

# D-MiSo: Editing Dynamic 3D Scenes using Multi-Gaussians Soup

**Joanna Waczyńska\***<sup>†</sup>

Doctoral School of Exact and Natural Sciences Faculty of Mathematics and Computer Science  
Jagiellonian University Jagiellonian University

**Piotr Borycki\***

**Joanna Kaleta**

Warsaw University of Technology  
Sano Centre for Computational Medicine

**Sławomir Tadeja**

Department of Engineering  
University of Cambridge

**Przemysław Spurek**

Jagiellonian University  
IDEAS NCBR

## Abstract

Over the past years, we have observed an abundance of approaches for modeling dynamic 3D scenes using Gaussian Splatting (GS). Such solutions use GS to represent the scene's structure and the neural network to model dynamics. Such approaches allow fast rendering and extracting each element of such a dynamic scene. However, modifying such objects over time is challenging. SC-GS (Sparse Controlled Gaussian Splatting) enhanced with Deformed Control Points partially solves this issue. However, this approach necessitates selecting elements that need to be kept fixed, as well as centroids that should be adjusted throughout editing. Moreover, this task poses additional difficulties regarding the re-productivity of such editing. To address this, we propose **Dynamic Multi-Gaussian Soup** (D-MiSo)<sup>3</sup>, which allows us to model the mesh-inspired representation of dynamic GS. Additionally, we propose a strategy of linking parameterized Gaussian splats, forming a Triangle Soup with the estimated mesh. Consequently, we can separately construct new trajectories for the 3D objects composing the scene. Thus, we can make the scene's dynamic editable over time or while maintaining partial dynamics.

## 1 Introduction

Recently introduced Gaussian Splatting (GS) [1] represents the 3D scene structure through Gaussian components. We can combine GS with the neural network (i.e., deform network) to model dynamic scenes [2]. This approach involves the joint training of both the GS components and the neural network. GS characterizes the 3D object's shape and color, while the neural network utilizes time embedding and Gaussian parameters to generate updated initial positions to model dynamic scenes. Such an approach allows for fast rendering and extracting each element of a dynamic scene.

Most existing methods can effectively model dynamic scenes, but generating new 3D objects'

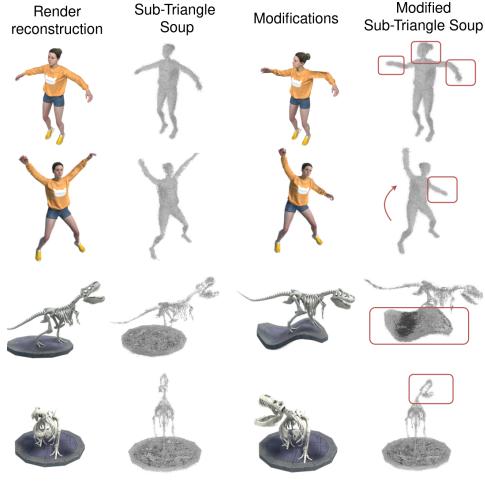


Figure 1: D-MiSo model parameterized dynamic scenes by Triangle Soup (disjoint triangles cloud), which allows modification of objects during time.

\*Equal contribution

<sup>†</sup>[joanna.waczynska@doctoral.uj.edu.pl](mailto:joanna.waczynska@doctoral.uj.edu.pl)

<sup>3</sup><https://github.com/waczjoan/D-MiSo>

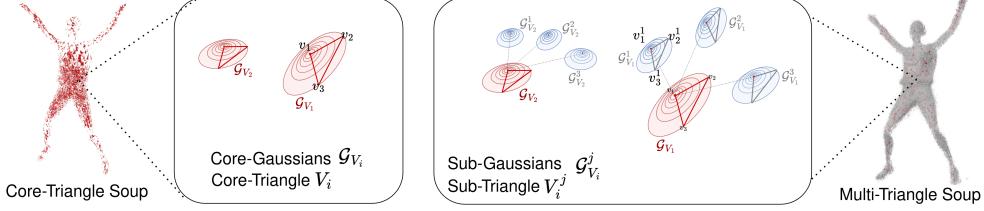


Figure 2: Each object using the D-MiSo model is represented by Core-Gaussians and Sub-Gaussians, which form Multi-Gaussians. Each Gaussian is related to a triangle using parameterization proposed in GaMeS [3]. Triangles define the Gaussian shape (i.e., location, scale, rotation), and triangles clouds form Triangles Soups.

positions remains challenging. Consequently, we cannot edit objects over time when using such approaches. To tackle this issue, SC-GS (Sparse Controlled Gaussian Splatting) [4] uses Deformed Control Points to manage Gaussians. After the training phase, we can manually modify the model at any point in time. However, this method requires identifying elements to remain static and adjusting 3D objects’ centroids (nodes) during editing when a relationship between the selected nodes is visible. For example, by moving the humanoid 3D model’s hand, the part of the head or leg is also changed.

To address this issue, we introduce **Dynamic Multi-Gaussian Soup** (D-MiSo), which is easier to modify (see Fig. 1) and obtain renders comparable to SC-GS. D-MiSo estimates the mesh as Triangle Soup, i.e. a set of disconnected triangle faces [5, 6], and uses a dynamic function to control the vertices.

Thanks to using Triangle Soup, we control two types of Gaussian components. D-MiSo employing Multi-Gaussians, defined as larger Core-Gaussians encompassing smaller ones termed Sub-Gaussians (Fig. 2). Sub-Gaussians are defined in the local coordinate system given by principal components of Core-Gaussian. Therefore, by modifying Core-Gaussian, we change all Sub-Gaussians, which allows scene modifications (Fig 3). Core-Gaussians are an alternative to the control points discussed in [4], with the added advantage of allowing individual modifications. Consequently, there is no necessity for static and dynamic markers. In Fig. 4 we present the difference between modification applied by SC-GS and D-MiSo. In D-MiSo, we can select and modify one part of the object like a mesh. In contrast, we must select static and dynamic points when using SC-GS, but editing only one part of the object is difficult.

In D-MiSo, we use flat Gaussians from GaMeS [3], which can be parametrized by triangle faces. Consequently, we obtain two Triangle Soups as shown in Fig. 2 where Core-Triangle Soup is marked by red color, and Sub-Triangle Soup is denoted in blue. During training, the positions of Core-Gaussians are managed by deformation MLP, while the Sub-Gaussians are collectively manipulated through global transformation and small local deformation. The former describes the general flow of objects in the scene, while the local deformation is responsible for modeling small changes like shadows and light reflections. After training, we can modify our model directly by using the vertex of the Sub-Triangle Soup, or we can generate mesh from the Core-Triangle Soup (Fig. 5).

The contributions of this paper are significant and are outlined as follows:

- We introduce the Multi-Gaussian components, which consist of a single large Gaussian response for global transformations and many small components dedicated to rendering. Multi-Gaussian components allow for the modeling of large 3D scene elements.

