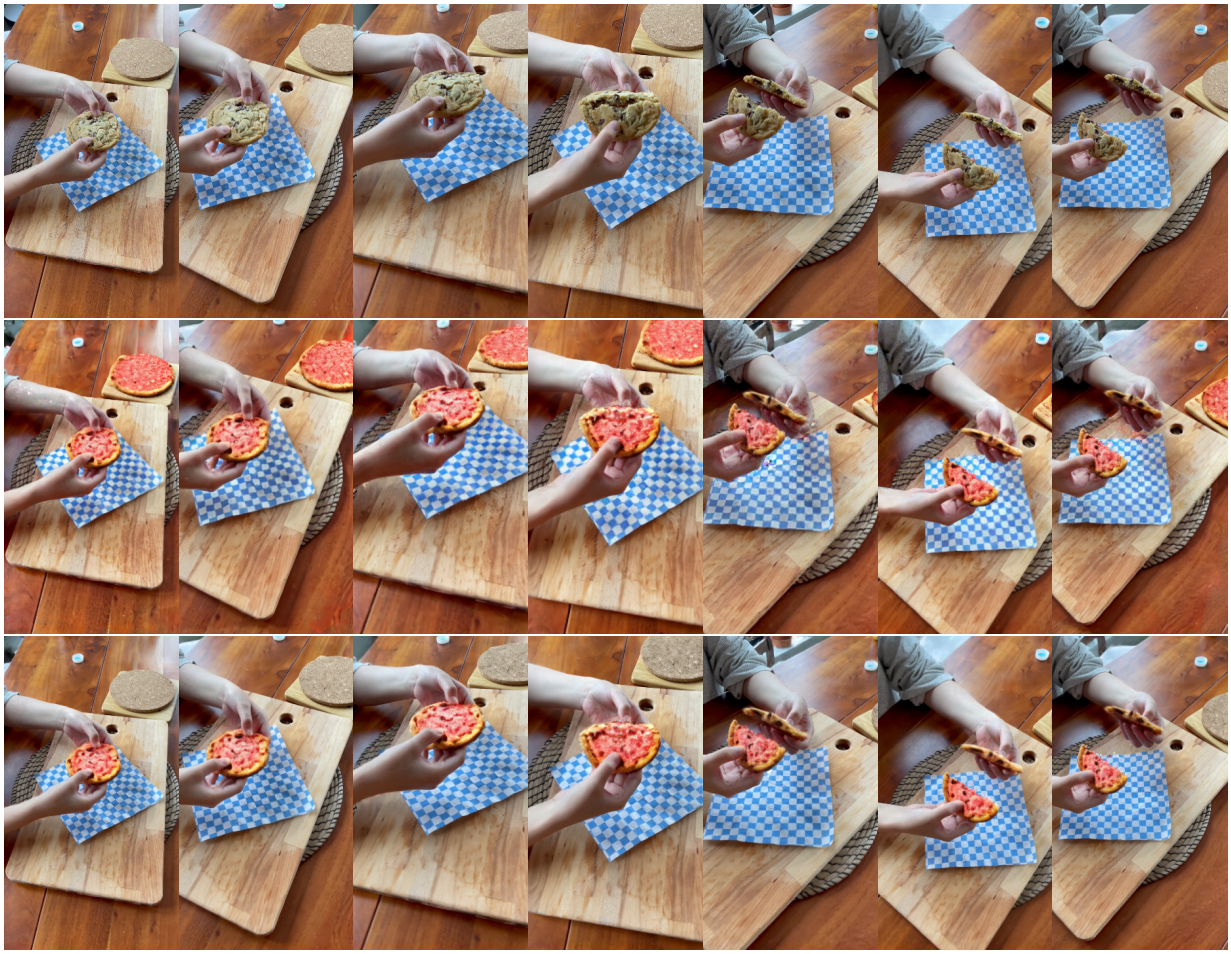

Figure 4.4: Frames 0, 39, 79, 119, 159, 199, and 222 of the input cookie video, followed by editing without segmentation pipeline and then edited with segmentation using the text prompt "change cookie to pizza".

the possible observable states of an object, the editing pipeline has less information and is unable to infer the resulting state of that object. Although the segmentation results show the broken half of the cookie has the 3D Gaussians we want to edit, the segmented 3D Gaussians are the set of 3D Gaussians that are allowed to be edited by the editing pipeline, they do not equate to the 3D Gaussians that are edited. The editing pipeline selects a subset of the segmentation set to perform its editing.

Therefore, without changing the editing pipeline, two fixes to the scene come to mind that could keep the edits consistent throughout the time frames. One, as alluded to by the previous suspicion, is to have the scene's cameras capture all angles of the cookie while it dynamically changes. This means a dataset should ensure the cameras are positioned strategically so that all viewpoints of the cookie can be captured. Two, we take the opposite approach, such that instead of focusing on the scene's cameras, we focus on the object in question. Given the cameras of a scene, we could limit the allowed dynamics of an object so that all its states always stay within the cameras' viewpoints. For example, this would include simple translations or rotations parallel to the cameras' views. Setting these two

