Neural Progressive Meshes

YUN-CHUN CHEN, University of Toronto, Canada
VLADIMIR G. KIM, Adobe Research, USA
NOAM AIGERMAN, Adobe Research, USA
ALEC JACOBSON, Adobe Research, University of Toronto, Canada

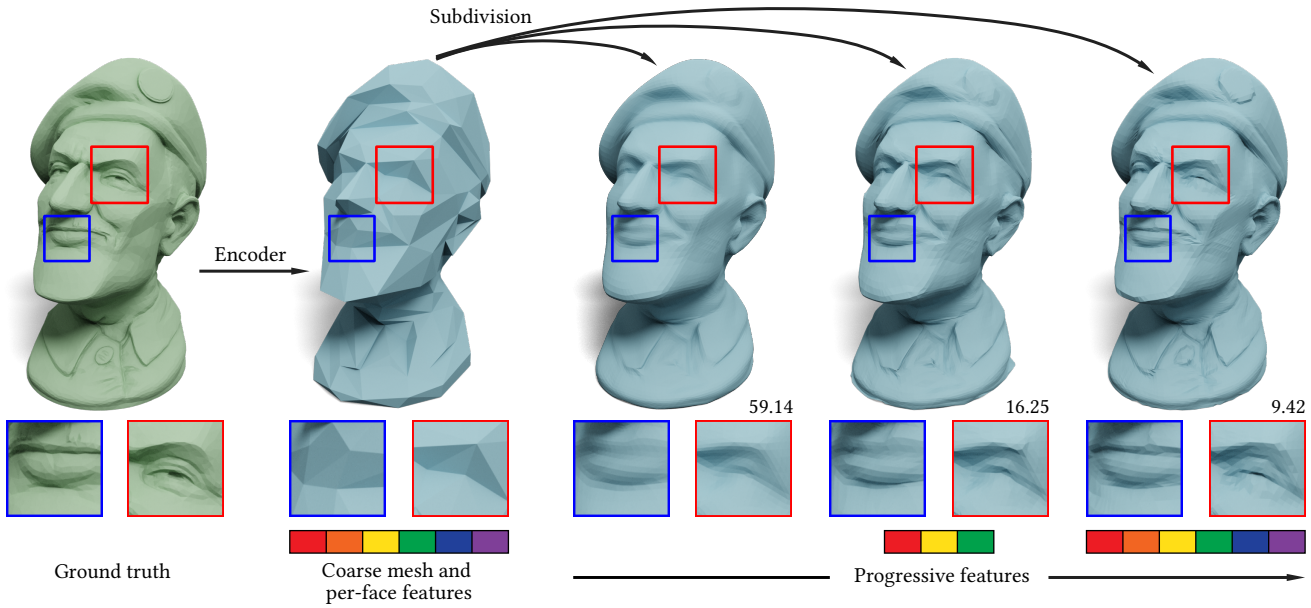

Fig. 1. **Neural Progressive Meshes.** We present a framework that learns a progressive compressed representation of meshes for transmission purposes. Given a high-resolution mesh in the database, the server trains a network that derives a compressed representation that can be transmitted to the client. The client reconstructs the low-resolution mesh using a pre-trained subdivision decoder. The reconstruction quality can be further improved with additional data transmitted progressively from the server to the client. The numbers shown in the corner of each example are the compression ratios.

The recent proliferation of 3D content that can be consumed on hand-held devices necessitates efficient tools for transmitting large geometric data, e.g., 3D meshes, over the Internet. Detailed high-resolution assets can pose a challenge to storage as well as transmission bandwidth, and level-of-detail techniques are often used to transmit an asset using an appropriate bandwidth budget. It is especially desirable for these methods to transmit data progressively, improving the quality of the geometry with more data. Our key insight is that the geometric details of 3D meshes often exhibit similar local patterns even across different shapes, and thus can be effectively represented with a shared learned generative space. We learn this space using a subdivision-based encoder-decoder architecture trained in advance on a large collection of surfaces. We further observe that additional residual features can be transmitted progressively between intermediate levels of subdivision that enable the client to control the tradeoff between bandwidth cost and quality of reconstruction, providing a *neural progressive mesh representation*. We evaluate our method on a diverse set of complex 3D shapes and demonstrate that it outperforms baselines in terms of compression ratio and reconstruction quality.

ACM Reference Format:
Yun-Chun Chen, Vladimir G. Kim, Noam Aigerman, and Alec Jacobson. 2023. Neural Progressive Meshes. *ACM Trans. Graph.* 1, 1 (May 2023), 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Authors' addresses: Yun-Chun Chen, University of Toronto, Canada, ycchen@cs.toronto.edu; Vladimir G. Kim, Adobe Research, USA, vokim@adobe.com; Noam Aigerman, Adobe Research, USA, aigerman@adobe.com; Alec Jacobson, Adobe Research, University of Toronto, Canada, jacobson@cs.toronto.edu.

0730-0301/2023/5-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

We propose a framework for learning a progressive compressed representation of meshes. Given a high-resolution mesh, our goal is to derive a compressed representation that can be transmitted to a client using a small bandwidth budget. The progressive nature of the compression entails the client can immediately reconstruct a meaningful mesh with lower reconstruction quality, and the server

can further progressively transmit additional data to improve the reconstruction quality while the asset is being used.

The need for such progressive compression techniques is consistently rising, following the rise in the need to transmit detailed 3D meshes from a centralized server repository to a client. Our method is especially suitable for virtual and augmented reality applications on mobile devices, which require selective transmissions of 3D content based on its visibility and available bandwidth. It additionally enables the client to stop decompression at a desired resolution.

Mesh decimation and level-of-detail (LoD) techniques are commonly used to reduce the size of a 3D asset either for rendering or transmission efficiency purposes [Garland and Heckbert 1997; Lescoat et al. 2020]. In addition to geometry simplification, remeshing can be used to optimize the size of a 3D asset [Surazhsky and Gotsman 2003; Szymczak et al. 2002]. Mesh simplification is a greedy process, and thus learning-based techniques have been proposed to devise a more efficient simplification approach [Potamias et al. 2022]. Most decimation techniques provide a single asset with a desired triangle budget and do not provide a way to progressively improve the quality of the asset. Progressive representations have been proposed to address this use case [Hoppe 1996], enabling incremental transmission of data to gradually improve the quality of the asset on the client side. All of these methods, however, inevitably lose details as they reduce the polygon count, approximating complex geometric details with planes. Surface subdivision [Loop 1987; Zorin et al. 1996] techniques could be used on the client side to increase the resolution of the transmitted low-resolution mesh, and the coarse mesh can be optimized specifically for a particular subdivision scheme [Hoppe et al. 1994]. These subdivision schemes, however, use simple hand-crafted filters, and thus subdivided geometry lacks any intricate original details. In this work, we propose to learn the space of geometric details by encoding them in progressive per-face features, which could be used to guide a neural subdivision process, enabling it to reconstruct complex geometry such as the eye or the lips of a character as shown in Figure 1.