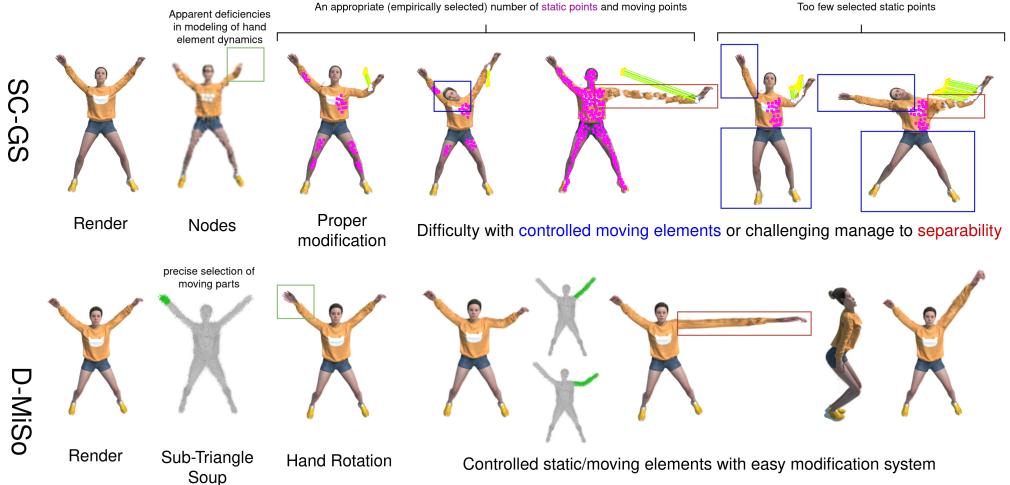


Figure 4: Comparison of possible modifications in D-MiSo and the SC-GS. In the latter, authors use nodes while D-MiSo apply Sub-Triangle Soup (see the second column). We also must add static (pink) and dynamic (yellow) points in SC-GS to obtain modification by editing dynamic points. In practice, we have to use many static points to stop artifacts. Moreover, SC-GS is not an affine invariant and produces space when we change the size of the objects. In the case of D-MiSo, we marked points and applied modifications. Our model is superior in handling object scaling.

- We propose D-MiSo a model that uses Multi-Gaussian components and two deformation networks for modeling dynamic scenes.
- Our D-MiSo allows an object to be edited at a selected moment in time. The edited components are independent, and the editing does not affect other parts of the object. In addition, it also allows for full or partial dynamics to be maintained. Modifications also include scaling and rotation.

## 2 Related Works

Recent advancements in view synthesis, particularly driven by NeRFs [7], have significantly contributed to the rapid development of novel view synthesis techniques. However, majority of these approaches model static scenes implicitly using multilayer perceptron (MLP). Moreover, several works have extended classical NeRF to dynamic scenes through the use of deformation fields [8, 9, 10] and [11]. The alternative approaches, such as [12] and [13], represent scenes as 4D radiance fields. However, NeRF-based solutions often suffer from long training and rendering times. To address this, grid-plane-based methods [14, 15, 16] have been proposed. In addition, several NeRF-based approaches have also been extended for scene editing purposes [17, 18, 19, 20].

The recently introduced Gaussian Splatting technique [1] addresses many limitations of other methods, offering multiple advantages due to their explicit geometry representation, enabling easier dynamics modeling. The efficient rendering of 3D GS also avoids densely sampling and querying neural fields, making downstream applications such as free-viewpoint video reconstruction more feasible. While the original GS was developed for static scenes, several extensions for dynamic scenes were proposed. For example, [21] utilizes a multiview dynamic dataset and a frame-by-frame approach to model such scenes. However, this method lacks inter-frame correlation and requires high storage overhead for long-term sequences. In [22, 2] MLP is introduced to model changes in Gaussians over time, and in [23] MLP together with decomposed neural voxel encoding algorithm are utilized for training and storage efficiency. In [24], dynamic scenes are divided into dynamic and static parts, optimized separately and rendered together to achieve decoupling. Other works enhance dynamic scene reconstruction using external priors. For example, the diffusion priors can be used as regularization terms during optimization [25].

Furthermore, GS was employed for mesh-based scene geometry editing. In [26] 3D Gaussians are defined over an explicit mesh and utilizes mesh rendering to guide adaptive refinement. This approach depends on the extracted mesh as a proxy and fails if the mesh cannot be extracted. In contrast, in [27] explicit meshes are extracted from 3D GS representations by regularizing Gaussians over surfaces. However, this method involves a costly optimization and refinement pipeline. Another

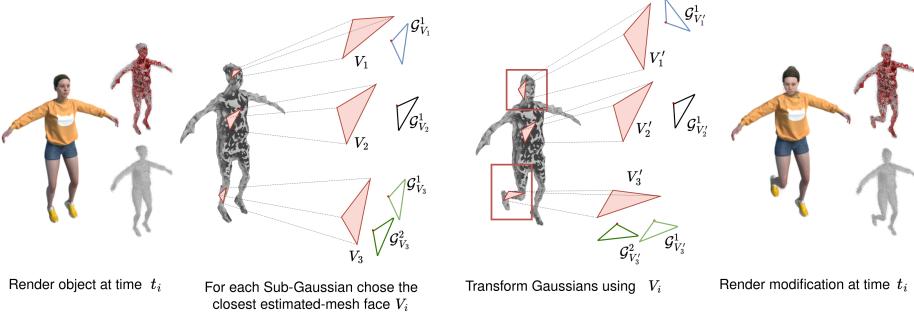


Figure 5: One way to modify the object at the selected time  $t_i$  is to take Core-Gaussians and apply a meshing strategy to obtain the correct mesh instead of Triangle Soup. Then, we can parametrize Sub-Gaussian in the coordinate system given by mesh faces instead of Core-Triangle Soup. Finally, we can modify our mesh to obtain new modifications.

example of [28] employs sparse control points for 3D scene dynamics, but this method struggles with intense edit movements and necessitates accurate static node selection. Also, [3] combines GS with mesh extraction. However, it only works for static scenes. In contrast to all these approaches, we propose a D-MiSo, a mesh-based method specifically designed to handle dynamic scenes. D-MiSo leverages a straightforward pipeline of GS techniques to enable real-time editing of dynamic scenes.

### 3 Dynamic Multi-Gaussian Soup

Here we present the main components of D-MiSo. We start with the classical GS to provide the foundations for our model. Next, we introduce the concept of Multi-Gaussians and present how to estimate the mesh for editing. Finally, we show D-MiSo, which uses Multi-Gaussians in dynamic 3D scenes.

**Gaussian Splatting** The Gaussian Splatting (GS) technique models 3D scenes using an array of 3D Gaussians, each specified by its mean position, covariance matrix, opacity, and color expressed using spherical harmonics (SH) [29, 30]. The GS algorithm constructs the radiance field by iteratively optimizing the parameters of all Gaussian components. Ultimately, the GS efficiency mainly depends on its rendering method, which involves projecting Gaussian components.

The GS framework employs a dense collection of 3D Gaussians:  $\mathcal{G} = \{(\mathcal{N}(\mathbf{m}_i, \Sigma_i), \sigma_i, c_i)\}_{i=1}^n$ , where  $\mathbf{m}_i$  denotes the position,  $\Sigma_i$  the covariance,  $\sigma_i$  the opacity, and  $c_i$  the SH colors for the  $i$ -th Gaussian. The GS optimization process involves a repetitive cycle of rendering and comparing the resultant images with the training views. In our work, we will use Multi-Gaussian approaches (Fig. 6).

**Multi-Gaussians** Multi-Gaussians  $\mathcal{G}_{multi}$  are dedicated to describing relatively large parts of the 3D scene to allow modification of large blocks instead of modifying each small Gaussian separately. The Multi-Gaussian model comprises a primary large 3D Gaussian (referred to as the Core-Gaussian  $\mathcal{G}_{core}$ ), which encompasses numerous smaller Gaussians (termed Sub-Gaussians  $\mathcal{G}_{sub}$ ), all of which are parameterized by the main Core-Gaussian. Multi-Gaussian are similar to anchor Gaussians from [31], but we do not use a neural network to produce child components. We parametrize Sub-Gaussians in a local coordinate system.