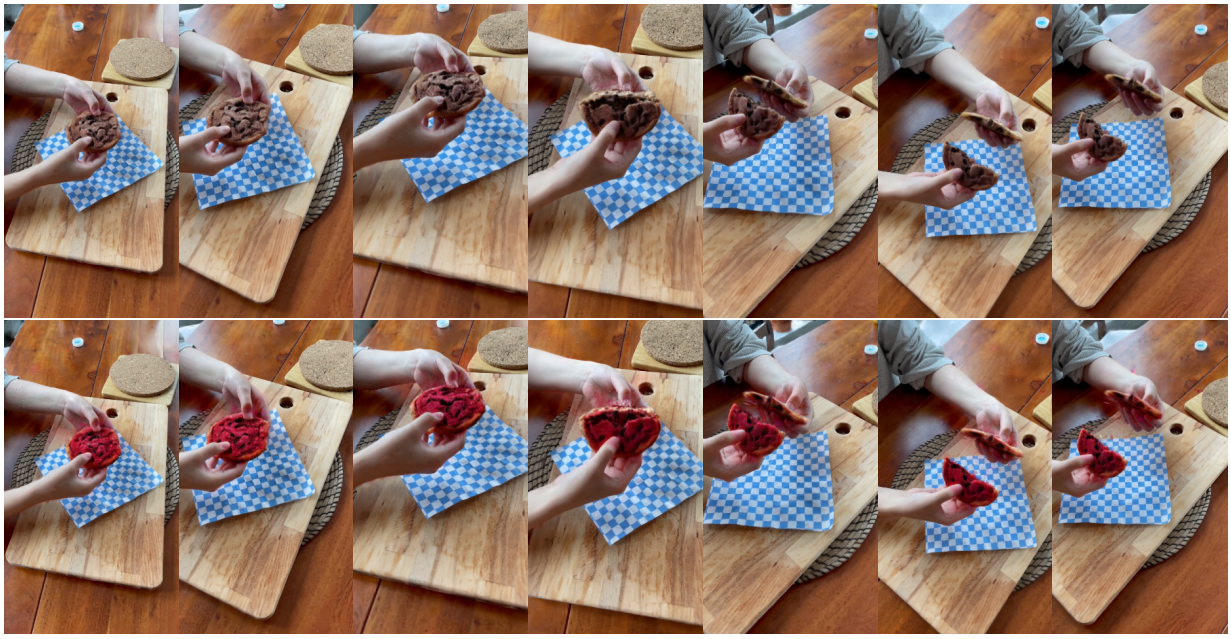

Figure 4.5: Frames 0, 39, 79, 119, 159, 199, and 222 of editing the cookie scene with prompts "change cookie to chocolate" and "change cookie to red velvet cookie".

fixes to the side, the editing pipeline's inability to infer from the absence of camera data is a major flaw, and changing a scene's data so it fits with the editing pipeline is not within the scope of this research.

Parameter	**Hand Base**	**Hand**	**Torch Base**	**Torch**
Resolution	268x480	268x480	268x480	268x480
Training Input Size	268x480	268x480	268x480	268x480
Frames	223	223	223	223
# Iterations	500	500	500	500
Points	[(110, 150)]	[(110, 150)]	[(30, 130)]	[(30, 130)]
θ	0.70	0.70	0.55	0.55
# 3D Gaussians	985,601	1,269,222	768,405	970,389
Training Time	13 min	13 min	11 min	10 min
Render Time	3 min	3 min	3 min	3 min

Table 4.4: Summary of Editing Parameters Used for the hand and torch scene where "Base" indicates editing without segmentation pipeline

We also evaluate the editing performance in the other scenes to determine if our editing results are consistent. Viewing the editing results for the hand scene in Figure 4.6, the outcome is what we expected, where the baseline edits more 3D Gaussians than what is desired, and our results are edits of 3D Gaussians as a subset of the segmented 3D Gaussians. However, this scene reveals another flaw in the editing pipeline; the quality of the edits is highly dependent on the provided text prompt. Specifically, we view the results of Figure 4.7 and see editing that does not change the hand to a "green monster". Thus, we claim

Figure 4.6: Frames 0, 39, 79, 119, 159, 199, and 222 of the input hand video, followed by editing without segmentation pipeline and then edited with segmentation using the text prompt "change hand to iron man".

that the editing pipeline only works with *strong* text prompts, or text prompts that induce greater change in the parameters of the 3D Gaussians. Perhaps *weaker* text prompts do not have large magnitudes of gradients during 3D Gaussian optimization, so this results in weak editing. We leave this gradient visualization for a future study.

4.2 Quantitative Studies

Following the evaluation of the segmentation and editing results, we also discuss the limitations of the dynamic scene editor in the context of required memory and compatible datasets. Determining the scope of experiments that the pipeline can be run on gives us insights into how applicable it is to our intended audience.

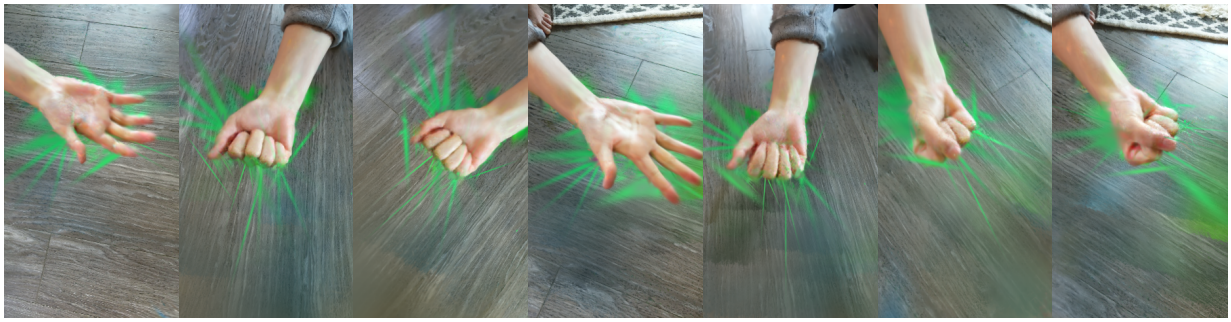

Figure 4.7: Frames 0, 39, 79, 119, 159, 199, and 222 of the edited hand scene using the text prompt "change hand to green monster".