We follow recent advances in subdivision-based learning techniques for mesh analysis [Hanocka et al. 2019; Hu et al. 2022] and upsampling [Hertz et al. 2020; Liu et al. 2020]. At inference time, given an input mesh, the server first uses TetWild [Hu et al. 2018] to preprocess it and then uses a subdivision-based encoder adapted from SubdivNet [Hu et al. 2022] to map geometric details of the original mesh to high-dimensional per-face features of a sequence of decimated meshes. The mesh at the lowest (coarsest) level of resolution can then be transmitted to the client, which uses a subdivision-based decoder adapted from Neural Subdivision [Liu et al. 2020] to reconstruct a high-resolution mesh. The per-face features can be additionally transmitted to the client, and our subdivision decoder is trained to use them to further improve the quality of the reconstruction. We train the encoder and the decoder jointly on a large and heterogeneous collection of shapes using a reconstruction loss. To allow progressive refinement, we also introduce a sparsity loss on the per-face features, favoring the irrelevant features to be zero.

We evaluate our network on the Thingi10K [Zhou and Jacobson 2016] dataset, split into the training, validation, and test sets. We create a benchmark, evaluating the reconstruction quality given a prescribed transmission limit. We demonstrate that our method outperforms various baselines that use mesh decimation, subdivision, or progressive representations.

2 RELATED WORK

Mesh compression and simplification. Lossless mesh compression techniques often use entropy encoding for geometry, connectivity, and surface attributes [Alliez and Desbrun 2001; Deering 1995; Rossignac 1999; Szymczak et al. 2001; Taubin and Rossignac 1998; Touma and Gotsman 1998]. These methods perfectly preserve the original details and thus are limited in their ability to reduce size. Geometry simplification techniques aim to reduce the polygon count while retaining the geometric features of the original mesh as much as possible [Garland and Heckbert 1997; Lescoat et al. 2020; Surazhsky and Gotsman 2003; Szymczak et al. 2002]. Neural mesh simplification techniques [Potamias et al. 2022] have also been proposed to address the greedy nature of classical techniques. All of these methods reduce the polygon count at the expense of losing original details. Any of these methods could be used to produce the initial coarse mesh in our framework. In this paper, we opted for QSlim [Garland and Heckbert 1997] due to its simplicity and since it preserves the manifoldness and the watertightness of the input.

Progressive mesh representations have also been proposed [Hoppe 1996] to transmit details incrementally, but this technique does not aim to compress data, and thus can only reconstruct the original high-fidelity shape by transmitting all geometric information.

Surface upsampling. Mesh subdivision is a common tool to refine coarse meshes [Catmull and Clark 1978; Loop 1987; Zorin et al. 1996]. However, these methods employ hard-coded priors and usually recover (piecewise) smooth shapes, losing non-trivial details. One can optimize the coarse shape with respect to a particular subdivision scheme to maximize the reconstruction quality [Hoppe et al. 1994], but it has limited capabilities since the scheme itself stays fixed.

Neural networks can be used to significantly expand the space of geometric details created with a subdivision scheme [Hertz et al. 2020; Liu et al. 2020]. We build upon the Neural Subdivision [Liu et al. 2020] framework. Unlike Neural Subdivision, which encodes the local geometric details into the weights of the neural network, our method takes advantage of a subdivision-based encoder that encodes the local geometric details into per-face features. We introduce skip connections [Ronneberger et al. 2015] between the same level in the encoder and the decoder to improve the reconstruction quality and allow for progressive learnable features as additional input.

Morreale et al. [2022] also propose to represent a surface via two neural networks: one is a generic multi-layer perception (MLP) network that reconstructs coarse surfaces and the other is a detailization convolutional architecture. Unlike our method, this technique requires training two separate neural networks for each shape instance and thus requires transmitting the entire neural network for each asset, preventing it from effectively compressing details shared across different shapes and increasing computational costs.

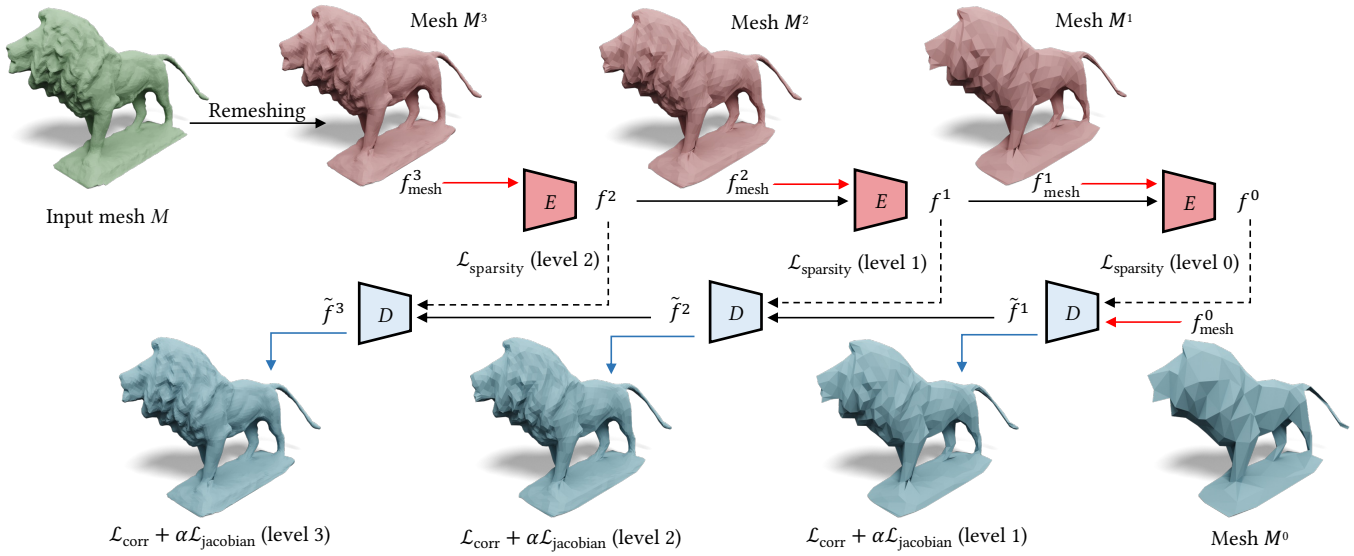

Fig. 2. **Overview of Neural Progressive Meshes.** Our network consists of an encoder E and a decoder D. Given an input mesh M to be transmitted, we first apply a remeshing scheme to obtain a sequence of LoD meshes $M^L \dots M^0$. The encoder learns per-face features that encode the local geometric details of each LoD mesh. The decoder learns to reconstruct a high-resolution mesh based on the transmitted coarsest mesh M^0. The reconstruction quality can be iteratively improved with features being progressively transmitted from the encoder to the decoder.

