GaussCraft: Precision Editing and Reconstruction with 2D Gaussian Splatting

Jiwon Park[1]

Master of Science in Computer Graphics, Vision and Imaging

Supervisor: Prof. Lourdes Agapito

Submission date: 09/09/2024

Abstract

2D Gaussian Splatting has recently provided a novel method for accurately reconstructing geometrically consistent radiance fields from multi-view images, improving surface representation and achieving high-quality, real-time rendering. However, existing 2D or 3D Gaussian splatting methods do not offer the capability for user-directed scene editing. While some 3D Gaussian-based methods exist for avatar-specific editing, they are limited to avatar applications and do not extend to general scenarios. Therefore, this paper introduces GaussCraft, an real-time scene editing framework that utilizes 2D Gaussian Splatting for high-quality mesh reconstruction. Unlike previous methods, GaussCraft only requires training once using the reconstructed mesh and does not need retraining for each edited scene. Specifically, GaussCraft reconstructs the mesh from multi-view images using 2D Gaussian Splatting, and then binds 2D Gaussians to each mesh face, allowing users to render scenes with user-edited, deformed meshes. This method has been tested on both synthetic and real-world captured data, showing significant potential for application across various fields. Our code can be found on: `https://github.com/jiwonhaha/GaussCraft`

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Professor Lourdes Agapito, for her invaluable guidance and support throughout this project. I am also grateful to my colleague and the research community for their constructive feedback and helpful discussions. Lastly, I acknowledge the resources and infrastructure provided by UCL, which made this research possible.

Contents

List of Figures

Chapter 1

Introduction

Novel view synthesis is a critical task in computer vision and graphics, where the capacity to interpolate and extrapolate views from sparse image data is paramount. Traditional Multi-View Stereo (MVS) methods [13] which reconstruct geometries from limited viewpoints often struggle with quality, yielding less realistic renderings when viewed from new angles. In recent years, Neural Radiance Fields (NeRF) [7] have emerged as a breakthrough in high-fidelity synthesis, inspiring a host of researches in the field. Despite their impressive performances, the implicit nature of NeRF representations makes direct object editing within scenes challenging.

To make scenes easier to modify, NeRF-editing technique [9] has been developed, allowing for the editing of implicit representations while still maintaining high output quality. Nevertheless, these NeRF-based methods often require extensive computational resources and training times, which limit their practicality for real-time applications. This has led to significant research efforts aimed at reducing these limitations [14, 15, 16, 17]. For instance, MiP-NeRF360 [18] significantly cuts down on training time, yet still demands extensive computational resources.

In response to the limitations of current synthesis methods, the 3D Gaussian Splatting (3DGS) model [8] emerges as a viable alternative. It employs an explicit representation by optimizing multiple 3D Gaussians, characterized by their position, opacity (α), anisotropic covariance, and spherical harmonic (SH) coefficients. This explicit representation not only accelerates training but also facilitates real-time

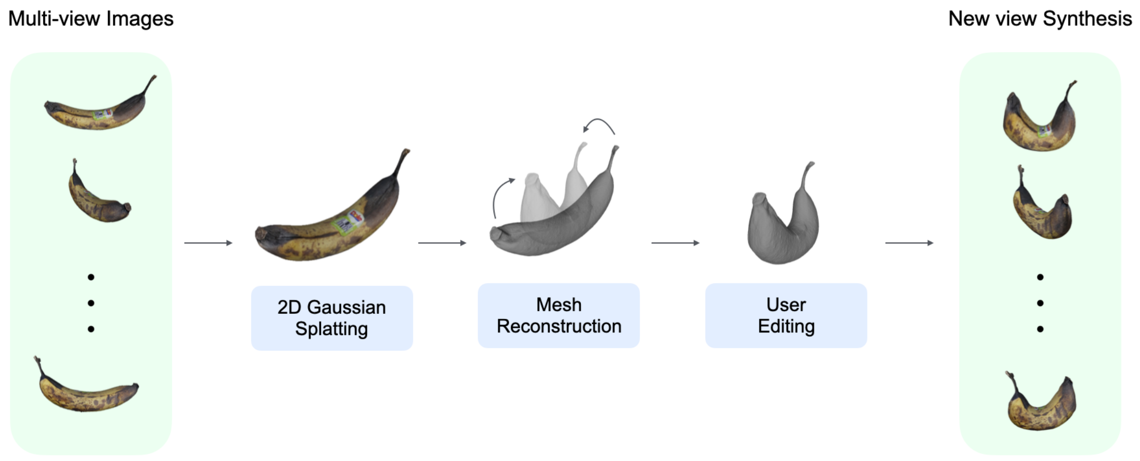

Figure 1.1: The pipeline of the 2D Gaussian Splatting editing framework

view synthesis. The 3DGS model has also inspired further investigations into various subfields, such as anti-aliasing [19], material modeling [20, 21], dynamic scene reconstruction [22], and the creation of animatable avatars [23, 11]. However, animatable avatar creation with Gaussian Avatar [11], are limited to specific case like editing Flame mesh models [24]. In contrast, NeRF-editing techniques [9] support editing from multi-view images, highlighting the need for broader applicability in Gaussian Splatting-based editing applications. Despite this need, the reliance of 3DGS on less adaptable mesh reconstruction methods imposes a significant limitation on its flexibility.

Building upon the concept of surfels (surface elements) which represent complex geometries effectively in SLAM and robotics by approximating surfaces with localized attributes based on known geometry [25, 26, 27, 28], 2D Gaussian Splatting (2DGS) introduces a refined technique that utilizes explicit ray-splat intersections. This method not only adheres to perspective accuracy but also significantly enhances the quality of reconstructions [10]. Unlike conventional approaches relying on ground truth geometry or depth sensors, which are limited by lighting conditions, 2DGS incorporates surface normals into the splatting process to facilitate robust surface regularization, improving both the accuracy and smoothness of the resultant surface meshes.

This paper introduces GaussCraft, a novel 2D Gaussian editing technique, as shown in Figure 1.1. The method begins by optimizing 2D Gaussians to reconstruct smooth meshes from point clouds, images, and camera data obtained via Structure-from-Motion (SfM) [29]. After the initial optimization, the model reconstructs a mesh and extracts key properties such as scaling, position, and rotation for each triangular face. 2D Gaussians are then placed at the center of each mesh face and bound to them, allowing a clear distinction between local and global spaces, similar to the method used in Gaussian Avatar [11]. However, unlike their approach, GaussCraft uses 2D Gaussians, omitting the z-axis in the local coordinate system and treating mesh faces as tangent planes. This simplification leads to faster optimization and more consistent results.

GaussCraft also integrates adaptive density control techniques to refine the mesh further. We adopt adaptive density control from Gaussian Avatar and 3DGS, with our unique method where excessive deviations in splat placement trigger pruning, ensuring a tightly integrated and efficiently edited mesh.

Finally, mesh deformation is handled using As-Rigid-As-Possible (ARAP) techniques [30], enabling dynamic updates to the mesh properties and their visualization during rendering.

In summary, the contributions of this paper are as follows:

- The introduction of flexible, real-time user-defined 2D Gaussian editing capabilities with mesh deformation.

- The simplification of local space dimensionality, enhancing both efficiency and result consistency.

- The implementation of novel adaptive density control to improve editing accuracy and performance.

This paper begins with a discussion of the preliminaries, explaining an in-depth look at traditional 3D representations, Neural Radiance Fields, 3D Gaussian Splatting, and ARAP deformation. Following that, we present a literature review, covering classical scene reconstruction methods, including point-based rendering and radiance fields. Next, we explore NeRF-Editing, which addresses a similar task to ours using implicit representation with NeRF [9]. We then move on to 2D Gaussian Splatting [10] and Gaussian Avatar [11], which serve as primary inspirations for our model. The methods section details the process of mesh reconstruction, user editing via mesh deformation, the approach for binding meshes and faces, and the optimization process, including adaptive Gaussian control. In the evaluation section, we apply our model to various datasets, providing both qualitative and quantitative assessments to demonstrate its efficiency and effectiveness. Finally, the conclusion summarizes our contributions and discusses potential directions for future work.

Chapter 2

Preliminaries

This chapter provides an overview of the foundational concepts and techniques essential for understanding the GaussCraft, which is focused on mesh reconstruction and editing. We begin with a detailed introduction to the primary methods of 3D representation in computer vision, exploring both implicit and explicit approaches. Subsequently, we delve into the specifics of 3D Gaussian Splatting, including discussions on Spherical Harmonics and the rasterization process crucial for rendering. The chapter concludes with an in-depth explanation of As-Rigid-As-Possible Deformation, focusing on techniques for managing rigid mesh deformation.

2.1 3D Representation and Reconstruction

3D reconstruction is the process of capturing the shape and appearance of real-world objects or scenes from sensor data, such as images, depth maps, or point clouds. This process can be categorized into two main types of representations: explicit and implicit. Each representation has its own strengths, limitations, and suitable use cases depending on the requirements of the application.

2.1.1 Explicit Representation

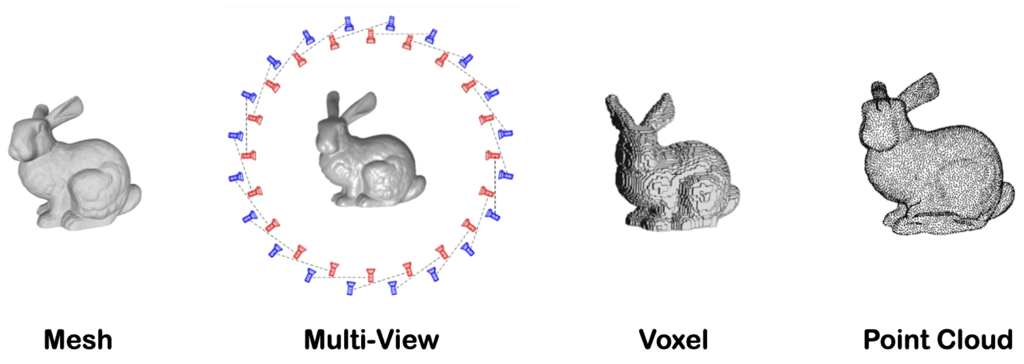

Figure 2.1: Explicit Representations of Stanford Bunny [1] (From left [1, 2, 3, 4])

Explicit representations involve directly storing and manipulating the geometric information of 3D objects. These methods provide a clear and direct way to describe the shape and structure of objects but often require significant storage and processing power.

Mesh

Meshes are the most commonly used explicit representation for 3D objects in computer graphics and computational geometry. They consist of vertices, edges, and faces, with each face typically being a triangle. Vertices are defined as 3D points in space, and edges connect these points to form triangular faces. This configuration allows for the representation of complex surfaces and is extensively applied in areas such as 3D modeling, animation, and finite element analysis.

Especially in triangular meshes, Numerous triangles discretize surface of a 3D object, and each triangles are specifically defined by the 3D coordinates of vertices. This collection of triangles provides a detailed approximation of the object's surface. Triangular meshes are particularly valued for their ability to represent complex surfaces with a relatively minimal number of elements. However, the creation and manipulation of high-resolution meshes can be computationally intensive, and the performance of the representation is heavily dependent on the mesh resolution, so it is often difficult to represent 3D object precisely. Despite these challenges, meshes remain a fundamental tool in many 3D reconstruction studies [31, 32, 33].

Multi-View Stereo (MVS)

Multi-view stereo (MVS) is a technique that reconstructs 3D geometry from multiple images taken from different viewpoints. By getting the correspondences between these images, MVS algorithms estimate the depth and surface normals of the scene, which are then used to build a 3D model. MVS is often used in conjunction with structure from motion (SfM) [29] to generate a dense point cloud or mesh from a sparse set of camera poses and feature points.

MVS can produce high-quality reconstructions with detailed surface geometry, especially when the input images have significant overlap and high resolution. However, the quality of the reconstruction can degrade in regions with textureless surfaces, occlusions, or reflective materials. Additionally, MVS requires careful calibration and alignment of the input images to produce accurate results.

Volumetric Representation

Volumetric representations is a 3D reconstruction method divide 3D space into a regular grid of voxels (volume elements), where each voxel stores information such as the presence, color, or other properties of the object within that volume. Typically, a voxel grid is used to represent the occupancy or color of space within a defined bounding box.

Volumetric methods like voxel grids [34, 35] are particularly effective for representing complex shapes and allowing topological changes during reconstruction. They are widely adopted in various applications, including real-time 3D reconstruction systems like KinectFusion [36]. However, the main limitation of traditional volumetric representations is their substantial memory consumption, especially at high resolutions. To address this challenge, sparse volumetric data structures such as octree [37] and quadtree [38] have been used. These structures only allocate memory for regions of the volume where data is present, significantly reducing memory usage while still capturing detailed geometrical information.

Point Cloud

A point cloud represents [39, 40] a 3D object as a set of discrete points in space, where each point is defined by its 3D coordinates, and sometimes with additional component such as color or normal vector. Point clouds are typically obtained from 3D scanning devices, such as LiDAR or depth cameras, or generated through techniques like MVS.

Point clouds are a flexible representation that can capture specific details of a surface and are suitable for applications like 3D scanning, object recognition, and surface reconstruction. However, point clouds do not essentially describe the connectivity between points, making it challenging to directly use them for tasks that require surface information, such as rendering or physical simulations. To overcome this limitation, point clouds are often used to convert into other representations, such as meshes or implicit surfaces.

2.1.2 Implicit Representation

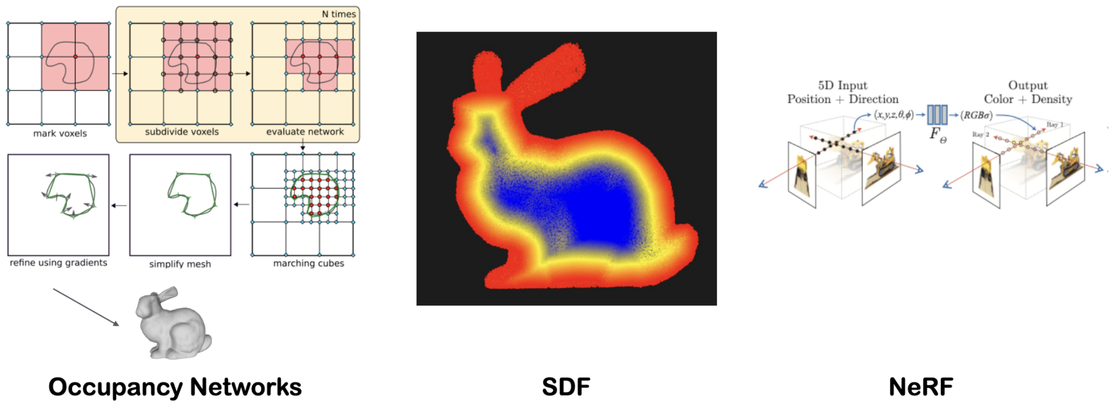

Figure 2.2: Implicit Representations of Stanford Bunny [1] and NeRF (From left [5, 6, 7])

Implicit representations encode the geometry of 3D objects using mathematical functions or neural networks rather than explicit surface descriptions. These methods offer a compact and continuous representation of shapes, which can be advantageous for certain types of 3D reconstruction and editing tasks.

