Supplementary Material for "PoNQ: a Neural QEM-based Mesh Representation"

Nissim Maruani
Inria, Université Côte d'Azur
nissim.maruani@inria.fr

Maks Ovsjanikov
LIX, École Polytechnique, IP Paris
maks@lix.polytechnique.fr

Pierre Alliez
Inria, Université Côte d'Azur
pierre.alliez@inria.fr

Mathieu Desbrun
Inria Saclay - Ecole Polytechnique
mathieu.desbrun@inria.fr

In this supplemental material, we discuss further evaluations of our PoNQ representation by comparing it with [9] and [10] in Sec. 1, before providing a number of implementation details in Sec. 2 which were not thoroughly covered in the main paper.

1. Discussing Other Related Works

To complement the series of comparisons presented in the main paper, we also discuss how PoNQ differs from two other (less) related works.

1.1. Reach for The Spheres [9]

A recent work on surface reconstruction of SDF, called Reach for The Spheres (RTS) [9], relies on the deformation of an initial mesh, thus requiring that its genus matches the genus of the underlying SDF-encoded shape. In practice, several shapes of the Thingi30 have a genus strictly higher than one. Consequently, and despite our best efforts, we could not run the code provided by the authors of Reach for The Spheres (RTS) [9] on the Thingi30 dataset: segmentation faults and inverted elements systematically appeared in the optimization process when starting from a unit sphere — and using the output of a marching-cube (MC) extraction instead did not solve this problem, since the resulting genus depends on the MC table being used. Nevertheless, we provide for completeness a visual comparison of the two methods in Fig. 1 for two shapes that RTS could handle. While RTS exhibits many artifacts and PoNQ is far more robust to the genus of shapes, we note that the RTS empty sphere constraint could potentially be added to our loss function.

(a) RTS [9] (b) PoNQ (c) Ground truth

Figure 1. Learning-based results for PoNQ. Optimization-based results for Reach For the Spheres [9]. Note that both method rely on the same input, here a 32^3 SDF grid. RTS can either miss thin structures (bottom rows) or fill thin voids (top rows).

1.2. Deep Marching Tetrahedra [10]

Regarding Deep Marching Tetrahedra (DMTet) [10], which relies on the deformation of a regular tetrahedra grid, two critical differences exist between their work and ours:

- strictly speaking, DMTet is not a point-based approach: their hybrid representation requires a neural network to model a continuous field of SDF and displacements;
- unless one restricts the amplitude of the deformation, it

cannot guarantee intersection-free output meshes.
DMTet is thus quite distinct from PoNQ, making it difficult to compare them fairly. Nonetheless, we tried to provide a comparison; but given the absence of DMTet implementation for surface reconstruction from SDF grids, we found ourselves unable to provide a meaningful visual comparison — just like VoroMesh [8] was unable to compare with DMTet for the simpler case of the optimization-based task.

2. Implementation Details

We now cover in detail a series of aspects of PoNQ that were not fully detailed in the main paper.

2.1. Training

For all the examples we show in our paper, we trained our PoNQ network with the AdamW optimizer and a linear combination of the losses, i.e.,

$$L = \alpha_{\text{CD}} L_{\text{CD}} + \alpha_{\mathbf{n}} L_{\mathbf{n}} + \alpha_{\mathbf{A}} L_{\mathbf{A}} + \alpha_{\mathbf{v}^*} L_{\mathbf{v}^*} + \alpha_{reg} L_{reg} + \alpha_{occ} L_{occ}$$

We rely on three training phases of 200 epochs each, with batches of size 16 with the following learning rates γ, weights $\boldsymbol{\alpha} = (\alpha_{\text{CD}}, \alpha_{\mathbf{n}}, \alpha_{\mathbf{A}}, \alpha_{\mathbf{v}^*}, \alpha_{reg}, \alpha_{occ})$ and point samples S:

- $\gamma = 6.4 \cdot 10^{-5}$, $\boldsymbol{\alpha} = (100, .1, .1, 100, 100, .1)$, $S = 5 \cdot 10^5$
- $\gamma = 3.2 \cdot 10^{-5}$, $\boldsymbol{\alpha} = (100, .1, .1, 100, 100, .1)$, $S = 7 \cdot 10^5$
- $\gamma = 3.2 \cdot 10^{-5}$, $\boldsymbol{\alpha} = (100, .1, .1, 100, 1, .1)$, $S = 7 \cdot 10^5$.

