Neural Geometry Fields For Meshes

Venkataram Sivaram
ves223@ucsd.edu
University of California San Diego
San Diego, CA, USA

Ravi Ramamoorthi
ravir@cs.ucsd.edu
University of California San Diego
San Diego, CA, USA

Tzu-Mao Li
tzli@ucsd.edu
University of California San Diego
San Diego, CA, USA

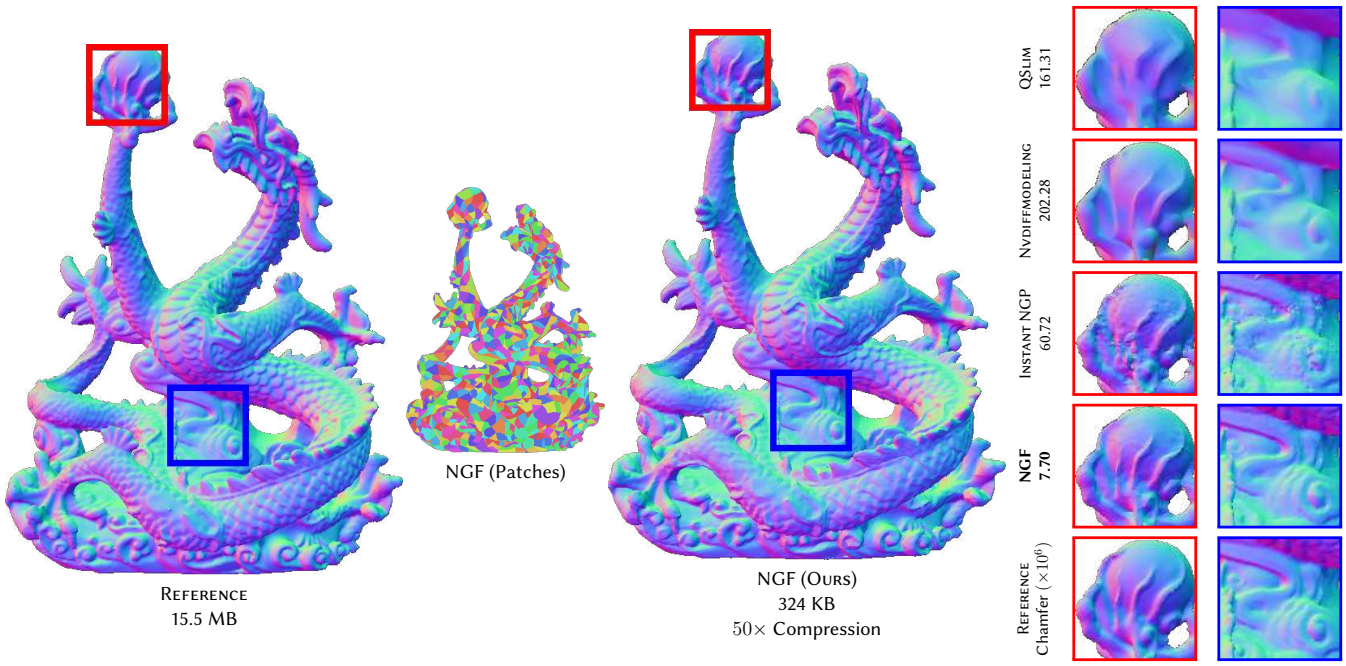

Figure 1: Neural Geometry Fields. We present a mesh-based neural representation for discrete surfaces that enjoys the benefits of both classical meshes (UV-parametrizable) and neural representations (compact). Given a target surface (Reference), we partition it into a set of quadrangular patches (NGF Patches). We then displace each patch with a coordinate neural network, and extract a standard triangle mesh from the patches and their displacement (NGF). An important application of our representation is mesh compression. On the right, we show comparison of discrete surfaces compressed by a classical mesh simplification algorithm QSlim [Garland and Heckbert 1997], an appearance-driven mesh processing method Nvdiffmodeling [Hasselgren et al. 2021], a neural implicit surface Instant NGP [Müller et al. 2022] (using marching cubes with similar triangle count to ours), and finally our representation (Ours). All methods are run with the same storage constraints as our method, and we show the Chamfer error next to the method. Despite the high compression rate of 50×, our method is capable of achieving significantly lower error and preserves visual appearance. The dragon model is courtesy of Thingi10K.

ABSTRACT

Recent work on using neural fields to represent surfaces has resulted in significant improvements in representational capability and computational efficiency. However, to our knowledge, most existing work has focused on implicit representations such as signed distance fields or volumes, and little work has explored their application to discrete surface geometry, i.e., 3D meshes, limiting the applicability of neural surface representations.

We present *Neural Geometry Fields*, a neural representation for discrete surface geometry represented by triangle meshes. Our idea is to represent the target surface using a coarse set of quadrangular patches, and add surface details using coordinate neural networks by displacing the patches. We then extract a traditional triangular mesh from a neural geometry field instance by sampling the displacement. We show that our representation excels in mesh compression, where it significantly reduces the memory footprint of meshes without compromising on surface details.

SIGGRAPH Conference Papers '24, July 27–August 01, 2024, Denver, CO, USA

ACM ISBN 979-8-4007-0525-0/24/07
https://doi.org/10.1145/3641519.3657399

CCS CONCEPTS

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

KEYWORDS

Neural representation, mesh simplification, mesh compression

ACM Reference Format:
Venkataram Sivaram, Ravi Ramamoorthi, and Tzu-Mao Li. 2024. Neural Geometry Fields For Meshes. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers '24 (SIGGRAPH Conference Papers '24), July 27–August 01, 2024, Denver, CO, USA*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3641519.3657399

1 INTRODUCTION

Neural surface representations have recently risen in popularity due to their advantages in information bandwidth and storage compactness, and compatibility with gradient-based optimization. The primary representation mechanism for recent works have been implicit functions such as signed distance fields [Park et al. 2019], occupancy grids [Mescheder et al. 2019] or volumes [Wang et al. 2021]. However, to use these representations in downstream tasks like scene modeling, surface texturing, or photorealistic rendering, an additional step is typically taken to convert these representations to meshes. This conversion step requires additional processing (e.g. marching cubes) and strips away the compact nature of these neural representations in favor of the more friendly meshes. We present a neural representation which foregoes this processing and storage overhead by directly generating meshes rather than implicit functions. As such, our representation is particularly suitable for mesh compression.

The primary challenge with representing meshes with neural graphics primitives is that additional connectivity information must be constructed for surfaces with different polygon schemes and topology. It is challenging to implement a gradient descent algorithm for optimizing connectivity data, and correspondingly there has been little previous work to our knowledge on representing meshes with neural networks. Fortunately, previous work on geometry images [Gu et al. 2002] unveils a regular image-based scheme that can be used to represent discrete meshes. We leverage this finding, and demonstrate that using a lightweight multilayer perceptron (MLP) to displace a coarse mesh can achieve state-of-the-art compression and representation of mesh geometry.

Specifically, we present a Neural Geometry Fields representation that consists of a set of quadrangular patches, which captures the shape of the given surface. Each patch is parametrized easily by construction, which allows us to attach a trainable feature field on the patches. We then feed the features to an MLP which outputs the displacement of the patch. To obtain a traditional mesh from our representation for both training and actual uses, we sample vertices from each patch and construct triangles within each patch. We then use an appearance-based loss [Hasselgren et al. 2021] to optimize for the patch vertices, the MLP weights, and the features. Figure 2 illustrates this process. We show that our representation can preserve details of surfaces even under significant compression rate, compared to both traditional mesh simplification and neural implicit surfaces (see Figure 1).