Occupancy Networks

The occupancy function [5] is one of the simplest implicit representations, and it assigns a binary value $o(\mathbf{x})$ to each point $\mathbf{x}$ in space to classify whether the point lies inside (occupied, $o(\mathbf{x}) = 1$) or outside (unoccupied, $o(\mathbf{x}) = 0$) the object. With this binary classification, reconstruct the surface of the object can be done by identifying the boundary between occupied and unoccupied regions.

Voxel grids with occupancy value are often used with occupancy function. The surface of the object can then be extracted using techniques like marching cubes [41]. While this approach is conceptually simple and straightforward, but it have limitation of suffering from limited resolution and aliasing artifacts in the resulting surface.

Signed Distance Function (SDF)

A Signed Distance Function (SDF) is a scalar field $d(\mathbf{x})$ that assigns a value to each point $\mathbf{x}$ in space, representing the distance to the nearest surface of an object. The sign of this value indicates whether the point is inside ($d(\mathbf{x}) < 0$) or outside ($d(\mathbf{x}) > 0$) the object. The surface itself corresponds to the zero-level set of the SDF, where $d(\mathbf{x}) = 0$.

SDFs provide a smooth and continuous representation of surfaces, making them useful for applications like shape deformation, collision detection, and level set methods. These functions can be stored in a volumetric grid or represented implicitly using neural networks, as seen in recent deep learning approaches to 3D shape representation. However, high-resolution SDFs is memory intensive because capturing specific details correctly often requires high computation cost.

To overcome given limitation, Truncated Signed Distance Function (TSDF) is introduced, an extension of the SDF designed to optimize memory usage and computation in 3D data representations. Instead of recording the exact distance to the nearest surface, the TSDF truncates this distance to a predefined threshold, τ, focusing computational resources on regions near the surface.

Definition and Properties The TSDF, $d_{\text{TSDF}}(\mathbf{x})$, is defined as:

$$d_{\text{TSDF}}(\mathbf{x}) = \min(\max(d(\mathbf{x}), -\tau), \tau)$$

where $d(\mathbf{x})$ is the signed distance, and τ is the truncation threshold. Points with distances beyond τ are clamped to $\pm\tau$, depending on whether they are inside or outside the object.

Neural Radiance Fields (NeRF)

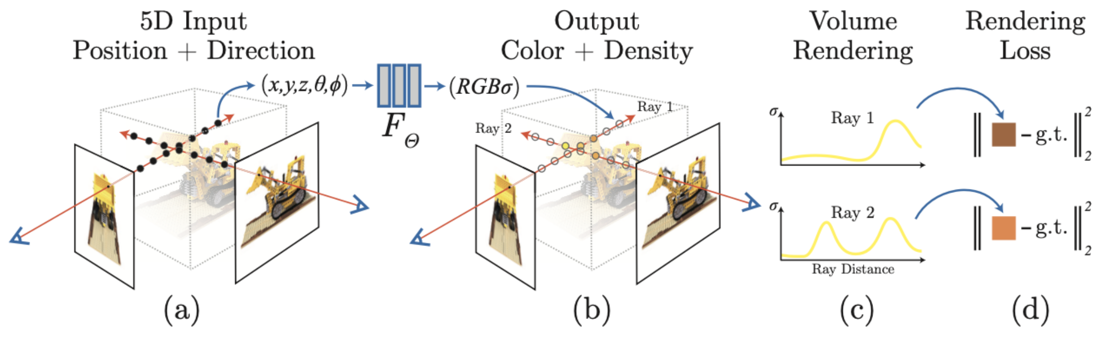

Figure 2.3: Overview of Neural Radiance Fields (NeRF) [7]

Neural Radiance Fields (NeRF) utilize a fully connected neural network to represent 3D scenes by modeling both color $\mathbf{c}$ and density σ at each spatial point. These quantities are functions of the 3D position $\mathbf{x} = (x, y, z)$ and the viewing direction $\mathbf{d} = (\theta, \phi)$, where θ and ϕ define the viewing angles.

Training Procedure NeRF is trained using a collection of images captured from various viewpoints. The neural network is designed to map a given pair of 3D coordinates and viewing direction $(\mathbf{x}, \mathbf{d})$ to corresponding RGB color $\mathbf{c}(\mathbf{x}, \mathbf{d})$ and density $\sigma(\mathbf{x})$. The objective of training is to minimize the error between the predicted views generated by NeRF and the actual observed images.

Volumetric Rendering NeRF's rendering process is based on volumetric rendering, where the color $C(\mathbf{r})$ of a ray $\mathbf{r}(t)$ passing through the scene is calculated. The ray is parameterized as:

$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$$

where $\mathbf{o}$ is the camera origin, t represents the distance along the ray, and $\mathbf{d}$ defines the ray direction.

The color of the ray $C(\mathbf{r})$ is obtained by integrating the contributions of color and density along the ray path:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})\, dt$$

Where:

- $T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))\, ds\right)$ is the accumulated transmittance, representing the probability that light is not absorbed prior to reaching point t.
- $\sigma(\mathbf{r}(t))$ is the volumetric density at point $\mathbf{r}(t)$, determining how much light is scattered or absorbed.
- $\mathbf{c}(\mathbf{r}(t), \mathbf{d})$ is the emitted color at point $\mathbf{r}(t)$ in direction $\mathbf{d}$.

Photorealistic Rendering NeRF excels in generating highly photorealistic images from novel viewpoints by learning a continuous and differentiable representation of the scene. Unlike traditional methods that rely on explicit geometry representations, NeRF encodes both geometry and appearance implicitly within the network.

Advantages and Limitations

NeRF offers several advantages, including its ability to provide a continuous and smooth 3D scene representation, making it particularly effective for tasks such as scene editing, view interpolation, and optimization. Additionally, NeRF excels at capturing fine surface details and complex lighting interactions, enabling the rendering of highly intricate scenes. However, it also presents some limitations. The model requires a substantial amount of images and significant computational resources for training, as it involves optimizing a large number of parameters. Moreover, despite its ability to produce high-quality renderings, NeRF's rendering process is relatively slow compared to traditional methods, as it relies on dense sampling and integration along each ray.

2.2 3D Gaussian Splatting

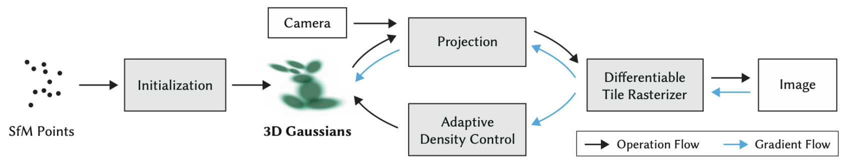

Figure 2.4: Overview of 3D Gaussian Splatting [8]

3D Gaussian Splatting (3DGS) [8] is a explicit representation in a way that allows efficient real-time rendering. In this section, we will discuss the initialization of 3D Gaussians, the use of spherical harmonics for representing view-dependent appearance, and the rasterization process that converts 3D Gaussians into 2D images.

2.2.1 Initialization

The initialization of 3D Gaussians in 3DGS [8] involves setting up the parameters that define each center of Gaussians M, covariances S, colors C, and opacities A.

The covariance matrix S is particularly important, as it defines the anisotropic shape of the Gaussian. It can be decomposed as:

Algorithm 1 Initialization of 3D Gaussian Splatting [8]

```
1: $M \leftarrow$ SfM Points                              ▷ Positions
2: $S, C, A \leftarrow$ InitAttributes()                  ▷ Covariances, Colors, Opacities
```

$$\Sigma = RSS^T R^T,$$

where R is a 4×1 quaternion vector converted into a 3×3 rotation matrix describing the orientation of the Gaussian, and S is a 3×1 scaling vector converted into a 3×3 scale matrix that controls the size along the principal axes. The initialization typically starts with setting μ at the center of the SfM points [29], R as an identity matrix (indicating no initial rotation), and S based on the expected spread of the Gaussian in each direction.

2.2.2 Spherical Harmonics

Spherical harmonics are used in 3DGS to represent the view-dependent appearance of each Gaussian. This approach allows the encoding of complex lighting effects, such as diffuse and specular reflections, in a compact form that is computationally efficient.

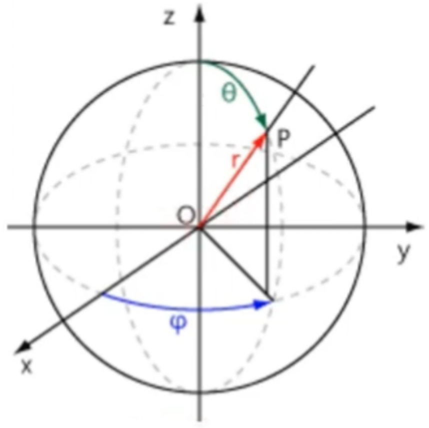

Figure 2.5: Spherical Harmonics in a spherical coordinate system

Spherical harmonics are mathematical functions defined on the surface of a sphere as figure 2.5, which can be used to approximate complex functions over spherical domains, such as the appearance of an object under varying lighting conditions. In a spherical coordinate system, a point P on the sphere is defined by two angles, θ (colatitude) and ϕ (longitude).

The spherical harmonics $Y_l^m(\theta, \phi)$ are given by:

$$Y_l^m(\theta, \phi) = \sqrt{\frac{(2l+1)(l-|m|)!}{4\pi(l+|m|)!}} P_l^{|m|}(\cos\theta) e^{im\phi},$$

where:

- l is the degree and m is the order of the harmonic.
- $P_l^{|m|}(\cos\theta)$ is the associated Legendre polynomial, defined as:

  $$P_l^{|m|}(\cos\theta) = (-1)^{|m|} \frac{(l+|m|)!}{(l-|m|)!} P_l^{(-|m|)}(\cos\theta),$$

  where $P_l^{|m|}(\cos\theta)$ is the standard Legendre polynomial.
- $e^{im\phi}$ accounts for the azimuthal variation.

These harmonics provide a basis for representing functions on a sphere, such as the color $c(\theta, \phi)$ of a Gaussian, which can be expressed as a linear combination of spherical harmonics:

$$c(\theta, \phi) = \sum_{l=0}^{L} \sum_{m=-l}^{l} c_l^m Y_l^m(\theta, \phi),$$

where:

- c_{lm} are the coefficients corresponding to each basis function.
- L is the order of the spherical harmonics, controlling the level of detail in the appearance representation.

These coefficients c_{lm} are learned during the training process, enabling the model to capture complex variations in appearance with respect to different viewing angles.

2.2.3 Rasterization

Rasterization in 3D Gaussian Splatting involves projecting the 3D Gaussians onto the 2D image plane, where they are converted into pixel values. Unlike traditional rasterization, which operates on triangles or polygons, 3DGS rasterization deals with ellipsoidal splats, defined by their Gaussian distributions.

Color Computation: The color C of a pixel is computed by blending the contributions of all Gaussians that project onto that pixel. The blending operation accounts for depth ordering to maintain correct visibility:

$$C = \sum_{i=1} c_i \alpha'_i \prod_{j=1}^{i-1} (1 - \alpha'_j),$$

where:

- c_i is the color contribution of the i-th Gaussian.
- α'_i is the opacity of the i-th Gaussian after considering its depth relative to the camera.
- The product term $\prod_{j=1}^{i-1}(1 - \alpha'_j)$ ensures that closer Gaussians occlude those further away.

Tile-based Rasterizer:

1. **Cull Gaussian**: Perform frustum culling to discard Gaussians that are outside the view frustum.
2. **Screen Space Gaussian**: Project 3D Gaussians onto 2D screen space. The covariance projection function is defined as where J is Jacobian of the affine approximation of the projective transformation and W is viewing transformation:

$$\Sigma' = JW\Sigma W^T J^T$$

3. **Create Tiles**: Split the screen into 16x16 pixel tiles, preparing the grid for further processing.

4. **Duplicate with Keys**: Generate keys for each Gaussian in every tile. Each key is a 64-bit value where the lower 32 bits represent the view-space depth of the Gaussian and the higher 32 bits represent the Tile ID.

5. **Sort by Keys**: Perform depth ordering by radix sorting the keys in each tile.

6. **Identify Tile Ranges**: Identify the range of Gaussians that belong to each tile based on the sorted keys.

7. **Get Tile Range**: Read the Gaussian list for each tile to maximize parallelism in loading, sharing, and forward/backward processing.

8. **Blend in Order**: Accumulate color and α values by traversing the lists front-to-back for each pixel. The process stops when a target saturation of α is reached in a pixel, ensuring proper blending of overlapping Gaussians.

9. **Backward Process**: During backpropagation, gradients are propagated following the ratio of the opacity of the Gaussian to ensure accurate gradient calculation across the splats.

This method optimizes for high-load scenarios with many small splats by efficiently sorting and processing Gaussians in parallel, utilizing the GPU's capabilities for real-time rendering.

2.2.4 Adaptive Control

Refinement Iteration Protocol: The system undergoes a refinement iteration every 100 iterations, ensuring periodic adjustments to the Gaussian components based on defined criteria.

- **Remove Gaussian**: Gaussians with opacity (α) less than a threshold $\epsilon = -0.005$ are removed from the system to prevent the processing of elements with negligible impact on the model.