2.2. Meshing

As explained in the main paper, the choices we made to design our QEM-based PoNQ representation allow for a simple meshing approach, inspired by computational geometry to ensure robustness: due to the local optimality of the positions $\mathbf{v}_i^*$ (in particular, their ability to capture corners and sharp features), our output mesh is produced by filtering the triangle facets of a 3D Delaunay triangulation of the optimal positions $\mathbf{v}_i^*$. Given the PoNQ data (produced by a trained network or through optimization), we now describe how a PoNQ mesh is extracted in full detail.

0. Pre-processing We first normalize the quadrics by dividing them by their largest eigenvalue. While this technically changes their meaning (they now measure Euclidean distances *up to a multiplicative constant*), it also removes the possible bias due to variable sampling density. Additionally, a bounding box B of all the points $\mathbf{v}_i^*$ is computed.

1. Triangulation of QEM optimal positions. We then compute the Delaunay tetrahedralization of all points $\mathbf{v}_i^*$, to which we add eight "protective" points defined as the corners of the bounding box B: all the adjacent tetrahedra to these protective points will be guaranteed to be outside of the shape we wish to reconstruct. The next two steps will tag each of the remaining tetrahedra as either *inside* or *outside* based on local geometric information, so that *our final PoNQ mesh will simply be the triangle mesh forming the boundary between the inside and outside regions*, ensuring watertightness and no self-intersections by design.

2. Tagging obvious inside/outside tetrahedra. We first tag all the tetrahedra adjacent to the eight protective point as *outside*. Tagging the rest of the tetrahedra seems daunting, but by leveraging the ideas put forth in the Crust algorithm [1], we note that the location of the circumcenter of a Delaunay tetrahedron whose four vertices are on the surface to mesh (which is the case we are in) can often determine the insideness or outsideness of this tetrahedron — corresponding to whether this circumcenter is on the inside (resp. outside) medial axis. We use a similar approach here, except that our Delaunay vertices have additional information to help us: we also know a local normal $\mathbf{n}_i$ in the vicinity of each vertex $\mathbf{v}_i^*$. Thus, each vertex and its assigned normal defines an oriented plane, forming the boundary between the outside half-space (the one pointed by the normal) and the inside half-plane. Using this local test to determine inside/outside, we tag a tetrahedron as *outside* (resp., *inside*) if both its circumcenter and barycenter are determined to be in the outside (resp., inside) half-space of each of its four vertices. We then go through every (non-protective) vertex that already has at least one tagged adjacent tetrahedron. If such a vertex does not have any adjacent *outside* (resp., *inside*) tetrahedron, we pick its untagged adjacent tetrahedron with the smallest edge; if the size of this smallest edge is below a certain large threshold (i.e., we are not in a very sparse region of the domain), we then tag this selected tetrahedron as *outside* (resp., *inside*). The rationale behind this last round of tagging is that we know that all vertices of our 3D Delaunay triangulation are *on* the surface, so unless the smallest tetrahedron is too big (in this case, there is clearly a large uncertainty), we can safely tag it to be on the opposite side of the surface than what the other tags had already determined. While this tagging procedure can, on rare occasions, tag *all* tetrahedra, there are often a few remaining untagged tetrahedra (typically caused by the presence of near sharp features or thin structures) that are too ambiguous to tag. We now need to lift the remaining uncertainty based on additional PoNQ data.