4.2.1 Memory Limitations

One of the major obstacles in running our experiments is ensuring there is enough memory on our device to run them. All experiments were run on a single NVIDIA RTX A6000, which supports up to 50 GB of device memory. Recall that segmentation on the scenes allowed dynamic image size inputs to allow the tuning of the parameter training. For example, Table 4.1 has the input cookie image resolution of 1072x1920 with the DINOv2 input as 1072x1920 and the training input size as 268x480. Originally, we tried to support the input image size within our training framework, but we did not have enough memory on the device to support this. Running with training image sizes of 536x960 was still insupportable, and the memory capacity was enough when the size 268x480 was used. We reached similar conclusions for the other trained scenes in Table 4.2 as we ended up reducing both DINOv2 and training input sizes. Given this context, we claim that if we generalize our method to any dynamic scene, it would require **at least 50 GB** of memory.

We return to the torch scene in Figure 4.3, and discuss the impacts of reducing the image size. By using the same parameters as other scenes, we achieved the results in Figure 4.9, which are undesirable compared to the input itself because we see multiple instances of 3D Gaussians as artifacts and scaled larger than they should be. This gave us the intuition to adjust the scale learning rate to 0.0020, shown in Table 4.2, which achieved the results in Figure 4.3. From the new results, the 3D Gaussians are sufficient to explore segmentation and editing, as evidenced by the previously discussed segmentation and editing results of the torch scene. Here, we saw another limitation of the dynamic scene editor, where, depending on the input scene, either its allocated memory for training is too large for the device to fit, or it requires extra fine-tuning to get a comprehensible result.

4.2.2 Dataset Limitations

It was noted before that the Hypernerf dataset was chosen as the representative set of data to run our methods. One of the main reasons to use Hypernerf was that [14] had already implemented support for Hypernerf in the segmentation pipeline, so it was easier to use. Thus, supporting Hypernerf in the editing pipeline involved simply porting this parsing, but due to the pipeline in [15] being built for specific datasets, not including Hypernerf, the functionality for computing perceptual loss was lost in optimizing 3D Gaussians. Here we