- **Gradient-Based Adaptive Decisions**:
  - If the gradient of the view-space position exceeds a threshold $\tau_{pos} = 0.0002$, the algorithm decides between splitting or cloning the Gaussian based on its scale.
  - **Split Gaussian**: If the scale exceeds τ_{scale}, the Gaussian is split into two, with each new Gaussian having its scale divided by 1.6. This step is taken to refine the model detail in areas of high complexity.
  - **Clone Gaussian**: If the scale is below the threshold, the Gaussian is cloned and repositioned based on the gradient direction, aiding in covering more space in the model efficiently.
- **Initialize Gaussian**: Every 3000 iterations, all Gaussians have their opacity reset to zero to moderate the increase in the number of Gaussians and remove redundant or floating elements.

2.3 As-Rigid-As-Possible Deformation

The ARAP deformation [30] is mainly focuses on minimizing energy. Specifically, the technique defines an energy function that measures the deviation of the mesh's local transformations from being perfectly rigid. The energy function is calculated as the sum of the squared differences between the original mesh's local transformations (which are ideally rigid) and the deformed mesh's local transformations.

In mathmatic formula, the energy E in ARAP can be expressed as:

$$E = \sum_{i} \sum_{j \in \mathcal{N}(i)} w_{ij} \left\| (\mathbf{R}_i(\mathbf{v}_j - \mathbf{v}_i)) - (\mathbf{v}'_j - \mathbf{v}'_i) \right\|^2$$

where:

- $\mathbf{v}_i$ and $\mathbf{v}_j$ are the positions of adjacent vertices in the original mesh.
- $\mathbf{v}'_i$ and $\mathbf{v}'_j$ are the positions of the corresponding vertices in the deformed mesh.

- $\mathbf{R}_i$ is the optimal rotation matrix that approximates the transformation of vertex i and its neighbors.
- w_{ij} is a weight that typically depends on the distance between vertices i and j, often derived from the original mesh's structure.
- $\mathcal{N}(i)$ denotes the set of neighboring vertices around vertex i.

The ARAP algorithm works iteratively to minimize this energy. The process involves local and global steps repeated until convergence:

1. **Local Step:** For each vertex, the algorithm computes the optimal rotation matrix $\mathbf{R}_i$ that best approximates a rigid transformation of the local neighborhood. This step ensures that each part of the mesh tries to move in a way that closely resembles a rigid body movement.
2. **Global Step:** The algorithm then updates the positions of the vertices in the deformed mesh by solving a linear system that minimizes the overall energy E, given the fixed rotation matrices $\mathbf{R}_i$ computed in the local step.

Through this iterative process, ARAP efficiently reduces the energy, leading to deformations that are visually smooth and natural while preserving the original mesh's local structures. Users can interactively manipulate specific vertices or regions of the mesh, and the ARAP technique ensures that the resulting deformations maintain the mesh's integrity and aesthetic quality.

Chapter 3

Literature Review

In this section, we will begin by exploring classic scene reconstruction methods, followed by an examination of neural radiance field-based editing techniques. After that, we will delve into two key studies that primarily inspired our method: 2D Gaussian Splatting and Gaussian Avatar.

3.1 Classic Scene Reconstruction

Traditional scene reconstruction techniques for novel view synthesis originated from light fields, often using dense sampling techniques [42]. These early methods faced challenges when handling unstructured capture data. The advent of Structure-from-Motion (SfM) [29] allowed for the collection of photos to synthesize novel views by estimating a sparse point cloud during camera calibration. Multi-view stereo (MVS)[13] techniques followed, producing impressive 3D reconstructions by blending the captured images into the novel view, using the geometry for re-projection [43, 44, 45, 46]. However, these methods often struggled with unstructured regions and over-reconstruction artifacts. Recent advancements in neural rendering algorithms [47] have reduced these artifacts and mitigated the issues associated with storing all input images on the GPU, outperforming traditional methods in most cases.

3.2 Point-Based Rendering and Radiance Fields

Point-based methods traditionally rendered disconnected and unstructured geometry samples, which resulted in visible artifacts like holes or aliasing. However, advancements in point-based rendering, such as splatting techniques and neural point rendering, have addressed many of these issues.

Recent work has integrated point-based methods with NeRF [7]-like volumetric rendering to take advantage of the strengths of both approaches, leading to improved rendering quality. These methods allow for more flexible scene representations and efficient rendering, particularly for scenes with complex depth or dynamic content.

Methods like Pulsar [48] and ADOP [49] focus on faster and more scalable point-based rendering, using techniques like diffuse point-based blending to maintain high-quality results even in challenging scenarios. Additionally, some approaches utilize anisotropic covariance optimization and efficient depth sorting to handle scenes with complex geometry, offering a robust solution for real-time novel view synthesis.

3.3 NeRF editing

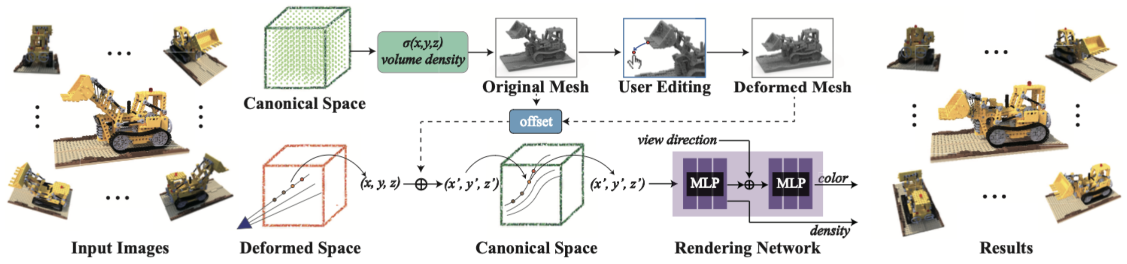

Figure 3.1: The pipeline of NeRF editing framework [9]

The neural radiance field (NeRF) has demonstrated remarkable performance in novel view synthesis, allowing the creation of highly realistic images from diverse viewpoints. Building on this foundation, this approach facilitates shape deformation within a scene, enabling the generation of new images from arbitrary perspectives after editing.

Initially, in NeRF-editing [9] the NeRF network is trained to reconstruct triangular mesh using Marching Cubes. The mesh from the original NeRF network often has a rough surface. To improve editing, the NeuS [50] method is used, which learns geometry as a neural signed distance function (SDF). The mesh from the SDF's zero-level set serves as the editing object, allowing intuitive scene modifications. Subsequently, an explicit triangular mesh representation from the scene's implicit representation is explained, allowing users to directly edit it.

After the user edits the triangular mesh of the scene, the deformation is transferred to the implicit volume in two steps. First, cage mesh is created, and it is converted to tetrahedral mesh by tetrahedronize the cage using TetWild. The displacement of the triangular mesh vertices v_i drives the deformation of the tetrahedral mesh, transferring the surface changes to the volume. The deformed tetrahedral mesh is denoted as T' and t' is vertices of tetrahedral mesh after deformation while v' is vertices of triangular mesh after deformation The ARAP deformation is applied to tetrahedral mesh based on surface constraints, calculating the barycentric coordinates of each triangular vertex within its corresponding tetrahedron. The optimization problem is:

$$\min E(T'), \text{ subject to } At' = v',$$

A is the barycentric weight matrix, solvable with the Lagrangian multiplier.

After applying surface deformation to the tetrahedral mesh, the discrete deformation field of the "effective space" which is the internal space of the cage mesh is derived and use it to bend the casting rays. Rays are cast into the space containing the deformed tetrahedral mesh to render the deformed radiance field. For each sampled point, the corresponding tetrahedron in the deformed mesh T' is located. Using the displacement of vertices before and after deformation, the displacement Δp is computed for each point via barycentric interpolation. This displacement is added to the input coordinates to predict the density and RGB values:

$$\zeta(\mathbf{p} + \Delta\mathbf{p}), \zeta(\mathbf{d}) \to (\sigma, \mathbf{c}).$$

3.4 2D Gaussian Splatting

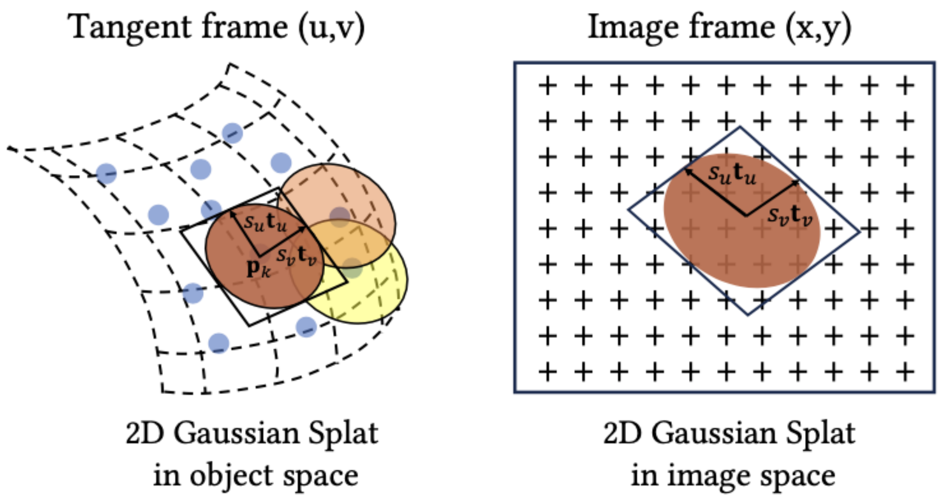

Figure 3.2: Illustration of 2D Gaussian Splatting [10]

To achieve accurate geometry reconstruction while ensuring high-quality novel view synthesis, the authors introduce differentiable 2D Gaussian splatting (2DGS). Unlike 3D Gaussian splatting (3DGS), which encapsulates the entire angular radiance in a volumetric blob, this method simplifies the modeling by employing "flat" 2D Gaussians within a 3D space. In this approach, the 2D Gaussian represents a density distribution over a planar disk, with the disk's normal vector indicating the direction of the steepest density gradient. This feature allows for more precise alignment with thin surfaces. While some earlier methods also use 2D Gaussians for geometry reconstruction, they typically require dense point clouds or ground-truth normals. In contrast, this technique reconstructs both appearance and geometry simultaneously, relying solely on a sparse calibration point cloud and photometric supervision.

The 2D Gaussian splat is defined by a central point p_k, two principal tangential vectors t_u and t_v, and a scaling vector $S = (s_u, s_v)$ that controls the Gaussian's variances. The primitive's normal vector is determined by the cross product of the two tangential vectors, $t_w = t_u \times t_v$. The orientation of the splat can be described by a 3×3 rotation matrix $R = [t_u, t_v, t_w]$, and the scaling factors are represented by a 3×3 diagonal matrix S, with the third entry being zero.

The 2D Gaussian is defined in a local tangent plane within the 3D world space and is parameterized as:

$$P(u, v) = p_k + s_u t_u u + s_v t_v v = H(u, v, 1, 1)^\top,$$

$$H = \begin{bmatrix} s_u t_u & s_v t_v & 0 & p_k \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} RS & p_k \\ 0 & 1 \end{bmatrix}$$

is a homogeneous transformation matrix that encapsulates the geometry of the 2D Gaussian. For any point $u = (u, v)$ within the uv space, the 2D Gaussian value is calculated using the standard Gaussian function:

$$\mathcal{G}(u) = \exp\left(-\frac{u^2 + v^2}{2}\right).$$

The parameters for the center p_k, scaling (s_u, s_v), and rotation (t_u, t_v) are all learnable. Similar to 3DGS, each 2D Gaussian primitive has an associated opacity α and a view-dependent color c, which are parameterized using spherical harmonics.

After that, 2D Gaussians are rendered by projecting them onto the image plane via an affine approximation. Each Gaussian is centered at a point in 3D space, with increasing projection error as the camera moves farther from this center. To reduce this error, a homogeneous transformation matrix is used to map world coordinates into screen space.

For efficient ray-splat intersections, the splatting process as finding the intersection of two planes is represented: the x-plane and the y-plane. The intersection is represented by a point in homogeneous coordinates, which is then projected onto the image space. This method helps handle degenerate cases where the splat degenerates into a line under extreme viewing angles. To stabilize the rendering of these degenerate splats, a low-pass filter is applied, ensuring that the Gaussian splat remains well-defined across different viewpoints.

3.4.1 Mesh Extraction

To extract meshes from the reconstructed 2D splats, the authors utilize a process that involves rendering depth maps of the training views and subsequently fusing these depth maps into a coherent 3D mesh using Truncated Signed Distance Function (TSDF) fusion, as implemented in Open3D [51].

Depth Map Rendering The first step in their approach is to render depth maps from the training views. For each view, the median depth value of the splats projected onto the pixels is calculated, providing a robust depth estimate that reduces the impact of noise and outliers.

TSDF Fusion The depth maps are then fused into a single 3D volumetric representation using TSDF fusion. In TSDF fusion, each voxel in a predefined 3D grid accumulates signed distance values from the depth maps. The fusion algorithm combines these signed distances by taking into account the confidence in each depth measurement, usually weighted by factors such as the viewing angle and distance from the camera. This fusion process results in a smooth, coherent surface representation of the reconstructed object.

In their implementation, the voxel size is set to 0.004, which determines the resolution of the resulting mesh, and the truncation threshold is set to 0.02, which controls the maximum distance from the surface that is considered during the fusion process. These parameters are crucial in balancing the trade-off between mesh detail and computational efficiency.

Surface Reconstruction Once the TSDF volume is constructed, the surface is extracted using the Marching Cubes algorithm [41], which converts the volumetric data into a mesh. The Marching Cubes algorithm iterates through the voxels in the grid, identifying how the surface intersects each voxel, and generates triangles to approximate the surface. This results in a triangular mesh that can be used for rendering or further processing.