3. Finishing up triangle selection. Delaunay-based meshing approaches (like *Crust*) require a dense point sampling (formally, an $\epsilon-$sampling) to offer topological and geometric guarantees, which is not compatible with our desire to deal with thin structures, sharp features and corners — and this is the main reason why our earlier phase often ends up not providing a tag for *every* tetrahedron. To finish our tetrahedron tagging based on the ones we already have, we propose to use a graph cut approach, inspired by existing

spectral graph partitioning [7]. For each Delaunay triangle T between two adjacent tetrahedra for which *at least* one of them is still undetermined, we compute a likelihood score $S(T)$ that evaluates how confident we are that this triangle is to appear on the final output mesh. We propose a score that evaluates the fitness of T based on the local PoNQ normals and the local PoNQ quadrics matrices:

$$S(T) = S_{\mathbf{n}}(T) + hS_{\mathbf{Q}}(T)$$

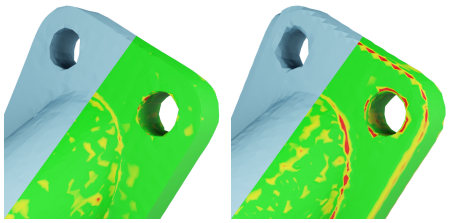
(with h set as the squared inverse of the edge length of the SDF grid), where:

$$S_{\mathbf{n}}(T) = \Big(\frac{2}{\pi} \sum_{\mathbf{v}_i^* \in T} \arccos(n_T^t \, \mathbf{n}_i)\Big)^2,$$

$$S_{\mathbf{Q}}(T) = \sum_{\mathbf{v}_i^* \in T} \sum_{\substack{\mathbf{v}_j^* \in T \\ i \neq j}} [\mathbf{v}_j^*, 1]^t \mathbf{Q}_i [\mathbf{v}_j^*, 1],$$

where n_T denotes the normal of triangle T. We can now tag the remaining undetermined tetrahedra with a definite *inside* or *outside* label: we compute a minimum cut of the Voronoi graph (in which each dual of a tetrahedron is a node, and each dual of a Delaunay triangle face is an edge) using the already-tagged "inside" ones as a source and the "outside" ones as a drain, and each edge weight between two tetrahedra (with a common face T) set to the score $S(T)$. Since most of tetrahedra are already tagged, we can merge Voronoi edges between marked tetrahedra to reduce the graph size and thus accelerate computations. We now just extract the final *PoNQ mesh* as the triangle mesh forming the boundary between the inside and outside regions.

2.3. Ablation studies



(a) PoNQ (b) PoNQ w/out QEM

(c) PoNQ (d) PoNQ w/out S_Q

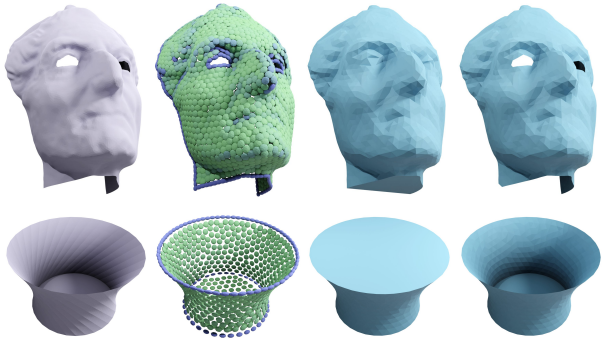
Figure 2. A network trained without QEM (b) fails to recover sharp edges. The second-order QEM information provided by S_Q helps to disambiguate tetrahedra labeling (c).

To further justify the need for QEM, we provide two ablation studies: we re-generated models with a simpler version of our meshing algorithm that does not rely on quadrics (thus removing S_Q), and we re-trained a network producing only points, normals, and voxel occupancies (thus removing the QEM part and its associated losses L_A, L_{v^*}, and L_{reg}). Figure 2 shows that the QEM-optimal vertex placement is essential to fit sharp features, and that using only S_n can mess up labeling in intricate/thin regions. Quantitatively, non-QEM versions fall behind our competitors in both surface and sharp-edge fitting scores, see Table 1.