Geometric deep learning. A large number of neural shape representations have recently been developed. Unsupervised representation learning techniques often follow an encoder-decoder architecture [Baldi 2012; Bengio et al. 2009], where the encoder network maps shapes to a high-dimensional feature code and the decoder reconstructs the original data. Existing shape encoders include 2D CNNs over projections of shapes [McCormac et al. 2017; Su et al. 2015], 3D CNNs over voxel grids [Tran et al. 2015], point-based architectures [Qi et al. 2017a,b], surface-based techniques [Mitchel et al. 2021], mesh-based approaches that learn directly over the discrete mesh structure [Hanocka et al. 2019]. SubdivNet [Hu et al. 2022] is a mesh-based approach that operates on meshes with Loop [Loop 1987] subdivision sequence connectivity. The goal is to learn per-face features for dense prediction tasks on mesh surfaces. We build upon SubdivNet and develop a subdivision-based encoder. Our goal is different from SubdivNet in that we aim to encode the local geometric details into per-face features and use them to guide the subdivision process in the decoder.

One can also directly optimize for shape codes with respect to some reconstruction objective without using an encoder [Park et al. 2019]. Existing decoders generate shapes by folding 2D atlases into 3D surfaces [Groueix et al. 2018b], deforming template meshes [Groueix et al. 2018a], or predicting occupancies [Mescheder et al. 2019] or SDFs [Chen and Zhang 2019; Park et al. 2019].

While one can view an encoder-decoder architecture as an extreme version of neural compression, where the entire shape is compressed to a single feature vector. Existing decoders usually do not perform well at reconstructing high-resolution mesh details in comparison to surface upsampling techniques. By transmitting an optimized low-resolution mesh, we can also guarantee the preservation of the original topology and the watertightness of the input.

3 METHOD

3.1 Overview

Given a triangle mesh $M = (V, F)$ with vertex positions V and faces F, our goal is to encode it into a data stream $d_{0\dots t}$ which could be progressively transmitted to the client, where the client should be able to reconstruct a low-resolution shape from initial transmission d_0 and then iteratively improve it with subsequent transmissions $d_{1\dots t}$ for some bandwidth and time budget t.

We first obtain a sequence of LoD meshes $M^0 \dots M^L$, where $M^i = (V^i, F^i)$ and M^0 is the coarsest mesh with a fixed number of faces. The triangulation of each subsequent mesh $F^1 \dots F^L$ is defined by a simple subdivision rule iteratively applied to F^0. We also preserve correspondences during the simplification step and use this mapping to define vertex positions at each level of subdivision $V^0 \dots V^L$. When using our progressive representation, the server first transmits the coarsest mesh $d_0 = M^0$ to the client. The client uses the same subdivision scheme to reconstruct the sequence of meshes $\tilde{M}^i = (\tilde{V}^i, F^i)$, where $i = 1 \dots L$ (note that we apply the same subdivision rule on the client side, and thus the triangulation is exactly the same). We assume that the mesh at the highest level of subdivision M^L is sufficiently close to the input for all practical purposes, and thus reduce our problem to designing a method that can efficiently compress vertex coordinates V^L on the server side and enable reconstructing the coordinates $\tilde{V}^L$ on the client side.

To take advantage of the shared structures in local mesh geometry, we use an encoder-decoder approach, where we first encode geometric details as per-face features at each level i: f^i, and then the decoder uses the features to reconstruct the mesh at the highest level of detail. Our encoder E directly leverages LoD meshes by

learning filters that map features and vertex coordinates from higher-resolution to lower-resolution level: $E : [V^i, f^i] \rightarrow [V^{i-1}, f^{i-1}]$. Our decoder D, in a similar manner, maps from lower-resolution to higher-resolution level: $D : [V^i, f^i] \rightarrow [V^{i+1}, f^{i+1}]$. To speed up training and improve the quality of learned features, we connect corresponding faces at the same level of detail with skip connections, akin to U-Net architectures for images [Ronneberger et al. 2015]. We train our network so that the decoder can still reconstruct a plausible shape via the learned subdivision process even before all features f^i are transmitted. Specifically, to facilitate compression, we introduce a feature sparsity loss, which aims to set per-face features to 0 if they do not aid in reconstruction. In addition, we also have classical reconstruction losses based on vertex coordinates and their differential properties. See Figure 2 for an illustration of our network architecture and training losses.

At inference time, after the initial coarse mesh M^0 is transmitted to the client, the decoder reconstructs the high-resolution mesh by running the learned subdivision process with per-face features f^i set to 0. Our subsequent transmissions $d_{1\ldots t}$ simply assign non-zero features to some selected faces, enabling us to progressively improve the quality of the reconstructed shape. Due to the sparsity loss, we can simply sort the features by magnitude.

3.2 Neural Progressive Meshes

In this subsection, we discuss our neural progressive representation, including preprocessing and the encoder and decoder architectures.

LoD preprocessing. To derive our LoD $M^0 \ldots M^L$ representation used in the encoder, we first decimate the input mesh M via QSlim [Garland and Heckbert 1997] to obtain a coarse mesh M^0 with $|F^0| = 400$ faces. The target number of faces for simplification is picked to yield sufficiently coarse meshes to facilitate compression but also retain enough topological details for subdivision. To subdivide the coarse mesh into higher-resolution meshes, we first split each edge at the midpoint, subdividing each triangle into 4. This gives us the triangulations $F^1 \ldots F^L$, where $|F^i| = 4|F^{i-1}|$. To get vertex coordinates at the subdivision levels, we use successive self-parameterization [Liu et al. 2020], which allows us to map each point on each mesh M^i to the original mesh M, and we use that coordinate for all vertices: $V^1 \ldots V^L$. We set $L = 3$ for all experiments, which is selected to give us enough triangle budget to reconstruct shapes in our dataset. We sometimes refer to this step as remeshing, as M^L could be viewed as a remeshed version of M, which has similar geometry but a different triangulation.

Encoder. Our encoder E operates on the sequence of LoD meshes: $M^L, M^{L-1} \ldots M^0$, where all triangles in a high-resolution mesh can be grouped into groups of four and mapped to a single triangle on the next level of resolution based on the LoD scheme. We define convolution and pooling operators based on this mapping following SubdivNet [Hu et al. 2022]. The input per-face features at the highest level in the encoder are 13-dimensional (i.e., $f^L_{\text{mesh}} \in \mathbb{R}^{13}$), composed of a 7-dimensional shape feature (face area, three interior angles, and the inner product between the face normal and the vertex normals) and a 6-dimensional pose feature (face center coordinate and face normal). Unlike SubdivNet, where the input per-face features at the subsequent levels are just the output per-face features from the previous level, in our encoder, the input per-face features at the subsequent levels are a concatenation of the output per-face features from the previous level (i.e., f^i) and the 13-dimensional per-face features computed based on mesh M^i (i.e., f^i_{mesh}). This design allows us to encode the local geometric details of each LoD mesh into the feature encoding process. Our encoder maps the input per-face features at level i to learned per-face features at the subsequent level (i.e., level $i-1$): $f^{i-1} \in \mathbb{R}^8,\ \forall\ 1 \leq i \leq L$.