3.5 Gaussian Avatar

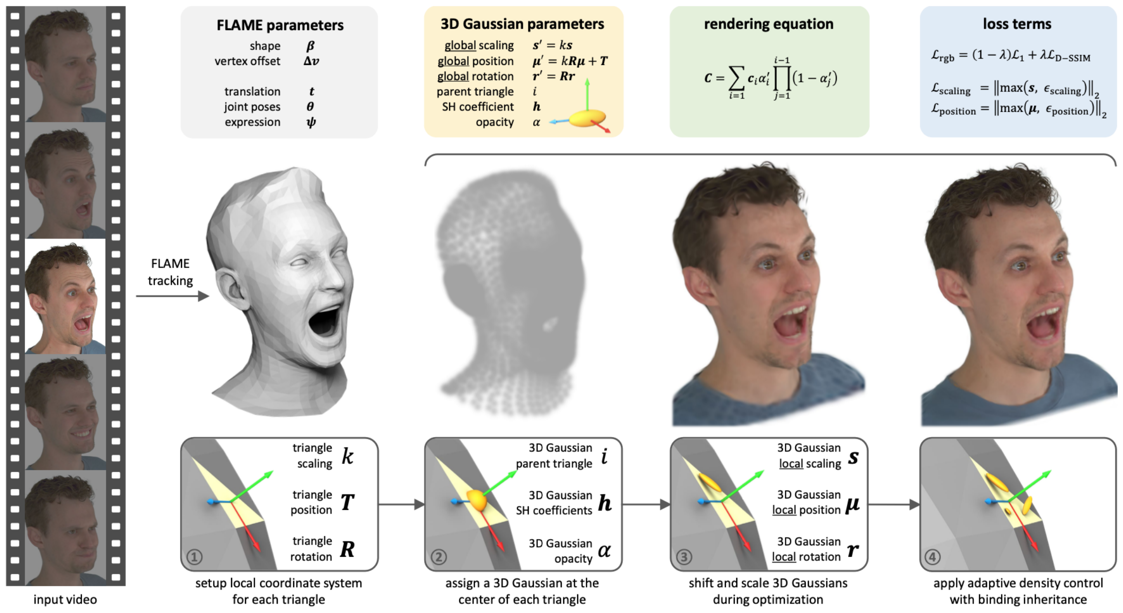
FLAME parameters
shape β
vertex offset Δv
translation t
joint poses θ
expression ψ
3D Gaussian parameters
global scaling $s' = ks$
global position $\mu' = kR\mu + T$
global rotation $r' = Rr$
parent triangle i
SH coefficient h
opacity α
rendering equation
$C = \sum_{i=1} c_i \alpha_i' \prod_{j=1}^{i-1} (1 - \alpha_j')$
loss terms
$\mathcal{L}_{\text{rgb}} = (1-\lambda)\mathcal{L}_1 + \lambda \mathcal{L}_{\text{D-SSIM}}$
$\mathcal{L}_{\text{scaling}} = \|\max(s, \epsilon_{\text{scaling}})\|_2$
$\mathcal{L}_{\text{position}} = \|\max(\mu, \epsilon_{\text{position}})\|_2$
FLAME tracking
triangle scaling k
triangle position T
triangle rotation R
3D Gaussian parent triangle i
3D Gaussian SH coefficients h
3D Gaussian opacity α
3D Gaussian local scaling s
3D Gaussian local position μ
3D Gaussian local rotation r
input video
setup local coordinate system for each triangle
assign a 3D Gaussian at the center of each triangle
shift and scale 3D Gaussians during optimization
apply adaptive density control with binding inheritance

Figure 3.3: Overview of Gaussian Avatar [11]

The main concept of the Gaussian Avatar method [11] is establishing links between the FLAME [24] mesh and 3D Gaussian splats. Initially, each triangle in the mesh is associated with a 3D Gaussian, and this Gaussian moves with the corresponding triangle over time. More specifically, the Gaussian remains fixed within the triangle's local space but dynamically changes position in the global (metric) space as the triangle moves. The mean position of the triangle's vertices, T, is used as the local space origin. The direction of one edge, the triangle's normal vector, and their cross product are concatenated to form the rotation matrix R, which defines the orientation of the triangle in global space. The scaling factor k is calculated using the mean edge length and its perpendicular to describe the triangle's size.

For each triangle's 3D Gaussian, its local position μ, rotation r, and anisotropic scaling s are defined in the local space. Initially, μ is set to the local origin, r is an identity matrix, and s is a unit vector. During rendering, these local parameters are transformed into global space by the following equations:

$$r' = Rr,$$

$$\mu' = kR\mu + T,$$

$$s' = ks.$$

The scaling of the triangle is incorporated into these transformations, which ensures that the 3D Gaussian's local position and scaling are relative to the triangle's size. This allows for an adaptive step size in metric space while maintaining a constant learning rate for parameters in the local space. As a result, 3D Gaussians associated with smaller triangles move slower during iteration steps compared to those paired with larger triangles, making the interpretation of parameters related to the distance from the triangle center more intuitive.

Chapter 4

Method

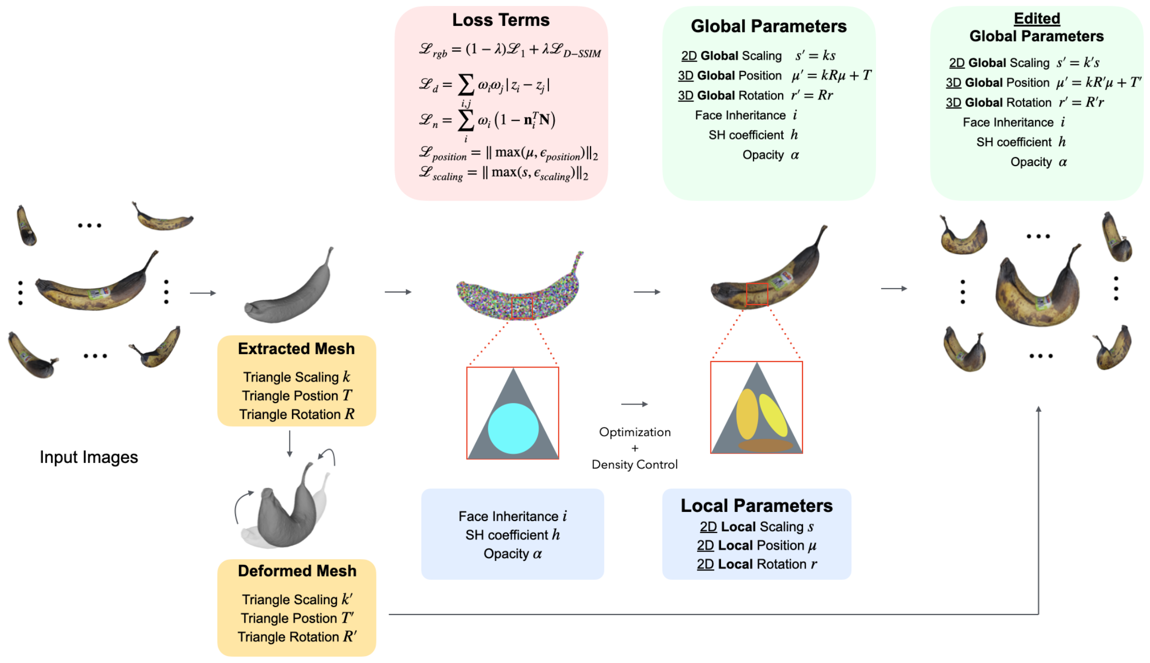
Loss Terms
$\mathscr{L}_{rgb} = (1-\lambda)\mathscr{L}_1 + \lambda\mathscr{L}_{D-SSIM}$
$\mathscr{L}_d = \sum_{i,j} \omega_i\omega_j |z_i - z_j|$
$\mathscr{L}_n = \sum_i \omega_i \left(1 - \mathbf{n}_i^T\mathbf{N}\right)$
$\mathscr{L}_{position} = \| \max(\mu, \epsilon_{position})\|_2$
$\mathscr{L}_{scaling} = \| \max(s, \epsilon_{scaling})\|_2$
Global Parameters
2D Global Scaling $s' = ks$
3D Global Position $\mu' = kR\mu + T$
3D Global Rotation $r' = Rr$
Face Inheritance i
SH coefficient h
Opacity α
Edited Global Parameters
2D Global Scaling $s' = k's$
3D Global Position $\mu' = kR'\mu + T'$
3D Global Rotation $r' = R'r$
Face Inheritance i
SH coefficient h
Opacity α
Input Images
Extracted Mesh
Triangle Scaling k
Triangle Postion T
Triangle Rotation R
Deformed Mesh
Triangle Scaling k'
Triangle Postion T'
Triangle Rotation R'
Optimization + Density Control
Face Inheritance i
SH coefficient h
Opacity α
Local Parameters
2D Local Scaling s
2D Local Position μ
2D Local Rotation r

Figure 4.1: Overview of GaussCraft

The input for GaussCraft consists of cameras calibrated via Structure-from-Motion (SfM) [29], and it employs 2D Gaussian splatting [10] for mesh extraction. Initially, each 2D Gaussian is initialized based on a sparse point cloud generated by the SfM process, with the Gaussian positions set to match the locations of the point cloud. If point clouds are not available alongside the cameras, the positions of the Gaussians are initialized randomly. The parameters of these 2D Gaussians are then optimized using the corresponding images and camera parameters. This optimization refines the geometric reconstruction, demonstrating the effectiveness of 2D Gaussian splatting in accurately capturing the scene's structure. Users can modify the reconstructed mesh using As-Rigid-As-Possible (ARAP) deformation, as discussed in this paper, or opt for alternative deformation techniques, such as skeleton or cage-based methods.

As depicted in Figure 4.1, after the deformation, the model processes the reconstructed mesh, images, and camera parameters as inputs. It employs the Gaussian Avatar methodology to adjust the scale, position, and rotation of each Gaussian in both global and local spaces. The model initializes a 2D Gaussian at the center of each mesh face. Contrary to Gaussian Avatar, this model treats each mesh face as a tangent plane to the bound 2D Gaussians, with local adjustments in scale, position, and rotation expressed in two dimensions. The loss function integrates RGB, Depth Distortion, and Normal Consistency losses from 2DGS with position and scale losses from Gaussian Avatar, ensuring that the Gaussians minimally deviate from their corresponding mesh faces and preventing excessive scaling.

The model also adopts an adaptive density control similar to Gaussian Avatar, implementing the Binding Inheritance theory which binds a new 3D Gaussian to the same triangle as the old one and prevents pruning of all Gaussians on a face by maintaining at least one per face. To address artifacts from the misalignment of 2D Gaussians with mesh faces, a pruning process includes a position threshold that significantly restricts the movement of each Gaussian's center away from the center of its face. This methodology ensures the accurate and efficient rendering of edited novel view synthesis scenes.

After the optimization of Gaussians with the reconstructed mesh, the user can input the deformed mesh for rendering. The model then utilizes the mesh properties of the deformed mesh instead of the reconstructed mesh to recalculate the modified global parameters of scale, position, and rotation. This allows the model to render the user-edited scene effectively in real-time.

4.1 Reconstruction

The reconstruction process begins by utilizing the optimized 2D Gaussians obtained from the input set of images, camera parameters, and the Structure-from-Motion (SfM) point cloud. The initial sparse point cloud derived from the SfM process is used to initialize these 2D Gaussians, which are then iteratively refined.

A crucial part of this process involves mitigating noise that can arise when optimizing purely with photometric losses, a common challenge in 3D reconstruction tasks. To enhance the accuracy of the geometry reconstruction, two regularization terms: depth distortion and normal consistency are introduced. Both of them will be deeply handled in optimization part, and this part it will explain their role briefly.

4.1.1 Depth Distortion

Unlike NeRF, which focuses on volumetric rendering, the 2D Gaussian splatting method doesn't inherently consider the distance between intersected Gaussian primitives. This can lead to inaccuracies in color and depth rendering, particularly when Gaussians overlap along a ray. To address this, a depth distortion loss (L_d) is introduced. This loss minimizes the distance between ray-splat intersections by concentrating the weight distribution along the rays, thus reducing discrepancies in depth representation. The efficient implementation of this regularization, utilizing CUDA, ensures that the 2D Gaussians are properly aligned along the ray depth.

4.1.2 Normal Consistency

To ensure that the 2D Gaussian splats accurately align with the surfaces they represent, it is necessary to enforce normal consistency. This involves aligning normals of the splats with the actual surface normals of the object. The normal consistency loss (L_n) encourages this alignment by penalizing deviations between the splat normal and the estimated surface normal. Specifically, the normal is computed based on the gradients of the depth maps, ensuring that the 2D splats adhere closely to the underlying geometry.

4.1.3 Mesh Extraction

To extract meshes from the reconstructed 2D splats, we render depth maps of the training views by using the median depth value of the splats projected to the pixels. We then utilize truncated signed distance fusion (TSDF) to fuse these reconstruction depth maps, following the approach implemented in Open3D [51]. During the TSDF fusion, voxel size and the truncated threshold are set to 0.004 and 0.02 as 2DGS.

4.2 User Editing

Once the mesh is obtained, users can edit it through mesh deformation techniques. In this paper, we primarily utilize As-Rigid-As-Possible (ARAP) mesh deformation due to its ability to offer intuitive and flexible manipulation while preserving the mesh's local geometric properties by minimizing distortion. ARAP is particularly effective in preventing artifacts that can arise from excessively large or irregularly shaped faces, which may challenge the Gaussians' ability to cover those areas accurately.

By using ARAP, the model produces a deformed mesh that retains the relative shape and size of each triangular face, ensuring that the Gaussians can more accurately represent the edited scene. This preservation of local structure is essential for maintaining the consistency of the reconstructed scene, allowing the Gaussians to capture and render fine details effectively. Again, other mesh deformation methods such as skeleton or cage-based deformations also can be used instead of ARAP deformation.

4.3 2D Gaussian Rigging

After mesh reconstruction, this model incorporates the concepts from the Gaussian Avatar [11], which uses global and local parameters to link triangular meshes with 2D Gaussian splats. While Gaussian Avatar is built on the principles of 3D Gaussian Splatting (3DGS), this project adapts these ideas specifically for 2D Gaussian Splatting (2DGS), tailoring the approach to operate effectively in a 2D environment.

In this approach, each triangle of the mesh is paired with a 2D Gaussian, which follows the triangle as it deforms. Unlike 3DGS, where Gaussians are associated with volumetric elements, 2DGS ties the Gaussians to the surfaces represented by the triangular mesh. The key difference from Gaussian Avatar is that, in 2DGS, the Gaussians are limited to two dimensions. Each face of the mesh is treated as the tangent space for its corresponding splat. This dimensional reduction minimizes the likelihood of splats moving excessively along the Z-axis, leading to both more stable behavior and reduced computational overhead.

The process begins by assigning a fixed 2D position to each Gaussian within the local space of its parent triangle, while allowing it to move dynamically in the global space as the triangle deforms. Initially, the mean position T of vertices of triangular face is chosen as the origin of the local space. A rotation matrix R is then constructed using the direction of one triangle edge, the normal vector of the triangle, and their cross product to establish the orientation of the triangle in global space. Additionally, a scalar k is derived from the mean length of one edge and its perpendicular distance, ensuring consistent scaling between the local and global spaces.