Method	Grid size	CD ↓ ($\times 10^{-5}$)	F1 ↑	NC ↑	ECD ↓	EF1 ↑
retrained w/o QEM	32^3	**1.327**	0.840	0.960	**0.184**	0.598
PoNQ w/o S_Q	32^3	1.801	0.851	**0.964**	0.191	**0.715**
PoNQ	32^3	1.514	**0.852**	**0.964**	**0.184**	0.713
retrained w/o QEM	64^3	0.921	0.891	0.979	0.115	0.837
PoNQ w/o S_Q	64^3	0.931	**0.892**	**0.980**	**0.103**	0.863
PoNQ	64^3	**0.886**	**0.892**	**0.980**	0.109	**0.866**
retrained w/o QEM	32^3	1.387	0.803	0.942	0.139	0.286
PoNQ w/o S_Q	32^3	1.475	**0.810**	**0.943**	**0.137**	**0.315**
PoNQ	32^3	**1.344**	**0.810**	0.942	**0.137**	0.314
retrained w/o QEM	64^3	0.784	0.922	**0.971**	0.102	0.489
PoNQ w/o S_Q	64^3	0.779	**0.924**	**0.971**	0.102	**0.511**
PoNQ	64^3	**0.758**	**0.924**	**0.971**	**0.100**	**0.511**
retrained w/o QEM	128^3	0.645	**0.939**	**0.984**	0.135	0.556
PoNQ w/o S_Q	128^3	0.671	0.938	**0.984**	0.126	0.584
PoNQ	128^3	**0.641**	**0.939**	**0.984**	**0.123**	**0.592**

Table 1. Ablation studies on ABC (top) and Thingi30 (bottom).

2.4. Open surfaces

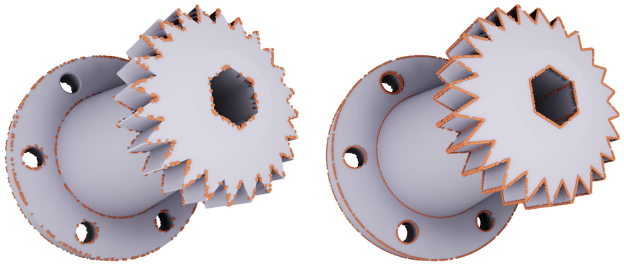


(a) Gr. Truth (b) PoNQ (c) Closed mesh (d) Open mesh

Figure 3. Optimization-based results for PoNQ when open boundaries are used.

We provide more results in Fig. 3, this time for open surfaces. Note that the QEM-optimized vertices $\mathbf{v}^*$ snap to sharp and boundary features, allowing for clean reconstructions. As explained in the main paper, we currently transit via a closed mesh (see Fig. 3c) that is then filtered to create holes between open boundaries, thus limiting our representation of open surfaces. However, since the PoNQ representation provides a precise fit of the target surface (see Fig. 3b), we believe a more sophisticated meshing could allow for arbitrary open surfaces.

3. Edge-based CD: Discussion



(a) Surface-based ECD (b) Edge-based ECD

Figure 4. Comparison between two different ECD evaluation methods: sampling surface points and identifying right-angle normal changes between neighbors (left image) is less accurate than sampling sharp edges (right image), and fails to capture sharp edges with angles different than 90 degrees.

The Edge Chamfer Distance was introduced in BSP-Net [4] to evaluate sharp reconstructions on ShapeNet [2]. It is based on a sampling of the surface, and only captures sharp edges featuring an angle close to 90 degrees, see Fig. 4a. While this metric might be sufficient for ShapeNet, in which most sharp angles lie on the edges of boxy objects (chairs, televisions...), it is no longer true for ABC [6] where sharp edges can have a variety of dihedral angles. As a result, we observed a large variance in this metric evaluation depending on the initial random surface sampling; moreover, it also failed to capture the spurious sharp edges of VoroMesh [8] due to faces of small area not being sampled. For these two reasons, and since most meshes of the ABC dataset have clean and well-defined sharp edges, we found that sampling the sharp edges (defined as those with a dihedral angle larger than $\frac{\pi}{6}$) directly provided a more robust and faithful metric (see cog teeth in Fig. 4b).