Similarly to classical GS, we parameterize the Core-Gaussian distribution by center  $\mathbf{m}$  and the covariance parameterized by factorization:  $\Sigma = RSSR^T$ , where  $R$  is the rotation matrix and  $S$  the scaling parameters. More precisely we consider  $p$  Core-Gaussians uses flat Guassinas as in [3], and

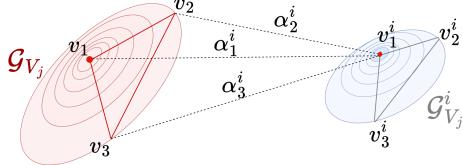


Figure 6: Multi-Gaussian, consisting of one Core-Gaussian  $\mathcal{G}_{V_j}$  and a Sub-Gaussian  $\mathcal{G}_{V_j}^i$ . The Core Gaussian is parametrized by a  $V_j$ -triangle, and the Sub-Gaussian by a  $V_j^i$ -triangle. The relative distance of the center of the Sub-Gaussian from the Core-Gaussian is indicated by  $\alpha^i = (\alpha_1^i, \alpha_2^i, \alpha_3^i)$ .

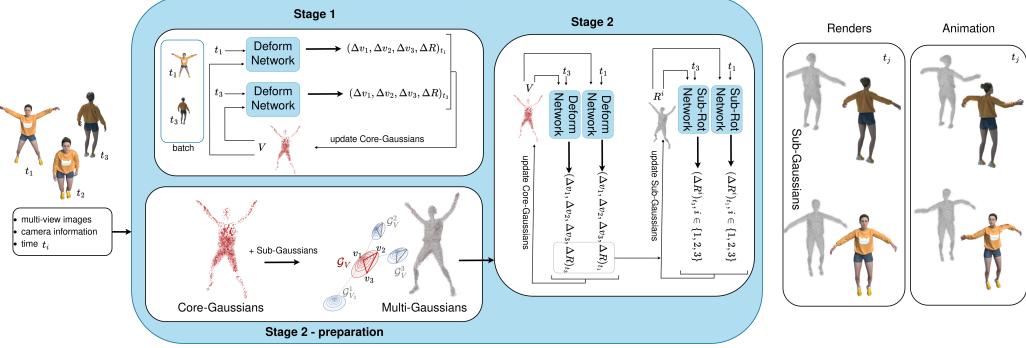


Figure 7: D-MiSo model diagram. The input of the model consists of images at different moments in time and information regarding the position of the camera. Training distinguishes two main phases (i.e., stages). The first includes the preparation of Core-Gaussians, describing the movement of the object. The second focuses on the fitting of Sub-Gaussians responsible for the render’s quality. The model has the ability to produce high-quality renders or create an animation/modification of the object due to Sub-Gaussians (i.e., Sub-Triangles Soup) shape modification.

is defined by:

$$\mathcal{G}_{core} = \{(\mathcal{N}_{core}(\mathbf{m}_i, R_i, S_i), \sigma_i, c_i)\}_{i=1}^p, \quad (1)$$

where  $S = \text{diag}(s_1, s_2, s_3)$ ,  $s_1 = \varepsilon$  and  $R$  is rotation matrix of Core-Gaussian defined as:  $R = (\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3)$ , where  $\mathbf{r}_i \in \mathbb{R}^3$  which can be interpreted as a local coordinate system used by Sub-Gaussian.

Sub-Gaussian can be interpreted as a child of Core-Gaussian. We define centers of Sub-Gaussian  $\mathcal{N}_{sub}(\mathbf{m}^i, R^i, S^i)$  in the local coordinate system of Core-Gaussian  $\mathcal{N}_{core}(\mathbf{m}, R, S)$  by:

$$\mathbf{m}^i = \mathbf{m} + R\boldsymbol{\alpha}^{iT} \quad (2)$$

where  $\mathbf{m}$ ,  $R$  is Core-Gaussian position and rotation; and  $\boldsymbol{\alpha}^i = (\alpha_1^i, \alpha_2^i, \alpha_3^i)$  are trainable parameters used to define the positions of the Sub-Gaussian relative to the Core-Gaussian (Fig. 6). Sub-Gaussians are used for rendering and can be seen as a main component of our model:

$$\mathcal{G}_{sub} = \{(\mathcal{N}_{sub}(\mathbf{m} + R\boldsymbol{\alpha}^{iT}, R^i, S^i), \sigma^i, c^i)\}_{i=1}^k, \quad (3)$$

where  $\mathbf{m}$ ,  $R$ ,  $S$  are parameters of Core-Gaussian and,  $\boldsymbol{\alpha}^i, S^i, R^i$  marks parameters of  $i$ -th Sub-Gaussians with opacity  $\sigma^i$ , and SH colors  $c^i$ .

Core-Gaussians is generally dedicated mainly to transformations, hence, in practice opacity or colors are not used during rendering. On the other hand, Sub-Gaussian is devoted to rendering and modification. It has its own opacity and colors.

One of our mode’s most important properties is its ability to model dynamic scenes in each time step. To obtain such properties, we parameterize all Gaussian using Triangle Soup proposed by GaMeS [3]. Thanks to a few simple transformations, we can convert the mean  $\mathbf{m}$  rotation  $R$  and scaling  $S$  into triangle  $V = (\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$ , which parametrize Gaussian distribution. Such transformation is unambiguous and reversible, for more details see Appendix A. GaMeS transformation between Gaussian parameters and triangle face we denote by  $\mathcal{T}$ :

$$\mathcal{N}(V) = \mathcal{N}(\mathcal{T}^{-1}(V)) = \mathcal{N}(\hat{\mathbf{m}}_V, \hat{R}_V, \hat{S}_V)$$

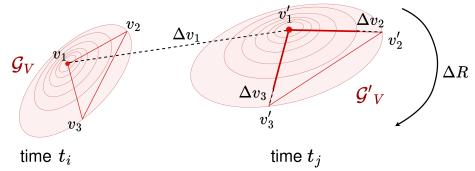


Figure 8: Representation of change over time acting on Core-Gaussians using a neural network responsible for movement. In practice,  $t_i$  is an abstract time. The network’s output returns information about the change in location  $\Delta v_1$ , scale ( $\Delta v_2$ ,  $\Delta v_3$ ), and rotation  $\Delta R$ .

In our D-MiSo we parameterize the  $p$  Core-Gaussians with  $k$  Triangle soup:

$$\mathcal{G}_{multi} = \left\{ \left( \mathcal{N}_{core}(V_j), \left\{ \left( \mathcal{N}_{sub} \left( \hat{\mathbf{m}}_{V_j} + \hat{R}_{V_j} \boldsymbol{\alpha}^{iT}, \hat{R}^i, \hat{S}^i \right), \sigma^i, c^i \right) \right\}_{i=1}^k \right) \right\}_{j=1}^p, \quad (4)$$

where  $V_j, \boldsymbol{\alpha}^i, R^i, \hat{S}^i, \sigma^i, c^i$  are trainable parameters and  $(\hat{\mathbf{m}}_{V_j}, \hat{R}_{V_j}, \hat{S}_{V_j}) = \mathcal{T}^{-1}(V_j)$ . Alternatively, we can parameterize Core-Gaussian and Sub Gaussians by Triangle Soup:

$$\mathcal{G}_{multi} = \left\{ \left( \mathcal{N}_{core}(V_j), \left\{ (\mathcal{N}_{sub}(V_j^i), \sigma^i, c^i) \right\}_{i=1}^k \right) \right\}_{j=1}^p, \quad (5)$$

where  $(\hat{\mathbf{m}}_{V_j}, \hat{R}_{V_j}, \hat{S}_{V_j}) = \mathcal{T}^{-1}(V_j)$ ,  $V^i = \mathcal{T}(\hat{\mathbf{m}}_V + \hat{R}_V \boldsymbol{\alpha}^{iT}, \hat{R}^i, \hat{S}^i)$ .