For each 2D Gaussian associated with a triangle, we define its 2D position μ, 2D rotation r, and anisotropic scaling s within the triangle's local space. The position μ is initialized at the origin of this local space, the rotation r is set as an identity matrix (indicating no initial rotation), and the scaling s is initialized as a unit vector, representing uniform scaling.

To enforce the 2D constraint and eliminate any rotation around the z-axis, we modify the quaternion $q = (q_w, q_x, q_y, q_z)$ used in the rotation matrix. In a 3D rotation, q_x, q_y, q_z define the axis of rotation, while q_w governs its magnitude. By setting $q_z = 0$, we restrict the rotation to the xy-plane, ensuring no contribution from the z-axis and effectively confining the system to a 2D transformation. This allows the 2D Gaussian to behave as if it were constrained to a tangent plane associated with each triangle.

Additionally, the position vector μ is treated as a 2D vector rather than a 3D one. This simplifies the transformation by keeping the 2D Gaussian aligned with the local tangent plane of the triangle. This setup treats each triangle face as a 2D plane for the Gaussian splat, which is particularly important for ensuring that the Gaussians maintain their intended orientation and scaling during the rendering process.

During rendering, these local properties position μ, rotation r, and scaling s are transformed into the global space. The transformation ensures that the 2D Gaussians follow the deformations of the corresponding mesh triangles in a way that preserves their local orientation and anisotropic scaling. The following transformations are applied to the local parameters:

$$r' = Rr, \tag{4.1}$$

$$\mu' = kR\mu + T, \tag{4.2}$$

$$s' = ks. \tag{4.3}$$

By incorporating triangle scaling, the local position and scaling of a 2D Gaussian are defined relative to the absolute scale of the associated triangle. This enables an adaptive step size in the metric space with a constant learning rate for parameters defined in the local space. For example, a 2D Gaussian paired with a smaller triangle will move slower during an iteration step compared to those paired with larger triangles. This approach also simplifies the interpretation of the parameters concerning the distance from the triangle's center.

This rigging process ensures that as the triangles deform or move, the corresponding 2D Gaussians adjust accordingly, maintaining their relative positions, orientations, and scales. By adapting the Gaussian Avatar methodology to 2DGS, this project achieves a more accurate and responsive scene representation tailored to 2D surfaces, allowing for robust and flexible rendering of novel views.

4.4 Optimization

This model employs five distinct loss functions: RGB loss [8], depth distortion loss, normal consistency loss [10], and position and scale loss [11].

4.4.1 RGB Loss

The RGB loss can be expressed as:

$$\mathcal{L}_{\text{rgb}} = (1-\lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{\text{D-SSIM}},$$

where the L1 loss is combined with the D-SSIM term. The parameter λ controls the balance between the L1 loss and the D-SSIM loss, allowing for a trade-off between pixel-level accuracy and perceptual similarity.

4.4.2 Depth Distortion Loss

The depth distortion loss, inspired by Mip-NeRF360 [18], addresses the issue of Gaussian primitives that are close in color and depth becoming overly spread out during the rendering process. This loss function reduces such artifacts by concentrating the weight distribution along rays, thereby enhancing depth accuracy. Specifically, the loss is formulated as:

$$\mathcal{L}_d = \sum_{i,j} \omega_i \omega_j |z_i - z_j|,$$

where ω_i represents the blending weight for the i-th intersection point along the ray:

$$\omega_i = \alpha_i \hat{G}_i(\mathbf{u}(\mathbf{x})) \prod_{j=1}^{i-1} \left(1 - \alpha_j \hat{G}_j(\mathbf{u}(\mathbf{x}))\right).$$

Here, α_i denotes the opacity of the i-th splat, $\hat{G}_i(\mathbf{u}(\mathbf{x}))$ is the Gaussian function evaluated at that point, and the product term accounts for the accumulated transparency from preceding splats along the ray. This approach ensures precise depth placement of splats and prevents overlapping Gaussians from introducing artifacts in the rendering process.

Unlike the distortion loss in Mip-NeRF360, which focuses on fixed sample points, this allows GaussCraft dynamically adjusts the depth of these intersections, resulting in a more accurate and coherent scene representation by effectively distributing depth information along the rays.

4.4.3 Normal Consistency Loss

The normal consistency loss ensures that the surface normals of all 2D splats align correctly with the actual surface geometry, especially when viewed from different angles. This is crucial for maintaining surface integrity, particularly in complex scenes where semi-transparent surfaces might overlap. The loss encourages the alignment of splat normals with the gradients of the depth maps, specifically at the point where the cumulative opacity reaches 0.5. It is defined as:

$$\mathcal{L}_n = \sum_i \omega_i \left(1 - \mathbf{n}_i^T \mathbf{N}\right),$$

where i refers to the index of the intersected splat along the ray, ω_i is the blending weight, $\mathbf{n}_i$ is the normal of the splat facing the camera, and $\mathbf{N}$ is the normal calculated from nearby depth points. The normal $\mathbf{N}$ is computed using a finite difference method:

$$\mathbf{N}(\mathbf{x}, \mathbf{y}) = \frac{\nabla_x p \times \nabla_y p}{|\nabla_x p \times \nabla_y p|}.$$

By enforcing this alignment, the normal consistency loss ensures that the rendered splats accurately follow the surface they represent, reducing artifacts and improving visual coherence.

4.4.4 Position Loss

To ensure that the 2D Gaussian splats accurately follow the geometry of the underlying mesh, a position loss is applied. This loss maintains the spatial relationship between the splats and the mesh triangles they represent. A key requirement is that each Gaussian splat remains close to its associated triangle. For example, a splat representing a feature on the hand should not drift toward an unrelated triangle, such as one on the arm. Although splats are initially placed at the center of their respective triangles, with additional splats introduced nearby during optimization, there is still a risk of misalignment. To mitigate this, we adopt a regularization term that constrains the local position of each Gaussian:

$$\mathcal{L}_{\text{position}} = \| \max(\mu, \epsilon_{\text{position}}) \|_2,$$

where $\epsilon_{\text{position}} = 1$ acts as a threshold, permitting minor positional shifts that remain within the bounds of the triangle's scale. This approach helps keep the splats properly aligned with the intended features of the mesh, preventing significant deviations that could lead to visual inaccuracies.

4.4.5 Scale Loss

The scale of the 2D Gaussian splats is crucial for maintaining visual coherence, particularly during animations. If the scale of a Gaussian splat differs significantly from that of its associated triangle, it can lead to undesirable artifacts, such as jittering, when the triangle undergoes even minor rotations. To prevent this, a scale loss is implemented to regularize the size of each Gaussian splat relative to its parent triangle:

$$\mathcal{L}_{\text{scaling}} = \| \max(s, \epsilon_{\text{scaling}}) \|_2,$$

where $\epsilon_{\text{scaling}} = 0.6$ serves as a threshold, deactivating the loss term when a Gaussian's scale is less than 60% of its triangle's scale. This tolerance is necessary to avoid excessive shrinkage of the splats, which could otherwise degrade rendering performance. By maintaining an appropriate scale, the splats effectively represent the scene without introducing jitter or other artifacts.

4.4.6 Final Loss Function

The final loss function is therefore expressed as:

$$\mathcal{L} = \mathcal{L}_{\text{rgb}} + \alpha \mathcal{L}_{\text{d}} + \beta \mathcal{L}_{\text{n}} + \gamma \mathcal{L}_{\text{p}} + \delta \mathcal{L}_{\text{s}},$$

where $\alpha = 100$ is suitable for bounded meshes as mentioned in 2DGS [10], $\beta = 0.05$, $\gamma = 0.5$, and $\delta = 1$. The terms $\mathcal{L}_{\text{p}}$ and $\mathcal{L}_{\text{s}}$ are only applied to visible splats.

These combined loss functions ensure a balanced and effective optimization process, allowing the model to produce accurate and visually coherent results.

4.5 Adaptive Density Control

We utilize the densification and pruning method for 2D Gaussian splatting as described in background. This process begins with an initial sparse set of Gaussians obtained from center of faces of mesh, which we iteratively refine to more accurately capture the scene's geometry.

During optimization, Gaussians are densified every 100 iterations starting from the 600th iteration. This process targets regions where geometric features are missing (under-reconstructed areas) and where Gaussians cover large portions of the scene (over-reconstructed areas). The densification is guided by the magnitude of the view-space position gradients, which are monitored using a threshold parameter, τ_{pos}, set to 0.0002 in our experiments.

For Gaussians in under-reconstructed regions, we clone them by creating duplicates with identical sizes and shifting them along the positional gradient. In contrast, for large Gaussians with high variance, we split them into smaller Gaussians by scaling down their size by a factor of $\phi = 1.6$, while maintaining a constant total volume.

To prevent an excessive increase in Gaussian density, we regularly prune Gaussians that are either too large or have a transparency level (α) below a threshold ϵ_α. This pruning helps maintain a balance between accurate representation and computational efficiency.

In addition, we employ a binding inheritance strategy inspired by the Gaussian Avatar technique [11]. This ensures that new Gaussians generated during densification remain linked to their original geometric context. Specifically, each 2D Gaussian created during densification is bound to the same triangle as its predecessor, preserving continuity and fidelity to the local scene geometry. This binding inheritance mechanism involves associating each Gaussian with the index of its parent triangle, ensuring it inherits the position and orientation within the global space.

We adopt an additional pruning technique to remove Gaussians that are too far from the center of the mesh, as 2D Gaussians can produce more inconsistent results with inappropriate positioning than 3D Gaussians. First, we compute the Euclidean distance for each point from the center, and then define an initial threshold. To make this threshold adaptable, we scale it based on the range between the maximum and minimum distances. This scaled threshold determines which points are pruned: any point exceeding the effective threshold is removed. This dynamic pruning ensures that the editing remains controlled by eliminating outliers, improving the overall stability and quality of the process.

This approach is particularly crucial in regions with complex geometry, where a higher Gaussian density is needed to accurately represent the edited scene. Additionally, the binding inheritance strategy ensures that each triangle consistently has at least one splat attached, even after pruning. This prevents issues such as floating artifacts or loss of detail in occluded regions. This comprehensive adaptive control strategy enables more accurate and efficient representation of complex scenes without unnecessary computational overhead.

Chapter 5

Experiment and Evaluation

In this chapter, we present the results and performance evaluation of our model, showcasing the editing outcomes and conducting an ablation study by comparing various ablated models with GaussCraft. Lastly, we address the limitations and discuss potential improvements.

5.1 Implementation

We have extended the renderer provided by 2DGS [10] to accept input meshes and render edited scenes based on the type of deformed mesh provided. All models were trained for 30,000 iterations, with the Adam optimizer [52] used for parameter optimization. The learning rates for position, feature, opacity, scaling, and rotation were initialized at 0.005, 0.0025, 0.05, 0.017, and 0.001, respectively. The position learning rate decays from 0.005 to 0.00005 over the course of training, with a delay multiplier of 0.01 and a maximum step count of 30,000. Also, We implement activating binding inheritance every 100 iterations from iteration 500 until the completion of training. All experiments were conducted on a single NVIDIA RTX 3090 TI GPU.

5.2 Dataset and Metrics

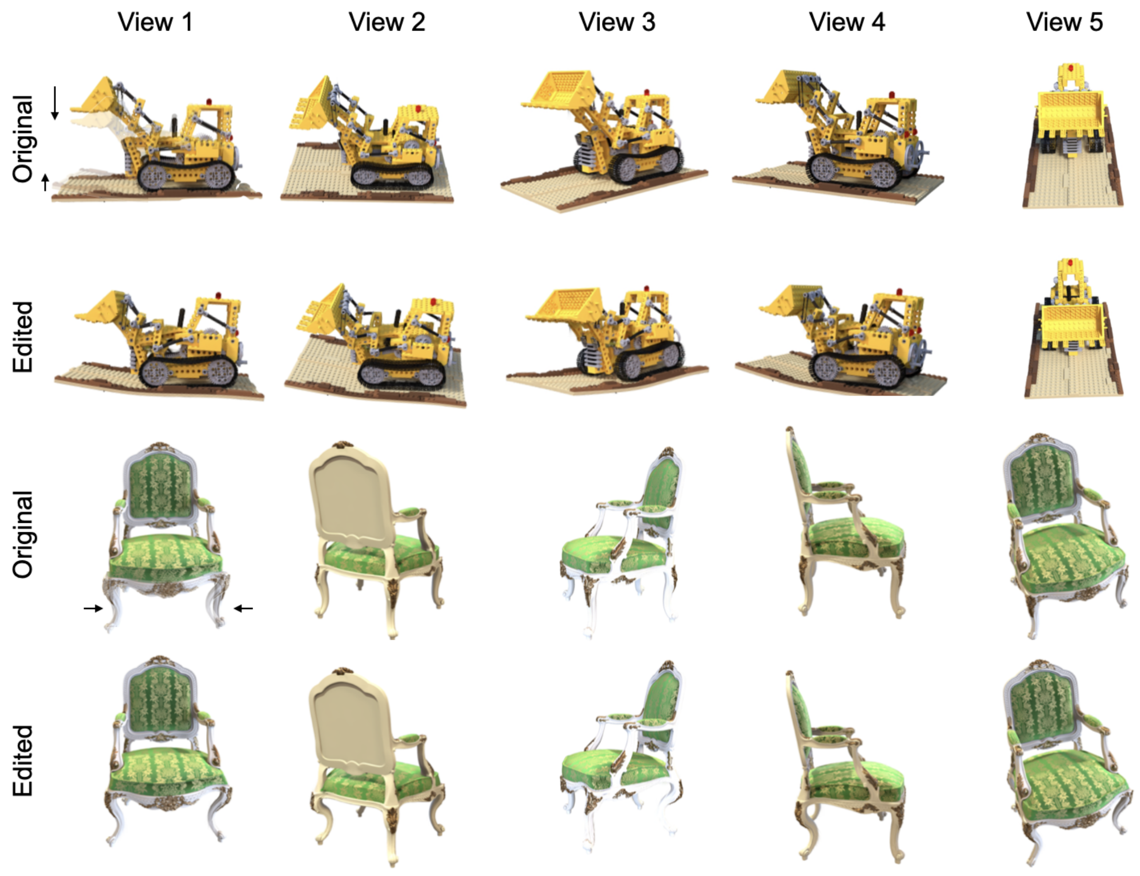

Figure 5.1: Comparison of Original rendering result and Edited rendering result on Lego bulldozer and a chair in NeRF Synthetic Dataset [7] (Lego: 14.9 FPS, Chair: 14.7 FPS)

To demonstrate the performance of our model, we utilized several datasets, including the Lego bulldozer and chair from the NeRF synthetic dataset [7], as well as scans 83 and 105 from the DTU dataset [12], and character from the Mixamo dataset [53]. Additionally, we tested our model on several custom dataset, which is generated with 100 random views from the upper hemisphere using Blender for training. Synthetic datasets were primarily used for both qualitative and quantitative experiments due to their ability to provide comprehensive views.