We present the following contributions:

(1) Combining surface partitioning and neural signal representations to forge a neural geometry fields representation for discrete surfaces.
(2) A coarse-to-fine appearance-driven optimization pipeline for overfitting Neural Geometry Fields to a particular reference mesh.
(3) A scheme for state-of-the-art mesh compression using our novel representation.

2 PREVIOUS WORK

Our work builds on classical mesh compression, subdivision techniques, geometry images and neural graphics primitives.

Classical mesh compression. We refer the readers to Maglo et al. [2015]'s survey for a comprehensive introduction. The most direct of the mesh compression methods applies single-rate compression techniques to reduce bit rates for vertex connectivity [Deering 1995; Gumhold and Straßer 1998; Touma and Gotsman 1998] and vertex data [Lee and Ko 2000; Taubin and Rossignac 1998]. Such methods are now incorporated into industry standard, e.g. Draco [Galligan et al. 2018]. These methods can be applied to our technique to further compress the patch representation. Neural geometry fields effectively act as a variable-rate compression on disjoint sections of the surface.

Other methods analyze the surface intrinsic information, using the mesh Laplacian [Lescoat et al. 2020] or error metrics [Cohen et al. 1998; Garland and Heckbert 1997; Hoppe 1996], to reduce the number of vertices or faces necessary for representing the shape of a reference mesh. Yet another class of method uses inverse rendering to find simpler meshes that render to the same images as the target surface [Hasselgren et al. 2021]. We build on both classes of methods: we apply QSlim [Garland and Heckbert 1997] to obtain a coarse representation of the target mesh that preserves the topology, and apply inverse rendering to refine our neural representation. Fundamentally, however, these methods build a lossy reconstruction of a target surface due to their sole use of polygonal meshes. By contrast, our representation preserves topological information using a coarse base mesh, while adding more details using coordinate neural networks.

Subdivision techniques. Subdivision surfaces splits polygonal faces into finer elements [Catmull and Clark 1978; Dyn et al. 1990; Hoppe et al. 1994; Loop 1987; Stam 1998]. In effect, this increases the vertex resolution of the mesh, and can be paired with displacement mapping to add details. In the context of mesh compression, a fixed subdivision scheme without displacement mapping [Lee et al. 2000] often leads to overly smooth reconstructions as a result of interpolating data with lower-order polynomials.

Neural networks have been applied to subdivision surfaces [Chen et al. 2023; Liu et al. 2020b] to achieve better matching results than classical subdivision. To extract a triangle mesh from neural subdivision surfaces, these methods rely on an expensive graph neural network. In contrast, our method sticks to simple interpolation and matrix multiply operations, and as a result is also efficient to evaluate interactively.

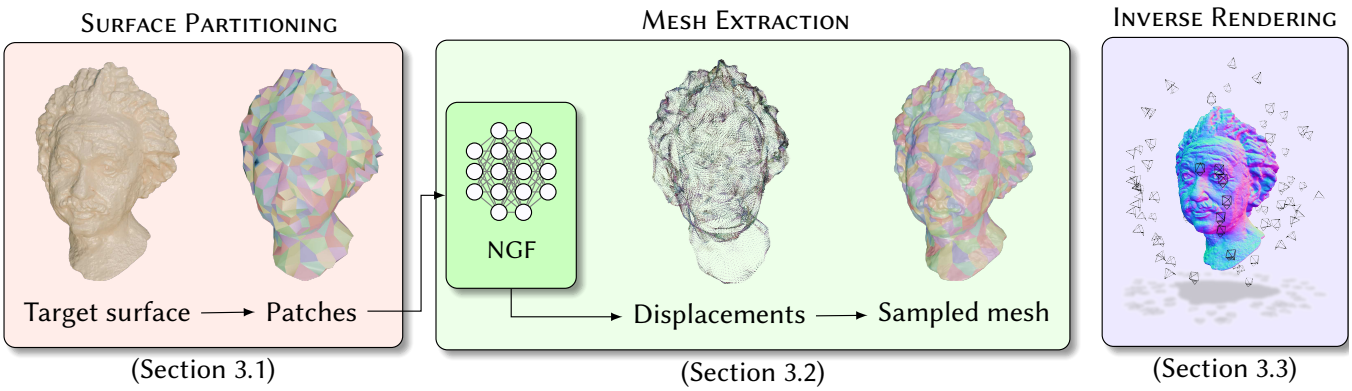

Figure 2: Overview. We overfit a neural geometry field representation to a specific target surface. We first pre-process the target mesh to obtain easily parametrizable patches, which enable the use of coordinate neural networks (Section 3.1). Then, we instantiate a neural geometry field with the patches and optimize the complete representation by repeatedly sampling triangle meshes (Section 3.2) and minimizing an inverse rendering loss (Section 3.3). The Einstein model is credited to Thingi10K.

Geometry images. The concept of representing surfaces as images, storing vertex information at each pixel, was first introduced with geometry images [Gu et al. 2002]. Geometry image construction cuts the surfaces so that they can be easily mapped to planar images, and then uniformly samples the surface to record 3D positions on the images.

The mesh can then be reconstructed implicitly by constructing quadrilaterals at each (2×2) set of pixels. Additionally, the constructed image can itself be compressed using traditional image compression techniques to reduce its footprint.

Our representation is derived from these works, in the sense that each patch operates similarly to a geometry image. Novel to our method is the introduction of a neural network that implicitly generates the vertices of the patches. This circumvents the problem of sampling and packing patches into texture atlases.

Neural graphics primitives. Coordinate neural networks have been used for representing 2D images, 3D scenes, and surface geometry [Martel et al. 2021; Mildenhall et al. 2020; Park et al. 2019; Tancik et al. 2020; Xie et al. 2022]. Previous works have shown that neural fields can represent high-fidelity signals with little use of memory. We build on the recent advances in neural fields and encode surface displacements using a feature field [Müller et al. 2022] followed by a shallow neural network.

Work so far in implicit surface representation has focused primarily on using neural networks to represent signed distance functions or volumetric occupancy grids [Mescheder et al. 2019; Park et al. 2019; Takikawa et al. 2021; Wang et al. 2021]. To improve computation and memory efficiency, hierarchical 3D spatial data structures are often used for storing features [Martel et al. 2021; Müller et al. 2022; Takikawa et al. 2021]. However, 3D voxels become inefficient when considering functions on surface manifolds, and primitives embedded within the surface itself would form a more ideal partitioning. In our method, we rely on a quadrilateral mesh to form the surface partition, as quadrilaterals provide simple and efficient parametrization domains for interpolation. Our meshes lead to more compact and efficient data structures for storing feature fields for surfaces.

Table 1: Notation. Table of notation used in describing our method.

Symbol	Definition
Γ	Target surface
Λ	Piecewise continuous surface defined by an NGF
Σ	Base surface
Ψ	Feature field on Σ
$\mathcal{P}$	A partition of Σ into patches
σ	A quadrilateral patch in $\mathcal{P}$
$\boldsymbol{v}$ or $\boldsymbol{v}_{ij}$	Arbitrary vertex or corner vertex
$\boldsymbol{f}$ or $\boldsymbol{f}_{ij}$	Arbitrary feature vector or corner feature vector
θ	MLP parameters
$\boldsymbol{u}$	Uniform sample in $[0, 1]^2$

3 NEURAL GEOMETRY FIELDS

As shown in Figure 2, our pipeline uses a neural network to continuously displace a base mesh Σ into the target surface Γ. We achieve this by first partitioning the surface into patches, and construct a continuous and trainable feature field $\Psi : \Sigma \to \mathbb{R}^F$, where each feature consists of F real components (Section 3.1). Next, we extract a traditional triangle mesh by sampling on the patches to obtain features, and feeding these features to along with 3D positions to a neural network to evaluate the displacement (Section 3.2). We optimize the feature fields and patches using a coarse-to-fine inverse rendering algorithm (Section 3.3). See Table 1 for an overview of the notation used in the subsequent discussion.