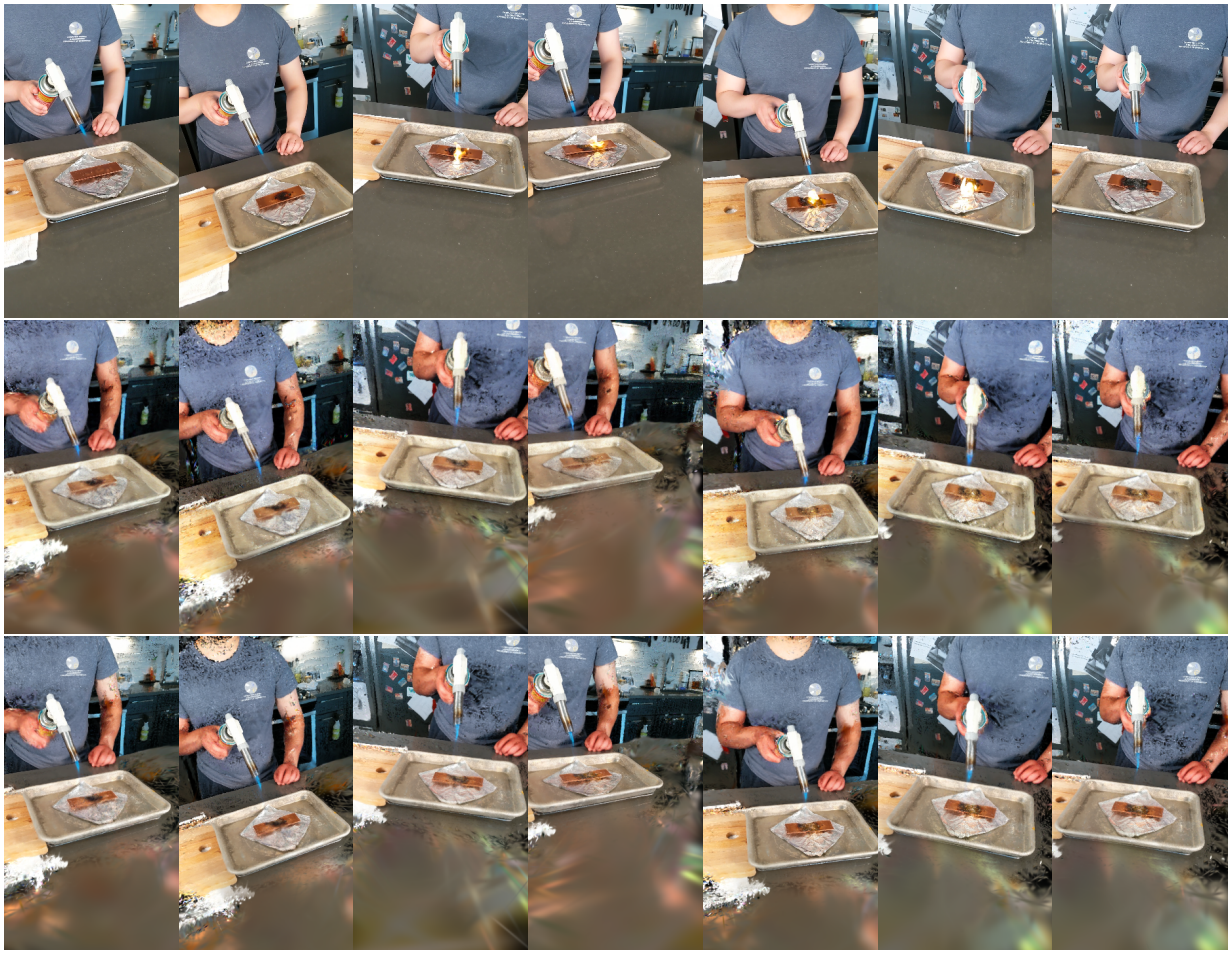

Figure 4.8: Frames 0, 39, 79, 119, 159, 199, and 222 of the input torch video, followed by editing without segmentation pipeline and then edited with segmentation using the text prompt "change hands to stone".

evaluate the disabling of the perceptual loss on the produced results.

We recall that the loss used in the method is L1 and SSIM loss, which are often used in combination to capture direct image differences. On the other hand, perceptual loss takes a different approach and computes errors based on features in a network. In Figure 4.10, we compare the difference between the input video with our rendered images. At first glance, we see that there are almost no differences between the two. It is only when we zoom in close and compare the two that we may notice the surface of the cookie in the last frame, $t = 222$, to be blurrier than the input video. For such an almost unnoticeable difference, it should be safe to claim that disabling perceptual loss has an insignificant effect on the segmentation and editing of the scenes. Thus, we view the disabling of the perceptual loss as having an insignificant impact on editing results, so image quality should not decrease regardless of the standard of the input dataset.

Figure 4.9: Frames 0, 39, 79, 119, 159, 199, and 222 of the rendered torch video using the default scale learning rate.

Figure 4.10: Frames 0, 39, 79, 119, 159, 199, and 222 of the input cookie video and renders without segmentation.

4.3 Application Studies

Now that we have evaluated the dynamic scene editor in its context, we evaluate the editor in the context of mainstream entertainment applications. Recall that the goal of this research is to achieve real-time, high-quality dynamic scene editing. In this section, we claim that our editor is **non-performant** for common use of these applications, and only effective in rare cases.

4.3.1 Video Editing

We evaluate the usage of the dynamic scene editor in the context of video editing as proposed in the introduction. Turning to the segmentation results in Table 4.1, we infer that for scenes

with similar complexity, it will take 12 hours to reconstruct the 3D scene and 11 minutes to edit. This efficiency is acceptable, given that a filmmaker can leave the reconstruction and segmentation overnight to run, and then work on other scenes while the editing happens.

However, the main problem arises with dataset compatibility. Specifically, most modern filming is shot in resolutions of 1080p (1920x1080) or 4K (3840x2160). Suppose a filmmaker has all of their clips in one of these resolutions and wants to create an edited clip via our method. Assuming the scene complexity is about the same as the scenes used here, this means the filmmaker needs **at least 50 GB** of memory to create an edited clip of down-scaled resolution. If we assume the filmmaker would want to create an edited clip to the scaled resolution and using this resolution in the pipeline would scale the required memory by the same amount, going from training input sizes of 268x480 to 1920x1080 would require at least 4x more memory. This is impractical, as most common filmmakers do not have access to computing power this large.

Furthermore, the memory limitations are also indicated by the number of frames that can be trained at once. Most modern filmmakers and viewers agree that frame progression feels normal when running at 30 frames per second. Therefore, given that about 223 frames mark the memory capacity, this gives users around 7-8 seconds of video editing per training session. With most modern shots lasting for a couple of seconds, 7-8 seconds of video editing is quite substantial for most cases, but unusable or needs to be reapplied for longer clips.

Memory aside, we also recall that the editing capabilities are limited to scenes where the dynamics of the object keep the object within the camera view. This means if a filmmaker has scenes with complex dynamics, they are unable to use this pipeline, or they would only be able to produce edits that do not span the entire scene. Thus, in the context of video editing, our dynamic scene editor is useful if a user is working with low-resolution, simple dynamic scenes.

4.3.2 Room Simulators

We evaluate the usage of the dynamic scene editor in the context of specific scene editing, such as room simulators. Following the same analysis as in video editing, we reach a similar conclusion that memory capacity is the main limitation for the dynamic editor to reach a practical state. However, we notice that in this case, room simulators are more likely to have scenes with less complex dynamics and shorter durations.

Consider a scene where a user is trying to envision how chairs in a building would look as students pass by or sit on them. We envision the students barely shifting the chairs throughout the scene, so the dynamics of the chairs remain simple. The editing pipeline can be quite effective because the text prompts will be applied to chairs with simple dynamics, and the edits themselves can contain only appearance changes, such as color.

