Interactive Geometry Editing of Neural Radiance Fields via Tetrahedral Mesh Deformation

Logan Dooley Rafael Hajjar

December 2025

Abstract

This paper presents a methodology for extending Neural Radiance Fields (NeRFs) to support geometric deformation without the need for network re-training. Building upon the foundational principles of "NeRF-Editing" (Yuan et al., 2022), we propose a pipeline that extracts an explicit mesh representation from an implicit field, applies tetrahedralization, and utilizes mesh deformation algorithms such as As-Rigid-As-Possible (ARAP). By calculating barycentric transformation matrices between the original and deformed states, we enable a ray-bending rendering approach that accurately reflects geometric edits. We demonstrate the efficacy of this pipeline on real-world scenes, address challenges regarding mesh noise, and discuss the limitations regarding temporal consistency in animation sequences.

1 Introduction

Since their introduction in 2020, Neural Radiance Fields (NeRFs) have become a dominant paradigm in computer vision and graphics for novel view synthesis. Unlike Gaussian Splatting, NeRFs utilize a volumetric representation, allowing them to accurately model complex physical media, such as semi-transparent objects or clouds. Furthermore, as an implicit representation, NeRFs avoid the level-of-detail gaps often observed in Gaussian Splatting when zooming in closely on a subject.

Despite these advantages, editing a pre-trained NeRF remains a challenge due to the implicit nature of the network weights. Recent research has explored modifying these models through color painting and geometry editing. This work focuses on recreating and extending geometry editing capabilities. Such tooling is critical for content creation, enabling applications ranging from animating static scans to architectural visualization where users might wish to alter object dimensions in a photorealistic scene.

While Large Language Models (LLMs) have shown promise in generative editing, editing via direct geometric manipulation of a NeRF offers distinct advantages. primarily, the output is deterministic with respect to the pre-trained model. Once a NeRF is trained, objects can be morphed without introducing hallucinations or artifacts common in generative approaches. Additionally, this method provides granular artistic control, integrating well with established computational geometry algorithms.

2 Methodology

Our proposed pipeline consists of six distinct stages, moving from implicit training to explicit manipulation and back to implicit rendering.

2.1 NeRF Training

We utilize the InstantNGP architecture for its computational efficiency. The model is pre-trained on the target scene to establish the baseline density and color fields required for subsequent geometry manipulation.

2.2 Mesh Extraction

To bridge the gap between implicit and explicit representations, we extract a mesh from the NeRF. Since the model outputs both color and density for spatial points, we employ the Marching Cubes algorithm. By defining a density threshold, we determine isosurfaces and construct a triangle mesh representing the scene geometry.

2.3 Tetrahedralization

The extracted surface mesh is insufficient for volumetric deformation. Therefore, we extend the representation into the volumetric domain using Delaunay triangulation to generate a tetrahedral mesh. This allows us to map the entire volume of the scene, not just the surface.

2.4 Mesh Deformation

We explore two categories of deformation:

1. **Geometric Transformations:** Simple scaling and translation of selected vertices to test pipeline validity.
2. **As-Rigid-As-Possible (ARAP):** A sophisticated energy minimization algorithm. ARAP allows a user to manipulate a subset of "handle" points while the rest of the mesh deforms naturally to preserve local rigidity. This creates intuitive, soft-body-like deformations.

2.5 Barycentric Transformation Matrices

To map points between the deformed space and the original canonical space, we compute barycentric transformation matrices for each tetrahedron. For a tetrahedron defined by vertices a, b, c, d, the transformation matrix is constructed as:

$$T = \begin{bmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \\ a_z & b_z & c_z & d_z \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{1}$$

This matrix transforms points from barycentric coordinates to world space coordinates, allowing us to compute the inverse mapping required for rendering.

