DOI: 10.1111/cgf.70074
EUROGRAPHICS 2025 / A. Bousseau and A. Dai
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 44 (2025), Number 2

Mesh Compression with Quantized Neural Displacement Fields

Sai Karthikey Pentapati[1,2], Gregoire Phillips[2], and Alan C. Bovik[1]

[1]Laboratory of Image and Video Engineering, The University of Texas at Austin
[2]Ericsson Inc., Santa Clara, CA

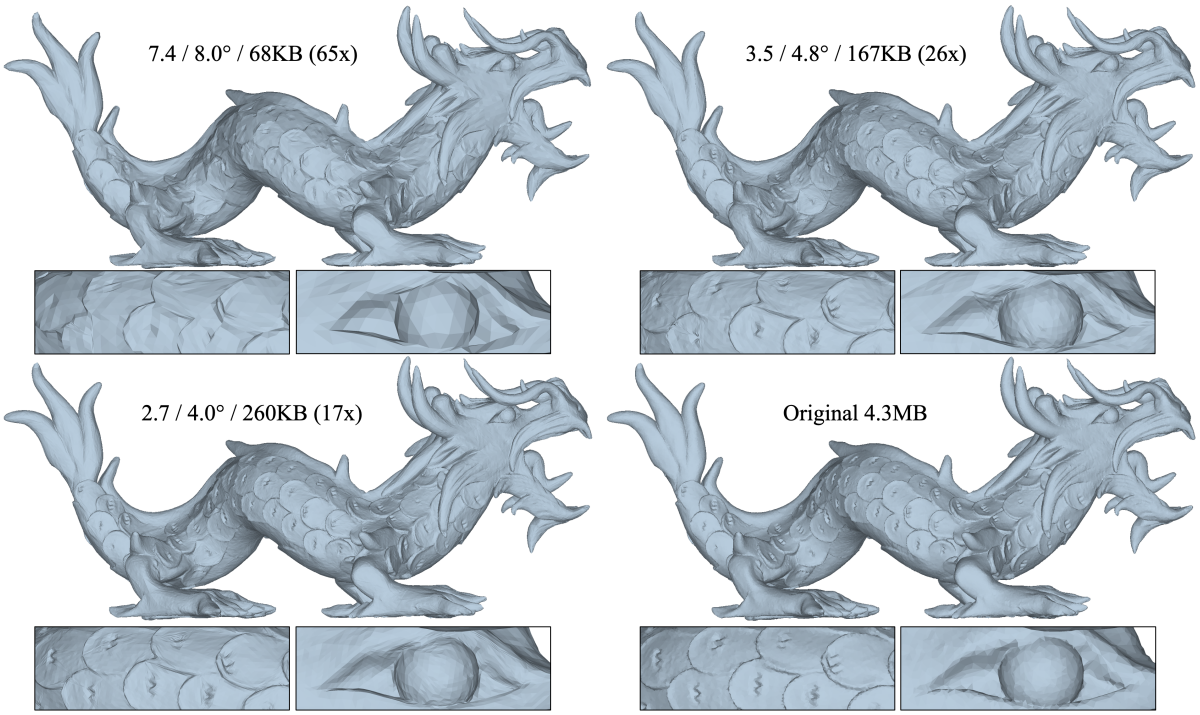

Figure 1: *We present a framework for compressing 3D meshes by compactly encoding a displacement field as an implicit neural representation. The displacement field can be applied to the surface of the coarse approximation of the original mesh to reconstruct it. The method achieves state-of-the-art reconstruction quality for a wide range of compression ratios. Here we show up to 65x compression of* `XYZ` *model courtesy of Stanford's 3D scanning repository. The point-to-mesh error, average normal error in degrees, and the compressed sizes are shown for each mesh.*

Abstract

Implicit neural representations (INRs) have been successfully used to compress a variety of 3D surface representations such as Signed Distance Functions (SDFs), voxel grids, and also other forms of structured data such as images, videos, and audio. However, these methods have been limited in their application to unstructured data such as 3D meshes and point clouds. This work presents a simple yet effective method that extends the usage of INRs to compress 3D triangle meshes. Our method encodes a displacement field that refines the coarse version of the 3D mesh surface to be compressed using a small neural network. Once trained, the neural network weights occupy much lower memory than the displacement field or the original surface. We show that our method is capable of preserving intricate geometric textures and demonstrates state-of-the-art performance for compression ratios ranging from 4x to 380x (See Figure 1 for an example).

CCS Concepts

• ***Computing methodologies*** → ***Mesh geometry models;*** *Neural networks;*

1. Introduction

3D meshes, consisting of vertices, edges, and faces, are used to represent detailed geometric structures in computer graphics and are ubiquitously used in gaming, virtual reality, and simulation applications. The use of meshes has grown in popularity compared to other 3D representations because their efficient rendering has been facilitated by advancements in hardware, notably graphics processing units (GPUs), which accelerate the process through parallel computing and optimized graphics pipelines. Despite these advancements in rendering technology, the inherent sizes and complexities of 3D meshes pose significant challenges in terms of memory use and transmission which are crucial for real-time communications applications. The development of compression algorithms is therefore crucial to mitigate these challenges to ensure efficient handling and transmission of complex and large-scale 3D data.

In recent years, compression algorithms that rely on building Implicit Neural Representations (INRs) have been shown to outperform traditional methods across various modalities. Among numerous 3D surface representations, the development of neural compression methods has been mostly focused on addressing the compression of Signed Distance Functions (SDFs) ([PFS*19, DNJ21]) and occupancy grids ([TLY*21, JYPH22]). However, the use of these representations is extremely limited in computer and mobile applications compared to meshes due to their inefficient rendering process given the available hardware.

Owing to these challenges, only a few deep compression techniques for meshes have been presented so far. The primary challenge of formulating INR-based compression algorithms for meshes is their discrete and unstructured nature. It is not straightforward to encode the connections between vertices as neural representations, due to the challenge of posing the connectivity reconstruction problem as a gradient-based optimization problem. To circumvent this challenge, remeshing and subdivision surfaces have been employed by [LKC*20] and [CKAJ23] to reconstruct the vertex connectivity when building large datasets of simplified meshes and their high-polygon count versions of them. Following a different direction, [SRL24] proposed a geometric primitive called the Neural Geometry Field to solve the problem of reconstructing the vertex connectivity.

In this paper, we present a simple method of constructing INRs of meshes and thereby compressing them. Similar to previous works by [CKAJ23] and [LKC*20], our method also leverages remeshing and surface subdivisions to construct an as-structured-as-possible representation of a mesh to facilitate the training of an INR. Specifically, our method compresses a mesh by converting it to a very coarse approximation of its surface and a displacement field encoded as a highly compact INR that allows reconstruction of the original surface from the coarse approximation. The major contributions of our work are the following:

1. A method for generating a structured representation suitable for building an INR from a 3D mesh.
2. Construction of a compact INR that very efficiently encodes a 3D surface represented as a mesh.
3. Compression of the trained INR to obtain state-of-the-art mesh compression results
4. The proposed overall scheme delivers very high compression outcomes which yield high-quality mesh reconstructions without significant increases in compute complexity.

2. Related Work

Our work furthers research on traditional and modern mesh compression algorithms, mesh surface subdivisions, and 3D surface representations using implicit neural networks.

Traditional methods of mesh compression can be broadly categorized into two types. The first type includes methods that aim for lossless preservation of vertex connectivity. These mostly involve algorithms that traverse the vertices of a mesh with minimal repeated visits, which avoids duplicate vertex references when representing connectivity. These either use data structures like triangle fans/ triangle strips([Dee95, Cho97, BPZ99]), spanning tree-based encoding ([TR98, DGGP05]), or triangle traversal encoding ([Ros99, GS98, SKR01]) These methods are often paired with quantization and predictive coding of 3D coordinates of the vertices ([Dee95, TR98, TG98]), and are now included in standard compression pipelines such as Google's Draco ([GHS*18]) which is used ubiquitously by the industry.

The second type of classical methods are *mesh simplification* methods, where the number of faces and vertices in the mesh are reduced while preserving as much geometric information as possible ([GH97, HG99]). However, The surface of a mesh can be *subdivided* to increase its vertex resolution ([CC78, Sab02]. In [Hop96], a method for progressively undoing the simplification is presented. A simplified surface can also be subdivided to recover the original vertex resolution, albeit with limited fidelity of the reconstructed approximation. Authors of [LMH00] use a displacement map along with a subdivision surface to reconstruct a mesh and as such is very related to our work. In their work, the displacement map is calculated as the signed distance between each vertex on the simplified and subdivided mesh, and the closest point on the surface of the original mesh. The major drawback is that if the simplified mesh has very low vertex resolution (which is desirable for compression), a greedy approach to locating the closest point on the original surface might yield unfavorable reconstructions. The survey [MLDH15] contains more comprehensive descriptions of such mesh compression techniques.