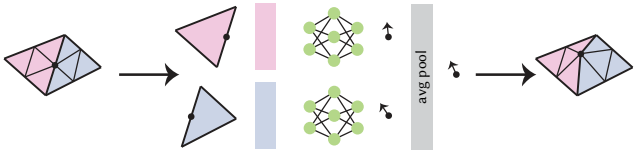

Fig. 3. **Vertex position prediction.** The decoder takes as input the features of the two adjacent faces and predicts the displacement for the midpoint.

Decoder. Our decoder D operates on the mesh transmitted at the coarsest level M^0 and can optionally leverage learned per-face features f^i. We first compute 13-dimensional shape and pose features (as described in the previous paragraph) of the coarse mesh M^0 to derive the per-face features f^0_{mesh}. We then concatenate features f^0_{mesh} with learned per-face features f^0. If features f^0 are not transmitted, we simply set $f^0 = 0$ for all faces. We treat the concatenated features as the input to the decoder. Instead of using half-flap representations as Neural Subdivision does, we adapt the Neural Subdivision architecture and develop a subdivision-based decoder that uses the features of the two adjacent triangles to predict vertex positions at the next level of subdivision $\tilde{V}^i, i = 1 \ldots L$, as shown in Figure 3. Our decoder maps the input per-face features at level i and optionally the learned per-face features transmitted from the same level in the encoder to per-face features at the next subdivision level in the decoder (i.e., level $i+1$): $f^{i+1} \in \mathbb{R}^8,\ \forall\ 0 \leq i \leq L-1$.

3.3 Network Training

We train our encoder-decoder network end-to-end using reconstruction and sparsity losses. The former favors higher quality of reconstruction and the latter favors sparser features and thus compression of the signal.

Reconstruction losses. Our reconstruction loss is composed of two terms. First, the ℓ_2 distance between vertex positions predicted by the decoder and true LoD positions:

$$\mathcal{L}_{\text{corr}} = \sum_{i=1}^{L} \frac{1}{|V^i|} \|\tilde{V}^i - V^i\|_2. \tag{1}$$

The second term is a loss in the gradient domain, measuring the similarity of Jacobians, which helps match differential properties of true and predicted LoD surfaces, such as normals and curvature:

$$\mathcal{L}_{\text{jacobian}} = \sum_{i=1}^{L} \frac{1}{|F^i|} \sum_{j=1}^{|F^i|} \|J^i_j - I\|_2, \tag{2}$$

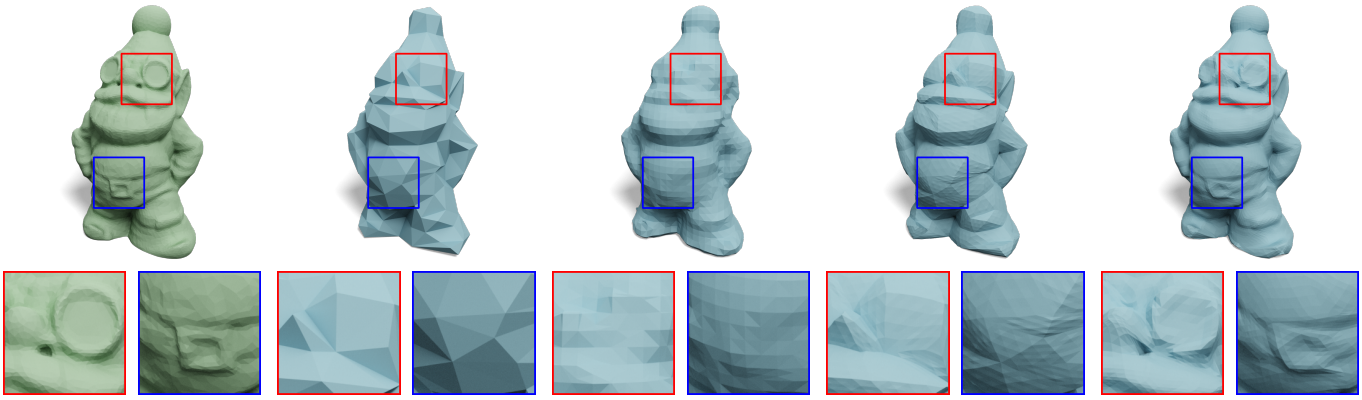

Fig. 4. **Visual comparisons with decimation and subdivision methods on Thingi10K.** The compression ratio (CR) is 61.39 for all methods.

where J_j^i is the Jacobian of the deformation that maps the j^{th} triangle of the true LoD mesh M^i to its predicted counterpart $\tilde{M}^i$, and I is the identity matrix.

Sparsity loss. To avoid transmitting features that encode redundant information in regions whose geometry could be inferred by the decoder without any aid, we introduce a sparsity loss:

$$\mathcal{L}_{\text{sparsity}} = \sum_{i=0}^{L-1} \frac{1}{|F^i|} \|f^i\|_1. \tag{3}$$

After network training, we sort the features based on the magnitude and transmit them progressively from the encoder to the decoder.

Total loss. We now define total training loss as a sum of weighted terms with $\alpha = 1$ and $\beta = 0.1$ for all experiments:

$$\mathcal{L} = \mathcal{L}_{\text{corr}} + \alpha \mathcal{L}_{\text{jacobian}} + \beta \mathcal{L}_{\text{sparsity}}. \tag{4}$$

Input preconditions. Our approach is not constrained to watertight, non-self-intersecting, near-delaunay triangulated meshes, and can be trained with such data by adding a preprocessing step. Given an input mesh, we use TetWild [Hu et al. 2018] to preprocess it.

4 EXPERIMENTS

4.1 Experimental Setup

Dataset. We evaluate our method on Thingi10K [Zhou and Jacobson 2016], a dataset of diverse models with interesting geometric and topological features. We start with the preprocessed watertight meshes provided by Hu et al. [2018] and filter out models that have more than 1 connected component or are not edge manifold, leaving us with 6,418 meshes. We sample 1,000 meshes and split them into training (80%), validation (10%), and test (10%) sets for experiments.

Evaluation metrics. We adopt the mean point-to-mesh distance d_{pm} and the average normal error d_{normal} to measure the similarity of the reconstructed mesh to the ground truth. The mean point-to-mesh distance first uniformly samples 1 million points on the surface of the final subdivided mesh $\tilde{M}^L$. Then it computes the average distance between each sampled point and the ground-truth mesh M. The normal error computes the average angle (in degrees) between the normal of the sampled point on mesh $\tilde{M}^L$ and the normal of the projected point on the ground-truth mesh M.