2.6 Re-rendering via Ray Bending

The final rendering step utilizes ray marching. For every sample point along a ray in the deformed space, we identify the enclosing tetrahedron in the deformed mesh. Using the barycentric coordinates, we map this sample back to the original tetrahedral mesh. The density and color values are then queried from the pre-trained NeRF at this "undeformed" location. This technique effectively bends the rays of the NeRF to match the explicit mesh deformation.

3 Implementation and Iterations

3.1 Baseline and Initial Deformation

Our initial baseline involved applying basic linear transformations to the NeRF model. Figure 1 demonstrates a "widening" operation applied to a truck scene.

Figure 1: Comparison of the original NeRF render (left) and the geometrically widened re-render (right).

3.2 ARAP Validation on Primitives

To validate the As-Rigid-As-Possible algorithm, we first applied it to a simple cube mesh. By constraining the bottom-left vertex and manipulating the top-right, we confirmed the solver's ability to minimize local distortion (Figure 2).

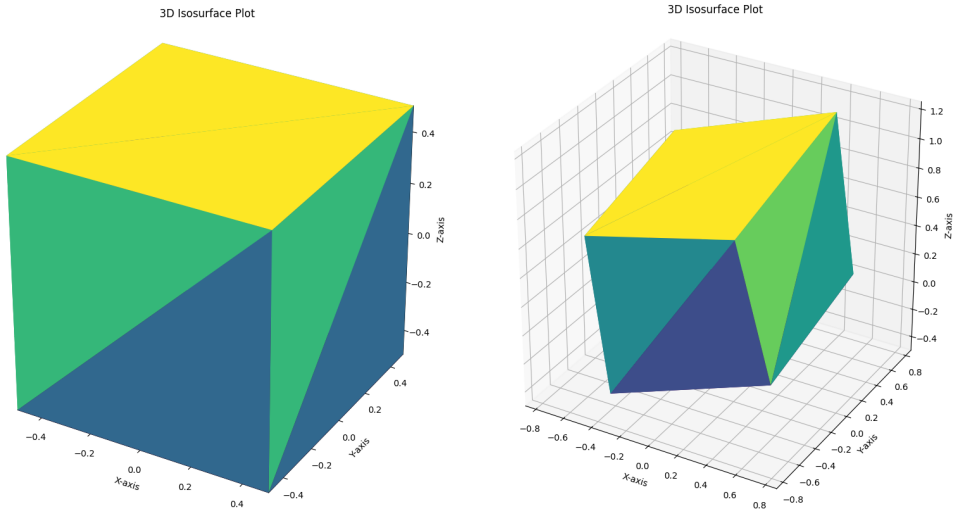

Figure 2: Original cube mesh (left) and ARAP deformation (right).

3.3 Mesh Filtering for NeRF Geometry

A significant challenge encountered was the noise inherent in meshes extracted via Marching Cubes from NeRFs. The raw meshes contained numerous isolated, small disconnected components which caused the ARAP solver to fail.

To resolve this, we implemented a filtering step to remove disconnected components below a volume threshold, isolating the primary object. Figure 3 illustrates the raw versus filtered mesh.