4.3.3 Virtual Sandboxes

We evaluate the usage of the dynamic scene editor in the context of virtual environments, where we imagine a user is placed in a sandbox setting and interacting with surrounding objects, whether this be in a video game or virtual reality. For example, suppose a user observes a car driving down a road, and the user wants to instantly see what the car would

look like if it swapped color palettes. In such a situation, we disregard the segmentation limitations from the user's perspective because we assume the 3D scene has already been trained before the user receives it. We focus our attention on the effectiveness of editing.

Our analysis of editing quality reaches the same conclusion as the editor evaluated within its context. Namely, the editing can perform consistent edits if the user interacts with an object via simple dynamics, and with *strong* prompts.

Furthermore, based on the results, editing time depends on the scene, but if we assume the video game environment has **at least** the same complexity as our scenes, then it follows that dynamic scene editing would take **at least** 13 minutes to edit and render the new changes. We computed this number by using the minimum of the sum of the training and render times for all the trained scenes (Table 4.3 and 4.4). This editing time is non-performant in this situation because usage in sandbox environments requires near real-time changes, as in a few seconds, so users can continuously iterate at an efficient pace.

Chapter 5

Concluding Remarks

This thesis has explored editing semantically trained 3D Gaussians to realize real-time dynamic 3D scene editing. Although it is feasible, the numerous limitations of such a pipeline deem it to be non-performant in its main applications. By having high memory demands and subpar editing times, the editing pipeline is held back from being practical.

However, we show that the selective editing nature of a dynamic scene editor is necessary. With the increased usage of text prompt editing in recent works, the problem of whether or not the edits align with the user's intention remains at large. Furthermore, the inconsistency of our notion of *strong* prompts supports the statement that text prompt editing alone is not sufficient for the common user. We demonstrated that some form of selective editing should be implemented, as it more closely aligns with a user's intentions. Given a scene with objects that can confuse a classification model or a scene where a user only wants to edit x out of n of the same objects, selective editing would yield better results.

We propose major directions of study to improve the performance of the dynamic scene editor within the context of usage in mainstream applications. Although an appealing feature of the dynamic scene editor is its ability to selectively edit objects via segmentation, the editing quality can be improved. Specifically, future investigations can tackle the limitations in inferring the absence of camera data and editing quality from text prompts. In recent research, breakthroughs have been made in developing image editors that produce high-quality results. Another improvement to the editing scheme is reducing the training time. We saw that 11 minutes of training a dynamic scene with new edits is non-performant in our mainstream application, and does not achieve the real-time rendering we set out initially. We leave the feasibility of integrating these image editors for further research.

We also observe that incorporating multi-object editing into the pipeline is quite straightforward, as it would only involve handling numerous input points and thresholds. The editing procedure can stay the same. However, achieving high-quality edits is nontrivial. In the situation that two objects are being edited by a user, it is better to edit the two objects as if they were a single object rather than editing them separately and forcing their interactions together. Forcing the interactions together would lead to artifacts at the point of contact between the two objects. Further study into a smart multi-object editing procedure would benefit rendered quality in all mainstream applications. Furthermore, following the same scenario where two objects are being simultaneously edited, the dynamics of the interacting objects could change because of material interactions or a change in object shapes. PhysDreamer

[63] proposed physics-based interaction with 3D objects via video generation, which could be applied to combined objects to create realistic, edited motion. We leave the feasibility and practicality of an integration for further research.

A key optimization to be made to the system is reducing the memory occupancy during training. Although the allowance of dynamic input sizes was implemented, the core problem of taking up too much memory is still present. In fact, during the experimentation of segmentation, we stopped 3D Gaussian cloning and pruning after several iterations, but still encountered increased memory usage as the iterations continued. Perhaps there is a memory leak in the system, so more research is needed to conclude whether high memory usage is due to a bug in the system or the intended behavior due to the sheer number of 3D Gaussians being optimized.

While memory demand currently poses a significant challenge for 3DGS, this limitation also highlights an exciting opportunity for breakthroughs. As optimization methods continue to evolve, we anticipate these barriers will be overcome, which not only unlocks more potential of 3DGS but also paves the way for new scientific and creative applications. We are optimistic that as research and collaboration continue, the future of 3DGS stays bright.

Appendix A

Code and Dataset Availability

All code and datasets are available for viewing at https://github.com/Jordlam/gsplat_code

References

[1] I. Zhura, D. Davletshin, N. D. W. Mudalige, A. Fedoseev, R. Peter, and D. Tsetserukou. *NeuroSwarm: Multi-Agent Neural 3D Scene Reconstruction and Segmentation with UAV for Optimal Navigation of Quadruped Robot*. 2023. arXiv: 2308.01725 [cs.RO]. URL: https://arxiv.org/abs/2308.01725.

[2] M. Khan, H. Fazlali, D. Sharma, T. Cao, D. Bai, Y. Ren, and B. Liu. *AutoSplat: Constrained Gaussian Splatting for Autonomous Driving Scene Reconstruction*. 2024. arXiv: 2407.02598 [cs.CV]. URL: https://arxiv.org/abs/2407.02598.

[3] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer. *D-NeRF: Neural Radiance Fields for Dynamic Scenes*. 2020. arXiv: 2011.13961 [cs.CV]. URL: https://arxiv.org/abs/2011.13961.

[4] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. "The lumigraph". In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. New York, NY, USA: Association for Computing Machinery, 1996, pp. 43–54. ISBN: 0897917464. DOI: 10.1145/237170.237200. URL: https://doi.org/10.1145/237170.237200.