Table 1. **Comparisons with baseline methods on Thingi10K.** All methods have the same compression ratio (CR = 61.39). Our method performs the best on both metrics that measure the quality of reconstruction.

Method	d_{pm} ($\times 10^{-4}$) ↓	d_{normal} ↓
QSlim	30.11	13.21°
Loop	68.47	14.98°
Butterfly	40.55	16.99°
SubdivFit	27.33	15.41°
Neural Subdivision	19.03	11.21°
Ours	4.12	7.19°

We use the compression ratio (CR) to evaluate the effectiveness of different methods in transmitting LoDs. Since we did not want our metric to be skewed by various additional potential post-processes: generic compression algorithms, changing floating-point resolution, and topology-specific compression, we decided not to account for the transmission of topological data. However, in practice, this puts our method at a disadvantage, since we only need to transmit the topology for the coarse mesh, where all other triangulations are defined by subdivision. For all methods, we measure CR as:

$$CR = \frac{3|V|}{3|V^0| + \sum_{f \in \mathcal{T}} \dim(f)},$$

where $\dim(f)$ is the size of all transmitted features $f \in \mathcal{T}$.

Implementation details. Each level in our encoder is composed of a mesh convolution [Hu et al. 2022], a batch normalization [Ioffe and Szegedy 2015], a ReLU [Nair and Hinton 2010], and an average pooling [Hu et al. 2022]. Each level in our decoder is composed of two modules, one for predicting vertex displacements and the other for feature learning. The vertex displacement prediction module is

CR / d_{pm} ($\times 10^{-4}$) / d_{normal}	1.15 / 2.80 / 10.15°	1.15 / 2.91 / 8.79°	1.15 / 2.52 / 4.73°	39.32 / 2.74 / 8.81°
Ground truth	ACORN	NGLOD	NCS	Ours

Fig. 5. **Visual comparisons with neural overfitting methods.** Our result is comparable to neural overfitting methods in d_{pm} and d_{normal} while having a much higher compression ratio.

a single fully connected layer. The feature learning module consists of a bilinear upsampling [Hu et al. 2022], a mesh convolution [Hu et al. 2022], a batch normalization, and a ReLU.

We train our network using the ADAM [Kingma and Ba 2014] optimizer in PyTorch [Paszke et al. 2019]. The initial learning rate is set to 1×10^{-3}, and the learning rate decay is set to 1×10^{-6}. We scale each mesh in the dataset to fit a unit cube. We then employ two data augmentation strategies. First, we randomly decimate each mesh to 10 different coarse meshes. Second, we randomly rotate the meshes at all levels of subdivision by a random rotation (in 90° increments) around each of the three axes.

4.2 Mesh Compression

We evaluate our method and several baselines for the mesh compression task using compression and reconstruction metrics. Since all methods will have a tradeoff between the compression ratio and the reconstruction quality, for all comparisons presented in this subsection we set the parameters of each method to reach the same compression ratio as ours, and only compare the reconstruction quality. We show quantitative results in Table 1 and qualitative results in Figures 4 and 9 for our method and baselines. See Section 3 in the supplemental material for more results of our method. We next detail the choice of our baselines.

We first compare to a mesh decimation approach, QSlim [Garland and Heckbert 1997], which reduces the mesh size by greedily collapsing edges with the lowest quadric error. This method is expected to lose details since it does not have an upsampling step. One can run a classical subdivision scheme (e.g., Loop [Loop 1987] or Butterfly [Zorin et al. 1996]). However, as observed in Table 1, it only worsens the performance, since these methods are not aware of the priors used by QSlim. One can specifically optimize the simplified mesh based on the subdivision method, e.g., SubdivFit [Hoppe et al. 1994], which improves the reconstruction accuracy. We find the learnable subdivision method (Neural Subdivision [Liu et al. 2020]) further improves the accuracy. Note that our method yields the highest accuracy at the same level of compression since it learns surface-specific features across a collection of shapes and can adaptively transmit features only in regions that need the most details.

4.3 Comparison to Mesh Compression by Neural Overfitting

A few recent techniques have been proposed for compressing data by overfitting neural representations to high-resolution meshes, e.g., NCS [Morreale et al. 2022], ACORN [Martel et al. 2021], and NGLOD [Takikawa et al. 2021]. These methods require transmitting all network weights for each mesh, do not learn a space of local details, and have been overfitted to very high-resolution meshes. Since these methods would not be very effective at our lower-resolution meshes, we run our method on their data and show results in Figure 5. The compression ratio for these methods is defined as the ratio between the ground truth mesh file size and the network file size. Even though our method was trained on much lower resolution data, it achieves a comparable quality of reconstruction (measured by d_{pm} and d_{normal}) compared to neural overfitting baselines while offering a much higher compression ratio.

4.4 Progressive Meshes

We now demonstrate progressive transmission, a key feature of our method. Note that all mesh compression techniques we discussed so far can only transmit a single shape with a particular bandwidth budget. To upgrade the resolution of the shape, the entire mesh needs to be re-transmitted at a higher resolution, potentially transmitting redundant information multiple times.

To evaluate the effectiveness of progressively transmitting features, we conduct an analysis by varying the number of features transmitted from the encoder to the decoder and look at the quality of the resulting reconstructions (see Table 2, Figures 6 and 11). See Section 2 in the supplemental material for more visual results. Note the gradual improvement in the quality of the reconstructed local details as more features are being transmitted.

We further compare our method to baselines for variable compression ratio in Figure 7, where the x-axis shows CR and the y-axis shows point-to-mesh distance. As noted, previously discussed mesh simplification baselines (QSlim) and different subdivision schemes applied to QSlim (Loop, Butterfly, Neural Subdivision) do not allow transmitting incremental data to progressively improve the subdivision quality. Thus, these methods are not directly comparable, and we render them as scatter points for context. It is worth noting,