[LKC*20] and [CKAJ23] augment the reconstruction of original surfaces from their subdivided versions using learning-based approaches. Both methods involve training large graph neural network-based deep-learning models that can upsample the vertex resolution of any given mesh by subdividing it, and predicting displacement offsets for each vertex based on their local geometry. These methods use *successive self-parameterization* (SSP) ([LKC*20] to build paired a dataset of vertex-aligned low and high polygon count meshes. Their reliance on using only local geometry to compute displacement offsets and a single trained model to reconstruct all meshes limits the quality of their reconstructed outputs. Instead of training one model to process all meshes, the methods devised by [HML*21] and [SRL24] perform per-mesh optimization. These methods rely on appearance-driven optimization of mesh simplification and building memory-efficient geometric fields, respectively. Due to the high efficacy of

per-mesh optimization, Neural Geometric Fields (NGF)([SRL24]) achieves much better compression ratio-to-quality trade-offs than [CKAJ23] and [LKC*20]. Despite current state-of-the-art compression, appearance-driven optimization methods are prone to inaccurate reconstruction because they rely on minimizing the rendering loss. The inaccuracies arise because minimizing the rendering loss to learn deformations to be applied to coarse surfaces as done in [SRL24] is akin to a greedy search for the closest points on the original surface to the points on the coarse surface, which might not be ideal.

Mesh surfaces can also be cut along edge-paths and flattened and fit in a 2D square to obtain a "Geometry Image" ([GGH02,HP05]). This "Geometry Image" can be compressed using any popular 2D image compression methods, such as wavelet-based image encoders. Authors of [MAG*22] present a method to build compressed neural representations of Geometry Images of 3D surfaces by fitting surface-wise convolutional neural networks.

Per-surface optimization methods for building INRs and geometric primitives have also been successfully applied to compress SDFs ([SMB*20, PFS*19, AATDJ21, YTY*24]) and occupancy grids ([TSM*20, LWZ*21, TCY*21, TLY*21, MLL*21]). Many works also leverage hybrid representations. For example, [JYPH22] builds a generalized surface representation by introducing Coordinate Fields that are hierarchical voxel grids with latent codes per each cell that allow decoding the SDF. [GYW*19] partition meshes into bounding boxes and use a network to apply deformations on them. INR-based image ([SPY*22]) and video ([CHW*21]) compression methods have also been shown to outperform traditional methods. Quantizing the parameters of INRs has often led to significantly increased compression ratios in many works of this nature. For instance, [TET*22] compress implicit feature grids that encode SDFs using vector-quantized dictionaries, [VSW*23] build compressed representations of mesh texture by quantizing neural features, while [ZLL*24,GCML23,KY22] quantize the parameters of a neural radiance field (NeRF) [MST*20] to build compact reconstructions of 3D scenes.

Inspired by the success of per-mesh and per-surface optimization methods and INRs for compression, we propose a method of mesh compression by encoding the displacement field as a compact neural network that is trained specifically for each mesh. The displacement field can refine the simplified version of the coarse mesh and build an accurate reconstruction of the original surface.

3. Neural Mesh Compression

3.1. Prerequisite: Successive Self-Parameterization

Successive self-parametrization, proposed by [LKC*20], is a method for obtaining a bijective mapping between the surface of a mesh and its simplified representation. We employ this technique in our method for remeshing the surface to be compressed and thus include this subsection for readers' ease of understanding. SSP takes a triangle mesh and an edge-collapse algorithm as input, then generates a decimated triangle mesh along with a bijective map between the surfaces of the original and decimated meshes. This is achieved by simultaneous collapse operations in the 3D domain and the UV domain consisting of the flattened 1-ring of the edge as shown in Figure 2 (borrowed from [LKC*20]). Due to the imposed boundary constraint in the UV domain, there is a bijective mapping between the UV patches before and after the edge collapse. The bijective map in the UV domain also allows a bijective mapping between the pre- and post-collapse surfaces in the 3D domain.

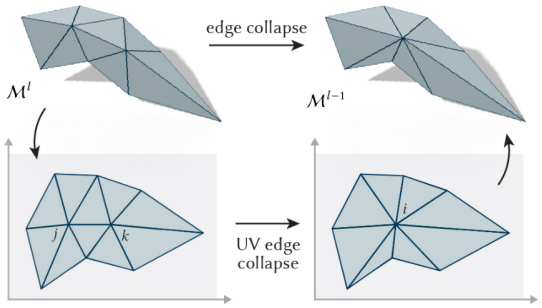

Figure 2: *Successive self-parameterization. (Figure borrowed from [LKC*20])*

3.2. Method Overview

Similar to methods like [TLY*21] and [SRL24], we train a neural network model specifically for each mesh to be compressed. Our method for building compressed representations of meshes can broken down into the following steps and is also illustrated in Figure 3:

1. Decimate the mesh to be compressed by reducing the number of faces by a large extent to obtain its coarse version, while simultaneously performing successive self-parametrization (see section 3.1) to obtain a bijective mapping between surfaces of the original and the decimated meshes.
2. Perform midpoint subdivision of the faces of the coarse mesh a desired number of times to increase the sampling resolution of the surface of the decimated mesh.
3. Use the bijective map generated by SSP to determine a displacement map for the vertices of the subdivided mesh, such that the displaced vertices reconstruct the original surface.
4. Overfit a simple multi-layer perceptron (MLP) that encodes the displacement of each vertex of the super-sampled subdivided mesh.
5. Prune, quantize, and entropy-encode the weights of the trained model.

The compressed representation of the mesh now consists of the coded weights of the overfitted network and the coarse mesh. To decode and reconstruct the mesh, the compression can be reversed according to the following steps:

1. Decode the entropy-coded quantized weights and load them to the model.
2. Perform midpoint subdivision of the coarse mesh the same number of times as in the encoding process.
3. Use the quantized model to compute the displacement that needs to be applied to each vertex on the subdivided mesh.
4. Apply the computed displacement to obtain a decoded approximation of the original surface.

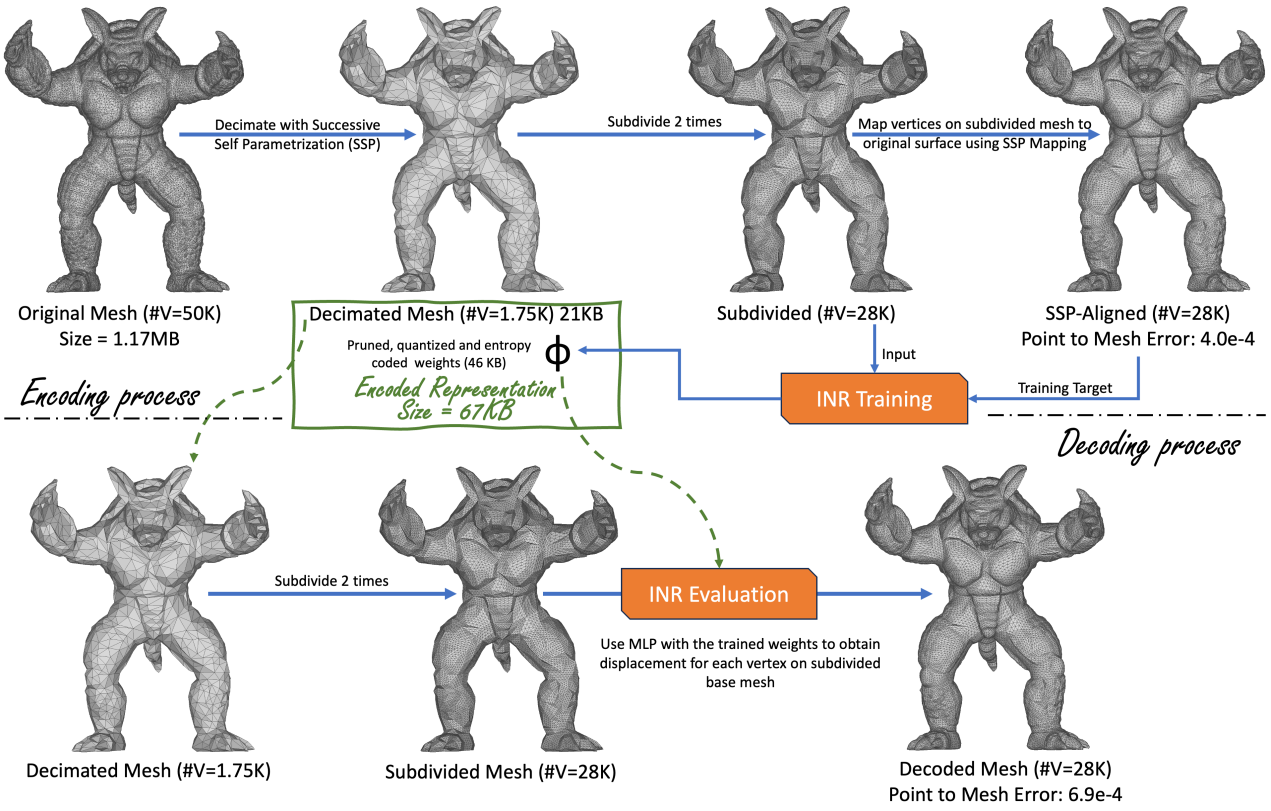

Figure 3: *Overview of Neural Mesh Compression*

We explain the design of the encoding process of Neural Mesh Compression in the subsequent subsections.