3.1 Surface Partitioning

Patches. To construct a feature field on the base mesh Σ, we build a partition $\mathcal{P}$ of disjoint quadrilateral patches σ whose union covers Σ. We specifically use quadrilaterals as they embed a simple interpolation domain. Although triangles are also simple in this

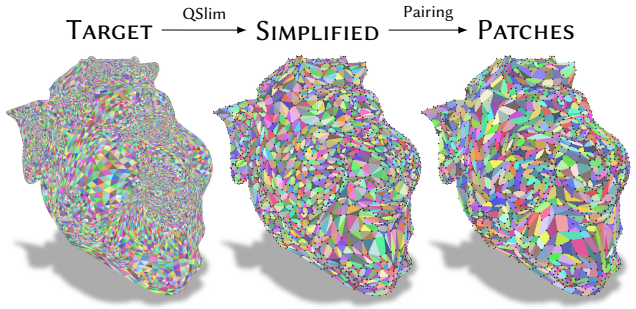

Figure 3: Surface Partitioning. Constructing the partition $\mathcal{P}$ of patches. The target surface Γ is first simplified using QSlim to reduce its polygon count. Adjacent triangles are then paired to form quadrilaterals, which builds the quadrilateral mesh Q representing $\mathcal{P}$.

regard, fewer quadrilaterals are needed in general to cover the surface.

Formally, each patch σ must be diffeomorphic to the unit square domain, so that it can be represented with four corner vertices, $\boldsymbol{v}_{00}, \boldsymbol{v}_{10}, \boldsymbol{v}_{01}, \boldsymbol{v}_{11}$, and a sample $\boldsymbol{u} \in [0,1]^2$ using a bilinear interpolation as follows:

$$\begin{aligned}\sigma(\boldsymbol{u}) = (1-\boldsymbol{u}_u)((1-\boldsymbol{u}_x)\cdot \boldsymbol{v}_{00} + \boldsymbol{u}_x \cdot \boldsymbol{v}_{10}) \\ + \boldsymbol{u}_y((1-\boldsymbol{u}_x)\cdot \boldsymbol{v}_{01} + \boldsymbol{u}_x \cdot \boldsymbol{v}_{11})\end{aligned} \tag{1}$$

Likewise, with four feature vectors $\boldsymbol{f}_{00}, \boldsymbol{f}_{10}, \boldsymbol{f}_{01}, \boldsymbol{f}_{11}$, we can define a smooth feature field within σ using the same bilinear interpolation formula:

$$\begin{aligned}\Psi(\boldsymbol{u}) = (1-\boldsymbol{u}_u)((1-\boldsymbol{u}_x)\cdot \boldsymbol{f}_{00} + \boldsymbol{u}_x \cdot \boldsymbol{f}_{10}) \\ + \boldsymbol{u}_y((1-\boldsymbol{u}_x)\cdot \boldsymbol{f}_{01} + \boldsymbol{u}_x \cdot \boldsymbol{f}_{11})\end{aligned} \tag{2}$$

These formulations are visualized in Figure 4 **(b)**.

Surface partitioning. We represent the base surface Σ with a non-degenerate quadrilateral mesh with vertices $\mathcal{V} = \cup_\sigma \{\boldsymbol{v}_{00}, \boldsymbol{v}_{10}, \boldsymbol{v}_{01}, \boldsymbol{v}_{11}\}$ and faces $\boldsymbol{Q}$. A shared edge between two quadrilaterals corresponds to a shared continuous boundary amongst two patches. On the other hand, we can have a patch that shares no edges with others, which allows us to represent non-manifold base surfaces Σ. We likewise assign each vertex in $\mathcal{V}$ with its corresponding feature in $\mathcal{F} = \cup_\sigma \{\boldsymbol{f}_{00}, \boldsymbol{f}_{10}, \boldsymbol{f}_{01}, \boldsymbol{f}_{11}\}$, and the resulting feature field Ψ will inherit the same discontinuities as Σ.

The evaluation of our surface involves using Σ as a base mesh that is later refined by a neural network MLP_θ. The resulting surface necessarily has the same topology as Σ, including its topological features such as holes and intersections. In the interest of surface compression, we wish to store a minimal amount of information for the base quadrilateral mesh. Thus, a method for constructing the base mesh is ideally topology preserving, even at low quadrilateral counts $|\boldsymbol{Q}|$. To do this, we **1.** simplify the mesh represented by Γ using QSlim, as it is robust and scalable, and **2.** greedily combine adjacent triangles to form near-rectangular quadrilaterals, removing non-manifold triangles in the process. Figure 3 demonstrates this process on a sample model. Note that this procedure can handle non-manifold input surfaces.

3.2 Mesh Extraction

In this section, we describe how we sample a mesh from a neural geometry field instance so that we can perform inverse rendering to optimize our representation.

Equipped with the base mesh and a feature field, we can build a piecewise continuous surface Λ by concatenating patches displaced by a neural network. First, we combine the vertex position and feature with an encoding **enc** derived from positional encoding [Mildenhall et al. 2020]:

$$\textbf{enc}(\boldsymbol{v}, \boldsymbol{f}) = \left(\sin(2^0\boldsymbol{v}), \cos(2^0\boldsymbol{v}), \ldots, \sin(2^L\boldsymbol{v}), \cos(2^L\boldsymbol{v}), \boldsymbol{f}\right) \tag{3}$$

The number of levels is controlled by a hyperparameter L. Then, at each patch σ and sample $\boldsymbol{u}$, the displaced surface coordinate on Λ is

$$\Lambda(\boldsymbol{u}) = \sigma(\boldsymbol{u}) + \text{MLP}_\theta \circ \textbf{enc}(\sigma(\boldsymbol{u}), \Psi(\boldsymbol{u})). \tag{4}$$

We extract the resulting representation as a discrete mesh by sampling $\boldsymbol{u}$ on each patch. In practice, this is done as follows:

(1) Tessellate each patch by sampling $\boldsymbol{u}$ in $[0,1]^2$ with k samples along each dimension, for a total of k^2 samples per patch. (Figure 4 **(a)**)
(2) Generate the corresponding vertices and features for each sample according to Equations (1) and (2). (Figure 4 **(b)**)
(3) Compute the displaced vertex by encoding and applying the neural network as in Equation (4). (Figure 4 **(c)** and **(d)**)
(4) Generate connectivity information independently for each patch to obtain the connectivity for the extracted surface. (Figure 4 **(e)**)

Locally, at each patch σ, we perform step **(4)** by laying out the vertices in a grid arrangement and applying a triangulation similar to height fields. The resulting mesh, when combined across multiple patches, results in a semi-regular mesh.

Jittering. During optimization, we aim to thoroughly sample each patch so that the neural geometry field obtains a better reconstruction of the target surface. We achieve this by randomly jittering uniform samples during mesh extraction by $\boldsymbol{u} \sim \hat{\boldsymbol{u}} + \mathcal{D}(\omega)$, where

$$\hat{\boldsymbol{u}} = \frac{\langle i, j\rangle}{k-1} \quad \text{for} \quad 0 \le i, j \le k-1 \tag{5}$$

are the original uniform samples and $\mathcal{D}(\omega)$ uniformly samples points in the origin-centered disk with radius ω. Note that to uphold the structure of the jittered mesh (i.e. prevent triangle fold overs), it is necessary that ω be less than $0.5/(k-1)$. To preserve boundaries, we additionally enforce $\omega = 0$ when $\hat{\boldsymbol{u}}$ is a boundary sample.

Jittering provides richer gradient signals to the optimization by virtue of better exploring the surface points of each patch. This is especially effective in regions of the neural geometry field where the vertex count is limited (i.e. with low patch counts). In Figure 5, while both strategies can coarsely recover the features of the reference, uniform sampling produces a noisier result compared to jittered sampling. As such, jittering can be thought of as a regularization mechanism for the reconstructed surface.