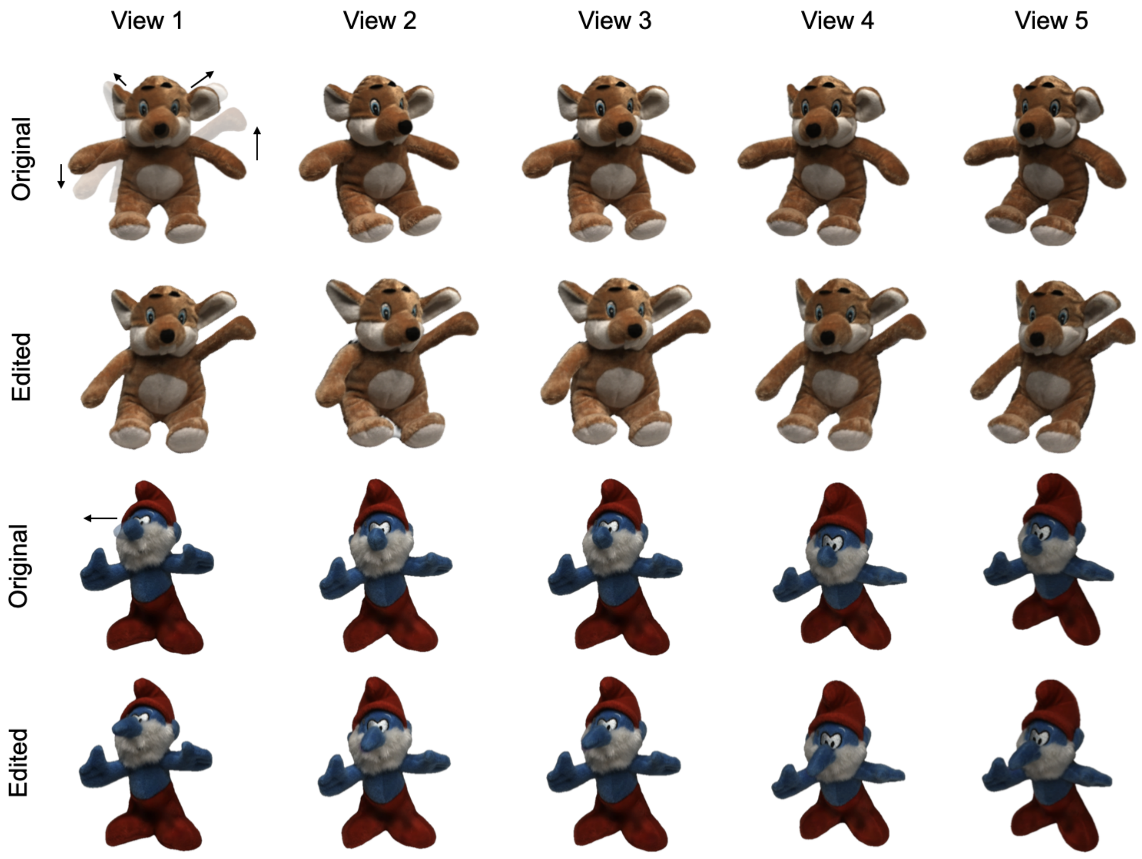

Figure 5.2: Comparison of Original rendering result and Edited rendering result on Scan 83 and 105 in DTU Dataset (Scan83: 14.0 FPS, Scan105: 14.6 FPS)[12]

Since there are no existing edited scenes, obtaining the ground truth for new view synthesis is challenging. To address this, we adopted the same quantitative evaluation method as NeRF Editing [9], incorporating character from the Mixamo [53]. This approach allowed us to generate ground truth for the edited scenes. We selected 100 random views for testing from the upper hemisphere of the edited ground truth of Mixamo. For quantitative evaluation, we assessed the model using the following metrics:

5.2.1 Structural Similarity Index Measure (SSIM)

The Structural Similarity Index Measure (SSIM) [54] is a perceptual metric used to evaluate the visual similarity between two images. It assesses differences in structural content, luminance, and contrast. SSIM between two images x and y is given by:

$$\mathrm{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

In this formula, μ_x and μ_y represent the mean intensities of images x and y, σ_x^2 and σ_y^2 denote their variances, and σ_{xy} is the covariance between them. The constants C_1 and C_2 are included to avoid instability when the denominator is close to zero. SSIM values range from -1 to 1, with a value of 1 indicating perfect structural similarity between the two images.

5.2.2 Learned Perceptual Image Patch Similarity (LPIPS)

The Learned Perceptual Image Patch Similarity (LPIPS) [55] metric quantifies the perceptual similarity between two images by comparing deep features extracted by a neural network. The LPIPS distance is computed as:

$$\mathrm{LPIPS}(x, y) = \sum_l \frac{1}{H_l W_l} \sum_{h,w} \|\hat{y}_l^h - \hat{x}_l^h\|_2^2$$

Here, $\hat{x}_l^h$ and $\hat{y}_l^h$ are the feature representations extracted from layer l of a pre-trained network, while H_l and W_l are the height and width of the feature map at that layer. A lower LPIPS value implies that the two images are more perceptually similar.

5.2.3 Peak Signal-to-Noise Ratio (PSNR)

Peak Signal-to-Noise Ratio (PSNR) is a commonly used metric for assessing the quality of image reconstructions. It is defined as:

$$\text{PSNR} = 10\log_{10}\left(\frac{L^2}{\text{MSE}}\right)$$

where L is the maximum possible pixel value (often 255 for 8-bit images), and MSE represents the Mean Squared Error between the original and reconstructed images. Higher PSNR values implies better image reconstruction quality.

Using these metrics, we were able to quantitatively evaluate the model's performance on the provided Mixamo models.

5.3 Qualitative Results

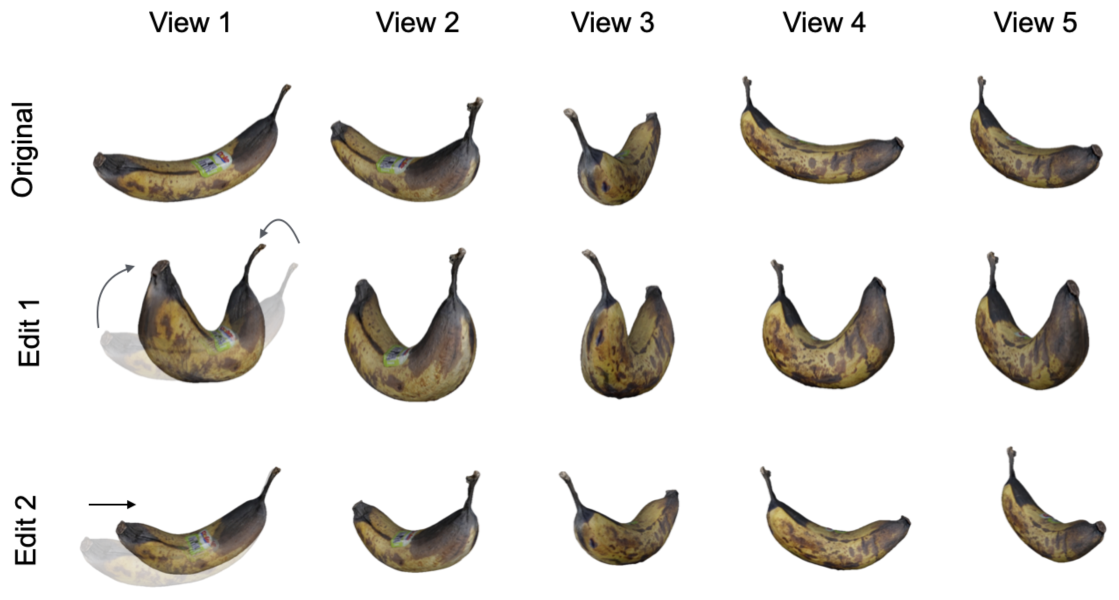

Figure 5.3: Comparison of Original rendering result and Edited rendering result on custom Dataset (14.7 FPS)

We compared the rendering results of 2D Gaussian Splatting with the edited scenes using GaussCraft from various viewpoints. Figure 5.1 shows the comparison using the Lego bulldozer and chair scenes from the NeRF synthesis dataset. For the Lego

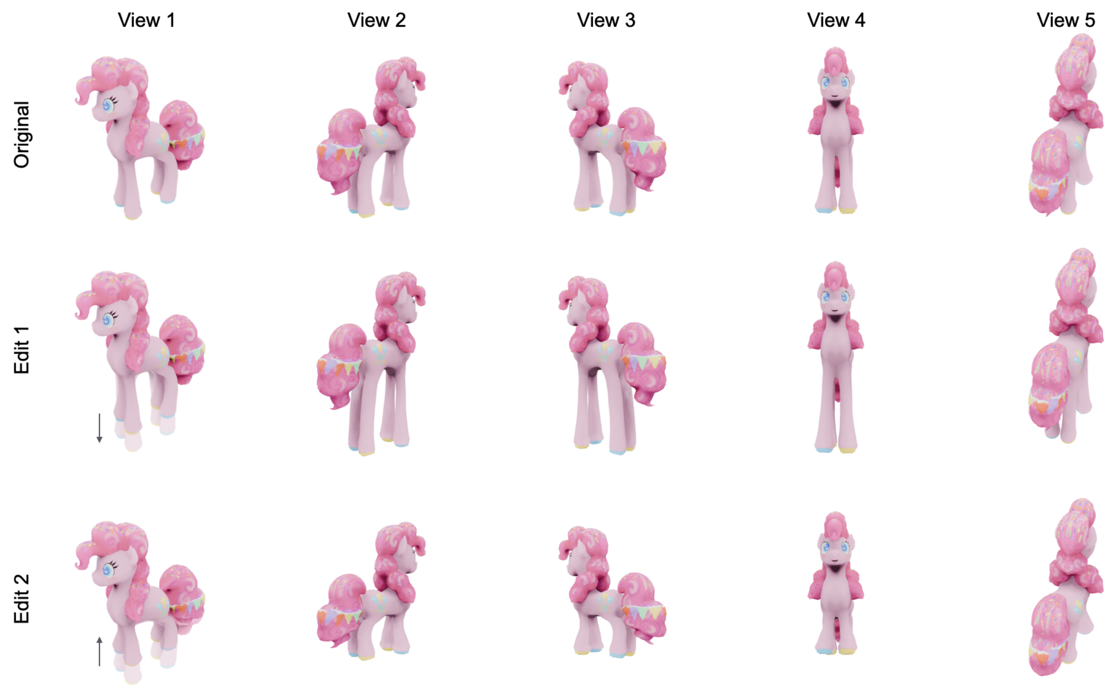

Figure 5.4: Comparison of Original rendering result and Edited rendering result on custom Dataset (15.1 FPS)

bulldozer, we lowered the bulldozer's shovel and demonstrated that the details are preserved as in the original scene. In the chair scene, we winded the legs of the chair, illustrating that our method works well across different types of edits.

Additionally, as illustrated in Figure 5.2, we compared two scenes from the DTU dataset. In the first scene, we stretched the ear and moved arms of a doll, and in the second scene, we stretched the doll's nose. Since the DTU dataset is generated by capturing real-world scenes, this comparison also verifies that our model performs well in real-world scenarios.

Finally, we compared results using a custom dataset in figure 5.3 and 5.4, with various edited scenes. These examples demonstrate that our model is versatile and capable of accommodating a wide range of edits, allowing users to customize scenes as they desire.

As shown in Figure 5.5, the reconstructed mesh from 2DGS with multi-view

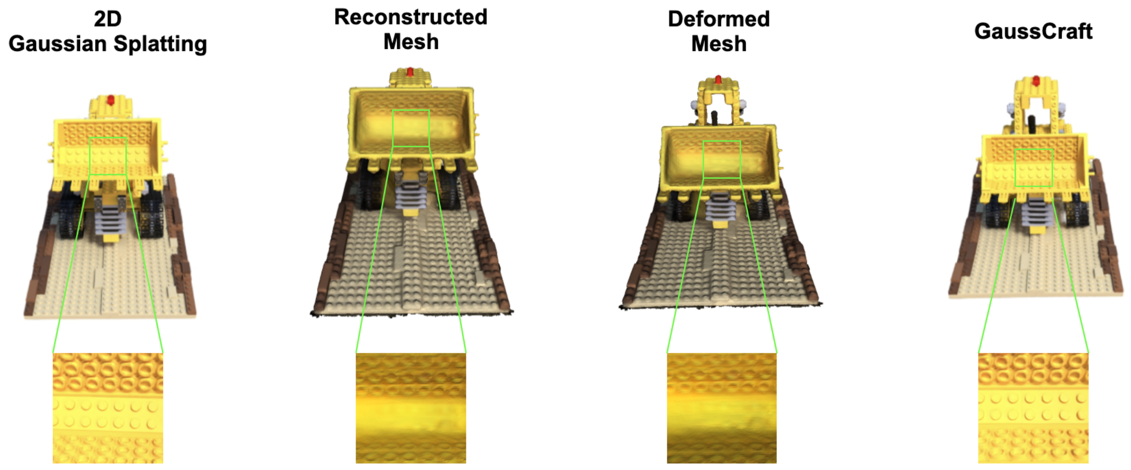

Figure 5.5: Difference of capturing detail between mesh and rendered scene.

images is not perfect. Despite these imperfections, users can still achieve reasonable performance in the edited scene by utilizing deformations of the mesh. This is because the mesh serves as an intermediate representation that enables interactive editing. Therefore, even if the mesh is not an exact representation, the model can still produce high-quality results. However, it is important to note that if the mesh reconstruction is too abstract, it may pose challenges for precise deformation.