3.1. Additional timings

VoroMesh [8] provides two implementations for their mesh extraction, based on SciPy and CGAL. Since our mesh extraction is also based on SciPy, we use their SciPy one out of fairness in Tab. 2 to evaluate and compare timings; we also added the timings of the meshing phase of NMC [5] and NDC [3] for comparison purposes. While our code is not yet optimized, an implementation of PoNQ in CGAL would undoubtedly allow for an even faster extraction.

Method	Grid	Optimization (s)	Meshing (s)
NDC [3]	32^3	-	0.001
NMC [5]	32^3	-	0.1
VoroMesh [8]	32^3	2.0	0.3
PoNQ	32^3	2.6	0.3
NDC [3]	64^3	-	0.02
NMC [5]	64^3	-	0.9
VoroMesh [8]	64^3	4.2	1.1
PoNQ	64^3	4.1	1.3
NDC [3]	128^3	-	0.2
NMC [5]	128^3	-	7.3
VoroMesh [8]	128^3	36.2	4.8
PoNQ	128^3	17.9	7.8

Table 2. Timings for optimization-based experiments.

4. Additional Renders

Finally, Figs. 5-9 exhibit further results comparing PoNQ to previous works.

References

[1] Nina Amenta, Marshall Bern, and Manolis Kamvysselis. A new Voronoi-based surface reconstruction algorithm. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques - SIGGRAPH '98*, pages 415–421, Not Known, 1998. ACM Press. 2

[2] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository, Dec. 2015. arXiv:1512.03012 [cs]. 4

[3] Zhiqin Chen, Andrea Tagliasacchi, Thomas Funkhouser, and Hao Zhang. Neural dual contouring. *ACM Transactions on Graphics*, 41(4):1–13, July 2022. 4

[4] Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. BSP-Net: Generating Compact Meshes via Binary Space Partitioning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 42–51, Seattle, WA, USA, June 2020. IEEE. 4

[5] Zhiqin Chen and Hao Zhang. Neural marching cubes. *ACM Transactions on Graphics*, 40(6):1–15, Dec. 2021. 4

[6] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. ABC: A Big CAD Model Dataset For Geometric Deep Learning, Apr. 2019. arXiv:1812.06216 [cs]. 4

[7] Ravikrishna Kolluri, Jonathan Richard Shewchuk, and James F. O'Brien. Spectral surface reconstruction from noisy point clouds. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 11–21, Nice France, July 2004. ACM. 3