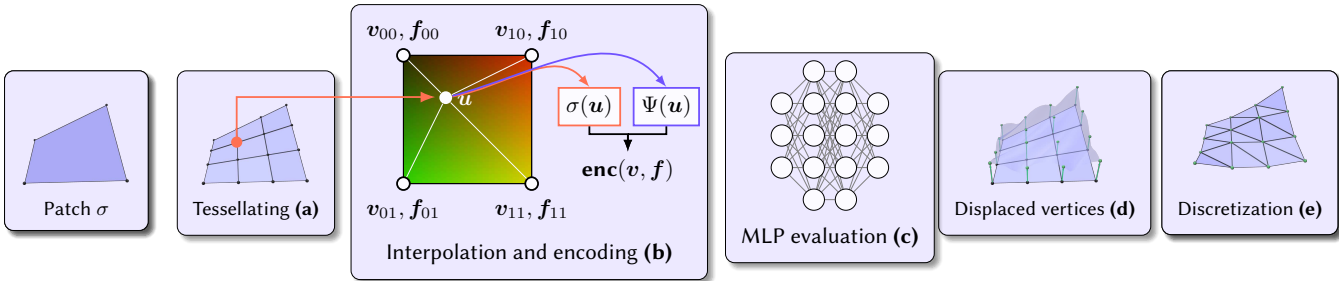

Figure 4: Mesh Extraction. Pipeline flow of a single patch during mesh extraction, where a traditional mesh is derived from our representation for use in optimization. A patch σ is uniformly tessellated, and corresponding vertices and features are computed. After positional encoding and concatenation, we evaluate the neural network to obtain the vector displacement on σ. Applying this displacement yields the displaced patch vertices on $\Lambda(\sigma)$, and finally we create a discretized mesh with a fixed scheme. Repeating this process over all patches constructs the full surface Λ.

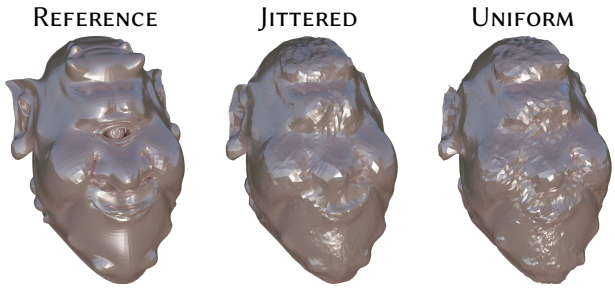

Figure 5: Sample Jittering. Jittering uniform samples during mesh extraction helps to regularize the optimization of a neural geometry field. Above, we demonstrate its impact when learning surfaces at low patch counts (10 patches), where omitting jittered results in a rougher surface.

Algorithm 1: Triangle mesh extraction on Λ

Input: Tessellation resolution k
Input: Neural geometry field Λ
Function *Discretize(V, k)*:
  $T \leftarrow [\,]$
  $V_{grid} \leftarrow V.reshape(k, k, 3)$
  for *Sections* $[[V_a, V_b], [V_c, V_d]]$ *in* V_{grid} **do**
    // Triangulate the quadrilateral
    $T.add(SplitQuad(V_a, V_b, V_c, V_d))$
  return T
Function *Tesselate(σ, k)*:
  $V \leftarrow [\,]$
  for $\hat{u} \in UniformSamples(k)$ **do**
    // Jittering the uniform samples (Equation 5)
    $u \leftarrow \hat{u} + IsInteriorSample(u) \cdot \mathcal{D}(\omega)$
    // Evaluate the neural geometry field (Equation 4)
    $V.add(\Lambda(u))$
  $T \leftarrow Discretize(V, k)$
  return V, T
Function *ExtractMesh(Σ, k)*:
  $V \leftarrow [\,], T \leftarrow [\,]$
  for $\sigma \in \mathcal{P}$ **do**
    $V_\sigma, T_\sigma \leftarrow Tesselate(\sigma, k)$
    $V.add(V_\sigma)$
    $T.add(T_\sigma)$
  return $Mesh(V, T)$

3.3 Optimization

Inverse rendering. Our optimization pipeline is a rasterization-based inverse rendering process that fine-tunes the appearance of our representation's surface to that of the target surface. We find that an appearance-based method for optimization is more stable than using distance-based methods such as Chamfer distance or signed distance field queries. To stabilize the training process, we apply a coarse-to-fine approach using inverse rendering. At a particular tessellation resolution, k, the mesh M will have a fixed connectivity defined by the triangulation of M. Its vertex positions, on the other hand, are differentiable with respect to the patch corners $\mathcal{V}$ and features $\mathcal{F}$, as well as the neural network parameters θ. This means that inverse rendering can indeed produce useful gradients for optimizing our representation.

Objective function. In contrast with conventional inverse rendering problems, our pipeline has access to various attributes of the reference surface. Of these, we use only the surface normal vectors.

With the sampled mesh, differentiable normal vectors can be evaluated from vertex cross products, as is typical. During rasterization, these normal are interpolated to generate the normal vector frame buffers $\mathcal{N}(\cdot)$ for both the reference and sampled surface. Using these buffers, an image-space loss can be defined. Additionally, to promote an even vertex distribution along the surface, we include

a Laplacian term. The unified objective function we use is

$$\mathcal{L} = \frac{1}{|\mathcal{N}(\cdot)|}\|\mathcal{N}(\Gamma) - \mathcal{N}(\Sigma)\|_1 + \frac{1}{|V|}\|LV\|_1, \tag{6}$$

where $\|\cdot\|_1$ is the L_1 metric and L is the (uniform) mesh Laplacian of M. Note that for the smoothing term, we apply the Laplacian on the uniformly sampled vertices.

Camera arrangement. As the reference surface can have arbitrary topology, it is crucial to have a set of views thoroughly cover its surface area. To distribute the cameras, we cluster the triangles of the reference by geodesic distance. We then construct a camera for each cluster, looking towards the centroid of the cluster. This approach is both efficient and scalable with respect to the number of cameras to instantiate. Additionally, to capture occluded geometry, we rasterize multiple depth layers using depth peeling.

General configuration. As a preprocessing step to our optimization pipeline, we normalize the coordinates of the reference mesh. We run the pipeline for tessellation resolutions $k \in \{4, 8, 12, 16\}$, typically using 200 cameras. During rendering, we use nvdiffrast to rasterize 10 views in a batch using 3 depth layers each. Our neural network consists of two hidden layers, each with 64 neurons, with $L = 8$ levels for positional encoding, and a feature vector size of $F = 20$. Lastly, we use a fixed learning rate of 10^{-3} with an ADAM optimizer [Kingma and Ba 2015].

4 RESULTS

We test our method on meshes with various surface and topological features. In Table 2, we display models with varying storage size, genus, and surface details. Models like Dragon exhibit hard to reach surfaces that are hidden from ordinary inspection; Einstein and Skull both contain immense surface details; Metatron and other high genus meshes are tricky for appearance-based methods to reconstruct due to frequent occlusions in camera views.

In our results, we evaluate how well resulting meshes compare with the target mesh, both visually and geometrically. We use the render image loss, where the surface is shaded with an environment map, and the Chamfer loss, using the vertices sampled from our representation. For these comparisons, we forego jittering when extracting final meshes.

Additional results, notably rasterization, *uv*-parameterization and comparison to other neural methods, are shown in the supplementary material. We urge the reader to take a look into these results to further understand the capabilities of our representation.