3.3. Generating Training Data

As our method involves training a neural network specific to each mesh, a training dataset with paired inputs and targets has to be generated for each mesh to be compressed. Given a mesh $M_{orig} := (V_{orig}, F_{orig})$ to be compressed, we define its surface ($S_{orig} \subset \mathbb{R}^3$) as the set of its vertices, all points lying on its edges, and all points lying within its faces. First, M_{orig} is simplified using `QSLIM` ([GH97]) to obtain M_{coarse}, which is a very coarse approximation of the original, such that $|V_{orig}| \gg |V_{coarse}|$. The simplification process is performed with SSP to obtain a bijective function $f_{SSP} : S_{coarse} \rightarrow S_{orig}$. Thus, the original surface can be reconstructed by transforming each point on the coarse surface using the mapping. This process can be formulated as $S_{orig} = \{f_{SSP}(p) | p \in S_{coarse}\}$. Then, a displacement field, $\mathcal{F} : S_{coarse} \rightarrow \mathbb{R}^3$ can be defined such that $\mathcal{F}(p) = f_{SSP}(p) - p, \forall p \in S_{coarse}$. With this proposed formulation, the task of mesh compression can be converted into the task of compressing the field $\mathcal{F}$ that is defined on a topological surface S_{coarse}. INRs have been shown to be amenable to compressing such fields in works by [SMB*20] and [YRSh22].

So far, with this method of point-wise surface mapping, infinite pairs of points and their corresponding displacements can be sampled, yielding data that can be used to train an INR that perfectly captures the geometry of the surface. This, however, still leaves the task of compressing the vertex connectivity of the mesh. When conducting reconstruction and decoding, the point cloud obtained by evaluating the INR can be triangulated to obtain a triangle mesh, but errors or inaccuracies in the trained INR can lead to poor triangulation. Also, an INR training pipeline with randomly sampled pairs in each batch tends to slow the training down significantly due to the high time cost of evaluating $\mathcal{F}(p)$ relative to performing gradient descent on a small MLP.

Pre-defining a subdivision scheme, and hence also a surface sampling scheme, addresses both of these issues. We employ a midpoint subdivision scheme to increase the vertex resolution of the coarse surface, whereby a new vertex is introduced at the midpoint of each edge of the mesh. We subdivide M_{coarse} a fixed number of times (say s times) to obtain $M_{subdivided}$, such that $|F_{subdivided}| = 4^s \times |F_{coarse}|$. Note that this subdivision scheme ensures that the same vertex connectivity is maintained during the encoding and decoding process. This implies that no additional encoding is required for reconstructing connectivity. Additionally, because the vertex coordinates are the same during the encoding and decoding processes, the vertices introduced by successive subdivisions are also exactly the same. This implies that the inputs to the INR during training and decoding can be predetermined. The predetermination of inputs allows caching the training dataset, since the

paired set $\{(\mathcal{F}(p), p) : \forall p \in V_{subdivided}]\}$ need only be computed once, and each sample can be used across multiple training batches. It is crucial to note that finite sampling of S_{coarse} implies that only an approximation of S_{orig} is used as the training target for the INR. In section 4.2, we find error bounds enforced by this approximation.

A neural network can now be fitted to this dataset to obtain an INR of the displacement field needed to reconstruct an accurate approximation of S_{orig} from $S_{subdivided}$.

3.4. Neural Mesh Compression Architecture

As deep neural networks are great universal function approximations, we employ a multi-layer perception (MLP) with parameters θ to approximate the displacement field $\mathcal{F}$ to be applied on $V_{subdivided}$. The proposed architecture for mesh compression consists of generating positional embedding of inputs fed to a dense multi-layer perceptron in which the first few layers are augmented with newly proposed one-ring feature aggregation.

Instead of feeding the 3D coordinates of vertices $p \in V_{subdivided}$ (where $V_{subdivided} \subset \mathbb{R}^3$) to the network directly, we first perform **positional encoding** to embed them in a higher dimensional frequency space, similar to [MST*20]. Specifically, we map each real-valued coordinate from $\mathbb{R}$ to the frequency space according to Equation 1 to get a vector in $\mathbb{R}^{2Q}$, where Q is a hyperparameter. As this transformation is performed on each of the three vertex coordinates, the coordinates $V_{subdivided} \subset \mathbb{R}^3$ get mapped to vectors having 6Q dimensions, which are then fed to the MLP. This addresses the issue of neural networks being biased towards learning low-frequency functions that was studied by ([RBA*18]).

$$\gamma(p) = (\sin(\pi p \cdot 2^0), \cos(\pi p \cdot 2^0), \ldots, \sin(\pi p \cdot 2^{Q-1}), \cos(\pi p \cdot 2^{Q-1})) \quad (1)$$

Additionally, we observe that having access to the local geometry of a vertex improves the ability of the neural network to encode $\mathcal{F}$. To that end, we devised **one-ring feature accumulation**. Given a vertex i being processed, its activations at layer l (f_i^l) are aggregated with the activations of the vertices in its one-ring neighborhood ($\hat{f}_j^l \forall j \in \mathcal{N}(i)$) as shown in Equation 2, where w^l and b^l are the weights and bias of the l^{th} layer of the main branch of the MLP, and $\hat{w}^l$, $\hat{b}^l$ are the weights and biases of the l^{th} auxiliary branches of the MLP responsible for computing the neighborhood features, and e_{ij} is the length of the edge connecting vertices i and j. This operation is also visualized in the Figure 3.4. This operation is a more localized and runtime-memory-efficient variant of graph convolutions. Performing this operation instead of regular graph convolutions reduces the time and GPU memory required to train the network drastically, while yielding similar reconstruction quality (See section 4.2). Only the first few layers of the multi-layer perceptron are augmented with one-ring feature accumulation. The feature accumulation process is:

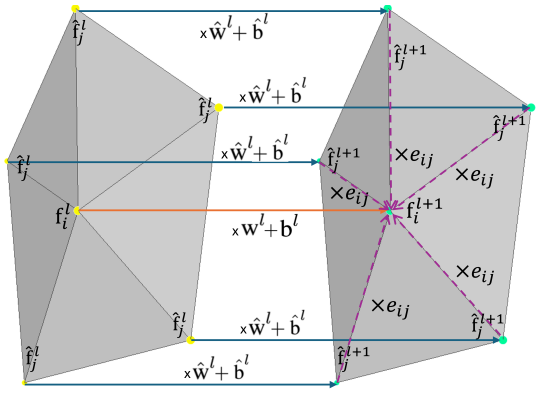

Figure 4: *One-ring feature accumulation*

$$\hat{f}_j^{l+1} = \hat{w}^l \cdot \hat{f}_j^l + \hat{b}^l, \forall j \in \mathcal{N}(i)$$
$$f_i^{l+1} = (w^l \cdot f_i^l + b^l + \sum_{j \in \mathcal{N}(i)} (\hat{f}_j^{l+1} \cdot e_{ij})) \times 0.5 \quad (2)$$

The difference between one-ring accumulation and graph convolution operations [KW17, MLR*20] is subtle yet significant. In a graph neural network built with N graph convolution layers in series, any vertex i will be influenced by all the vertices that are at most N edges away from i. However, in the case of an N-layered MLP with one-ring accumulation, any vertex i is only influenced by vertices that are at most one (and not N) edges away. This is because the features $\hat{f}$ of each vertex are obtained from an independent MLP before being aggregated with features "f" of their one-ring neighbors that are extracted from the main MLP, as described in Equation 2.

We advise against using batch normalization in the neural network. Since building an INR involves overfitting on the training dataset, using batch normalization modifies the computed intermediate features based on the other values in the training mini-batch. This creates an effect of pseudo-randomization of the features of a sample during training, due to the impact of other samples in its batch which hinders the overfitting process. In our implementation, we use layer-normalization to ensure deterministic training.

The overall architecture of the INR is quite simple. It is a multi-layer perceptron with l hidden layers and k features. Among the h hidden layers, the first g layers are augmented with one-ring feature accumulation. Section 4.1 provides the exact values of these and other hyper-parameters.

3.5. Compressing The INR Parameters

We employ pruning, quantization, and entropy coding to further compress the parameters θ of the MLP that encodes the displacement field.

L1 unstructured pruning zeroes those parameters with the lowest $(1-S) \times 100$ percentile of L1-norm, to obtain a sparsity factor of S (the fraction of non-zero parameters). Instead of pruning the

model to obtain a sparsity factor of S in one shot after training, we prune it progressively, which involves pruning the model in multiple steps during the course of model training. To reach a target sparsity of $S\%$, the model can be pruned and fine-tuned alternatively z times where each iteration of pruning introduces sparsity by a factor of $s = \sqrt[z]{S}$.

All the parameters of the INR that were originally represented in a 32-bit precision floating point format, are quantized to 8-bit integers. We employ a simple **post-training quantization** strategy, where all parameters are first normalized to $[0,1]$, scaled by 256, and rounded to the nearest integer. However, we dynamically quantize the activations based on their range during inference to improve accuracy.