5.4 Quantitative Results

Model	SSIM ↑	LPIPS ↓	PSNR ↑	Time ↓
GaussCraft	**0.9642**	**0.0292**	**24.1437**	**10m**
GaussCraft 3D	0.9640	0.0296	24.0084	10m 20s
NeRF-Editing	0.9463	0.0372	21.0037	16h

Table 5.1: Quantitative Comparison between various methods. Bold indicates the best performance

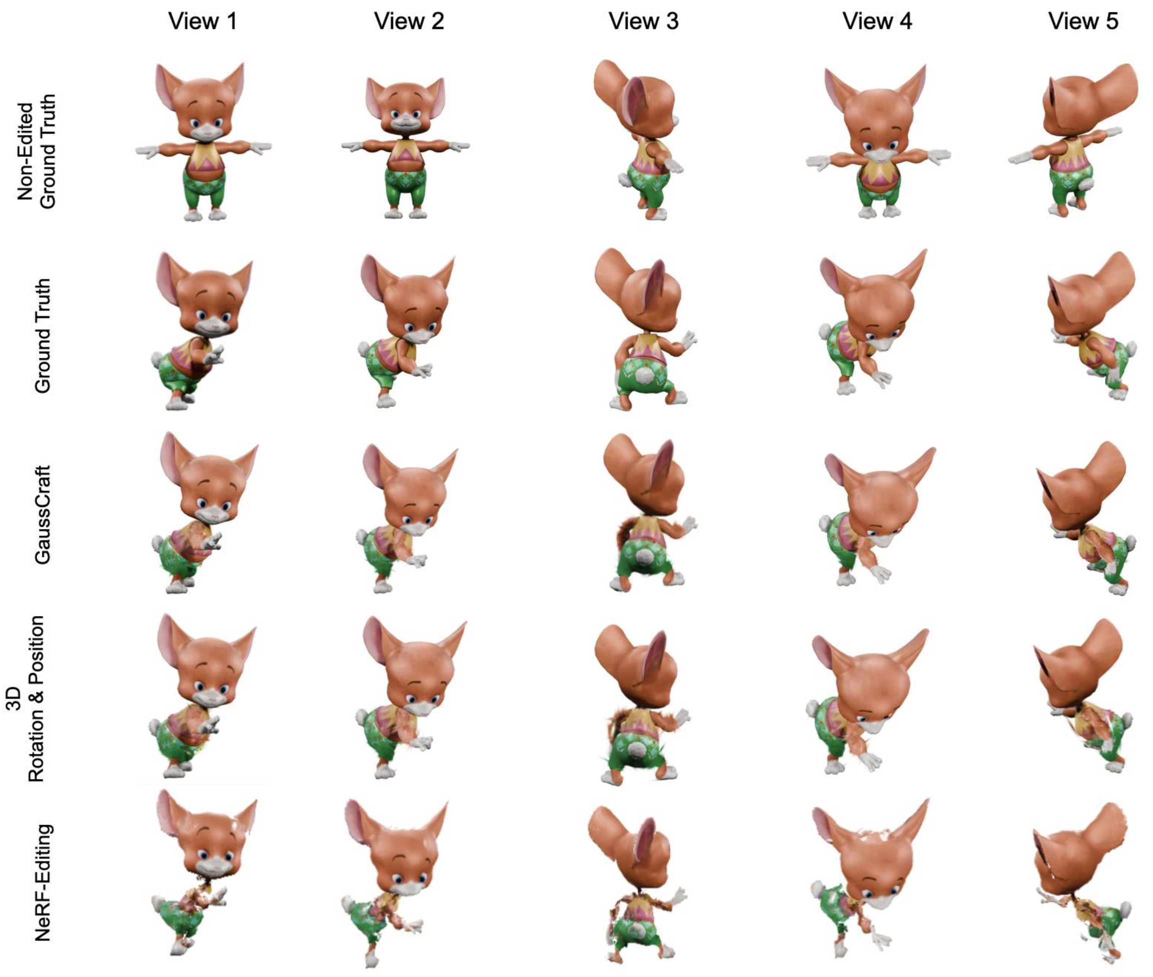

Figure 5.6: Visual comparisons of the Ground Truth and models

In our quantitative analysis, we employed two triangular meshes provided by Mixamo. One mesh was used during training instead of the reconstructed mesh from 2D Gaussian Splatting, while the other served as a stand-in for the deformed mesh. It is important to focus on that these provided meshes are simpler than the reconstructed mesh, primarily due to their lower vertex and face counts. This simplification may introduce additional artifacts in the final output compared to using a fully reconstructed mesh.

Given the limited prior work specifically addressing explicit representation or edit-

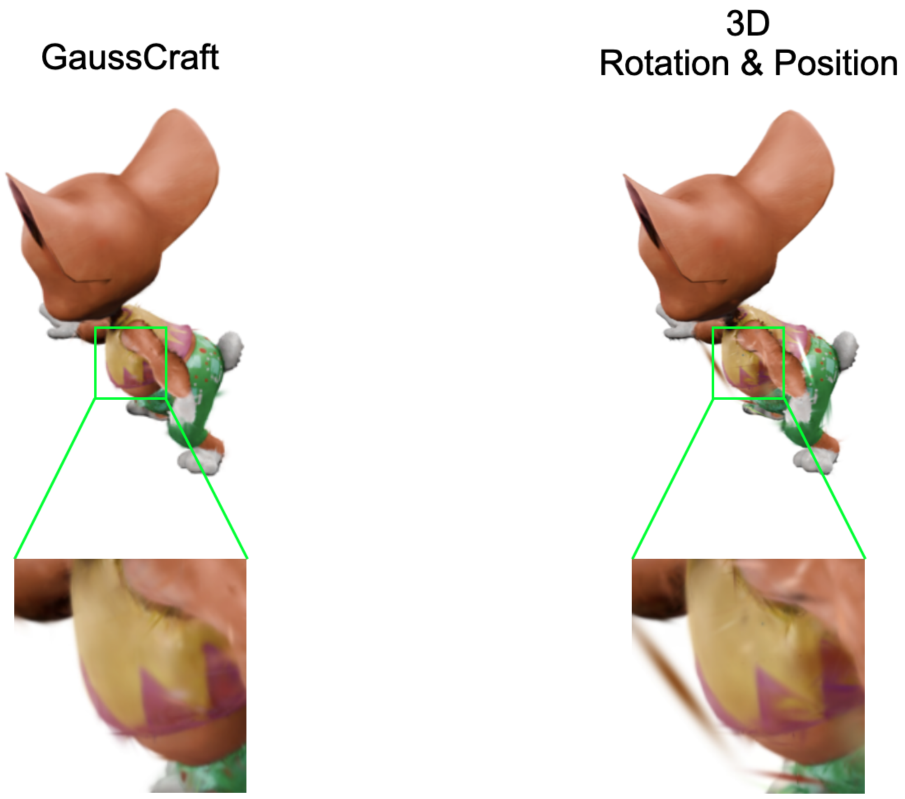

Figure 5.7: Zoom in Comparison between GaussCraft 3D and GaussCraft

ing using 2D/3D Gaussian splatting, our method introduces the first user-editable real-time rendering approach in this domain. To evaluate its performance, we compare our quantitative results with an alternative method that retains the z-axis in 2D Gaussian splatting within local space, which we refer to as "GaussCraft 3D." Additionally, we compare our approach to the NeRF-based implicit editing method, NeRF-editing [9]. For NeRF-editing, we trained for 300,000 iterations and used two meshes provided by Mixamo, rather than reconstructed meshes using the NeuS method [50], to ensure a fair comparison. As shown in Table 5.1, GaussCraft consistently outperforms the other methods across all metrics. This demonstrates that, with shorter training time and real-time rendering capabilities, it can achieves better quality results.

We also present qualitative results in Figures 5.6 and 5.7. Notably, in view 3 of Figure 5.6, GaussCraft produces fewer artifacts than GaussCraft 3D. Furthermore, a closer inspection in Figure 5.7 reveals more pronounced artifacts in GaussCraft 3D that are not well-aligned with the surface. These findings suggest that eliminating the z-axis in local scaling and rotation during 2D Gaussian splatting leads to better outcomes, with fewer artifacts.

5.5 Ablation Study

Model	**SSIM ↑**	**LPIPS ↓**	**PSNR ↑**
Ours	**0.9642**	<u>0.0292</u>	**24.1437**
w/o ADC	0.9586	0.0379	23.1135
w/o $\mathcal{L}_{\textbf{position}}$ & $\mathcal{L}_{\textbf{scale}}$	<u>0.9642</u>	0.0292	<u>24.1393</u>
w/o pruning	0.9641	**0.0291**	24.1328

Table 5.2: Ablation study on the performance of different model variants. Bold indicates the best performance, while underlining indicates the second-best performance.

In this ablation study, we systematically evaluate the impact of key components of our model by selectively removing them and observing the resulting changes in performance metrics.

Adaptive Density Control (ADC): When ADC is removed from GaussCraft, the model's ability to accurately represent complex regions, such as surfaces with intricate patterns, hair, or repetitive textures, is significantly impaired. As shown in Table 5.2, the absence of ADC results in a noticeable decline across all metrics, particularly in SSIM and LPIPS. This underscores the crucial role of ADC in capturing fine details and maintaining high visual quality.

Position and Scale Loss Functions: Next, we assess the impact of removing the position and scale loss functions, which are essential for preventing Gaussians from becoming excessively large or deviating from their intended positions. Although the model continues to produce reasonable results in trained scenes without these loss functions, the lack of constraints can lead to artifacts when scenes are modified, due to Gaussians that are either too large or poorly positioned. As indicated by the metrics in Table 5.2, this manifests as a slight degradation in performance, particularly in the PSNR metric, which measures the structural similarity of the images.

Pruning: Finally, we evaluate the effect of not pruning Gaussians that have moved too far from their expected locations. Without pruning, the model becomes less robust, as Gaussians that are misaligned contribute to artifacts, particularly in 2D representations. This is reflected in the slight decrease in SSIM and PSNR metrics compared to the complete GaussCraft model. The pruning process is therefore essential for maintaining the integrity of the rendered images, ensuring that Gaussians do not introduce unintended visual distortions.

Overall, these findings highlight the importance of adaptive density control, as well as the position and scale loss functions, in achieving high-quality, artifact-free renderings.

5.6 Limitations

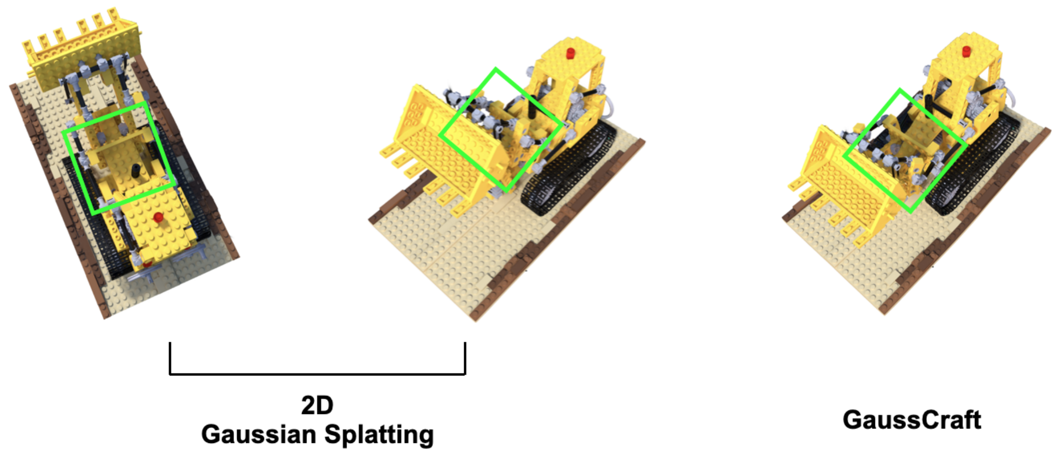

Figure 5.8: Lighting failure case

The limitations of our method can be categorized into two main areas: lighting and mesh issues.

First, our method is trained using multi-view images captured in a static lighting environment. As a result, all Gaussians are optimized based solely on the specific scene provided during training. Since our method binds Gaussians to the faces of the mesh, we do not account for changes in shadows or lighting based on edited results. As we can observe from figure 5.8, after editing the scene, shadow and lighting still remain same. To address this limitation, future work could explore enabling the model to perform relighting based on the edited scenes. Existing relighting techniques using Gaussian splatting [56, 57] could be integrated into our model to overcome this challenge.

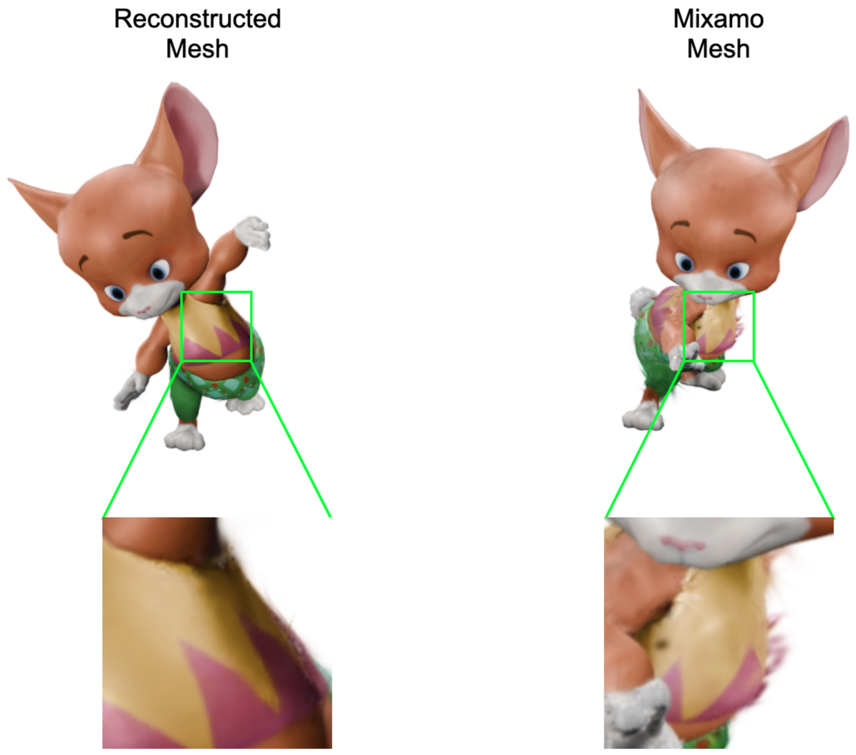

Figure 5.9: Result Comparison between two different meshes

Secondly, since this method binds 2D Gaussians to mesh faces, it can encounter challenges when the mesh is not well-defined. For example, if the reconstructed mesh has too few faces, even with adaptive density control, the model may fail to generate a sufficient number of Gaussians to support effective editing. This limitation is generally not an issue when working with meshes reconstructed from 2D Gaussian splatting. However, when using pre-existing meshes, such as those from Mixamo, artifacts may arise, resulting in an inability to accurately capture the edited scene. As shown in Figure 5.9, which compares results from training with a reconstructed mesh and a Mixamo-provided mesh (used for evaluation), there is a noticeable difference in quality. To address this, future work could focus on subdividing mesh faces, especially those with larger scales or more complex geometries, to ensure the mesh can better represent the scene after editing.