In D-MiSo, we use the collation of Multi-Gaussian distribution for rendering and Sub-Triangle Soup for editing. The formal definition of our model uses equation (4) since in training, we store Core-Gaussian as a triangle face (Core-Gaussian does not have colours) and Sub-Gaussian as a collection from classical GS components with colour and opacity. After training, we parametrize our model to equation (5) for editing.

### 3.1 Dynamic Multi-Gaussian Soup (D-MiSo)

Previously, we defined Multi-Gaussians and their parametrization using Triangle Soup. Now, we have all the tools to present the D-MiSo model. The overview of our method is illustrated in Fig. 7. The input to our model is a set of images of a dynamic scene, together with the time label and the corresponding camera poses. Our training is divided into two stages. In the first stage, we initialise the Core-Gaussians. In the second, we add Sub-Gaussian components.

**Stage 1** First, we train only Core-Gaussians to obtain good initialization for Multi-Gaussians. As Core-Gaussians are mainly employed to capture motion, the model does not require much of them, see Fig. 7. In our approach, the Core-Triangle Soup, constructed via the Core-Gaussians parameterization (as depicted in Fig. 2), is adjusted depending on the time  $t$ .

Practically, when random initialization of Gaussians is necessary, redundant Gaussians must be pruned first to ensure that the remaining ones emphasize the object’s shape. To reduce the number of Gaussians and obtain consistent Core-Gaussians, we train GS on a batch containing a few views instead of one. In practice, we render few views from different positions and use back-propagation.

In particular, we parameterize Gaussians  $\mathcal{N}(\mathbf{m}, R, S)$  by face  $V = (\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$  to obtain  $\mathcal{N}(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$ . Our *Deform Network* takes as input the triangle vertexes  $V$  assigned to the parameterized 3D Core-Gaussians and the current time  $t$  and returns updated  $\psi(V, t) = (\Delta \mathbf{v}_1(t), \Delta \mathbf{v}_2(t), \Delta \mathbf{v}_3(t), \Delta R_V(t))$ . Such updates consist of

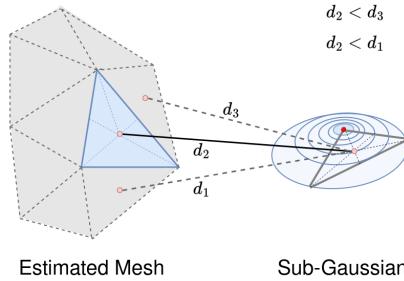


Figure 9: One way to animate is to assign the nearest triangle (face) to each Sub-Gaussian from the estimated mesh. The mesh modification changes the assigned Gaussian.

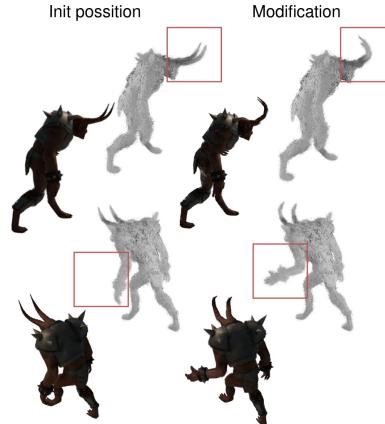


Figure 10: An example of Sub-Triangle Soup modification and the render obtained by this change from a different viewpoint. It is possible not only to change the position of the hand but also to raise the thumb.

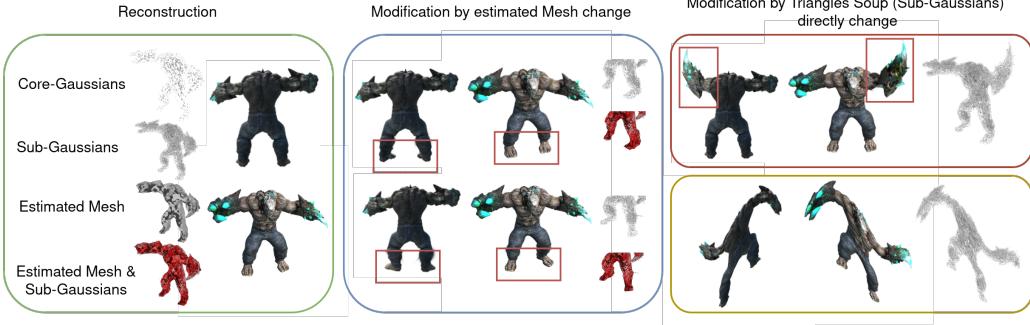


Figure 11: Reconstruction and three ways of modification of the output object. The first involves modifying the estimated mesh, which does not have to be accurate. The next two focus on Sub-Triangle Soup editing. The red box shows the direct modification of the Triangle Soup in a logical way (e.g., raising a hand). The yellow box shows a change in space, i.e., giving fluidity to an object by creating an abstract modification.

translation and rotation (see Fig. 8):

$$V(t) = V \odot \psi(V, t) = (\mathbf{v}_1 + \Delta\mathbf{v}_1(t), \mathbf{v}_2 + \Delta\mathbf{v}_2(t), \mathbf{v}_3 + \Delta\mathbf{v}_3(t)) \cdot \Delta R_V(t).$$

D-MiSo parameterized Core-Gaussians in time  $t$  by:  $\mathcal{N}_{core}(V(t), \sigma, c) = \mathcal{N}_{core}(V \odot \psi(V, t), \sigma, c)$  where  $\psi$  is a deformable network that moves triangle  $V$  according to time  $t$ . The opacity and colour of the Core-Gaussian are used only in the first stage.

**Stage 2: Preparation** To move to the second phase of the model, it is imperative to prepare the Multi-Gaussians. This involves attaching  $k$  Sub-Gaussians to each Core-Gaussian generated in Stage 1, as shown in Fig. 6. Henceforth, Sub-Gaussians assume responsibility for the resultant rendering. Initially, the Sub-Gaussian adopts the same features as the Core-Gaussian, except the position.

**Stage 2** The primary objective of the second phase is to parallelized Core-Gaussians (Core-Triangles Soup) to enhance understanding of movement and increase rendering quality through the precise training of Sub-Gaussians. Since the centers of Sub-Gaussians are parameterized by the rotation matrix of Core-Gaussian, when the *Deform Network*  $\psi$  changes the Core-Gaussian all Sub-Gaussians (attached to this Core-Gaussian) are modified by global transformation  $\psi(V, t)$ . D-MiSo use an additional deformation network *Sub-Rot Network*  $\phi$  dedicated to each  $i$ -th Sub-Gaussian's small changes. *Sub-Rot Network* takes the Sub-Gaussian rotation matrix  $R_V^i$  and the current time  $t$  as input and produces an updated rotation matrix  $\Delta R_V^i(t)$ .

The position  $\hat{\mathbf{m}}_V^i(t) = \hat{\mathbf{m}}_{V(t)} + \hat{R}_{V(t)} \boldsymbol{\alpha}^{iT}$  of the Sub-Gaussian in time  $t$  is determined by the position  $\hat{\mathbf{m}}_{V(t)}$ , and rotation  $\hat{R}_{V(t)}$  of the Core-Gaussian (parameterized by triangle  $V$ ) and the learning parameter  $\boldsymbol{\alpha}^i$ . It should be noted that scale  $S^i$ , color  $c_i$ , and opacity  $\sigma_i$  of Sub-Gaussian are trainable and do not depend on time. *Sub-Rot Network* produce updated  $\phi(R^i, t) = \Delta R^i(t)$  for rotation parameter

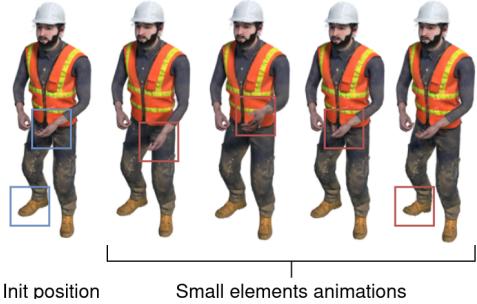


Figure 12: Limbs render and modification obtained with D-MiSo. It is worth noting that it is also possible to close the hand.

by the local coordinate system given by the rotation matrix of Core-Gaussian, when the *Deform Network*  $\psi$  changes the Core-Gaussian all Sub-Gaussians (attached to this Core-Gaussian) are modified by global transformation  $\psi(V, t)$ . D-MiSo use an additional deformation network *Sub-Rot Network*  $\phi$  dedicated to each  $i$ -th Sub-Gaussian's small changes. *Sub-Rot Network* takes the Sub-Gaussian rotation matrix  $R_V^i$  and the current time  $t$  as input and produces an updated rotation matrix  $\Delta R_V^i(t)$ .