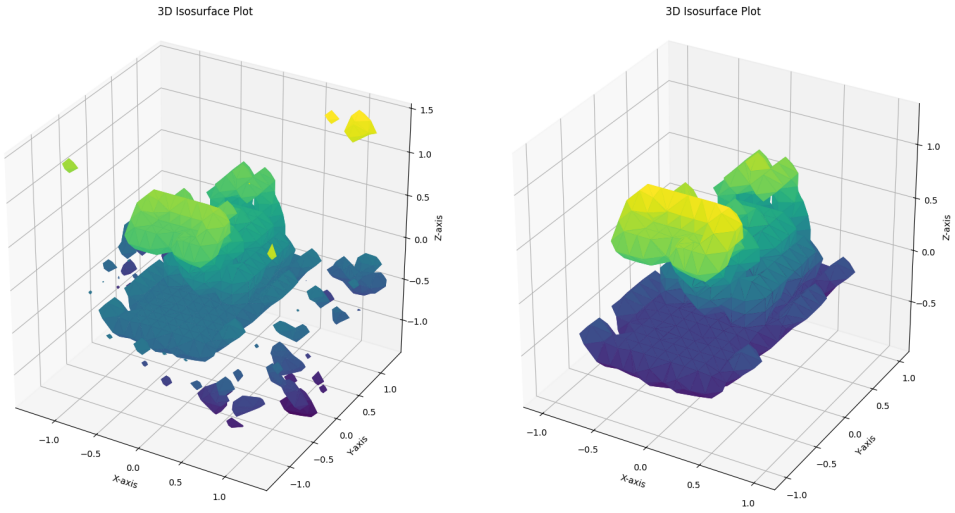

Figure 3: Raw Marching Cubes output (left) vs. filtered mesh suitable for ARAP (right).

3.4 Complex Deformation Results

Upon successfully filtering the mesh, we applied ARAP deformations to the NeRF scene. In the example shown in Figure 4, the bottom of the object was fixed while a point on the basket was pulled upwards. This deformation was then passed through the re-rendering pipeline (Figure 5).

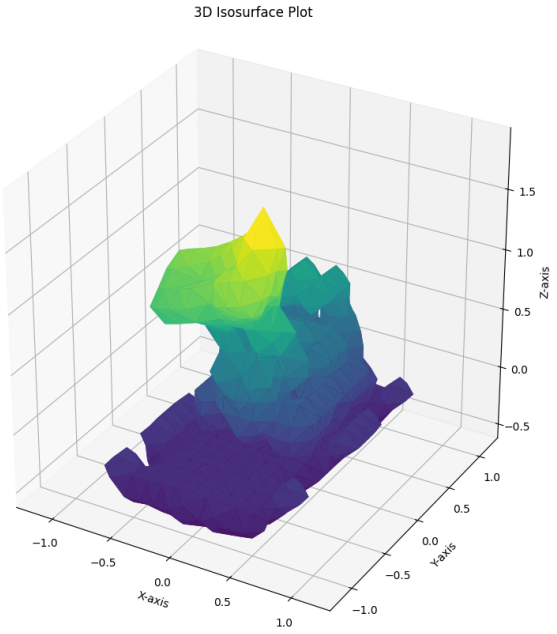

Figure 4: Visualization of the tetrahedral mesh deformed via ARAP.

Figure 5: Original NeRF (left) vs. ARAP deformed re-render (right), showing non-rigid deformation.

3.5 Temporal Animation Challenges

We attempted to generate an animation (GIF) of the deformation process. However, we observed significant artifacting (Figures 6, 7). This is attributed to inconsistent tetrahedralization between frames and potential tetrahedron inversion, which disrupts the barycentric mapping logic.

Figure 6: Sequential frames 1-3 of the deformation, showing artifact onset.

Figure 7: Sequential frames 4-5 showing significant rendering artifacts.

4 Discussion and Future Work

The results demonstrate that explicit mesh deformation can be successfully coupled with implicit NeRF rendering to achieve deterministic geometry editing. However, the reliance on high-quality mesh extraction is a limiting factor; manual filtering was required to make the ARAP solver viable. Future iterations should explore automated mesh cleaning or extraction methods that are robust to the noise inherent in density fields.

Regarding the temporal artifacts observed in animation attempts, future work must focus on maintaining topological consistency. Instead of re-tetrahedralizing at every frame, the system should compute the tetrahedralization once and update vertex positions within the existing simplices. Additionally, handling tetrahedron inversion is critical for robust rendering.

Performance remains an area for optimization. Currently, the identification of the containing tetrahedron for each ray sample occurs on the CPU, creating a bottleneck. Porting the tetrahedral search and barycentric computation to CUDA would significantly accelerate the pipeline, potentially enabling real-time deformation. Furthermore, integrating skeletal rigging systems could provide more intuitive control for character animation within NeRF scenes compared to general soft-body ARAP.