4.1 Compression

We evaluate our method for mesh compression against previous works. In particular, we compare against QSlim [Garland and Heckbert 1997] and nvdiffmodeling [Hasselgren et al. 2021], which rely on surface extrinsic and visual metrics, respectively. In particular, we use the symmetric Chamfer distance to compare geometric quality, and use mutli-view rendering for visual quality. For rendering, we use a random arrangement of camera views which are distinct from the training views, and shade using spherical-harmonics-based environment mapping [Ramamoorthi and Hanrahan 2001].

Both methods have shown strong results in compression. The storage for our representation consist of the base quadrilateral mesh and features $(\mathcal{V}, \mathcal{F}, \mathcal{Q})$, as well as the MLP, θ. All primitive elements are stored with 32 bits. The target surface and the outputs for each baseline are all meshes, so we calculate their storage costs using only the vertex data and connectivity information.

Our representations are optimized at increasing patch counts while fixing the tessellation resolution to $k = 16$. The representations require different amounts of storage, and to provide a fair comparison, we generate the baseline results at similar sizes. Figure 8 demonstrates the result of our method on the XYZ, Einstein, and Ganesha models, showing sections of the surface in insets for each method as well as plotting error metrics for each representation with respect to the compression ratio.

For these models, neural geometry fields consistently outperforms the baselines across all size variations of our methods. In the xyz model, for example, our method is better able to recover the scales on the dragon whereas QSlim and nvdiffmodeling struggle due to limited vertex count.

In Table 2, we show further results of our method with others. Our method scales well with the number of patches we partition the reference into. Even at its highest quality, at 2.5K patches, our representation remains under a megabyte in storage.

Additionally, in Figure 6, we compare our method to Draco [Galligan et al. 2018], a state-of-the-art mesh compression method [Doumanoglou et al. 2019] which performs compression of both vertex data and connectivity. At high amounts of quantization, our method still outperforms this baseline in terms of visual metrics. We observe that the advantage of our method over Draco correlates with the size of the reference mesh. In such cases, neural geometry fields can adaptively allocate primitives for particular regions of the surface, while quantization methods sacrifice vertex precision for the preservation of topology.

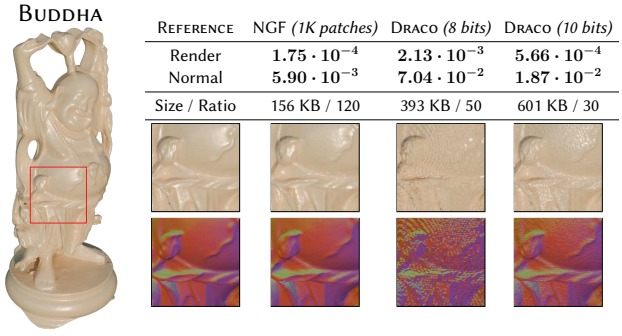

Figure 6: Mesh Compression (Pt. 3). We compare neural geometry fields to Draco [Galligan et al. 2018] which relies on data quantization. In this particular example, with the Buddha mesh, our method is notably more compact than this state-of-art compression method, even when using heavy amounts of quantization. Furthermore, our representation maintains visual similarity with the reference, whereas numerous visual artifacts can be seen from the results of Draco. For our method, we use 1K patches with the default configuration; for Draco, we encode the connectivity and vertex positions only, quantized with 8 and 10 bits per vertex. This mesh is taken from the Stanford 3D Scanning Repository.

Table 2: Mesh Compression (Pt. 1). Comparison of our method with the baselines at varying patch counts. We evaluate each method using a multi-view rendering loss (left columns, scaled by 10^2) and the Chamfer distance (right columns, scaled by 10^5). Our method outperforms the baselines consistently throughout models with varying features, including high triangle count and genus. Furthermore, the quality of our method scales as we increase its allocation of patches for reconstruction. In the following table, Lod refers to the number of primitives (patches or triangles) and overall size of the representation, Tris refers to the number of triangles in the reference mesh, and Nvdiff is shorthand for nvdiffmodeling. The models Dragon, Einstein, and Metratron are from Thinkg10K; Buddha, Armadillo, Xyz and Lucy are courtesy of the Stanford 3D Scanning Repository; Ganesha credited to peel3d.

	Lod	Dragon		Ganesha		Buddha		Einstein		Armadillo		Metatron		Xyz		Lucy	
Size		15.46 MB		41.03 MB		18.66 MB		14.73 MB		1.72 MB		2.38 MB		4.29 MB		1.72 MB	
Tris		900.5 K		2.4 M		1.1 M		858.0 K		100.0 K		138.4 K		249.9 K		100.0 K	
Ours	100 / 65 KB	23.18	47.26	8.72	1.42	5.65	1.94	13.03	9.16	5.01	6.48	-	-	8.37	7.60	5.97	8.83
	250 / 80 KB	16.07	11.46	6.39	0.82	2.76	15.60	10.78	4.00	3.44	2.85	8.20	9.22	6.84	2.77	4.00	3.62
	1.0K / 160 KB	9.54	1.42	0.30	3.52	1.48	1.10	6.49	1.34	2.38	1.66	4.62	3.18	4.17	0.94	2.51	1.97
	2.5K / 320 KB	**6.76**	**0.74**	**1.66**	**0.26**	**0.74**	**0.25**	**4.78**	**0.77**	**1.93**	**1.43**	**3.74**	**2.59**	**3.36**	**0.65**	**1.86**	**1.55**
QSlim	4K / 65 KB	23.42	39.92	21.75	44.77	23.75	46.80	24.57	76.61	19.10	63.47	15.25	35.92	12.95	32.37	22.58	63.99
	5K / 80 KB	21.26	32.25	20.25	42.22	23.75	46.80	23.02	62.39	17.11	52.56	14.22	29.68	12.16	26.01	21.73	53.50
	10K / 160 KB	17.53	15.93	14.83	37.03	12.38	6.85	19.77	32.30	13.14	27.60	10.27	15.73	9.78	13.38	17.70	29.46
	19K / 320 KB	14.93	8.21	9.68	20.72	6.15	2.66	15.96	16.13	9.39	14.95	7.19	8.48	7.56	6.76	13.83	16.30
Nvdiff	4K / 65 KB	21.27	57.02	17.31	29.93	25.31	52.22	23.81	83.85	18.95	82.51	14.65	44.20	12.52	40.65	22.28	100.76
	5K / 80 KB	19.83	45.58	15.26	28.97	25.31	53.16	23.94	71.41	16.86	76.56	13.85	36.54	11.39	33.49	21.56	89.16
	10K / 160 KB	17.32	24.58	10.58	26.87	12.09	7.18	18.25	38.87	10.14	39.02	10.31	18.81	9.70	18.30	17.04	45.47
	19K / 320 KB	14.13	11.46	8.45	19.65	5.74	3.04	15.32	20.46	7.77	20.71	8.90	10.03	8.32	9.96	13.21	21.37

4.2 Evaluations

Tessellation. During mesh extraction, we permit arbitrary tessellation rates. In Figure 9, we compare the quality of these meshes with respect to their tessellation resolutions. While the geometric quantities show steady improvements with increasing tessellation, the visual metrics plateau around $k = 16$. We thus cap our pipeline to this resolution to minimize additional computational and memory overhead during optimization.

Features. Our method is flexible with respect to the number of feature vector dimensions F. The primary trade-off lies between quality and storage overhead. As shown in Figure 10, more features lead to improved reconstruction. However, as the storage cost grows linearly with the feature size, this improvement eventually diminishes. We find that a feature size of $F = 20$ strikes a good balance between quality and storage size; with 1000 patches, the total storage cost typically remains below 200 kilobytes.

4.3 Patch-Based Representations

Without the neural component, our representation can be stored as a collection of square geometry images in a single, tightly packed atlas. Two interesting points of comparison arise; **(1)** how does our representation compare to a packed geometry image atlas; **(2)** how does it compare against a neural representation of the atlas?