One must note that, while pruning may suppress a parameter to zero, it still occupies 32 bits (or 8 bits after quantization) of storage. To reap the benefits of improved compression, the network must be pruned to a high factor (≤ 0.6 for instance), leading to a high frequency of zeroes in the list of parameters. **Entropy coding** can then yield a significantly more compact representation. To this end, we apply Huffman Coding on the quantized weights, to obtain a bit-stream that can be decoded losslessly.

4. Experiments & Results

We demonstrate the efficacy of our compression algorithm on meshes spanning a diverse range of surface characteristics, genera, and triangle counts. All meshes are normalized such that the largest dimension of their bounding boxes is one unit. Our method produces better compression outcomes compared to the baseline methods over a broad range of compression ratios, as shown in Table 1, Table 2, and Figure 9. Our method outperforms the baselines by much larger extents when the compression ratios are high or the mesh to be compressed has high geometric complexity (see Figure 10). Our choice of baselines includes the following:

1. `NGF` (Neural Geometric Fields) ([SRL24]): A method for neural mesh representations and the current state-of-the-art for mesh compression.
2. `QS-DRC`: As our method and NGF allow remeshing the meshes during the compression, it is unfair to compare these methods with methods like Draco ([GHS*18]) that encode the vertex connectivity losslessly and use quantization of vertex coordinates. Thus for a fairer comparison, we first simplify meshes to a certain extent using QSLIM ([GH97]) and then compress the simplified connectivity using Draco. We perform a grid search for each mesh to find the target faces for decimation with QSLIM and the number of bits for vertex quantization that minimize the compression error. We dub this method as `QS-DRC`, shorthand for QSLIM-Draco. Using QS-DRC provides a much more competitive baseline compared to using QSLIM and Draco independently.

`NGF` minimizes the rendering loss to reconstruct the geometry of meshes. This process is sensitive to hyperparameters such as rendering resolution, camera intrinsics, camera distance, and (anti-)aliasing. On the other hand, our method utilizes a simpler training objective of fitting an MLP to the displacement field generated using SSP, allowing it to achieve better metrics. However, It is important to note that `NGF` does not employ pruning, post-training quantization, or entropy coding of parameters. Thus for a fair comparison, we also included the performance of our method without applying the post-training model compression pipeline in the "Ours NPQC" (No Pruning Quantization and Coding) columns of Tables 1 and 2. Ours NPQC outperformed `NGF` in terms of d_{pm}. On the other hand, Ours NPQC lags `NGF` in terms of d_{norm}, which may be due to the explicit minimization of normal error in the pipeline of `NGF`. Conversely, applying similar post-training compression to NGF does not yield better results. This is because `NGF` consists of an MLP with just two hidden layers, which only consumes a small fraction of the overall memory, while most of the used memory is occupied by per-vertex features. When vertex-wise features are pruned and quantized, their performance deteriorates severely as shown in Figure 6.

We also compared our method against Geometry Images [GGH02] and Neural Convolutional Surfaces (NCS) [MAG*22] as shown in Figure 7. These methods involve encoding position or displacement maps of mesh surfaces in the two-dimensional UV plane, using standard image compression techniques and are thus broadly related to our work. Our method performs much better than those owing to the higher compressibility of INRs and the more accurate ground truth displacements predicted by SSP.

Additionally, we also compare our method with Neural Progressive Meshes (NPM) ([CKAJ23]) in Figure 5. With its code, trained model, and training dataset not being available publicly, we present comparisons only with a few of the meshes whose images are available in their paper and were easily found on the internet.

The size of original meshes reported in Tables 1 and 2 is equal to $32 \times 3v + 3f \log_2 v$ bits, where v and f are the number of vertices and faces in the mesh respectively. The size of the coarse meshes which are a part of the compressed representation are also governed by the same formula. In practice, the coarse meshes included in our compressed representation can be represented much more compactly by applying any of the lossless mesh compression methods (such as the ones mentioned in Section 2) to yield even higher compression ratios without any reduction of quality.

To evaluate the quality of a surface's reconstruction, we measure the mean point to mesh distance (d_{pm}) and the mean normal error (d_{norm}). To calculate d_{pm} between meshes M_a and M_b, we first sample 1 million points on the surface of M_a, then for each of these points we find the closest point on the surface of M_b and measure the average distance between them to obtain $d_{a\to b}$. Similarly, we also evaluate $d_{b\to a}$. Finally, $d_{pm}(\mathrm{M_a}, \mathrm{M_b}) = d_{a\to b} + d_{a\to b}$. In a similar fashion, d_{norm} is calculated to be the average difference between the angles of the normal vectors (in degrees) of the same set of points used during the evaluation of d_{pm}.

These metrics for evaluation are different from those used by authors of recent works. [SRL24] measure the Chamfer distance between the vertices of M_a and M_b. This method is sensitive to vertex resolution, which is not truly representative of the geometric information a mesh depicts. Measuring Chamfer distance between the sets of vertices in the meshes also fails to account for the continuous surface manifold. On the other hand, [CKAJ23] only report

Compressed Sizes	85KB				130KB				187KB				260KB			
Mesh (Size MB)	Ours	Ours NPQC	NGF	QS-DRC	Ours	Ours NPQC	NGF	QS-DRC	Ours	Ours NPQC	NGF	QS-DRC	Ours	Ours NPQC	NGF	QS-DRC
Armadillo (1.14)	**6.92**	12.03	13.54	8.71	**4.91**	9.55	9.80	6.07	**3.46**	7.25	8.42	5.01	2.70	5.53	7.67	**2.55**
Dragon (12.51)	**18.15**	26.44	26.66	29.11	**11.24**	20.72	18.60	16.70	**8.51**	15.33	14.97	11.83	**6.46**	13.88	13.37	7.50
Einstein (8.44)	**7.90**	14.42	17.90	19.12	**5.23**	9.89	8.90	9.29	**3.90**	8.04	8.05	5.53	**2.63**	6.11	5.41	3.12
Ganesha (31.61)	**7.50**	14.06	13.75	16.61	**4.80**	9.28	9.89	10.77	**3.87**	8.46	8.66	7.38	**2.89**	6.42	5.81	6.03
Gargoyle (2.32)	**4.38**	9.30	12.27	16.36	**2.85**	6.87	8.70	7.35	**2.33**	4.97	6.69	5.62	**1.69**	3.76	5.11	2.07
Gnome (2.25)	**7.22**	10.61	8.43	12.93	**3.73**	7.75	6.49	8.33	**2.16**	4.98	4.97	5.94	**1.65**	3.52	3.24	2.25
Head (4.23)	**3.61**	6.75	10.04	13.61	**2.37**	5.00	6.16	8.23	**1.82**	3.67	5.32	4.36	**1.43**	2.82	4.57	1.67
Lucy (6.08)	**5.22**	9.69	12.94	21.69	**3.16**	7.16	10.57	8.74	**2.40**	5.23	4.54	3.99	**1.79**	3.94	4.28	2.88
Metatron (1.63)	**3.90**	7.39	14.65	8.95	**2.71**	5.35	12.36	6.15	**2.04**	3.91	11.65	3.17	**1.53**	2.89	9.53	1.74
XYZ (2.95)	**6.90**	12.92	14.41	22.60	**4.61**	8.34	12.64	9.63	**3.36**	6.14	8.04	6.84	**2.70**	5.37	6.79	3.98
Mean Error	**7.17**	12.36	14.46	16.97	**4.56**	8.99	10.41	9.12	**3.39**	6.80	8.13	5.97	**2.55**	5.42	6.58	3.38

Table 1: *Each mesh is first compressed to 4 bitrates using our method, our method without pruning, quantization, and entropy coding of parameters (Ours NPQC), NGF, and QS-DRC. Each cell denotes d_{pm} scaled $\times 10^4$ evaluated between the original mesh and the reconstructed meshes. The number of faces in the selected meshes varies from 50K in **Armadillo** to 2.1M in **Ganesha**. **Armadillo**, **Lucy**, and **XYZ** are courtesy of Stanford's 3D scanning repository; **Ganesha** is courtesy of Peel3D, while **Dragon**, **Einstein**, **Metatron**, and **Gargoyle** are available in Thingi10K ([ZJ16]).*