[5] M. Levoy and P. Hanrahan. "Light field rendering". In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. New York, NY, USA: Association for Computing Machinery, 1996, pp. 31–42. ISBN: 0897917464. DOI: 10.1145/237170.237199. URL: https://doi.org/10.1145/237170.237199.

[6] C. Buehler, M. Bosse, L. McMillan, S. Gortler, and M. Cohen. "Unstructured lumigraph rendering". In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. New York, NY, USA: Association for Computing Machinery, 2001, pp. 425–432. ISBN: 158113374X. DOI: 10.1145/383259.383309. URL: https://doi.org/10.1145/383259.383309.

[7] L. Wu, G. Cai, S. Zhao, and R. Ramamoorthi. "Analytic spherical harmonic gradients for real-time rendering with many polygonal area lights". In: *ACM Trans. Graph.* 39.4 (Aug. 2020). ISSN: 0730-0301. DOI: 10.1145/3386569.3392373. URL: https://doi.org/10.1145/3386569.3392373.

[8] N. Snavely, S. M. Seitz, and R. Szeliski. "Photo tourism: exploring photo collections in 3D". In: *ACM Trans. Graph.* 25.3 (July 2006), pp. 835–846. ISSN: 0730-0301. DOI: 10.1145/1141911.1141964. URL: https://doi.org/10.1145/1141911.1141964.

[9] M. Goesele, N. Snavely, B. Curless, H. Hoppe, and S. M. Seitz. "Multi-View Stereo for Community Photo Collections". In: *2007 IEEE 11th International Conference on Computer Vision*. 2007, pp. 1–8. DOI: 10.1109/ICCV.2007.4408933.

[10] 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. arXiv: 2003.08934 [cs.CV]. URL: https://arxiv.org/abs/2003.08934.

[11] S. J. Garbin, M. Kowalski, M. Johnson, J. Shotton, and J. Valentin. *FastNeRF: High-Fidelity Neural Rendering at 200FPS*. 2021. arXiv: 2103.10380 [cs.CV]. URL: https://arxiv.org/abs/2103.10380.

[12] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis. *3D Gaussian Splatting for Real-Time Radiance Field Rendering*. 2023. arXiv: 2308.04079 [cs.GR]. URL: https://arxiv.org/abs/2308.04079.

[13] T. Wu, Y.-J. Yuan, L.-X. Zhang, J. Yang, Y.-P. Cao, L.-Q. Yan, and L. Gao. *Recent Advances in 3D Gaussian Splatting*. 2024. arXiv: 2403.11134 [cs.CV]. URL: https://arxiv.org/abs/2403.11134.

[14] I. Labe, N. Issachar, I. Lang, and S. Benaim. *DGD: Dynamic 3D Gaussians Distillation*. 2024. arXiv: 2405.19321 [cs.CV]. URL: https://arxiv.org/abs/2405.19321.

[15] M. Chen, I. Laina, and A. Vedaldi. *DGE: Direct Gaussian 3D Editing by Consistent Multi-view Editing*. 2024. arXiv: 2404.18929 [cs.CV]. URL: https://arxiv.org/abs/2404.18929.

[16] G. Chaurasia, S. Duchene, O. Sorkine-Hornung, and G. Drettakis. "Depth synthesis and local warps for plausible image-based navigation". In: *ACM Trans. Graph.* 32.3 (July 2013). ISSN: 0730-0301. DOI: 10.1145/2487228.2487238. URL: https://doi.org/10.1145/2487228.2487238.

[17] P. Hedman, J. Philip, T. Price, J.-M. Frahm, G. Drettakis, and G. Brostow. "Deep blending for free-viewpoint image-based rendering". In: *ACM Trans. Graph.* 37.6 (Dec. 2018). ISSN: 0730-0301. DOI: 10.1145/3272127.3275084. URL: https://doi.org/10.1145/3272127.3275084.

[18] G. Kopanas, J. Philip, T. Leimkühler, and G. Drettakis. *Point-Based Neural Rendering with Per-View Optimization*. 2021. arXiv: 2109.02369 [cs.CV]. URL: https://arxiv.org/abs/2109.02369.

[19] A. Tewari et al. *Advances in Neural Rendering*. 2022. arXiv: 2111.05849 [cs.GR]. URL: https://arxiv.org/abs/2111.05849.

[20] E. Penner and L. Zhang. "Soft 3D reconstruction for view synthesis". In: *ACM Trans. Graph.* 36.6 (Nov. 2017). ISSN: 0730-0301. DOI: 10.1145/3130800.3130855. URL: https://doi.org/10.1145/3130800.3130855.

[21] P. Henzler, N. Mitra, and T. Ritschel. *Escaping Plato's Cave: 3D Shape From Adversarial Rendering*. 2021. arXiv: 1811.11606 [cs.CV]. URL: https://arxiv.org/abs/1811.11606.

[22] V. Sitzmann, J. Thies, F. Heide, M. Nießner, G. Wetzstein, and M. Zollhöfer. *DeepVoxels: Learning Persistent 3D Feature Embeddings*. 2019. arXiv: 1812.01024 [cs.CV]. URL: https://arxiv.org/abs/1812.01024.

[23] Y. Xie, T. Takikawa, S. Saito, O. Litany, S. Yan, N. Khan, F. Tombari, J. Tompkin, V. Sitzmann, and S. Sridhar. *Neural Fields in Visual Computing and Beyond*. 2022. arXiv: 2111.11426 [cs.CV]. URL: https://arxiv.org/abs/2111.11426.