Figure 13: Examples of object modifications. The first method allows for a smooth modification (bending) and also removes (e.g. plate), scales and/or adds (e.g. small blue balls) objects.

of Sub-Gaussians. Finally parameters of Sub-Gaussians in time  $t$  depends on *Deform Network*  $\psi$ , and *Sub-Rot Network*  $\phi$ , and the Corr-Gaussians parameter  $V$ , and Sub-Gaussian parameters  $R^i, S^i, c_i$  and  $\sigma_i$ :

$$\mathcal{G}_{Sub}(t) = \{(\mathcal{N}_{sub}(\hat{\mathbf{m}}_{V \odot \psi(V, t)} + \hat{R}_{V \odot \psi(V, t)} \boldsymbol{\alpha}^{iT}, R^i + \phi(R^i, t), S^i), c_i, \sigma_i)\}_{i=1}^k.$$

D-MiSo final model consists of two deformable networks and two levels of Gaussian distributions. Such a solution allows effectively modeling time in dynamic scenes and modifying objects in each time frame.

The result of the model’s inference depends on the camera’s view angle as well as on the selected time. Only Sub-Gaussian features are used in the rendering process. Hence, the implementation of generating an output image from Gaussians is no different from a vanilla GS.

The modifications involve the appropriate transformation of Sub-Gaussians. We base this on the fact that the transformed Gaussians have exactly the same color properties as before editing. Only the shape of the Gaussians is changed. In our work, we present three editing methods (Fig. 11).

## 4 Experiments

The experimental section is divided into two parts. First, we show that D-MiSo can model dynamic scenes with high quality; in the second part, that our model allows an easy editing procedure, which is our main contribution.

### 4.1 Reconstruction of dynamic scenes

Here, we outline the specifics of our implementation and provide a detailed description of the used datasets. We demonstrated the core advantages of our model by conducting experiments using three different datasets. The source code is available in GitHub. Our code is developed on top of the GS vanilla code, according to their license. We used NVIDIA GeForce RTX 4090 and A100 GPUs. The experiments focus on a benchmark tasks of reconstruction. PSNR metric is used to compare our model with another ones. Furthermore, additional numerical comparisons, i.e., SSIM/LPIPS, are available in the supplementary material.

**D-NeRF Datasets:** Contains seven dynamic objects with realistic materials described with a single camera [11]. This means the model had access to only one view at a given moment. Tab. 1 shows we are very comparable to other methods, and we achieve a higher PSNR on one object. The differences in metrics are small, however, our method presents an easier way to edit (Fig. 4). Providing, among other things, better scaling.

**NeRF-DS[34]:** This dataset contains again seven real-world scenarios containing a moving or deforming specular object. Each scene was recorded using two cameras, with the video captured by one of them being used as a training set, while the footage from the second one was treated as a test set. The camera pose was estimated using COLMAP for both cameras. Tab. 2 shows that our method for four objects achieves the SOTA results.

**PanopticSports Datasets:** The dataset comprises six dynamic scenes featuring significant object and actor movements [21]. The scenes are categorized by the activities performed in the video sequences, i.e., juggling, boxing, softball, tennis, football, and basketball. Each scene was recorded using 31 cameras over 150 timesteps. Following the official data split, we use footage from 27 cameras for training and the remaining 4 cameras for testing. Numerical results are shown in Tab. 3. The results show that the model achieved for five objects SOTA according to PSNR metrics, and six times according to LPIPS metrics.

Table 1: Quantitative comparisons (PSNR) on a D-NeRF dataset showing that D-MiSo gives comparable results with other models.

	PSNR ↑					
	Hook	Jumpin.	Trex	Bounc.	Hell.	Mutant
D-NeRF [11]	29.25	32.80	31.75	38.93	25.02	31.29
TiNeuVox-B [32]	31.45	34.23	32.70	40.73	28.17	33.61
Tensor4D [16]	29.03	24.01	23.51	25.36	31.40	29.99
K-Planes [15]	28.59	32.27	31.41	40.61	25.27	33.79
FF-NVS [33]	32.29	33.55	30.71	40.02	27.71	34.97
4D-GS [23]	30.99	33.59	32.16	38.59	31.39	35.98
DynMF [2]	33.94	38.04	35.82	41.92	37.51	41.68
Editable						
SC-GS [28]	39.87	41.13	41.24	44.91	42.93	45.19
<b>D-MiSo (our)</b>	38.13	42.05	40.88	41.49	41.49	44.38
						47.66

We show that D-MiSo is comparable to other methods. However, our main contribution in comparison to other methods is the very easy editing of the resulting object (Fig. 4 11).

## 4.2 Editing of dynamic scenes

As mentioned earlier, we proposed three new methods of output object modification. The first one focuses on moving and preparing a formal mesh, which is connected in contrast to Triangle Soup. We apply a simple meshing strategy on Core-Gaussians. We used the basic Alpha Shape algorithm [35, 36] and showed that the mesh does not have to estimate the surface perfectly to be effective. Then, we reparametrize the Sub-Gaussian by finding the closest face from the mesh (Fig. 9). In practice, we represent each Sub-Gaussian in a local coordinate system (analogically to Multi-Gaussians). Therefore, each Sub-Gaussian is assigned to the nearest face (Fig. 9). Each face can have a different number of Sub-Gaussians prescribed in such a configuration. This method allows us to maintain the consistency bestowed by the mesh. The whole process is shown in Fig. 5. Thanks to such representation, we can edit our connected mesh to produce the correct edition of the dynamic scene (Fig. 2).

The second editing method allows us to define the connections, i.e., editing Sub-Triangles Soup directly e.g. moving a hand or bending horns (Fig. 10) or rotating a human body (Fig. 3). Changes are possible because we can also transform a group of Sub-Triangles instead of individual ones. This way allows for even very subtle changes like raising the thumb (Fig. 10), turning the hand over (Fig 4), opening or closing the hand (Fig. 12). These edits would be difficult in other approaches based on adjusting the 3D objects’ centroids (nodes) since the space of nodes is limited in details area (Fig. 4). Nodes use is preeminent for defining movement, which was insignificant in these places.

With this method, we can also change complicated objects like a 360° scene without losing the dynamics-related model. For example, the rotation of a person stacking boxes (Fig. 3).

Table 3: Comparison on PanopticSports dataset.