Compressed Sizes	85KB				130KB				187KB				260KB			
Mesh (Size MB)	Ours	Ours NPQC	NGF	QS-DRC	Ours	Ours NPQC	NGF	QS-DRC	Ours	Ours NPQC	NGF	QS-DRC	Ours	Ours NPQC	NGF	QS-DRC
Armadillo (1.14)	**4.38**	6.18	5.57	5.20	**3.56**	5.28	4.12	3.83	**3.07**	4.74	3.82	3.17	2.93	4.15	3.80	**2.76**
Dragon (12.51)	**19.16**	21.12	19.69	28.29	16.38	19.84	**16.12**	19.79	**11.54**	14.33	11.79	12.57	**8.76**	11.63	10.36	10.16
Einstein (8.44)	**7.54**	8.66	7.89	12.46	**5.55**	7.23	6.32	6.25	**4.11**	6.86	4.55	5.01	**3.27**	4.89	3.98	4.19
Ganesha (31.61)	**8.85**	10.41	9.76	10.76	7.66	8.54	**7.60**	8.79	**5.14**	7.07	6.25	7.97	**4.16**	6.16	5.25	5.29
Gargoyle (2.32)	**2.54**	3.67	2.73	5.38	2.51	3.38	**2.50**	4.17	2.47	3.12	**2.39**	2.75	2.21	2.89	2.35	**1.91**
Gnome (2.25)	**6.02**	7.95	6.38	10.30	**3.62**	5.12	4.03	4.10	**2.32**	3.77	2.88	2.68	1.92	2.92	1.89	**1.77**
Head (4.23)	**2.18**	3.05	2.68	9.78	**1.83**	2.40	2.19	4.86	**1.40**	2.01	1.81	1.96	1.11	1.84	1.24	**0.95**
Lucy (6.08)	**6.39**	8.29	8.06	16.65	**4.98**	7.07	5.96	7.80	**3.64**	4.97	4.02	4.61	**2.93**	4.41	3.47	3.70
Metatron (1.63)	**3.95**	5.60	5.26	5.43	**3.10**	3.89	4.05	3.83	**2.20**	2.91	3.89	2.57	1.97	2.67	3.46	**1.84**
XYZ (2.95)	**7.59**	9.71	8.97	15.91	**6.16**	8.65	7.46	7.73	**4.67**	6.86	5.56	5.87	**4.05**	5.85	5.13	4.91
Mean Error	**6.86**	8.48	7.70	12.02	**5.53**	7.14	6.04	7.12	**4.06**	5.66	4.70	4.92	**3.33**	4.74	4.09	3.75

Table 2: *d_{norm} evaluated between the reconstructed and original meshes in degrees ($^\circ$) is shown for all meshes and methods.*

$d_{reconstructed \to original}$, which does not account for directional asymmetry.

4.1. Experiment Setup

We used our method to compress all meshes to 4 bitrates - 85, 130, 187, and 260 kilobytes(KB). The hyperparameters of our method to obtain these bitrates are shown in table 4. The number of layers with one-ring feature accumulation, $g = 4$, was the same for all configurations. We also fixed Q, the dimensionality of the positional embedding space to be 10 in all configurations. In all cases, the network was optimized using the AdamW optimizer [LH19] to minimize the mean squared error between the displacements predicted by the MLP, and the ground truth vectors computed using SSP. The optimization was performed over 3500 epochs with a batch size of 2048, and an initial learning rate of $1e-3$ with a cosine decay scheduler. We obtained an overall pruning ratio of 50% through $z = 5$ progressive pruning iterations, with an 87% pruning ratio introduced in each step. The pruning was performed after epoch numbers 200, 400, 600, 800, and 1000. Please visit the project website for the complete implementation.

V_{coarse}	2 subdivisions				3 subdivisions				4 subdivisions			
	SSP GT	Without Pruning	With Pruning	After Quantizing	SSP GT	Without Pruning	With Pruning	After Quantizing	SSP GT	Without Pruning	With Pruning	After Quantizing
500	28.64/30.24	29.23/30.24	29.23/30.25	31.01/30.43	15.57/26.25	16.15/26.65	16.42/26.67	18.48/27.20	8.21/22.42	8.79/22.86	8.88/22.89	9.72/23.41
1000	22.46/26.25	23.51/26.28	23.55/26.30	24.01/26.48	12.11/21.69	13.27/21.98	13.35/22.02	14.24/22.23	6.15/17.03	6.59/17.27	6.84/17.47	7.55/18.40
2000	9.31/17.58	9.89/17.64	9.93/17.64	10.63/17.65	4.42/11.54	6.24/11.67	6.29/12.04	7.59/12.62	2.00/6.33	5.84/8.03	6.11/8.41	6.81/8.98
4000	4.91/11.86	5.56/11.88	5.95/12.03	6.52/12.14	2.22/6.37	5.63/8.04	5.95/8.28	6.46/8.76	0.96/3.07	5.42/7.55	5.81/7.97	6.22/8.83
8000	2.68/7.72	4.80/8.18	5.09/8.58	5.64/8.97	1.24/3.64	5.16/7.09	5.49/7.78	6.01/8.51	0.63/2.49	4.93/6.79	5.33/7.61	5.95/8.23

Table 3: *Effect of the size of the coarse mesh and the number of subdivisions on the reconstruction quality when the hyperparameters of the INR model are fixed at $l = 32, k = 96$. **SSP GT** shows the reconstruction quality of the re-meshed approximation of the **dragon** mesh used as ground truth for training the INR. The reconstruction quality before and after pruning and quantization are also shown. Each cell shows d_{pm} and d_{norm} (in degrees) for the corresponding configuration.*

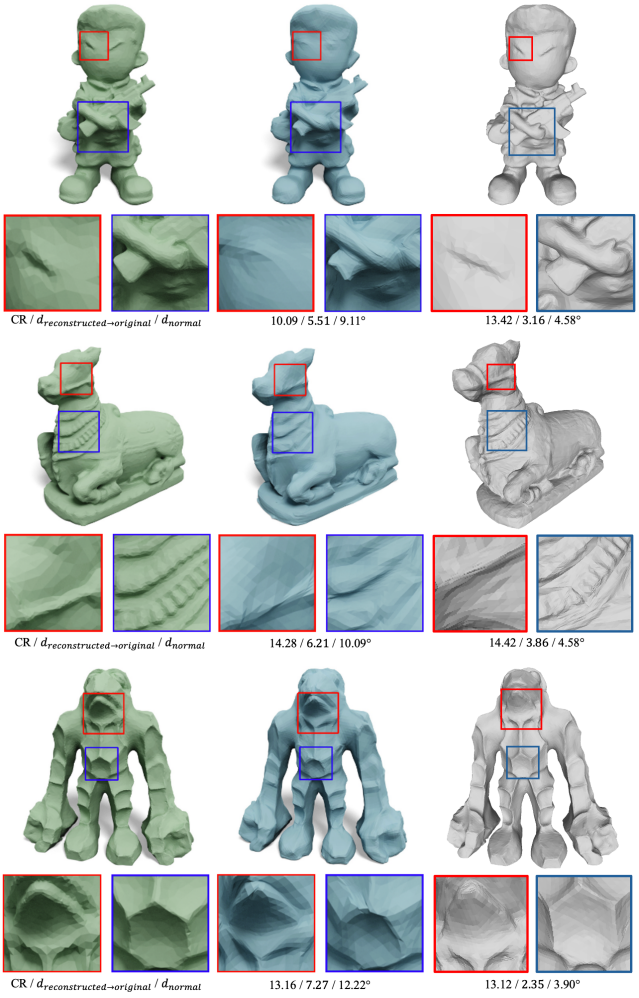

Figure 5: *Comparison with Neural Progressive Meshes (NPM). The mesh in the left column (green) is the original, the middle column (blue) is NPM, and the right (gray) is ours. The point-to-mesh errors are scaled $\times 10^4$*

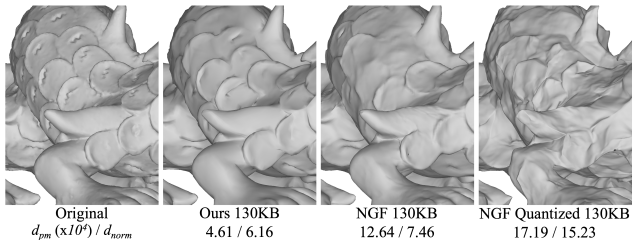

Figure 6: *Pruning and quantizing NGF leads to a rough and spiky geometry reconstruction.*

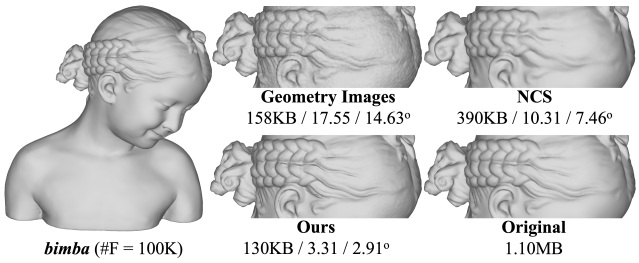

Figure 7: *Geometry Images [GGH02] shows quantization-like artifacts and NCS [MAG*22] fails to reconstruct finer details while our method retains the geometry more faithfully. The size of the compressed representation, $d_{pm}(\times 10^4)$ and d_{norm} are displayed for each method*

4.2. Ablation Studies

As explained in section 3.3, performing a finite number of subdivisions of a coarse mesh will only lead to an approximate reconstruction of the original mesh surface. Using this approximation as the dataset for training the INR introduces bounds on reconstruction quality. This is because the sampling error cannot be recovered even if the INR encodes the sampled displacement field perfectly. The **SSP GT** columns of table 3 show the reconstruction accuracy of the re-meshed approximations generated using SSP which were

Compressed Size	Vertices in coarse mesh (= V_{coarse})	Number of sub-divisions (= s)	Number of hidden layers (= l)	Hidden layer size (= k)
85KB	2000	2	20	56
130KB	2500	2	24	70
187KB	3000	3	28	82
260KB	3500	3	32	96