To perform the first of these benchmarks, we run multichart geometry images, where charts are packed into squares of a fixed resolution (32×32). For the second benchmark, we compare against a neural-hybrid implicit network which operates on Fourier encoded uv coordinates (with $L = 16$) and a feature field. Feature vectors are embedded on the corners of each square geometry image in the atlas and are interpolated within each patch similar to our method.

In Figure 7, we display the size and quality of these two benchmarks along with our method. Our method outperforms both of these benchmarks in terms of both size and quality. Whereas the ordinary atlas representation becomes comparable to our method with more charts, we observe that its neural counterpart struggles overall. We believe this is largely due to the numerous discontinuities along chart boundaries on the atlas. The difference in chart and patch boundary characteristics for a neural multichart geometry image and a neural geometry field is to note. In the former, these boundaries are typically discontinuous, whereas in our representation these boundaries are necessarily continuous.

4.4 Runtime

Our optimization pipeline is implemented using PyTorch and relies on nvdiffrast [Laine et al. 2020] for differentiable rasterization. In Table 3, we analyze the runtime of this process by comparing the GPU execution times reported by PyTorch; all evaluations are done using an NVIDIA GeForce RTX 2080 Ti graphics card.

At the most expensive configuration, 2500 patches at a tessellation resolution of $k = 16$, mesh extraction still remains interactive. The compression time, measured by the time taken to optimize our representation, typically lies in the minutes for most scenes. We find that this is a reasonable time to spend in comparison to the

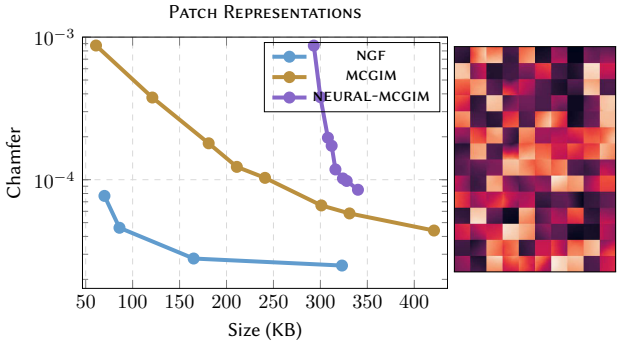

Figure 7: Multichart Geometry Images. On the left, we compare our method to using a multichart geometry image consisting of square geometry images, and a neural implicit representing the same atlas. Our method is better than both these approaches in terms of both size and quality. On the right, we display the x component of a multichart atlas; the discontinuities along patch boundaries makes regression on it more difficult.

Table 3: Runtime. The table shows the time it takes to perform mesh extraction and optimization for our representation. Even at the most intensive configuration, the mesh extraction process remain interactive. The training time for our representation is comparable to previous neural methods.

Patches	$k = 4$ (ms)	$k = 8$ (ms)	$k = 16$ (ms)	Training (min)
100	0.10	0.23	0.76	4
250	0.21	0.51	1.98	6
1000	0.51	1.97	8.14	8
2500	1.19	4.79	21.22	12

quality that can be retrieved afterward. In the supplementary, we also present a real-time rendering pipeline for rasterizing neural geometry fields that incurs little additional memory overhead.

5 CONCLUSION

In this paper, we propose a novel compact neural representation for discrete surface geometry. Our method effectively combines previous work in signal representations with neural implicit functions and mesh processing to produce state-of-the-art results. We present a concrete and robust pipeline for constructing an overfit instance of our representation to arbitrary surfaces, and demonstrate that it outperforms previous baselines for compressing meshes.

Limitations. The hybrid structure of neural geometry fields enables it to effectively fill in the details embedded within each patch. However, when these patches are extremely scarce with respect to the surface topology, our inverse rendering pipeline may fail to appropriately reconstruct the basic form of the target mesh.

Future work. The representation we present here consists of a bare minimum of extrinsic mesh data, $\mathcal{V}$, and $\mathcal{Q}$. It is possible to additionally including normal vectors for each vertex, so that each patch σ becomes a Bezier patch that better captures the curvature on the target surface.

Additionally, while we present a low-cost rasterization algorithm in the supplementary, we speculate that a similar pipeline can be developed for raytracing. In particular, works for tessellation-free [Thonat et al. 2021] and non-linear [Ogaki 2023] ray tracing have enabled memory efficient ray tracers for detailed surfaces, a similar method can potentially be applied for neural geometry fields.

ACKNOWLEDGMENTS

This work was supported in part by NSF grants 2105806 and 2212085. We also acknowledge gifts from Adobe, Google, Qualcomm and Rembrand, the Ronald L. Graham Chair, and the UC San Diego Center for Visual computing. We would also like to thank the anonymous reviewers for their insightful suggestions.

REFERENCES

Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. 2021. Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields. In *International Conference on Computer Vision*. 5855–5864.

Marcel Campen. 2017. Partitioning Surfaces into Quad Patches. In *Eurographics Tutorials*.

Marcel Campen, David Bommes, and Leif Kobbelt. 2012. Dual loops meshing: quality quad layouts on manifolds. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 31, 4, 1–11.

Nathan A. Carr, Jared Hoberock, Keenan Crane, and John C. Hart. 2006. Rectangular Multi-Chart Geometry Images. In *Symposium of Geometry Processing*. 181–190.

E. Catmull and J. Clark. 1978. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design* 10, 6 (1978), 350–355.

Yun-Chun Chen, Vladimir Kim, Noam Aigerman, and Alec Jacobson. 2023. Neural Progressive Meshes. In *SIGGRAPH Conference Proceedings*. 1–9.

Paolo Cignoni, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton, Federico Ponchio, and Roberto Scopigno. 2004. Adaptive Tetrapuzzles: Efficient Out-of-Core Construction and Visualization of Gigantic Multiresolution Polygonal Models. *ACM Transactions on Graphics (TOG)* 23, 3 (2004), 796–803.

Jonathan Cohen, Marc Olano, and Dinesh Manocha. 1998. Appearance-Preserving Simplification. In *SIGGRAPH*. 115–122.

Michael Deering. 1995. Geometry compression. In *SIGGRAPH*. 13–20.

Alexandros Doumanoglou, Petros Drakoulis, Nikolaos Zioulis, Dimitrios Zarpalas, and Petros Daras. 2019. Benchmarking Open-Source Static 3D Mesh Codecs for Immersive Media Interactive Live Streaming. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 1 (2019), 190–203.

Nira Dyn, David Levine, and John A. Gregory. 1990. A Butterfly Subdivision Scheme for Surface Interpolation with Tension Control. *ACM Transactions on Graphics (TOG)* 9, 2 (1990), 160–169.

Frank Galligan, Michael Hemmer, Ondrej Stava, Fan Zhang, and Jamieson Brettle. 2018. Google/draco: a library for compressing and decompressing 3D geometric meshes and point clouds. https://github.com/google/draco.

William Gao, April Wang, Gal Metzer, Raymond A Yeh, and Rana Hanocka. 2022. TetGAN: A Convolutional Neural Network for Tetrahedral Mesh Generation. In *Proceedings British Machine Vision Conference*.

Michael Garland and Paul S. Heckbert. 1997. Surface Simplification Using Quadric Error Metrics. In *SIGGRAPH*. 209–216.

Xianfeng Gu, Steven J. Gortler, and Hugues Hoppe. 2002. Geometry Images. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 21, 3 (2002), 355–361.

Stefan Gumhold and Wolfgang Straßer. 1998. Real Time Compression of Triangle Mesh Connectivity. In *SIGGRAPH*. 133–140.

Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. 2019. MeshCNN: a network with an edge. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 38, 4 (2019).