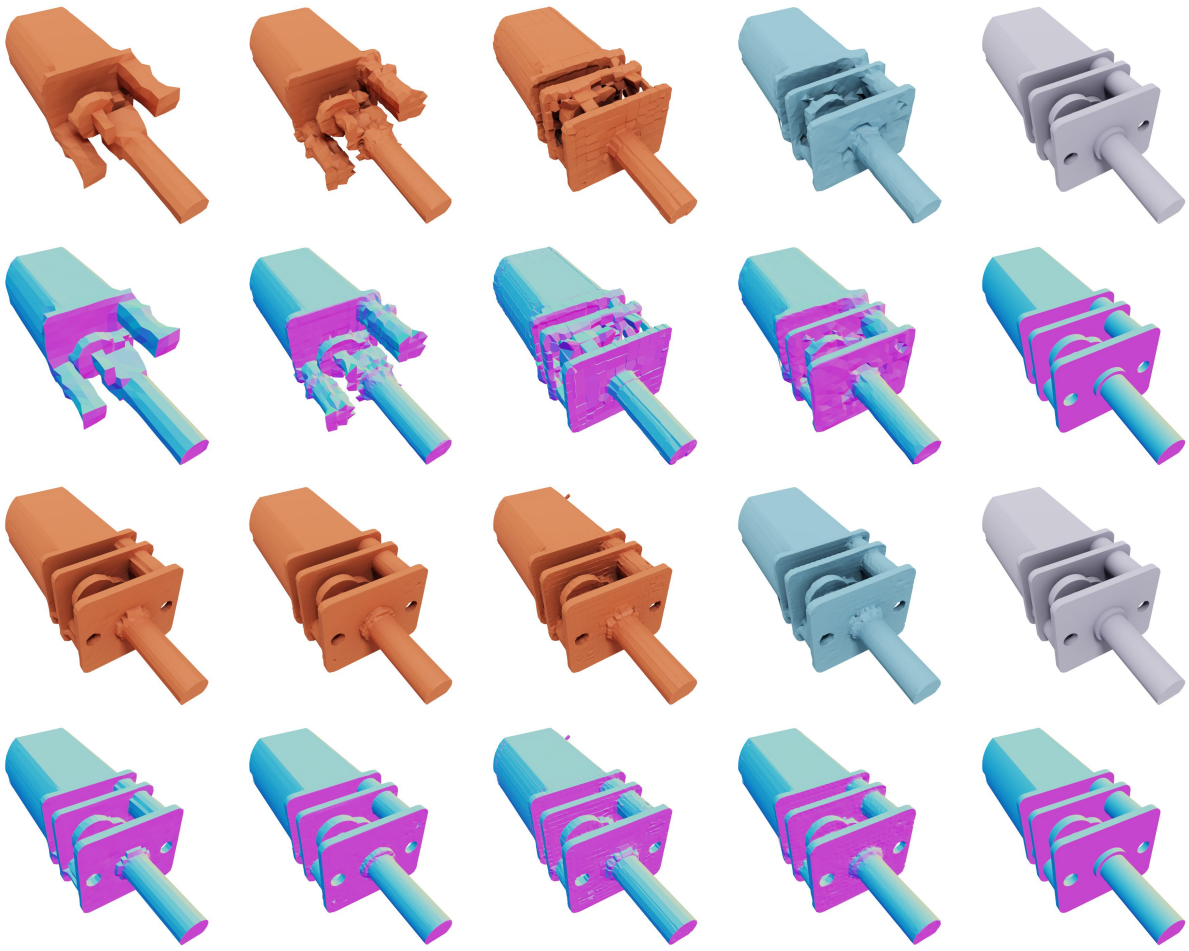
[8] Nissim Maruani, Roman Klokov, Maks Ovsjanikov, Pierre Alliez, and Mathieu Desbrun. VoroMesh: Learning Watertight Surface Meshes with Voronoi Diagrams. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 14565–14574, 2023. 2, 4

[9] Silvia Sellán. Reach For the Spheres: Tangency-Aware Surface Reconstruction of SDFs. *ACM Transactions on Graphics*, 2023. 1

[10] Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. Deep Marching Tetrahedra: a Hybrid Representation for High-Resolution 3D Shape Synthesis. In *Advances in Neural Information Processing Systems*, volume 34, pages 6087–6101. Curran Associates, Inc., 2021. 1

(a) SAP (b) DPF (c) VoroMesh (d) PoNQ (e) Gr. Truth

Figure 5. Optimization-based results (top to bottom: 32^3, 64^3, 128^3) on Thingi30.