Table 4: *Hyper-parameters for different configurations that were used to compress meshes to different bitrates yielding the results in table 1.*

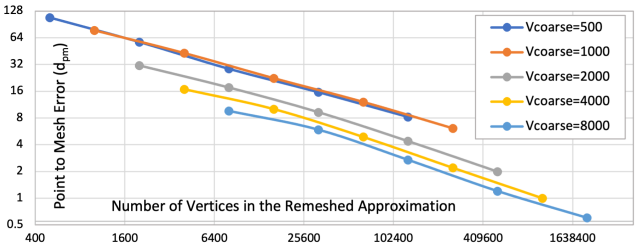

Figure 8: ***Dragon*** *mesh is first decimated to different resolutions (V_{coarse} and subdivided 0-4 times reaching different vertex resolutions of the remeshed approximations. Using a larger coarse mesh provides a better remeshed version with lower d_{pm} than having more subdivisions. The d_{pm} axis is scaled $\times 10^4$.*

used as the ground truth when training the neural representations. These act as an upper bound on the reconstruction quality achievable by the INR for a given configuration of coarse mesh size and number of subdivisions. The plot in Figure 8 shows that remeshing quality is better when larger coarse meshes are used compared to performing more subdivisions while the number of vertices in the remeshed approximation stays the same. However, unlike increasing the number of vertices in the coarse mesh, performing more subdivisions incurs no additional increase in the size of the compressed representation. Table 3 also shows the drop in reconstruction quality with the application of pruning and quantization while the size of the INR is fixed at $l = 32, k = 96$. It may be noted that pruning has minimal or no effect when the size of the INR used to encode the sampled displacement field has adequate learning capacity. While in such cases, the reconstruction quality of the remeshed ground truth (SSP GT) is the bottleneck, when the sampling rate of the ground truth displacement field increases (such as when V_{coarse} and number of subdivision s are high), the size of the INR model becomes the bottleneck with the reconstruction quality of the INR being much higher than the quality of the remeshed ground truth.

Prune+EC	Quantize	l=20,k=56	l=24,k=70	l=28,k=82	l=32,k=96
No	**No**	300KB	536KB	836KB	1284KB
No	**Yes**	75KB	134KB	209KB	321KB
Yes	**No**	226KB	453KB	685KB	804KB
Yes	**Yes**	45KB	79KB	102KB	160KB

Table 5: *Variation in storage size of MLP with pruning+entropy coding and quantization*

Method	Encoding Time	Decoding Time	GPU Memory
Ours (s=2)	17-22 mins	270 ms	1.7 GB
Ours (s=3)	65-70 mins	800 ms	2.5 GB
NGF (<140KB)	8-9 mins	200 ms	8 GB
NGF (>140KB)	25-28 mins	750 ms	8 GB
QS-DRC	3 mins	10 ms	None

Table 6: *Encoding and decoding times with GPU Memory Usage for various methods for the Ganesha mesh with 2.1M triangles. Note that the time for encoding using QS-DRC includes the time taken for performing the grid search which involves multiple decimation operations.*

Performing model quantization always incurs a reconstruction quality cost of approximately $+1.2$ points increase in d_{pm} and approximately $+0.5^\circ$ increase in d_{norm}. The reduction of memory due to quantization and pruning+entropy coding is shown in table 5.

Not using the one-ring feature accumulation reduces the d_{pm} and d_{norm} by factors of 15% and 4%, respectively on average. Using full graph convolutions improves d_{pm} by a factor of only 3% while increasing the training time by around 70% and the GPU memory usage by 300%.

4.3. Runtime and Memory Usage

Table 6 shows the runtime and memory usage of our method and the previous SOTA baseline when run on Nvidia GTX 1080Ti GPU for the Ganesha mesh with 2.1M triangles. The main hyperparameter that affects the runtime is the number of times the coarse mesh is subdivided (s). Increasing the s by one increases the number of samples in the training dataset four times, increasing the training time proportionally, while changing the size of the INR does not affect the training time drastically. The training dataset is generated only once per training at the beginning and involves subdividing the coarse mesh a fixed number of times and applying SSP for all vertices on the subdivided mesh. This process takes less than a minute. While providing significant improvement in compression quality, our method can compress meshes in the same order of magnitude of time and memory. The decoding speed of our method is practically instantaneous and similar to other methods that rely on per-mesh optimization. To improve the speed of encoding of our method, the number of epochs used for fitting the network can be reduced albeit at the cost of quality. We demonstrate these results in Table 7 While the encoding times are high for methods that rely on mesh-wise neural network optimization rendering them impractical for real-time and low-latency applications, they are much more suitable for distributing graphical assets to a broad audience, such as the distribution of video games, given the low decoding times.

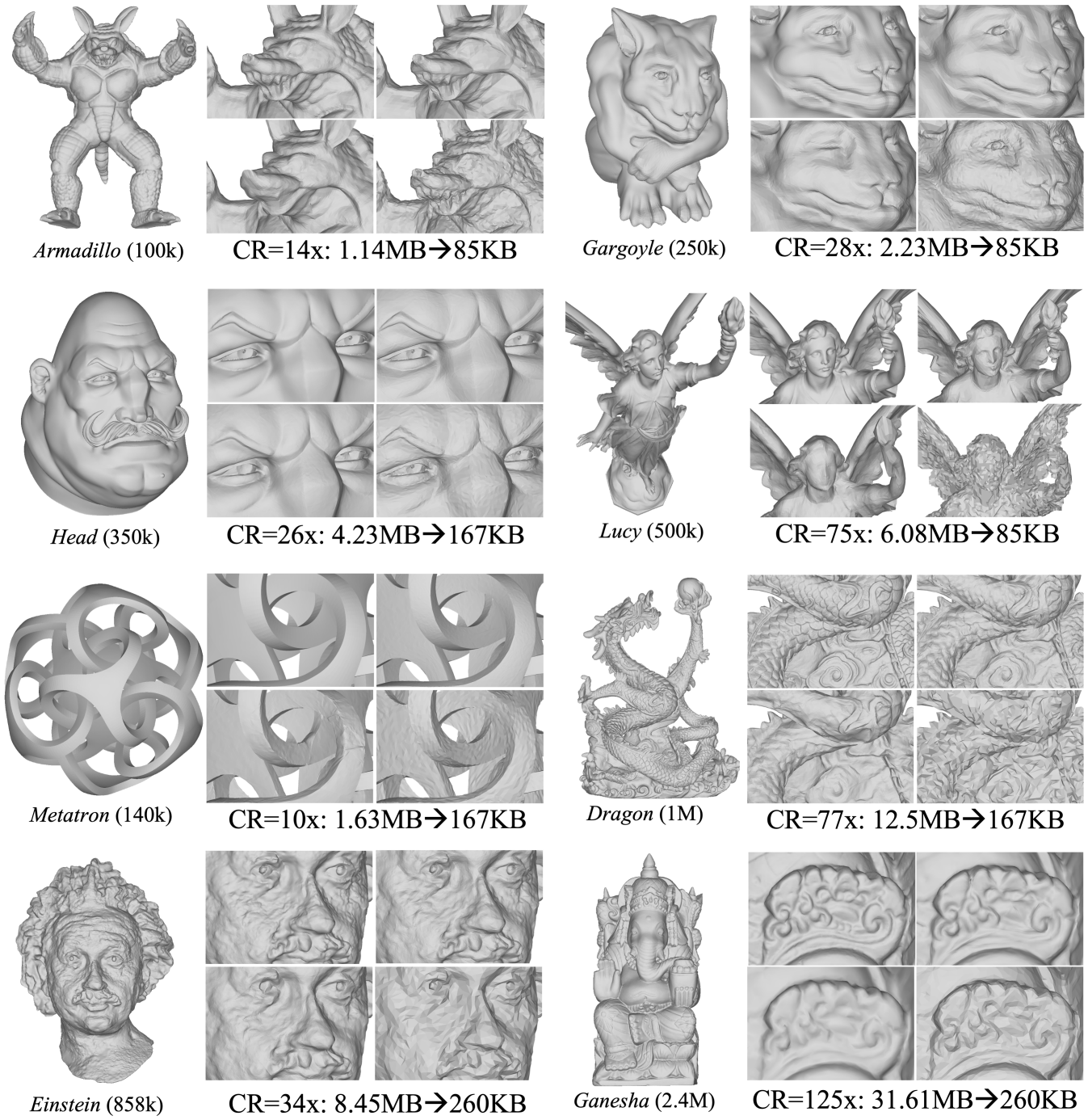

Figure 9: *Images of the outputs obtained by compressing and reconstructing various meshes (# faces in brackets) using different methods are shown for visual comparison. Starting from the top left in a clockwise order we show images corresponding to the original, our method, NGF, and QS-DRC respectively. Our method produces reconstructions with noticeably better visual fidelity compared to the baselines over a wide range of compression ratios by preserving finer details accurately. While NGF also performs admirable compression, it is prone to errors and reconstructing with reduced fidelity in complex geometric regions. When a mesh has high geometric complexity and/or when the compression ratio is high, QS-DRC tends to produce rough-looking surfaces due to the quantization of vertex coordinates and insufficient geometric resolution if the action of QSLIM is prominent.*