Jon Hasselgren, Jacob Munkberg, Jaakko Lehtinen, Miika Aittala, and Samuli Laine. 2021. Appearance-Driven Automatic 3D Model Simplification. In *Eurographics Symposium on Rendering*. 85–97.

Hugues Hoppe. 1996. Progressive Meshes. In *SIGGRAPH*. 99–108.

Hugues Hoppe, Tony DeRose, Tom Duchamp, Mark Halstead, Hubert Jin, John McDonald, Jean Schweitzer, and Werner Stuetzle. 1994. Piecewise smooth surface reconstruction. In *SIGGRAPH*. 295–302.

Shi-Min Hu, Zheng-Ning Liu, Meng-Hao Guo, Jun-Xiong Cai, Jiahui Huang, Tai-Jiang Mu, and Ralph R Martin. 2022. Subdivision-based mesh convolution networks. *ACM Trans. Graph.* 41, 3 (2022), 25:1–25:16.

Benjamin T Jones, Michael Hu, Milin Kodnongbua, Vladimir G Kim, and Adriana Schulz. 2023. Self-supervised representation learning for CAD. In *Computer Vision and Pattern Recognition*. 21327–21336.

Diederick P Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.

Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. 2020. Modular primitives for high-performance differentiable rendering. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 39, 6, Article 194 (2020), 11-14 pages.

Aaron Lee, Henry Moreton, and Hugues Hoppe. 2000. Displaced Subdivision Surfaces. In *SIGGRAPH*. 85–94.

Eung-Seok Lee and Hyeong-Seok Ko. 2000. Vertex Data Compression for Triangular Meshes. In *Pacific Graphics*. 225–234.

Thibault Lescoat, Hsueh-Ti Derek Liu, Jean-Marc Thiery, Alec Jacobson, Tamy Boubekeur, and Maks Ovsjanikov. 2020. Spectral mesh simplification. *Comput. Graph. Forum (Proc. Eurographics)* 39, 2 (2020), 315–324.

Zhaoshuo Li, Thomas Müller, Alex Evans, Russell H Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin. 2023. Neuralangelo: High-Fidelity Neural Surface Reconstruction. In *Computer Vision and Pattern Recognition*. 8456–8465.

Hsueh-Ti Derek Liu, Vladimir G. Kim, Siddhartha Chaudhuri, Noam Aigerman, and Alec Jacobson. 2020b. Neural Subdivision. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 39, 4 (2020), 124:1–124:16.

Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. 2020a. Neural Sparse Voxel Fields. In *Advances in Neural Information Processing Systems*.

Stephen Lombardi, Tomas Simon, Gabriel Schwartz, Michael Zollhoefer, Yaser Sheikh, and Jason Saragih. 2021. Mixture of volumetric primitives for efficient neural rendering. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 40, 4 (2021), 1–13.

Charles Loop. 1987. *Smooth Subdivision Surfaces Based on Triangles*. Master's thesis. Department of Mathematics, The University of Utah.

Andrea Maggiordomo, Henry Moreton, and Marco Tarini. 2023. Micro-mesh construction. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 42, 4 (2023), 1–18.

Adrien Maglo, Guillaume Lavoué, Florent Dupont, and Céline Hudelot. 2015. 3D Mesh Compression: Survey, Comparisons, and Emerging Trends. *Comput. Surveys* 47, 3 (2015), 44:1–44:41.

Julien N. P. Martel, David B. Lindell, Connor Z. Lin, Eric R. Chan, Marco Monteiro, and Gordon Wetzstein. 2021. ACORN: Adaptive Coordinate Networks for Neural Scene Representation. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 40, 4 (2021), 58:1–58:13.

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. 2019. Occupancy Networks: Learning 3D Reconstruction in Function Space. In *Computer Vision and Pattern Recognition*. 4460–4470.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *European Conference on Computer Vision*. 405–421.

Luca Morreale, Noam Aigerman, Paul Guerrero, Vladimir G. Kim, and Niloy J. Mitra. 2022. Neural Convolutional Surfaces. In *Computer Vision and Pattern Recognition*.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 41, 4 (2022), 102:1–102:15.

Shinji Ogaki. 2023. Nonlinear Ray Tracing for Displacement and Shell Mapping. In *SIGGRAPH Asia Conference Proceedings*. Article 93, 10 pages.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *Computer Vision and Pattern Recognition*. 165–174.

Nico Pietroni, Stefano Nuvoli, Thomas Alderighi, Paolo Cignoni, Marco Tarini, et al. 2021. Reliable feature-line driven quad-remeshing. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 40, 4 (2021), 1–17.

Rolandos Alexandros Potamias, Stylianos Ploumpis, and Stefanos Zafeiriou. 2022. Neural Mesh Simplification. In *Computer Vision and Pattern Recognition*. 18583–18592.

Ravi Ramamoorthi and Pat Hanrahan. 2001. An efficient representation for irradiance environment maps. *SIGGRAPH Conference Proceedings*, 497–500.

P. V. Sander, Z. J. Wood, S. J. Gortler, J. Snyder, and H. Hoppe. 2003. Multi-Chart Geometry Images. In *Symposium of Geometry Processing*. 146–155.

Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. 2021. Deep Marching Tetrahedra: A Hybrid Representation for High-Resolution 3D Shape Synthesis. In *Advances in Neural Information Processing Systems*, Vol. 34. 6087–6101.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. 2020. Implicit Neural Representations with Periodic Activation Functions. In *Advances in Neural Information Processing Systems*, Vol. 33. 7462–7473.

Jos Stam. 1998. Exact Evaluation of Catmull-Clark Subdivision Surfaces at Arbitrary Parameter Values. In *SIGGRAPH*. 395–404.

Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. 2021. Neural geometric level of detail: Real-time rendering with implicit 3D shapes. In *Computer Vision and Pattern Recognition*. 11358–11367.

Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. 2020. Fourier features let networks learn high frequency functions in low dimensional domains. In *Advances in Neural Information Processing Systems*, Vol. 33. 7537–7547.

Marco Tarini, Nico Pietroni, Paolo Cignoni, Daniele Panozzo, and Enrico Puppo. 2010. Practical quad mesh simplification. *Comput. Graph. Forum (Proc. Eurographics)* 29, 2 (2010), 407–418.

Gabriel Taubin and Jarek Rossignac. 1998. Geometric compression through topological surgery. *ACM Trans. Graph.* 17, 2 (1998), 84–115.

Theo Thonat, Francois Beaune, Xin Sun, Nathan Carr, and Tamy Boubekeur. 2021. Tessellation-free displacement mapping for ray tracing. 40, 6 (2021), 282:1–282:16.

Costa Touma and Craig Gotsman. 1998. Triangle mesh compression. In *Graphics Interface*. 26–34.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. In *Advances in Neural Information Processing Systems*. 6000–6010.

Luiz Velho and Denis Zorin. 2001. 4–8 Subdivision. *Computer Aided Geometric Design* 18, 5 (2001), 397–427.

Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. 2021. NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction. In *Advances in Neural Information Processing Systems*, Vol. 34. 27171–27183.

Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. 2022. Neural fields in visual computing and beyond. *Comput. Graph. Forum (Proc. Eurographics STAR)* 41, 2 (2022), 641–676.

Qingnan Zhou and Alec Jacobson. 2016. Thingi10K: A Dataset of 10,000 3D-Printing Models. *arXiv preprint arXiv:1605.04797* (2016).