Chapter 6

Conclusion

In conclusion, this paper introduces GaussCraft, a pioneering approach in real-time scene editing that utilizes 2D Gaussian Splatting for high-quality, deformable mesh reconstruction. Our method distinguishes itself by simplifying local space dimensionality and incorporating user-defined editing capabilities, offering a more efficient and consistent alternative to traditional techniques. By omitting the z-axis in the local coordinate system, GaussCraft achieves faster optimizations and smoother results, making it particularly suitable for real-time applications. The integration of densification and pruning techniques further enhances the precision of mesh editing, ensuring high fidelity in the final output.

GaussCraft is applicable to both real and synthetic scenes for object editing, and it is highly adaptable to a wide range of scenarios. Compared to previous editing methods based on implicit representations [9], GaussCraft offers significantly faster training times, real-time rendering capabilities, and produces high-quality results. Its efficiency makes it well-suited for interactive applications, where users can quickly manipulate scenes without long processing delays.

Looking ahead, there are numerous directions for extending GaussCraft's functionality. Future work could explore the integration of relighting or color modification within the 2D Gaussian-based editing framework, allowing for more dynamic adjustments to lighting and color schemes directly within scenes. Additionally, GaussCraft's potential could be expanded by incorporating more advanced user interaction features. For instance, the model could serve as a real-time, interactive scene editor, enabling users to make immediate adjustments to objects within a scene.

Further possibilities include combining scene editing with style transfer techniques, allowing users to modify the visual appearance of objects by applying specific artistic styles. Another interesting future work is editable object manipulation within a single image instead of multi-view images, possibly through the use of transformer-based models, which would enable to generate multi-view images based on simgle image. Moreover, GaussCraft could be extended to support text-based scene editing, allowing users to describe desired modifications using natural language, further enhancing its ease of use and accessibility for a broader range of applications.

By building on these ideas, GaussCraft could evolve into a various tool for scene editing across many different domains, from game development and animation to AR/VR applications, providing users with a powerful, efficient, and flexible editing platform.

Bibliography

[1] G. Turk and M. Levoy, "The stanford bunny," https://faculty.cc.gatech.edu/~turk/bunny/bunny.html, 1994, accessed: 2024-08-23.

[2] J. Park, S. N. Sinha, Y. Matsushita, Y.-W. Tai, and I. S. Kweon, "Robust multiview photometric stereo using planar mesh parameterization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 8, pp. 1591–1604, 2017.

[3] E. Camuffo, D. Mari, and S. Milani, "Recent advancements in learning algorithms for point clouds: An updated overview," *Sensors*, vol. 22, p. 1357, 02 2022.

[4] P. Agarwal and B. Prabhakaran, "Robust blind watermarking of point-sampled geometry," *Information Forensics and Security, IEEE Transactions on*, vol. 4, pp. 36 – 48, 04 2009.

[5] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, "Occupancy networks: Learning 3d reconstruction in function space," 2019. [Online]. Available: https://arxiv.org/abs/1812.03828

[6] C. Moloney, "Ice: Fast volume emission using signed distance fields. and bunnies." https://blog.blackredking.org/?p=47, 2010, accessed: 2024-08-23.

[7] 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

[8] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," 2023. [Online]. Available: https://arxiv.org/abs/2308.04079

[9] 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. [Online]. Available: https://arxiv.org/abs/2205.04978

[10] B. Huang, Z. Yu, A. Chen, A. Geiger, and S. Gao, "2d gaussian splatting for geometrically accurate radiance fields," in *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers '24*, ser. SIGGRAPH '24. ACM, Jul. 2024. [Online]. Available: http://dx.doi.org/10.1145/3641519.3657428

[11] S. Qian, T. Kirschstein, L. Schoneveld, D. Davoli, S. Giebenhain, and M. Nießner, "Gaussianavatars: Photorealistic head avatars with rigged 3d gaussians," *arXiv preprint arXiv:2312.02069*, 2023.

[12] H. Aanæs, R. R. Jensen, G. Vogiatzis, E. Tola, and A. B. Dahl, "Large-scale data for multiple-view stereopsis," *International Journal of Computer Vision*, pp. 1–16, 2016.

[13] J. L. Schönberger, E. Zheng, J.-M. Frahm, and M. Pollefeys, "Pixelwise view selection for unstructured multi-view stereo," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 501–518.

[14] S. J. Garbin, M. Kowalski, M. Johnson, J. Shotton, and J. Valentin, "Fastnerf: High-fidelity neural rendering at 200fps," 2021. [Online]. Available: https://arxiv.org/abs/2103.10380

[15] P. Hedman, P. P. Srinivasan, B. Mildenhall, J. T. Barron, and P. Debevec, "Baking neural radiance fields for real-time view synthesis," 2021. [Online]. Available: https://arxiv.org/abs/2103.14645

[16] C. Reiser, S. Peng, Y. Liao, and A. Geiger, "Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps," 2021. [Online]. Available: https://arxiv.org/abs/2103.13744

[17] A. Yu, R. Li, M. Tancik, H. Li, R. Ng, and A. Kanazawa, "Plenoctrees for real-time rendering of neural radiance fields," 2021. [Online]. Available: https://arxiv.org/abs/2103.14024

[18] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, "Mip-nerf 360: Unbounded anti-aliased neural radiance fields," *CoRR*, vol. abs/2111.12077, 2021. [Online]. Available: https://arxiv.org/abs/2111.12077

[19] Z. Yu, A. Chen, B. Huang, T. Sattler, and A. Geiger, "Mip-splatting: Alias-free 3d gaussian splatting," 2023. [Online]. Available: https://arxiv.org/abs/2311.16493

[20] Y. Jiang, J. Tu, Y. Liu, X. Gao, X. Long, W. Wang, and Y. Ma, "Gaussianshader: 3d gaussian splatting with shading functions for reflective surfaces," 2023. [Online]. Available: https://arxiv.org/abs/2311.17977

[21] Y. Shi, Y. Wu, C. Wu, X. Liu, C. Zhao, H. Feng, J. Zhang, B. Zhou, E. Ding, and J. Wang, "Gir: 3d gaussian inverse rendering for relightable scene factorization," 2024. [Online]. Available: https://arxiv.org/abs/2312.05133

[22] Y. Yan, H. Lin, C. Zhou, W. Wang, H. Sun, K. Zhan, X. Lang, X. Zhou, and S. Peng, "Street gaussians: Modeling dynamic urban scenes with gaussian splatting," 2024. [Online]. Available: https://arxiv.org/abs/2401.01339

[23] W. Zielonka, T. Bagautdinov, S. Saito, M. Zollhöfer, J. Thies, and J. Romero, "Drivable 3d gaussian avatars," 2023. [Online]. Available: https://arxiv.org/abs/2311.08581

[24] T. Li, T. Bolkart, M. J. Black, H. Li, and J. Romero, "Learning a model of facial shape and expression from 4d scans," *ACM Trans. Graph.*, vol. 36, no. 6, nov 2017. [Online]. Available: https://doi.org/10.1145/3130800.3130813

[25] 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*, ser. SIGGRAPH ’00. USA: ACM Press/Addison-Wesley Publishing Co., 2000, p. 335–342. [Online]. Available: https://doi.org/10.1145/344779.344936

[26] M. Zwicker, H. Pfister, J. Baar, and M. Gross, “Ewa volume splatting,” *IEEE Visualization*, 01 2001.

[27] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger, “Elasticfusion: Real-time dense slam and light source estimation,” *The International Journal of Robotics Research*, vol. 35, no. 14, pp. 1697–1716, 2016. [Online]. Available: https://doi.org/10.1177/0278364916669237

[28] T. Schöps, T. Sattler, and M. Pollefeys, “Surfelmeshing: Online surfel-based mesh reconstruction,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 10, pp. 2494–2507, 2020.

[29] A. Eltner and G. Sofia, “Chapter 1 - structure from motion photogrammetric technique,” in *Remote Sensing of Geomorphology*, ser. Developments in Earth Surface Processes, P. Tarolli and S. M. Mudd, Eds. Elsevier, 2020, vol. 23, pp. 1–24. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780444641779000011

[30] O. Sorkine and M. Alexa, “As-rigid-as-possible surface modeling,” in *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, ser. SGP ’07. Goslar, DEU: Eurographics Association, 2007, p. 109–116.

[31] A. Chen, M. Wu, Y. Zhang, N. Li, J. Lu, S. Gao, and J. Yu, “Deep surface light fields,” *Proc. ACM Comput. Graph. Interact. Tech.*, vol. 1, no. 1, jul 2018. [Online]. Available: https://doi.org/10.1145/3203192

[32] G. Riegler and V. Koltun, “Stable view synthesis,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 12 216–12 225.

[33] R. Gernot and K. Vladlen, "Free view synthesis," *CoRR*, vol. abs/2008.05511, 2020. [Online]. Available: https://arxiv.org/abs/2008.05511

[34] S. Lombardi, T. Simon, J. M. Saragih, G. Schwartz, A. M. Lehrmann, and Y. Sheikh, "Neural volumes: Learning dynamic renderable volumes from images," *CoRR*, vol. abs/1906.07751, 2019. [Online]. Available: http://arxiv.org/abs/1906.07751

[35] V. Sitzmann, J. Thies, F. Heide, M. Nießner, G. Wetzstein, and M. Zollhöfer, "Deepvoxels: Learning persistent 3d feature embeddings," *CoRR*, vol. abs/1812.01024, 2018. [Online]. Available: http://arxiv.org/abs/1812.01024

[36] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, 2011, pp. 127–136.

[37] D. Meagher, "Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer," 10 1980.

[38] E. Kawaguchi and T. Endo, "On a method of binary-picture representation and its application to data compression," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-2, pp. 27–35, 1980. [Online]. Available: https://api.semanticscholar.org/CorpusID:1953670

[39] K. Aliev, D. Ulyanov, and V. S. Lempitsky, "Neural point-based graphics," *CoRR*, vol. abs/1906.08240, 2019. [Online]. Available: http://arxiv.org/abs/1906.08240

[40] P. Dai, Y. Zhang, Z. Li, S. Liu, and B. Zeng, "Neural point cloud rendering via multi-plane projection," 2020. [Online]. Available: https://arxiv.org/abs/1912.04645

[41] W. E. Lorensen and H. E. Cline, “Marching cubes: A high resolution 3d surface construction algorithm,” *SIGGRAPH Comput. Graph.*, vol. 21, no. 4, p. 163–169, aug 1987. [Online]. Available: https://doi.org/10.1145/37402.37422

[42] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen, *The Lumigraph*, 1st ed. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3596711.3596760

[43] G. Chaurasia, S. Duchene, O. Sorkine-Hornung, and G. Drettakis, “Depth synthesis and local warps for plausible image-based navigation,” *ACM Trans. Graph.*, vol. 32, no. 3, jul 2013. [Online]. Available: https://doi.org/10.1145/2487228.2487238

[44] M. Eisemann, B. Decker, M. Magnor, P. Bekaert, E. Aguiar, N. Ahmed, C. Theobalt, and A. Sellent, “Floating textures,” *Computer Graphics Forum*, vol. 27, pp. 409 – 418, 04 2008.

[45] P. Hedman, J. Philip, T. Price, J.-M. Frahm, G. Drettakis, and G. Brostow, “Deep blending for free-viewpoint image-based rendering,” *ACM Trans. Graph.*, vol. 37, no. 6, dec 2018. [Online]. Available: https://doi.org/10.1145/3272127.3275084

[46] G. Kopanas, J. Philip, T. Leimkühler, and G. Drettakis, “Point-based neural rendering with per-view optimization,” *CoRR*, vol. abs/2109.02369, 2021. [Online]. Available: https://arxiv.org/abs/2109.02369

[47] A. Tewari, J. Thies, B. Mildenhall, P. Srinivasan, E. Tretschk, Y. Wang, C. Lassner, V. Sitzmann, R. Martin-Brualla, S. Lombardi, T. Simon, C. Theobalt, M. Niessner, J. T. Barron, G. Wetzstein, M. Zollhoefer, and V. Golyanik, “Advances in neural rendering,” 2022. [Online]. Available: https://arxiv.org/abs/2111.05849

[48] C. Lassner and M. Zollhöfer, “Pulsar: Efficient sphere-based neural rendering,” 2020. [Online]. Available: https://arxiv.org/abs/2004.07484

[49] D. Rückert, L. Franke, and M. Stamminger, "Adop: approximate differentiable one-pixel point rendering," *ACM Trans. Graph.*, vol. 41, no. 4, jul 2022. [Online]. Available: https://doi.org/10.1145/3528223.3530122

[50] P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura, and W. Wang, "Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction," *CoRR*, vol. abs/2106.10689, 2021. [Online]. Available: https://arxiv.org/abs/2106.10689

[51] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3d: A modern library for 3d data processing," 2018. [Online]. Available: https://arxiv.org/abs/1801.09847

[52] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017. [Online]. Available: https://arxiv.org/abs/1412.6980

[53] Mixamo, "Mixamo - 3d characters, animations, and models for games, vr, ar, and more," https://www.mixamo.com/, 2024, accessed: 2024-08-29.

[54] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[55] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 586–595.

[56] S. Saito, G. Schwartz, T. Simon, J. Li, and G. Nam, "Relightable gaussian codec avatars," 2024. [Online]. Available: https://arxiv.org/abs/2312.03704

[57] J. Gao, C. Gu, Y. Lin, Z. Li, H. Zhu, X. Cao, L. Zhang, and Y. Yao, "Relightable 3d gaussians: Realistic point cloud relighting with brdf decomposition and ray tracing," 2024. [Online]. Available: https://arxiv.org/abs/2311.16043