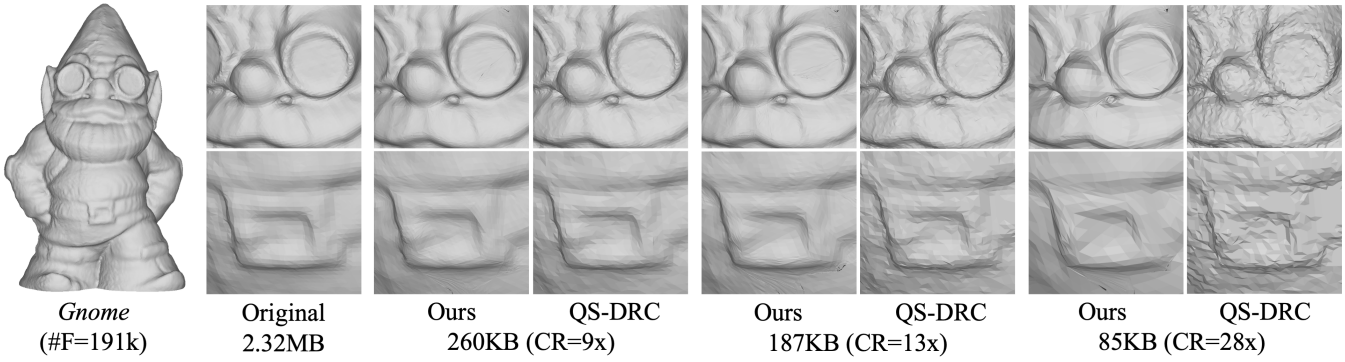

Figure 10: *The margin of improved reconstruction quality yielded by our method compared to QS-DRC increases with the compression ratio. For a mesh with relatively less geometric complexity such as the one shown here, QS-DRC performs relatively well compared to our method for lower compression ratios. In such cases, QS-DRC could be a more time- and energy-efficient option.*

Epochs	1000	2000	3000	4000	5000
Train Time (mins)	6	12	17	24	30
$d_{pm}(x10^4)$	5.89	5.55	5.19	5.13	5.08
d_{normal}	7.07	6.79	6.47	6.36	6.34

Table 7: *Effect of varying the number of epochs on the encoding quality for the Lucy model.*

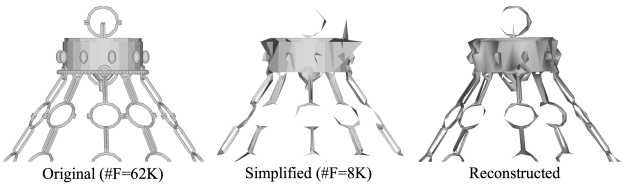

Figure 11: *The displacement fails to recover the geometry sufficiently after meshes with thin geometric structures undergo severe simplification.*

4.4. Failure Cases

The first step of our method involves building a coarse approximation of the surface to be compressed by simplifying the original mesh and reducing the number of triangles in it drastically (at least 50×). When simplified to a large factor, some meshes, such as the one shown in Figure 11, suffer from broken connectivity and structure. In such cases, the displacement field is insufficient to recover the geometric structure. Other methods that also rely on neural overfitting, such as [SRL24,JYPH22,CKAJ23], also suffer from the same limitations.

Our method also proves to be redundant when the mesh to be encoded is too simple and contains few geometrically detailed elements. In such cases, the displacement field would not significantly improve the reconstruction quality of the simplified surface and thus be redundant. For these meshes, QS-DRC would be more efficient due to its much faster encoding process and the loss in quality due to simplification and vertex quantization being negligible.

5. Conclusion and Scope for Future Research

This paper presents a state-of-the-art method for compressing 3D meshes that obtains up to 3× less d_{pm} than baselines for the same compression ratios. The margin of increase in the performance of our method over the existing method increases with the compression ratio allowing generation of more compact encoding of 3D triangle meshes. Despite the exceptional performance, our method has some current limitations that may motivate further research on this topic:

1. While per-mesh optimization methods like ours, [SRL24], and [TLY*21] can achieve impressive compression ratios, there is still a need to find ways to further accelerate the encoding speed of these methods to improve their practicality for more use cases.
2. While our method can be easily modified to compress UV parameters of vertices by performing mesh simplification in a manner that is UV-parameterization aware, there is also scope for extending the algorithm to compress the attribute maps as well by incorporating techniques proposed by [PRT*24] or [KPwG23].
3. As SSP relies on flattening mesh patches onto a 2D plane, it requires the surface to be edge-manifold. Because our method relies on SSP to build a training dataset, it has the same limitation. If the non-manifold aspect of the mesh needs to be preserved, as a workaround, the mesh could be partitioned by splitting the mesh along such edges and compressing the components independently. Otherwise, the surface could be remeshed to obtain a triangulation amenable to compression with our algorithm.

Acknowledgements
The authors thank the Texas Advance Computing Center and the National Science Foundation AI Institute for Foundations of Machine Learning (Grant 2019844) for providing compute resources that contributed to our research.

References

[AATDJ21] AUMENTADO-ARMSTRONG T., TSOGKAS S., DICKINSON S. J., JEPSON A. D.: Representing 3d shapes with probabilistic directed distance fields. *Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), 19321–19332. URL: https://api.semanticscholar.org/CorpusID:245117960. 3

[BPZ99] BAJAJ C. L., PASCUCCI V., ZHUANG G.: Single resolution compression of arbitrary triangular meshes with properties. *Computational Geometry 14*, 1 (1999), 167–186. doi:10.1016/S0925-7721(99)00026-7. 2

[CC78] CATMULL E. E., CLARK J.: Recursively generated b-spline surfaces on arbitrary topological meshes. *Seminal graphics: pioneering efforts that shaped the field* (1978). doi:10.1016/0010-4485(78)90110-0. 2

[Cho97] CHOW M. M.: Optimized geometry compression for real-time rendering. In *Proceedings of the 8th Conference on Visualization '97* (1997), VIS '97, p. 347–ff. doi:10.1109/VISUAL.1997.663902. 2

[CHW*21] CHEN H., HE B., WANG H., REN Y., LIM S. N., SHRIVASTAVA A.: Nerv: Neural representations for videos. In *Advances in Neural Information Processing Systems* (2021), Ranzato M., Beygelzimer A., Dauphin Y., Liang P., Vaughan J. W., (Eds.), vol. 34, pp. 21557–21568. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/b44182379bf9fae976e6ae5996e13cd8-Paper.pdf. 3

[CKAJ23] CHEN Y.-C., KIM V. G., AIGERMAN N., JACOBSON A.: Neural progressive meshes, 2023. arXiv:2308.05741. 2, 3, 6, 11

[Dee95] DEERING M.: Geometry compression. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques* (1995), SIGGRAPH '95, p. 13–20. doi:10.1145/218380.218391. 2

[DGGP05] DIAZ-GUTIERREZ P., GOPI M., PAJAROLA R.: Hierarchyless simplification, stripification and compression of triangulated two-manifolds. *Computer Graphics Forum 24*, 3 (2005), 457–467. doi:10.1111/j.1467-8659.2005.00871.x. 2

[DNJ21] DAVIES T., NOWROUZEZAHRAI D., JACOBSON A.: On the effectiveness of weight-encoded neural implicit 3d shapes, 2021. arXiv:2009.09808. 2

[GCML23] GORDON C., CHNG S.-F., MACDONALD L., LUCEY S.: On quantizing implicit neural representations. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)* (2023), pp. 341–350. doi:10.1109/WACV56688.2023.00042. 3

[GGH02] GU X., GORTLER S. J., HOPPE H.: Geometry images. *ACM Trans. Graph. 21*, 3 (2002), 355–361. doi:10.1145/566654.566589. 3, 6, 8

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (1997), SIGGRAPH '97, p. 209–216. doi:10.1145/258734.258849. 2, 4, 6

[GHS*18] GALLIGAN F., HEMMER M., STAVA O., ZHANG F., BRETTLE J.: Google/draco: a library for compressing and decompressing 3d geometric meshes and point clouds, 2018. URL: https://google.github.io/draco/. 2, 6

[GS98] GUMHOLD S., STRASSER W.: Real time compression of triangle mesh connectivity. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (1998), SIGGRAPH '98, p. 133–140. doi:10.1145/280814.280836. 2

[GYW*19] GAO L., YANG J., WU T., YUAN Y.-J., FU H., LAI Y.-K., ZHANG H.: Sdm-net: deep generative network for structured deformable mesh. *ACM Trans. Graph. 38*, 6 (Nov. 2019). doi:10.1145/3355089.3356488. 3

[HG99] HECKBERT P. S., GARLAND M.: Optimal triangulation and quadric-based surface simplification. *Comput. Geom. Theory Appl. 14*, 1–3 (1999), 49–65. doi:10.1016/S0925-7721(99)00030-9. 2

[HML*21] HASSELGREN J., MUNKBERG J., LEHTINEN J., AITTALA M., LAINE S.: Appearance-driven automatic 3d model simplification. In *Eurographics Symposium on Rendering* (2021). doi:10.2312/sr.20211293. 2