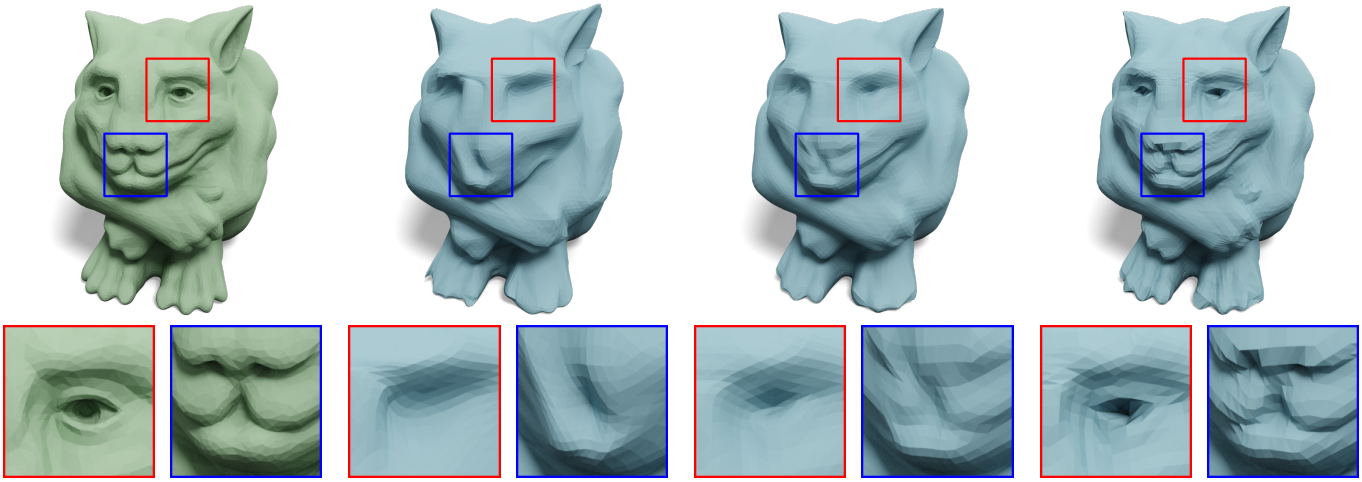

Fig. 6. **Progressive features.** Our results can be progressively improved if additional features are transmitted.

Table 2. **Ablation study on progressive features.**

Method	d_{pm} ($\times 10^{-4}$) ↓	d_{normal} ↓
Ours w/o features	14.56	12.36°
Ours + 40 features	6.81	9.20°
Ours + 400 features	4.12	7.19°

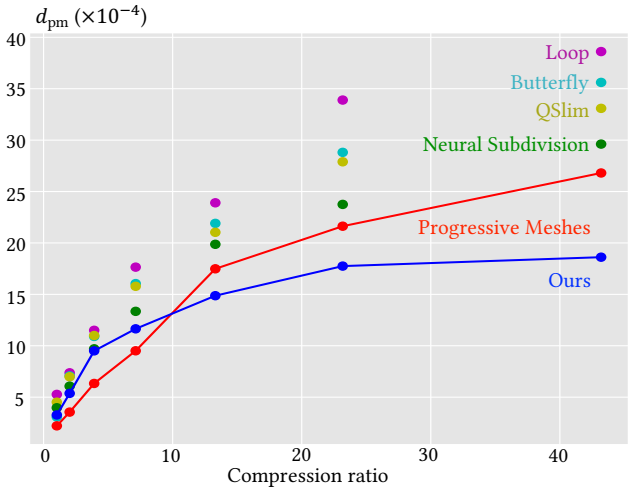

Fig. 7. **Compression ratio vs. point-to-mesh distance curve.**

nevertheless, that for any compression ratio, our method provides a higher reconstruction quality than these alternatives.

Progressive Meshes [Hoppe 1996] is the pioneering method that inspired our work and allowed us to use incremental features to progressively recover the details of a mesh. Since Progressive Meshes is a lossless method, it outperforms our approach for a smaller CR. However, we observe that the gap widens for $CR > 10$, suggesting that our method is especially effective when an asset needs to be significantly reduced in size. See Figure 10 for a visual comparison.

4.5 Levels of Detail

Our method design provides flexibility to the user on the client side to determine the resolution of the subdivided mesh. This choice does not affect the compression ratio but can be used to optimize the use of computational resources (e.g., displaying lower-level of subdivision for far-away assets). We conduct an ablation study that evaluates the quality of the subdivided mesh at each subdivision level (see Table 3 and Figure 8). As expected, the quality increases with each subdivision level, but returns are diminishing, even though the number of triangles increases exponentially by a factor of 4.

4.6 Limitations and Failure Cases

Our method fails when applied to a shape with complex topological details and intricate thin features. See Section 4 in the supplemental material for an example. Although not suitable for lossless compression, our method provides a superior lossy compression for a wide section of the size-accuracy spectrum.

4.7 Runtime

We train and test the network on an Intel(R) Core(TM) i7-12700K CPU machine with one NVIDIA A40 GPU. The network training takes around 2 days. At test time, the server needs 4.84s to remesh a 100,000-face mesh to 400 faces (CPU-only), and the encoder forward pass takes 4.02s to predict per-face features on the GPU. It takes 4.58s for the client to subdivide a 400-face mesh to 25,600 faces (i.e., 3 subdivision levels) on the GPU. Our method can also run on the CPU. In the CPU-only case, the encoder forward pass takes 7.42s to

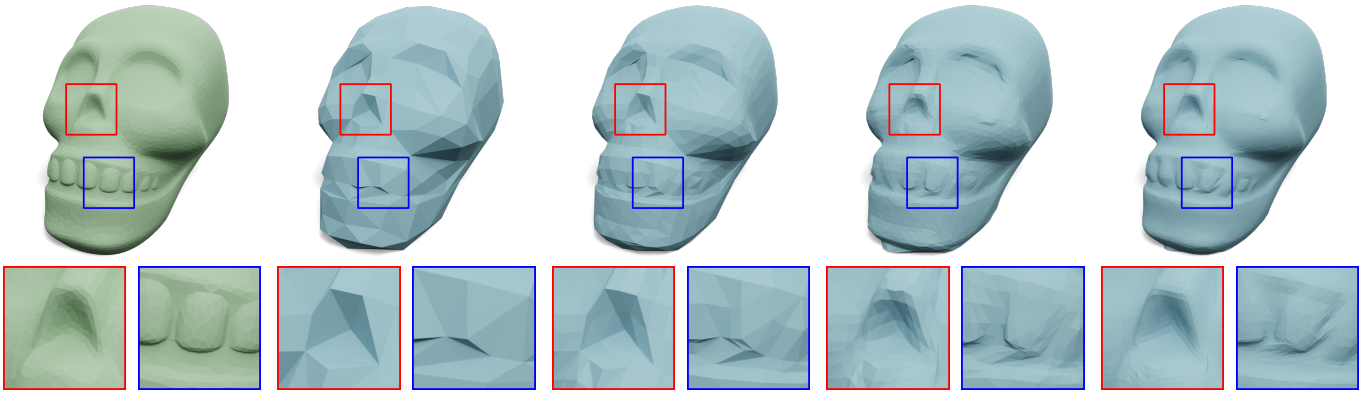

Fig. 8. **Levels of detail.** Our decoder is able to subdivide a coarse mesh to different LoD meshes, providing the user with the flexibility to determine the resolution of the subdivided mesh.

Table 3. **Ablation study on levels of detail.**

Method	# triangles	d_{pm} ($\times 10^{-4}$) ↓	d_{normal} ↓
Ours level 1	1,600	12.47	11.23°
Ours level 2	6,400	4.36	8.66°
Ours level 3	25,600	4.12	7.19°

predict per-face features, and the decoder forward pass takes 6.93s to subdivide a 400-face mesh to 25,600 faces.