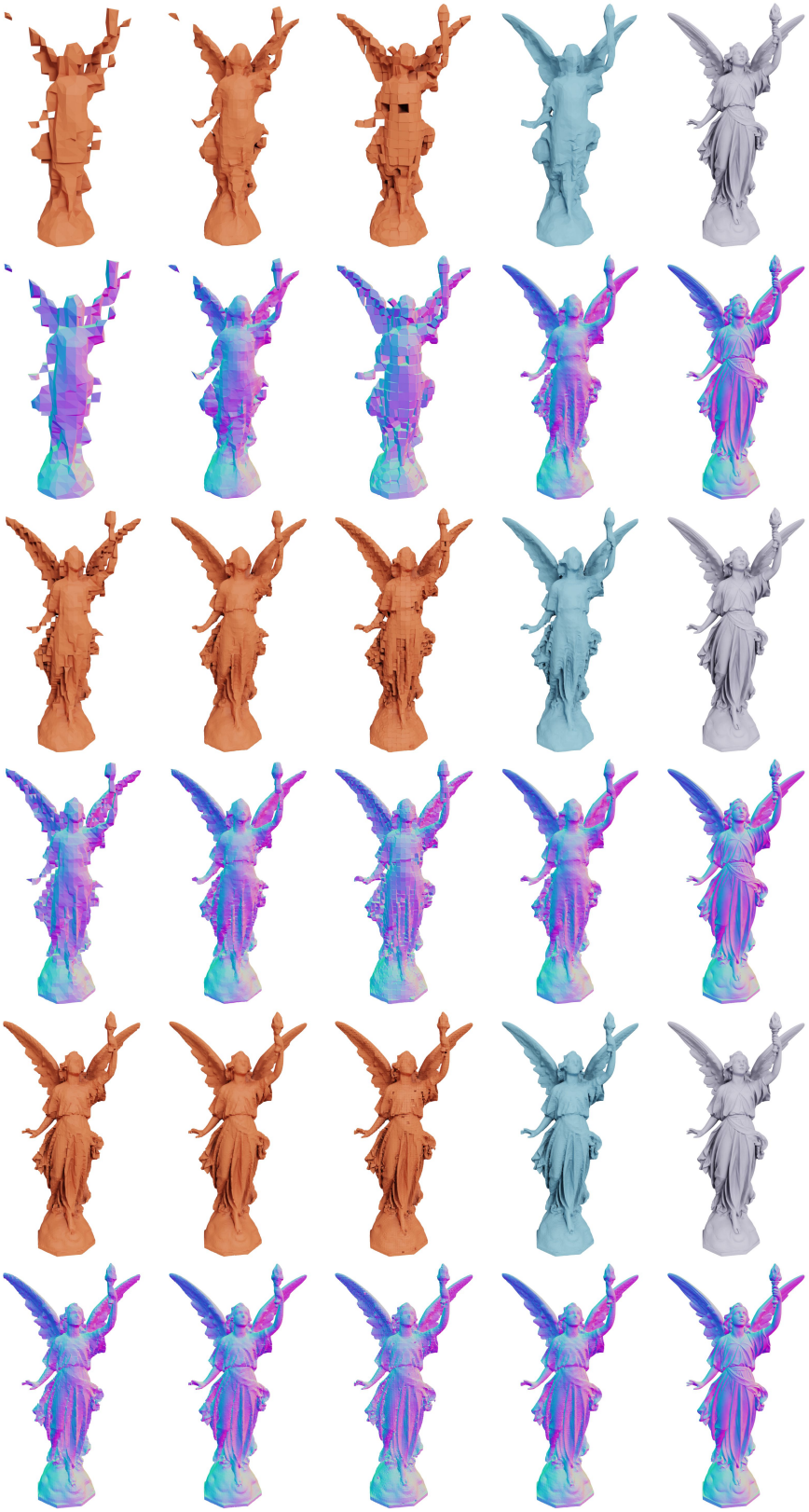


(a) NDC (b) NMC (c) VoroMesh (d) PoNQ (e) Gr. Truth

Figure 6. Learning results (top: 32^3; bottom: 64^3) on ABC.