[Hop96] HOPPE H.: Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (1996), SIGGRAPH '96, p. 99–108. doi:10.1145/237170.237216. 2

[HP05] HOPPE H., PRAUN E.: Shape compression using spherical geometry images. In *Advances in Multiresolution for Geometric Modelling* (2005), Dodgson N. A., Floater M. S., Sabin M. A., (Eds.), pp. 27–46. doi:10.1007/3-540-26808-1_2. 3

[JYPH22] JIANG H., YANG H., PAVLAKOS G., HUANG Q.: Cofie: Learning compact neural surface representations with coordinate fields. arXiv:2406.03417. 2, 3, 11

[KPwG23] KNODT J., PAN Z., WU K., GAO X.: Joint uv optimization and texture baking. *ACM Transactions on Graphics 43* (2023). doi:10.1145/3617683. 11

[KW17] KIPF T. N., WELLING M.: Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)* (2017). URL: https://api.semanticscholar.org/CorpusID:3144218. 5

[KY22] KANG S., YOO S.: Ternarynerf: Quantizing voxel grid-based nerf models. In *2022 IEEE International Workshop on Rapid System Prototyping (RSP)* (2022), pp. 8–14. doi:10.1109/RSP57251.2022.10039009. 3

[LH19] LOSHCHILOV I., HUTTER F.: Decoupled weight decay regularization, 2019. arXiv:1711.05101. 7

[LKC*20] LIU H.-T. D., KIM V. G., CHAUDHURI S., AIGERMAN N., JACOBSON A.: Neural subdivision, 2020. arXiv:2005.01819. 2, 3

[LMH00] LEE A., MORETON H., HOPPE H.: Displaced subdivision surfaces. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (2000), SIGGRAPH '00, p. 85–94. doi:10.1145/344779.344829. 2

[LWZ*21] LIU Q., WANG P.-S., ZHU C., GAINES B. B., ZHU T., BI J., SONG M.: Octsurf: Efficient hierarchical voxel-based molecular surface representation for protein-ligand affinity prediction. *Journal of Molecular Graphics and Modelling 105* (2021), 107865. doi:10.1016/j.jmgm.2021.107865. 3

[MAG*22] MORREALE L., AIGERMAN N., GUERRERO P., KIM V. G., MITRA N. J.: Neural convolutional surfaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2022). URL: https://api.semanticscholar.org/CorpusID:247958185. 3, 6, 8

[MLDH15] MAGLO A., LAVOUÉ G., DUPONT F., HUDELOT C.: 3d mesh compression: Survey, comparisons, and emerging trends. *ACM Comput. Surv. 47*, 3 (2015). doi:10.1145/2693443. 2

[MLL*21] MARTEL J. N. P., LINDELL D. B., LIN C. Z., CHAN E. R., MONTEIRO M., WETZSTEIN G.: Acorn: adaptive coordinate networks for neural scene representation. *ACM Trans. Graph. 40*, 4 (2021). doi:10.1145/3450626.3459785. 3

[MLR*20] MILANO F., LOQUERCIO A., ROSINOL A., SCARAMUZZA D., CARLONE L.: Primal-dual mesh convolutional neural networks. *Advances in Neural Information Processing Systems 33* (2020), 952–963. URL: https://proceedings.neurips.cc/paper/2020/file/0a656cc19f3f5b41530182a9e03982a4-Paper.pdf. 5

[MST*20] MILDENHALL B., SRINIVASAN P. P., TANCIK M., BARRON J. T., RAMAMOORTHI R., NG R.: Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV* (2020). doi:10.1145/3503250. 3, 5

[PFS*19] PARK J. J., FLORENCE P., STRAUB J., NEWCOMBE R., LOVEGROVE S.: Deepsdf: Learning continuous signed distance functions for shape representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2019). doi:10.1109/CVPR.2019.00025. 2, 3

[PRT*24] PENTAPATI S. K., RAI A., TEN A., ATLURU C., BOVIK A.: Geoscaler: Geometry and rendering-aware downsampling of 3d mesh textures, 2024. arXiv:2311.16581. 11

[RBA*18] RAHAMAN N., BARATIN A., ARPIT D., DRÄXLER F., LIN M., HAMPRECHT F. A., BENGIO Y., COURVILLE A. C.: On the spectral bias of neural networks. In *International Conference on Machine Learning* (2018). URL: https://api.semanticscholar.org/CorpusID:53012119. 5

[Ros99] ROSSIGNAC J.: Edgebreaker: connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics 5*, 1 (1999), 47–61. doi:10.1109/2945.764870. 2

[Sab02] SABIN M. A.: Subdivision surfaces, 2002. doi:10.1016/B978-044451104-1/50013-7. 2

[SKR01] SZYMCZAK A., KING D., ROSSIGNAC J.: An edgebreaker-based efficient compression scheme for regular meshes. *Computational Geometry 20*, 1 (2001), 53–68. doi:10.1016/S0925-7721(01)00035-9. 2

[SMB*20] SITZMANN V., MARTEL J., BERGMAN A., LINDELL D., WETZSTEIN G.: Implicit neural representations with periodic activation functions. In *Advances in Neural Information Processing Systems* (2020), Larochelle H., Ranzato M., Hadsell R., Balcan M., Lin H., (Eds.), vol. 33, pp. 7462–7473. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/53c04118df112c13a8c34b38343b9c10-Paper.pdf. 3, 4

[SPY*22] STRÜMPLER Y., POSTELS J., YANG R., GOOL L. V., TOMBARI F.: Implicit neural representations for image compression. In *ECCV* (2022). doi:10.1007/978-3-031-19809-0_5. 3

[SRL24] SIVARAM V., RAMAMOORTHI R., LI T.-M.: Neural geometry fields for meshes. SIGGRAPH '24. doi:10.1145/3641519.3657399. 2, 3, 6, 11

[TCY*21] TANG J.-H., CHEN W., YANG J., WANG B., LIU S., YANG B., GAO L.: Octfield: Hierarchical implicit functions for 3d modeling. In *The Thirty-Fifth Annual Conference on Neural Information Processing Systems (NeurIPS)* (2021). URL: https://api.semanticscholar.org/CorpusID:240354073. 3

[TET*22] TAKIKAWA T., EVANS A., TREMBLAY J., MÜLLER T., MCGUIRE M., JACOBSON A., FIDLER S.: Variable bitrate neural fields. In *ACM SIGGRAPH 2022 Conference Proceedings* (2022). doi:10.1145/3528233.3530727. 3

[TG98] TOUMA C., GOTSMAN C.: Triangle mesh compression. In *Proceedings of the Graphics Interface 1998 Conference, June 18-20, 1998, Vancouver, BC, Canada* (1998), pp. 26–34. doi:10.20380/GI1998.04. 2

[TLY*21] TAKIKAWA T., LITALIEN J., YIN K., KREIS K., LOOP C., NOWROUZEZAHRAI D., JACOBSON A., MCGUIRE M., FIDLER S.: Neural geometric level of detail: Real-time rendering with implicit 3D shapes. doi:10.1109/CVPR46437.2021.01120. 2, 3, 11

[TR98] TAUBIN G., ROSSIGNAC J.: Geometric compression through topological surgery. *ACM Trans. Graph. 17*, 2 (1998), 84–115. doi:10.1145/274363.274365. 2

[TSM*20] TANCIK M., SRINIVASAN P. P., MILDENHALL B., FRIDOVICH-KEIL S., RAGHAVAN N., SINGHAL U., RAMAMOORTHI R., BARRON J. T., NG R.: Fourier features let networks learn high frequency functions in low dimensional domains. *NeurIPS* (2020). URL: https://papers.neurips.cc/paper_files/paper/2020/file/55053683268957697aa39fba6f231c68-Paper.pdf. 3

[VSW*23] VAIDYANATHAN K., SALVI M., WRONSKI B., AKENINE-MÖLLER T., EBELIN P., LEFOHN A.: Random-Access Neural Compression of Material Textures. In *Proceedings of SIGGRAPH* (2023). doi:10.1145/3592407. 3

[YRSh22] YIFAN W., RAHMANN L., SORKINE-HORNUNG O.: Geometry-consistent neural shape representation with implicit displacement fields. In *International Conference on Learning Representations* (2022). URL: https://openreview.net/forum?id=yhCp5RcZD7. 4

[YTY*24] YENAMANDRA T., TEWARI A., YANG N., BERNARD F., THEOBALT C., CREMERS D.: Fire: Fast inverse rendering using directional and signed distance functions. In *Winter Conference on Applications of Computer Vision (WACV)* (2024), pp. 3077–3087. URL: https://api.semanticscholar.org/CorpusID:254246228. 3

[ZJ16] ZHOU Q., JACOBSON A.: Thingi10k: A dataset of 10,000 3d-printing models. arXiv:1605.04797. 7

[ZLL*24] ZHANG Z., LU G., LIANG H., CHENG Z., TANG A., SONG L.: Rate-aware compression for nerf-based volumetric video. In *Proceedings of the 32nd ACM International Conference on Multimedia* (2024), pp. 3974–3983. URL: https://api.semanticscholar.org/CorpusID:273640488. 3