5 CONCLUSIONS

We propose Neural Progressive Meshes, a novel representation that allows us to learn the space of surface details and efficiently compress them into per-face features. These features can be transmitted progressively, enabling our method to iteratively improve the quality of the reconstructed mesh as more data is transmitted. We demonstrate that our method is especially effective when only a small fraction of the original shape can be transmitted and outperforms other compression techniques, mesh simplification and subdivision approaches, and progressive mesh representations.

ACKNOWLEDGMENTS

This project is funded in part by NSERC Discovery (RGPIN–2022–04680), the Ontario Early Research Award program, the Canada Research Chairs Program, a Sloan Research Fellowship, the DSI Catalyst Grant program and gifts by Adobe Systems. We thank Hsueh-Ti Derek Liu for help with the Neural Subdivision code and Silvia Sellán, Abhishek Madan and Selena Ling for help with making figures.

REFERENCES

Pierre Alliez and Mathieu Desbrun. 2001. Valence-driven connectivity encoding for 3D meshes. In *Computer graphics forum*.
Pierre Baldi. 2012. Autoencoders, unsupervised learning, and deep architectures. In *ICMLW*.
Yoshua Bengio et al. 2009. Learning deep architectures for AI. *Foundations and trends in Machine Learning* (2009).
Edwin Catmull and James Clark. 1978. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-aided design* (1978).
Zhiqin Chen and Hao Zhang. 2019. Learning implicit fields for generative shape modeling. In *CVPR*.
Michael Deering. 1995. Geometry compression. In *Conference on Computer graphics and interactive techniques*.
Michael Garland and Paul S Heckbert. 1997. Surface simplification using quadric error metrics. In *Computer graphics and interactive techniques*.
Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. 2018a. 3d-coded: 3d correspondences by deep deformation. In *ECCV*.
Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. 2018b. A papier-mâché approach to learning 3d surface generation. In *CVPR*.
Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. 2019. Meshcnn: a network with an edge. *ACM TOG* (2019).
Amir Hertz, Rana Hanocka, Raja Giryes, and Daniel Cohen-Or. 2020. Deep Geometric Texture Synthesis. *ACM TOG* (2020).
Hugues Hoppe. 1996. Progressive meshes. In *Conference on Computer graphics and interactive techniques*.
Hugues Hoppe, Tony DeRose, Tom Duchamp, Mark Halstead, Hubert Jin, John McDonald, Jean Schweitzer, and Werner Stuetzle. 1994. Piecewise smooth surface reconstruction. In *Annual conference on Computer graphics and interactive techniques*.
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. *TOG* (2022).
Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2018. Tetrahedral meshing in the wild. *ACM TOG* (2018).
Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*.
Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *ICLR*.
Thibault Lescoat, Hsueh-Ti Derek Liu, Jean-Marc Thiery, Alec Jacobson, Tamy Boubekeur, and Maks Ovsjanikov. 2020. Spectral mesh simplification. In *Computer Graphics Forum*.
Hsueh-Ti Derek Liu, Vladimir G Kim, Siddhartha Chaudhuri, Noam Aigerman, and Alec Jacobson. 2020. Neural subdivision. *ACM TOG* (2020).
Charles Loop. 1987. Smooth subdivision surfaces based on triangles. *Master's thesis, University of Utah, Department of Mathematics* (1987).
Julien NP 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 TOG* (2021).
John McCormac, Ankur Handa, Andrew Davison, and Stefan Leutenegger. 2017. Semanticfusion: Dense 3d semantic mapping with convolutional neural networks. In *ICRA*.
Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. 2019. Occupancy networks: Learning 3d reconstruction in function space. In *CVPR*.
Thomas W Mitchel, Vladimir G Kim, and Michael Kazhdan. 2021. Field convolutions for surface CNNs. In *ICCV*.

Luca Morreale, Noam Aigerman, Paul Guerrero, Vladimir G. Kim, and Niloy Mitra. 2022. Neural Convolutional Surfaces. In *CVPR*.
Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *ICML*.
Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. Deepsdf: Learning continuous signed distance functions for shape representation. In *CVPR*.
Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*.
Rolandos Alexandros Potamias, Stylianos Ploumpis, and Stefanos Zafeiriou. 2022. Neural mesh simplification. In *CVPR*.
Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017a. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*.
Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. 2017b. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*.
Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*.
Jarek Rossignac. 1999. Edgebreaker: Connectivity compression for triangle meshes. *TVCG* (1999).
Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. 2015. Multi-view convolutional neural networks for 3d shape recognition. In *ICCV*.
Vitaly Surazhsky and Craig Gotsman. 2003. Explicit surface remeshing. In *SGP*.
Andrzej Szymczak, Davis King, and Jarek Rossignac. 2001. An Edgebreaker-based efficient compression scheme for regular meshes. *Computational Geometry* (2001).
Andrzej Szymczak, Jarek Rossignac, and Davis King. 2002. Piecewise regular meshes: Construction and compression. *Graphical Models* (2002).
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 *CVPR*.
Gabriel Taubin and Jarek Rossignac. 1998. Geometric compression through topological surgery. *ACM TOG* (1998).
Costa Touma and Craig Gotsman. 1998. Triangle mesh compression. In *Proceedings-Graphics Interface*.
Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. 2015. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*.
Qingnan Zhou and Alec Jacobson. 2016. Thingi10k: A dataset of 10,000 3d-printing models. In *SGP*.
Denis Zorin, Peter Schröder, and Wim Sweldens. 1996. Interpolating subdivision for meshes with arbitrary topology. In *Computer graphics and interactive techniques*.

Fig. 9. **Visual comparisons with decimation and subdivision methods on Thingi10K.** We report the d_{pm} ($\times 10^{-4}$) / d_{normal} results under each method. The compression ratio (CR) is the same for all methods on the same shape and is reported under the ground-truth example.