Metrics	Method	JuggleBoxes	Softball	Tennis	Football	Basketball	Mean
PSNR ↑	3DGs [1]	28.19	28.74	28.77	28.03	28.49	27.02
	Dyn3DG [21]	29.48	29.46	28.43	28.11	28.49	28.22
	D-MiSo (our)	29.79	29.39	28.6	29.02	28.99	28.49
SSIM ↑	3DGs [1]	0.91	0.91	0.91	0.90	0.90	0.89
	Dyn3DG [21]	0.92	0.91	0.91	0.91	0.91	0.91
	D-MiSo (our)	0.93	0.92	0.92	0.92	0.92	0.91
LPIPS ↓	3DGs [1]	0.15	0.15	0.14	0.16	0.16	0.18
	Dyn3DG [21]	0.15	0.17	0.19	0.17	0.19	0.18
	D-MiSo (our)	0.13	0.13	0.15	0.14	0.13	0.15

Our methods are also scalable, so we can easily remove or duplicate elements from an image. Moreover, the duplicated elements can be given their own dynamics. Examples of these effects are shown in Fig. 13 where we removed the plate, and both duplicated and rescaled the blue balls multiple times.

## 5 Conclusion

D-MiSo is a novel method based on Gaussian Splatting parameterization, which produces a cloud of triangles called Triangle Soup. The method allows easy editing of objects created in inference with the possible transformations including moving, scaling, and rotating. By defining Multi-Gaussians, the obligatory separability of modified elements seen in other models is combated. In addition, certain elements of objects can be duplicated and removed (Fig. 13). Furthermore, the D-MiSo method can facilitate giving different dynamics to separate parts of an object (Fig. 1).

Table 2: PSNR comparisons on a NeRF-DS dataset showing that D-MiSo gives comparable results with other models.

	PSNR ↑					
	Bell	Sheet	Press	Basin	Cup	Sieve Plate
HyperNeRF [9]	24.0	24.3	25.4	20.2	20.5	25.0
NeRF-DS [34]	23.3	25.7	26.4	20.3	24.5	26.1
TiNeuVox-B [32]	23.1	21.1	24.1	20.7	20.5	20.1
Editable						
SC-GS [4]	25.1	26.2	26.6	19.6	24.5	26.0
D-MiSo (our)	25.3	25.8	25.6	19.8	24.5	26.5

Nodes use is preeminent for defining movement, which was insignificant in these places.

The third method focuses on transforming the space in which the object is located. Similar to the second method, the third one also works directly on the Sub-Triangle soup. We can achieve such an effect by applying a certain function to the selected plane. In practice, the object, or a portion of it, is modified, for instance, through the use of sinusoidal functions. It allows us to obtain fluidity and more easily define the physical nature of the movement (Fig. 1, 11, 13).

**Limitation** The method allows for complex changes at a given moment in time. However, if some area is not well represented in the training set, it is not possible to edit them. For example, a person’s hand can be changed but not fingers (Fig. 4). This is due to the liminality of Triangle Soup relative to a well-fitted mesh.

## References

- [1] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), 2023.
- [2] Agelos Kratimenos, Jiahui Lei, and Kostas Daniilidis. Dynmf: Neural motion factorization for real-time dynamic view synthesis with 3d gaussian splatting. *arXiv*, 2023.
- [3] Joanna Waczyńska, Piotr Borycki, Sławomir Tadeja, Jacek Tabor, and Przemysław Spurek. Games: Mesh-based adapting and modification of gaussian splatting. *arXiv preprint arXiv:2402.01459*, 2024.
- [4] Yi-Hua Huang, Yang-Tian Sun, Ziyi Yang, Xiaoyang Lyu, Yan-Pei Cao, and Xiaojuan Qi. Sc-gs: Sparse-controlled gaussian splatting for editable dynamic scenes. *arXiv preprint arXiv:2312.14937*, 2023.
- [5] Hayong Shin, J.C. Park, B.K. Choi, Y.C. Chung, and S. Rhee. Efficient topology construction from triangle soup. In *Geometric Modeling and Processing, 2004. Proceedings*, pages 359–364, 2004.
- [6] Felix Paulano, Juan Jimenez-Delgado, and Ruben Pulido. An application to interact with 3d models reconstructed from medical images. volume 3, 01 2014.
- [7] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*, 2020.
- [8] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. *ICCV*, 2021.
- [9] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M. Seitz. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *ACM Trans. Graph.*, 40(6), dec 2021.
- [10] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6498–6508, 2021.
- [11] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-NeRF: Neural Radiance Fields for Dynamic Scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.
- [12] Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. Dynamic view synthesis from dynamic monocular video. In *Proceedings of the IEEE International Conference on Computer Vision*, 2021.
- [13] T. Li, M. Slavcheva, M. Zollhoefer, S. Green, C. Lassner, C. Kim, T. Schmidt, S. Lovegrove, M. Goesele, R. Newcombe, and Z. Lv. Neural 3d video synthesis from multi-view video. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5511–5521, Los Alamitos, CA, USA, jun 2022. IEEE Computer Society.
- [14] Ang Cao and Justin Johnson. Hexplane: A fast representation for dynamic scenes. *CVPR*, 2023.
- [15] Sara Fridovich-Keil, Giacomo Meanti, Frederik Rahbæk Warburg, Benjamin Recht, and Angjoo Kanazawa. K-planes: Explicit radiance fields in space, time, and appearance. In *CVPR*, 2023.
- [16] Ruizhi Shao, Zerong Zheng, Hanzhang Tu, Boning Liu, Hongwen Zhang, and Yebin Liu. Tensor4d: Efficient neural 4d decomposition for high-fidelity dynamic reconstruction and rendering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2023.

- [17] Yu-Jie Yuan, Yang-Tian Sun, Yu-Kun Lai, Yuewen Ma, Rongfei Jia, and Lin Gao. Nerf-editing: geometry editing of neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18353–18364, 2022.
- [18] Chengwei Zheng, Wenbin Lin, and Feng Xu. Editablenerf: Editing topologically varying neural radiance fields by key points, 2023.
- [19] Kacper Kania, Kwang Moo Yi, Marek Kowalski, Tomasz Trzciński, and Andrea Tagliasacchi. CoNeRF: Controllable Neural Radiance Fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2022.
- [20] Yuze Wang, Junyi Wang, Yansong Qu, and Yue Qi. Rip-nerf: Learning rotation-invariant point-based neural radiance field for fine-grained editing and compositing. *Proceedings of the 2023 ACM International Conference on Multimedia Retrieval*, 2023.
- [21] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. In *3DV*, 2024.
- [22] Ziyi Yang, Xinyu Gao, Wen Zhou, Shaohui Jiao, Yuqing Zhang, and Xiaogang Jin. Deformable 3d gaussians for high-fidelity monocular dynamic scene reconstruction. *arXiv preprint arXiv:2309.13101*, 2023.
- [23] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Wang Xinggang. 4d gaussian splatting for real-time dynamic scene rendering. *arXiv preprint arXiv:2310.08528*, 2023.
- [24] Yiqing Liang, Numair Khan, Zhengqin Li, Thu Nguyen-Phuoc, Douglas Lanman, James Tompkin, and Lei Xiao. Gaufre: Gaussian deformation fields for real-time dynamic novel view synthesis, 2023.
- [25] Tingyang Zhang, Qingzhe Gao, Weiyu Li, Libin Liu, and Baoquan Chen. Bags: Building animatable gaussian splatting from a monocular video with diffusion priors, 2024.
- [26] Lin Gao, Jie Yang, Bo-Tao Zhang, Jia-Mu Sun, Yu-Jie Yuan, Hongbo Fu, and Yu-Kun Lai. Mesh-based gaussian splatting for real-time large-scale deformation, 2024.
- [27] Antoine Guédon and Vincent Lepetit. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. *arXiv preprint arXiv:2311.12775*, 2023.
- [28] Yi-Hua Huang, Yang-Tian Sun, Ziyi Yang, Xiaoyang Lyu, Yan-Pei Cao, and Xiaojuan Qi. Sc-gs: Sparse-controlled gaussian splatting for editable dynamic scenes. *arXiv preprint arXiv:2312.14937*, 2023.
- [29] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, pages 5501–5510, 2022.
- [30] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (ToG)*, 41(4):1–15, 2022.
- [31] Ziyi Yang, Xinyu Gao, Yangtian Sun, Yihua Huang, Xiaoyang Lyu, Wen Zhou, Shaohui Jiao, Xiaojuan Qi, and Xiaogang Jin. Spec-gaussian: Anisotropic view-dependent appearance for 3d gaussian splatting, 2024.
- [32] Jiemin Fang, Taoran Yi, Xinggang Wang, Lingxi Xie, Xiaopeng Zhang, Wenyu Liu, Matthias Nießner, and Qi Tian. Fast dynamic radiance fields with time-aware neural voxels. In *SIGGRAPH Asia 2022 Conference Papers*, SA ’22. ACM, November 2022.
- [33] Xiang Guo, Jiadai Sun, Yuchao Dai, Guanying Chen, Xiaoqing Ye, Xiao Tan, Errui Ding, Yumeng Zhang, and Jingdong Wang. Forward flow for novel view synthesis of dynamic scenes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 16022–16033, October 2023.
- [34] Zhiwen Yan, Chen Li, and Gim Hee Lee. Nerf-ds: Neural radiance fields for dynamic specular objects, 2023.
- [35] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29(4):551–559, 1983.
- [36] Herbert Edelsbrunner. Alpha shapes—a survey. *Tessellations in the Sciences*, 01 2010.
- [37] Ake Björck. Numerics of Gram-Schmidt orthogonalization. *Linear Algebra and Its Applications*, 197:297–316, 1994.