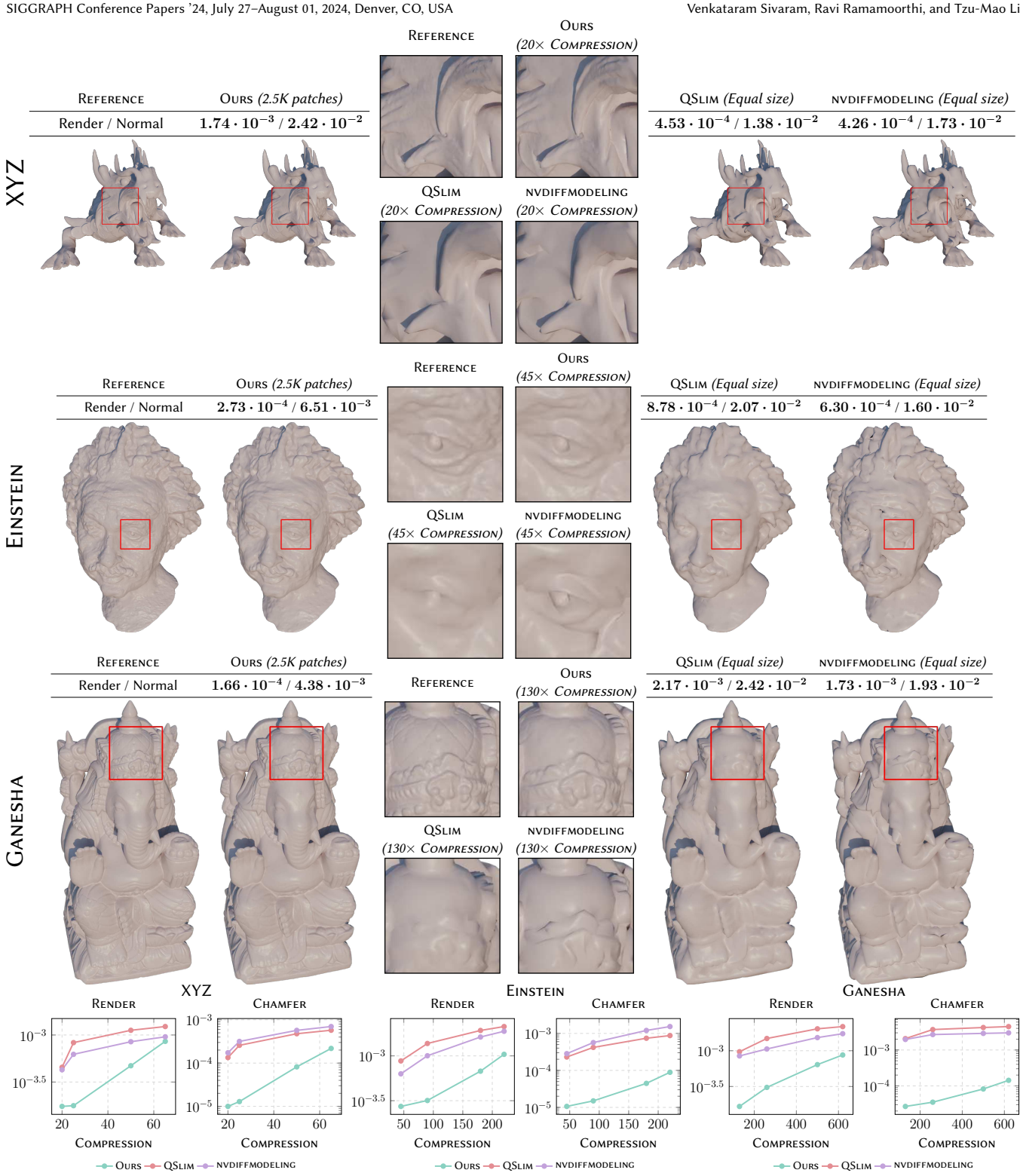

Figure 8: Mesh Compression (Pt. 2). Compressing high detail surface geometry with various methods. We visually compare the results produced by our method at increasing patch counts against the meshes produced by QSlim [Garland and Heckbert 1997], and nvdiffmodeling [Hasselgren et al. 2021] at equal storage. Our recovers fine surface details even at compression rates of over ×100. Visual metrics are based on the L_1 loss. Above models are xyz ©Stanford 3D Scanning Repository, Einstein ©Thingi10k, and Ganesha ©peel3d.

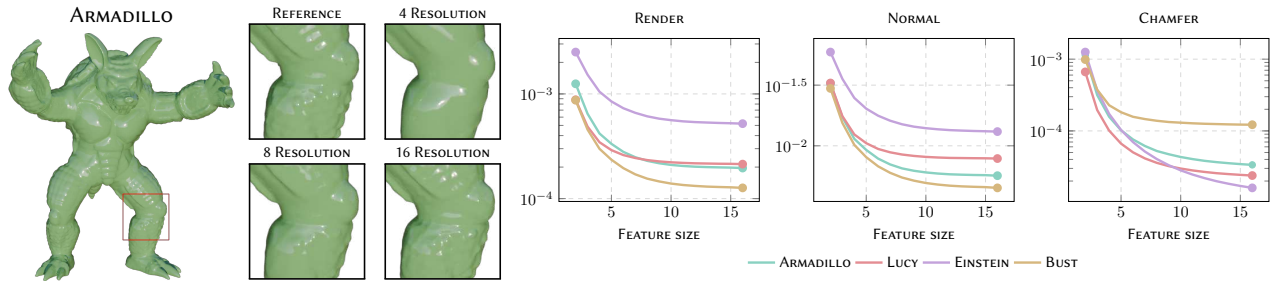

Figure 9: Tessellation Ablations. Neural geometry fields enable arbitrary level of detail with respect to the chosen tessellation resolution. Above, we show how the quality of the uniformly sampled extracted mesh scales with the resolution for the Armadillo mesh. On the right, we also plot the visual and geometric metrics of reconstructed neural geometry fields for resolutions $2 \leq k \leq 16$. We note that our choice to limit the maximal resolution to $k = 16$ is influenced by the fact that these metrics tend to plateau around this resolution. The mesh is credited to the Stanford 3D Scanning Repository.

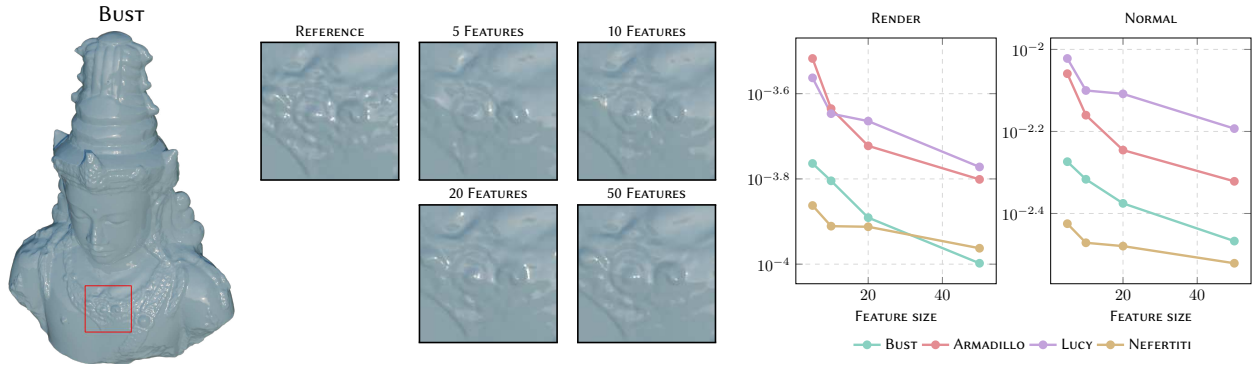

Figure 10: Feature Ablations. The quality of surface reconstruction of our method improves as more data is allotted for the feature vectors. Above, we show the visual effect of increasing feature vector size for the Bust model, for $F \in \{5, 10, 20, 50\}$. The plots on the right inform that, along with other models that we benchmarked, the visual metrics of our representation do indeed improve with higher feature sizes. The Chamfer distance remains similar, since the vertex resolution is constant, hence we omit it here. This model is courtesy of Thingi10K.