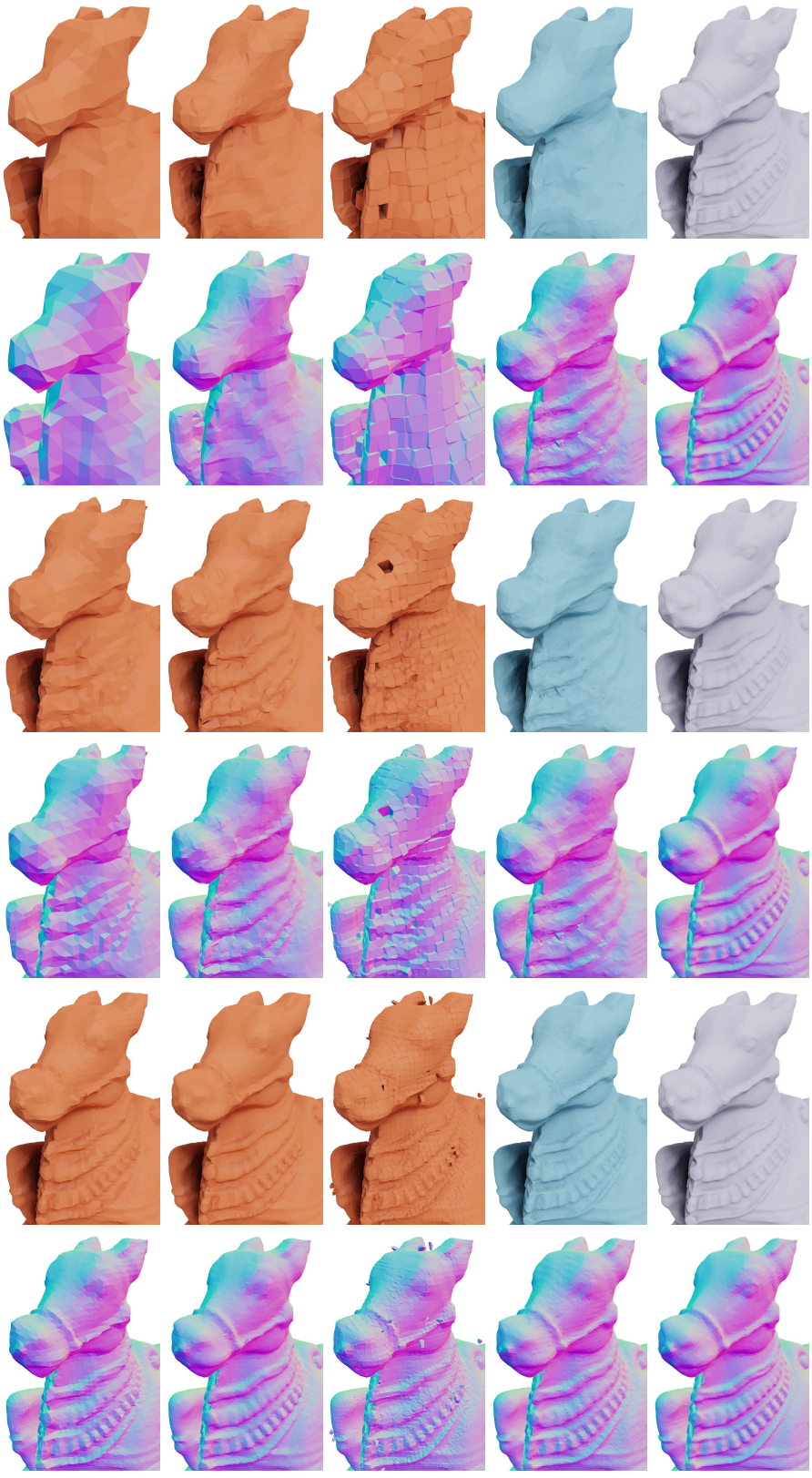
(a) NDC (b) NMC (c) VoroMesh (d) PoNQ (e) Gr. Truth

Figure 7. Learning results (top: 32^3; bottom: 64^3) on ABC.



(a) NDC (b) NMC (c) VoroMesh (d) PoNQ (e) Gr. Truth

Figure 8. Learning results (top to bottom: 32^3, 64^3, 128^3) on Thingi30. Networks trained on ABC.



(a) NDC (b) NMC (c) VoroMesh (d) PoNQ (e) Gr. Truth

Figure 9. Learning results (top to bottom: 32^3, 64^3, 128^3) on Thingi30. Networks trained on ABC.