## A GaMeS [3] parametrisation of Gaussian component

Multi-Gaussians describe 3D scenes using solid blocks rather than tiny Gaussian distributions. This method enhances our model’s suitability for dynamic environments. Additionally, we desire the capability to modify 3D objects at any moment. As a result, we employ a mesh-inspired to condition our model. In order to do that, we incorporate the parametrization concept from GaMeS [3].

As it was mentioned earlier, our model uses flat Gaussians. Therefore, we can approximate Gaussian components using triangle face mesh by parameterizing Gaussian components by the vertices of the mesh face. We denote such transformation by  $\mathcal{T}(\cdot)$ . In practice as a consequence of parameterizing each Gaussian, we obtain a cloud of triangles called Triangle Soup [5]. Accordingly, we get: Sub-Triangle Soup, Core-Triangle Soup and Multi-Triangle Soup, see Fig. 2.

Let us assume that we have a Gaussian component parameterized by mean  $\mathbf{m}$ , rotation matrix  $R = [\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3]$  and scaling  $S = \text{dig}(\varepsilon, s_2, s_3)$ . We define three vertex of triangle (face):

$$V = [\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3],$$

where  $\mathbf{v}_1 = \mathbf{m}$ ,  $\mathbf{v}_2 = \mathbf{m} + s_2 \mathbf{r}_2$ ,  $\mathbf{v}_3 = \mathbf{m} + s_3 \mathbf{r}_3$ .

Now we froze the vertex of the face  $V = [\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$  and reparameterize the Gaussian component by defining  $\hat{\mathbf{m}}$ ,  $\hat{R} = [\hat{\mathbf{r}}_1, \hat{\mathbf{r}}_2, \hat{\mathbf{r}}_3]$  and  $\hat{S} = \text{dig}(\hat{s}_1, \hat{s}_2, \hat{s}_3)$ . First, we put  $\hat{\mathbf{m}} = \mathbf{v}_1$ . The first vertex of  $\hat{R}$  is given by a normal vector:

$$\hat{\mathbf{r}}_1 = \frac{(\mathbf{v}_2 - \mathbf{v}_1) \times (\mathbf{v}_3 - \mathbf{v}_1)}{\|(\mathbf{v}_2 - \mathbf{v}_1) \times (\mathbf{v}_3 - \mathbf{v}_1)\|},$$

where  $\times$  is the cross product. The second one is defined by  $\hat{\mathbf{r}}_2 = \frac{(\mathbf{v}_2 - \mathbf{v}_1)}{\|(\mathbf{v}_2 - \mathbf{v}_1)\|}$ . The third one is obtained as a single step in the Gram–Schmidt process [37]:

$$\hat{\mathbf{r}}_3 = \text{orth}(\mathbf{v}_3 - \mathbf{v}_1; \mathbf{r}_1, \mathbf{r}_2).$$

Scaling parameters can also be easily calculated as  $s_1 = \varepsilon$ ,  $\hat{s}_2 = \|\mathbf{v}_2 - \mathbf{v}_1\|$  and  $\hat{s}_3 = \langle \mathbf{v}_3 - \mathbf{v}_1, \hat{\mathbf{r}}_3 \rangle$ .

Consequently, the covariance of Gaussian distribution positioned to face is given by:

$$\hat{\Sigma}_V = \hat{R}_V \hat{S}_V \hat{S}_V \hat{R}_V^T,$$

and correspond with the shape of a triangle  $V$ . For one face  $(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$ , we define the corresponding Gaussian component:

$$\mathcal{N}((\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)) = \mathcal{N}(\hat{\mathbf{m}}_V, \hat{R}_V, \hat{S}_V).$$

Finally, the Gaussian component is derived from the mesh face parameters. This approach can be applied in a Multi-Gaussian framework. Therefor we will use invertible transformation between Gaussian parameters and triangle face  $T$  and notation

$$\mathcal{N}(V) := \mathcal{N}(\mathcal{T}^{-1}(V)) = \mathcal{N}(\hat{\mathbf{m}}_V, \hat{R}_V, \hat{S}_V).$$

## B Extension of numerical results from main paper

This section contains a numerical comparison of the D-MiSo with others models, regarding the experiments described in the main document. We used additional LPIPS and SSIM metrics. We see that we get comparable competitive results. We can see that our method, because of the LPIPS metric, achieves better results for five objects from D-NeRF dataset (Tab. 4). Numerical comparison with metrics for the NeRF-DS dataset show that we are comparable to other methods in the reproduction task ( Tab. 5).

In our method, a batch of images is taken as an input to the model. Table 6 shows a numerical comparison using batch: 4, 8, 10 on D-NeRF. Training takes 80 thousand iterations and second stages started at the 5 thousandth iteration. Each Core-Gaussian have attached 25 Sub-Gaussians. This is comparable to the SC-GS implementation. Similar study have been done for NeRF-DS dataset presented in 7. We can see that batch is playing a bigger role in the dataset, which is considering a more pronounced move.

Methods	Hook			Jumpingjacks			Trex			BouncingBalls		
	PSNR ↑ SSIM ↑ LPIPS ↓											
D-NeRF	29.25	.968	.1120	32.80	.981	.0381	31.75	.974	.0367	38.93	.987	.1074
TiNeuVox-B	31.45	.971	.0569	34.23	.986	.0383	32.70	.987	.0340	40.73	.991	.0472
Tensor4D	29.03	.955	.0499	24.01	.919	.0768	23.51	.934	.0640	25.36	.961	.0411
K-Planes	28.59	.953	.0581	32.27	.971	.0389	31.41	.980	.0234	40.61	.991	.0297
FF-NVS	32.29	.980	.0400	33.55	.980	.0300	30.71	.960	.0400	40.02	.990	.0400
4D-GS	30.99	.990	.0248	33.59	.990	.0242	32.16	.988	.0216	38.59	.993	.0267
SC-GS	39.87	.997	.0076	41.13	.998	.0067	41.24	.998	.0046	44.91	.998	.0166
D-MiSo (our)	38.13	.990	.0086	42.05	.995	.0049	40.88	.996	.0029	41.49	.993	.0079
Methods	Hellwarrior			Mutant			Standup			Average		
	PSNR ↑ SSIM ↑ LPIPS ↓											
D-NeRF	25.02	.955	.0633	31.29	.978	.0212	32.79	.991	.0241	31.69	.975	.0575
TiNeuVox-B	28.17	.978	.0706	33.61	.982	.0388	35.43	.991	.0230	33.76	.983	.0441
Tensor4D	31.40	.925	.0675	29.99	.951	.0422	30.86	.964	.0214	27.62	.947	.0471
K-Planes	25.27	.948	.0775	33.79	.982	.0207	34.31	.984	.0194	32.32	.973	.0382
FF-NVS	27.71	.970	.0500	34.97	.980	.0300	36.91	.990	.0200	33.73	.979	.0357
4D-GS	31.39	.974	.0436	35.98	.996	.0120	35.37	.994	.0136	34.01	.987	.0316
SC-GS	42.93	.994	.0155	45.19	.999	.0028	47.89	.999	.0023	43.31	.997	.0063
D-MiSo (our)	41.49	.986	.0173	44.38	.997	.0026	47.66	.998	.0016	42.27	.993	.0065