[24] K. Park, U. Sinha, J. T. Barron, S. Bouaziz, D. B. Goldman, S. M. Seitz, and R. Martin-Brualla. *Nerfies: Deformable Neural Radiance Fields*. 2021. arXiv: 2011.12948 [cs.CV]. URL: https://arxiv.org/abs/2011.12948.

[25] D. Verbin, P. Hedman, B. Mildenhall, T. Zickler, J. T. Barron, and P. P. Srinivasan. *Ref-NeRF: Structured View-Dependent Appearance for Neural Radiance Fields*. 2021. arXiv: 2112.03907 [cs.CV]. URL: https://arxiv.org/abs/2112.03907.

[26] R. Liang, H. Chen, C. Li, F. Chen, S. Panneer, and N. Vijaykumar. *ENVIDR: Implicit Differentiable Renderer with Neural Environment Lighting*. 2023. arXiv: 2303.13022 [cs.CV]. URL: https://arxiv.org/abs/2303.13022.

[27] T. Müller, A. Evans, C. Schied, and A. Keller. "Instant neural graphics primitives with a multiresolution hash encoding". In: *ACM Trans. Graph.* 41.4 (July 2022). ISSN: 0730-0301. DOI: 10.1145/3528223.3530127. URL: https://doi.org/10.1145/3528223.3530127.

[28] A. Yu, S. Fridovich-Keil, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa. *Plenoxels: Radiance Fields without Neural Networks*. 2021. arXiv: 2112.05131 [cs.CV]. URL: https://arxiv.org/abs/2112.05131.

[29] J. Grossman, H. Sc, and W. Dally. "Point Sample Rendering". In: (May 1999).

[30] M. Sainz and R. Pajarola. "Point-based rendering techniques". In: *Comput. Graph.* 28.6 (Dec. 2004), pp. 869–879. ISSN: 0097-8493. DOI: 10.1016/j.cag.2004.08.014. URL: https://doi.org/10.1016/j.cag.2004.08.014.

[31] S. Laine and T. Karras. "High-performance software rasterization on GPUs". In: *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*. HPG '11. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2011, pp. 79–88. ISBN: 9781450308960. DOI: 10.1145/2018323.2018337. URL: https://doi.org/10.1145/2018323.2018337.

[32] M. Schütz, B. Kerbl, and M. Wimmer. *Software Rasterization of 2 Billion Points in Real Time*. 2022. arXiv: 2204.01287 [cs.GR]. URL: https://arxiv.org/abs/2204.01287.

[33] M. Botsch, A. Sorkine-Hornung, M. Zwicker, and L. Kobbelt. "High-Quality Surface Splatting on Today?s GPUs." In: Jan. 2005, pp. 17–24. DOI: 10.2312/SPBG/SPBG05/017-024.

[34] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. "Surfels: surface elements as rendering primitives". In: *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '00. USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 335–342. ISBN: 1581132085. DOI: 10.1145/344779.344936. URL: https://doi.org/10.1145/344779.344936.

[35] L. Ren, H. Pfister, and M. Zwicker. "Object Space EWA Surface Splatting: A Hardware Accelerated Approach to High Quality Point Rendering". In: *Computer Graphics Forum* 21.3 (2002), pp. 461–470. DOI: https://doi.org/10.1111/1467-8659.00606. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-8659.00606. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-8659.00606.

[36] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. "Surface splatting". In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. New York, NY, USA: Association for Computing Machinery, 2001, pp. 371–378. ISBN: 158113374X. DOI: 10.1145/383259.383300. URL: https://doi.org/10.1145/383259.383300.

[37] O. Wiles, G. Gkioxari, R. Szeliski, and J. Johnson. "SynSin: End-to-End View Synthesis From a Single Image". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[38] W. Yifan, F. Serena, S. Wu, C. Öztireli, and O. Sorkine-Hornung. "Differentiable surface splatting for point-based geometry processing". In: *ACM Transactions on Graphics* 38.6 (Nov. 2019), pp. 1–14. ISSN: 1557-7368. DOI: 10.1145/3355089.3356513. URL: http://dx.doi.org/10.1145/3355089.3356513.

[39] K.-A. Aliev, A. Sevastopolsky, M. Kolos, D. Ulyanov, and V. Lempitsky. *Neural Point-Based Graphics*. 2020. arXiv: 1906.08240 [cs.CV]. URL: https://arxiv.org/abs/1906.08240.

[40] D. Rückert, L. Franke, and M. Stamminger. "ADOP: approximate differentiable one-pixel point rendering". In: *ACM Trans. Graph.* 41.4 (July 2022). ISSN: 0730-0301. DOI: 10.1145/3528223.3530122. URL: https://doi.org/10.1145/3528223.3530122.

[41] G. Chen and W. Wang. *A Survey on 3D Gaussian Splatting*. 2025. arXiv: 2401.03890 [cs.CV]. URL: https://arxiv.org/abs/2401.03890.

[42] D. Charatan, S. Li, A. Tagliasacchi, and V. Sitzmann. *pixelSplat: 3D Gaussian Splats from Image Pairs for Scalable Generalizable 3D Reconstruction*. 2024. arXiv: 2312.12337 [cs.CV]. URL: https://arxiv.org/abs/2312.12337.