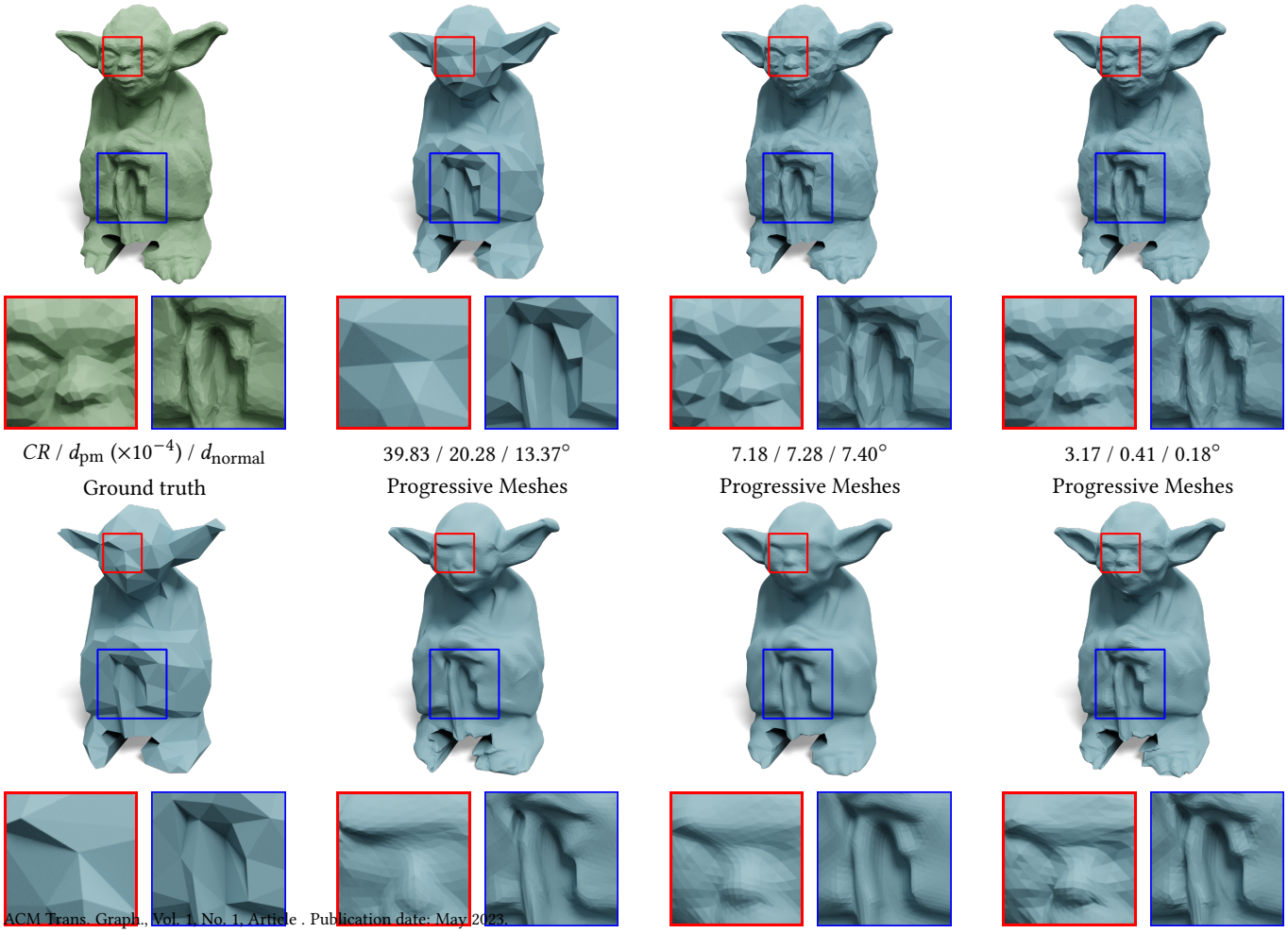

Fig. 10. **Comparison to Progressive Meshes.**

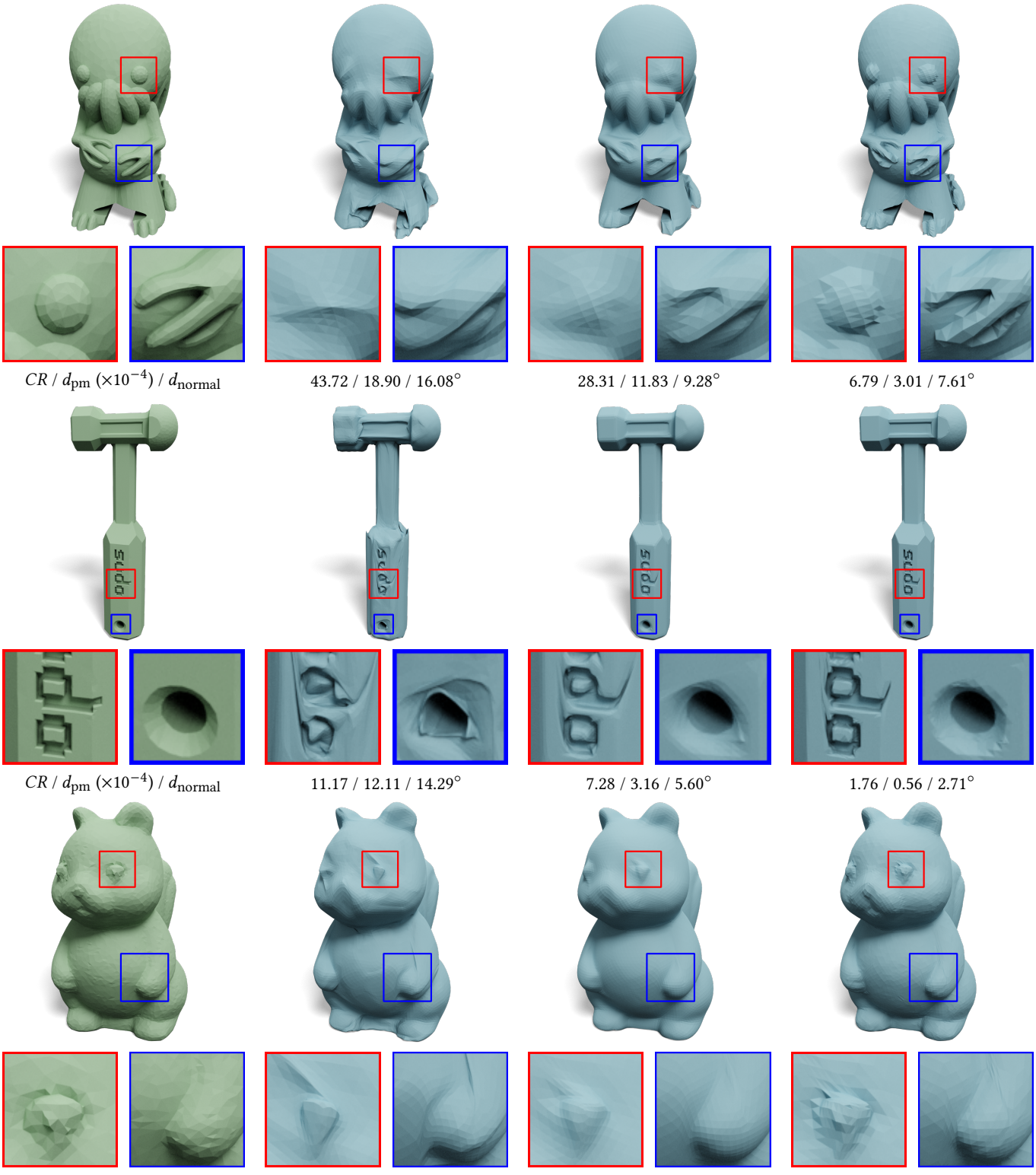

Fig. 11. **Progressive features.** We show more examples where transmitting more features leads to better quantitative and qualitative results.