Table 4: Quantitative comparison on a D-NeRF dataset

Methods	Bell			Sheet			Press			Basin		
	PSNR ↑ SSIM ↑ LPIPS ↓											
HyperNeRF	24.0	.884	.159	24.3	.874	.148	25.4	.873	.164	20.2	.829	.168
NeRF-DS	23.3	.872	.134	25.7	.918	.115	26.4	.911	.123	20.3	.868	.127
TiNeuVox-B	23.1	.876	.113	21.1	.745	.234	24.1	.892	.133	20.7	.896	.105
SC-GS	25.1	.918	.117	26.2	.898	.142	26.6	.901	.135	19.6	.846	.154
D-MiSo (our)	25.3	.846	.174	25.8	.875	.211	25.6	.867	.206	19.8	.788	.217
Methods	Cup			Sieve			Plant			Average		
	PSNR ↑ SSIM ↑ LPIPS ↓											
HyperNeRF	20.5	.705	.318	25.0	.909	.129	18.1	.714	.359	22.5	.827	.206
NeRF-DS	24.5	.916	.118	26.1	.935	.108	20.8	.867	.164	23.9	.898	.127
TiNeuVox-B	20.5	.806	.182	20.1	.822	.205	20.6	.863	.161	21.5	.843	.162
SC-GS	24.5	.916	.115	26.0	.919	.114	20.2	.837	.202	24.1	.891	.140
D-MiSo (our)	24.5	.874	.185	26.5	.881	.159	20.8	.815	.234	24.0	.849	.198

Table 5: Quantitative comparison on a NeRF-DS dataset

Batch size	Hook			Jumpingjacks			Trex			BouncingBalls		
	PSNR ↑ SSIM ↑ LPIPS ↓											
10	37.02	.986	.0108	42.05	.995	.0049	40.88	.996	.0029	39.19	.988	.0132
8	38.13	.990	.0086	41.52	.995	.0052	40.60	.994	.0043	41.49	.993	.0079
4	37.68	.989	.0105	40.44	.994	.0063	39.58	.995	.0047	39.76	.993	.0073
Batch size	Hellwarrior			Mutant			Standup			Average		
	PSNR ↑ SSIM ↑ LPIPS ↓											
10	40.02	.980	.0276	44.29	.997	.0027	47.66	.998	.0016	41.58	.990	.0091
8	41.32	.986	.0174	44.38	.997	.0026	47.17	.998	.0018	42.08	.993	.0068
4	41.49	.986	.0173	43.32	.996	.0034	45.91	.997	.0025	41.16	.993	.0074

Table 6: Batch study on a D-NeRF dataset

Batch size	Bell			Sheet			Press			Basin		
	PSNR ↑ SSIM ↑ LPIPS ↓											
8	25.1	.851	.165	24.8	.863	.219	25.2	.858	.205	19.7	.788	.217
4	24.5	.832	.190	25.1	.873	.202	25.3	.860	.215	19.8	.781	.245
2	25.3	.846	.174	25.8	.875	.211	25.6	.867	.206	19.6	.785	.207
1	25.1	.844	.190	25.6	.873	.216	24.3	.842	.273	19.7	.785	.229
Batch size	Cup			Sieve			Plant			Average		
	PSNR ↑ SSIM ↑ LPIPS ↓											
8	24.0	.884	.163	26.5	.881	.158	20.2	.812	.226	23.6	.848	.193
4	24.2	.886	.163	25.8	.871	.167	20.8	.815	.234	23.6	.845	.202
2	24.4	.883	.169	26.0	.875	.169	20.6	.817	.232	23.9	.849	.195
1	24.5	.874	.185	25.8	.864	.195	20.5	.808	.245	23.6	.841	.219

Table 7: Batch study on a NeRF-DS dataset

## C Extension of examples modification

Below is an example of scene modification from the PanopticSports dataset (Fig. 14). The dynamics of the object has been stopped, so as to show the possibility of changing the position of the ball at a given time instant  $t$ .



Figure 14: Example of a ball position modification on a 360° scene from PanopticSports Datasets

Due to the characteristics of Multi-Gaussians and the ease of their control – we show that we can easily duplicate and/or scale selected objects (Fig. 15).

Third method of modification allows to give new dynamics and fluidity to the object (Fig. 16). We can see that even difficult edits like bending the back, changing the shape of the face, bending the hand are possible and the result is natural from the point of view of graphics, this is made possible by editing the scale and rotation of Gaussians in an explicitly defined way.

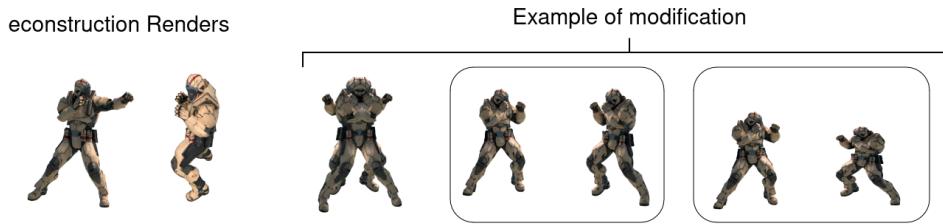


Figure 15: Example of animation by duplicating elements or changing their scale

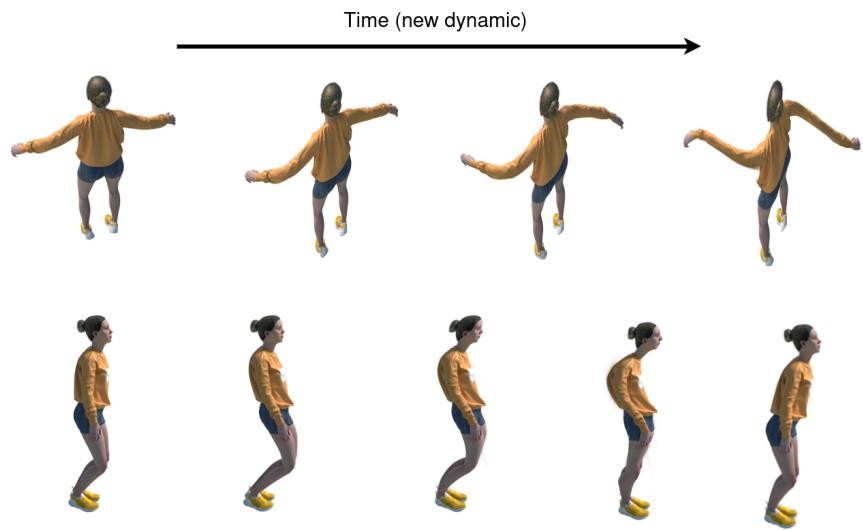


Figure 16: Example of third way of modification – Transformation of space, allowing to give new dynamics.