[43] Z. Liang, Q. Zhang, Y. Feng, Y. Shan, and K. Jia. *GS-IR: 3D Gaussian Splatting for Inverse Rendering*. 2024. arXiv: 2311.16473 [cs.CV]. URL: https://arxiv.org/abs/2311.16473.

[44] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli. "Image quality assessment: from error visibility to structural similarity". In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861.

[45] J. L. Schonberger and J.-M. Frahm. "Structure-From-Motion Revisited". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[46] K. Cheng, X. Long, K. Yang, Y. Yao, W. Yin, Y. Ma, W. Wang, and X. Chen. *GaussianPro: 3D Gaussian Splatting with Progressive Propagation*. 2024. arXiv: 2402.14650 [cs.CV]. URL: https://arxiv.org/abs/2402.14650.

[47] S. Qian, T. Kirschstein, L. Schoneveld, D. Davoli, S. Giebenhain, and M. Nießner. *GaussianAvatars: Photorealistic Head Avatars with Rigged 3D Gaussians*. 2024. arXiv: 2312.02069 [cs.CV]. URL: https://arxiv.org/abs/2312.02069.

[48] D. Das, C. Wewer, R. Yunus, E. Ilg, and J. E. Lenssen. *Neural Parametric Gaussians for Monocular Non-Rigid Object Reconstruction*. 2024. arXiv: 2312.01196 [cs.CV]. URL: https://arxiv.org/abs/2312.01196.

[49] L. Gao, J. Yang, B.-T. Zhang, J.-M. Sun, Y.-J. Yuan, H. Fu, and Y.-K. Lai. *Mesh-based Gaussian Splatting for Real-time Large-scale Deformation*. 2024. arXiv: 2402.04796 [cs.GR]. URL: https://arxiv.org/abs/2402.04796.

[50] Z. Yang, X. Gao, W. Zhou, S. Jiao, Y. Zhang, and X. Jin. *Deformable 3D Gaussians for High-Fidelity Monocular Dynamic Scene Reconstruction*. 2023. arXiv: 2309.13101 [cs.CV]. URL: https://arxiv.org/abs/2309.13101.

[51] B. Poole, A. Jain, J. T. Barron, and B. Mildenhall. *DreamFusion: Text-to-3D using 2D Diffusion*. 2022. arXiv: 2209.14988 [cs.CV]. URL: https://arxiv.org/abs/2209.14988.

[52] C. Vachha and A. Haque. *Instruct-GS2GS: Editing 3D Gaussian Splats with Instructions*. 2024. URL: https://instruct-gs2gs.github.io/.

[53] 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.

[54] Y. Chen, Z. Chen, C. Zhang, F. Wang, X. Yang, Y. Wang, Z. Cai, L. Yang, H. Liu, and G. Lin. *GaussianEditor: Swift and Controllable 3D Editing with Gaussian Splatting*. 2023. arXiv: 2311.14521 [cs.CV]. URL: https://arxiv.org/abs/2311.14521.

[55] G. Luo, T.-X. Xu, Y.-T. Liu, X.-X. Fan, F.-L. Zhang, and S.-H. Zhang. *3D Gaussian Editing with A Single Image*. 2024. arXiv: 2408.07540 [cs.CV]. URL: https://arxiv.org/abs/2408.07540.

[56] Y.-J. Yuan, Y.-T. Sun, Y.-K. Lai, Y. Ma, R. Jia, and L. Gao. *NeRF-Editing: Geometry Editing of Neural Radiance Fields*. 2022. arXiv: 2205.04978 [cs.GR]. URL: https://arxiv.org/abs/2205.04978.

[57] T. Brooks, A. Holynski, and A. A. Efros. *InstructPix2Pix: Learning to Follow Image Editing Instructions*. 2023. arXiv: 2211.09800 [cs.CV]. URL: https://arxiv.org/abs/2211.09800.

[58] K. Park, U. Sinha, P. Hedman, J. T. Barron, S. Bouaziz, D. B. Goldman, R. Martin-Brualla, and S. M. Seitz. *HyperNeRF: A Higher-Dimensional Representation for Topologically Varying Neural Radiance Fields*. 2021. arXiv: 2106.13228 [cs.CV]. URL: https://arxiv.org/abs/2106.13228.

[59] M. Oquab et al. *DINOv2: Learning Robust Visual Features without Supervision*. 2024. arXiv: 2304.07193 [cs.CV]. URL: https://arxiv.org/abs/2304.07193.

[60] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng. *Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains*. 2020. arXiv: 2006.10739 [cs.CV]. URL: https://arxiv.org/abs/2006.10739.

[61] Y.-C. Guo et al. *threestudio: A unified framework for 3D content generation*. https://github.com/threestudio-project/threestudio. 2023.

[62] G. G. Pihlgren, K. Nikolaidou, P. C. Chhipa, N. Abid, R. Saini, F. Sandin, and M. Liwicki. *A Systematic Performance Analysis of Deep Perceptual Loss Networks: Breaking Transfer Learning Conventions*. 2024. arXiv: 2302.04032 [cs.CV]. URL: https://arxiv.org/abs/2302.04032.

[63] T. Xie, Z. Zong, Y. Qiu, X. Li, Y. Feng, Y. Yang, and C. Jiang. *PhysGaussian: Physics-Integrated 3D Gaussians for Generative Dynamics*. 2024. arXiv: 2311.12198 [cs.GR]. URL: https://arxiv.org/abs/2311.12198.