ExMesh: EXplicit Mesh Reconstruction with Topology Adaptation

Supplementary Material

7. Methodology

7.1. Pseudocode for Topology Operations

Sections 3.2 and 3.3 introduce our core adaptive vertex splitting and merging strategies. This section provides detailed pseudocode for these operations and their corresponding UV maintenance routines.

Algorithm 1 Adaptive Vertex Splitting

Input: Mesh $\mathcal{M}(\mathcal{V}, \mathcal{F})$, UV coordinates $\mathcal{U}$, UV map $\mathcal{U}^{\text{map}}$, Vertex EMA Gradients $\mathcal{G}_{\mathcal{V}}$, Face Curvatures $\mathcal{K}$, Num splits N_{split}, Hyperparameters $\alpha, \beta, \tau_{\text{area\_hi}}, \tau_{\text{pos}}, \tau_{\text{uv}}$
Output: Updated Mesh $\mathcal{M}'(\mathcal{V}', \mathcal{F}')$, Updated UVs $\mathcal{U}'$

$\mathcal{F}_{\text{cand}} \leftarrow \emptyset$ *// Set of candidate faces*
$\mathcal{F}_{\text{mod}} \leftarrow \emptyset$ *// Set of modified faces*
for each face $\mathcal{F}_i \in \mathcal{F}$ **do**
  if $Area(\mathcal{F}_i) \geq \tau_{\text{area\_hi}}$ **then**
    $\mathcal{G}_i \leftarrow \frac{1}{3}\sum_{v \in \mathcal{F}_i} \mathcal{G}_{\mathcal{V}}(v)$
    $S_i \leftarrow \alpha \mathcal{G}_i + \beta \mathcal{K}_i$
    $\mathcal{F}_{\text{cand}} \leftarrow \mathcal{F}_{\text{cand}} \cup \{(\mathcal{F}_i, S_i)\}$
$\mathcal{F}_{\text{split}} \leftarrow \text{SampleTopN}(\mathcal{F}_{\text{cand}}, N_{\text{split}})$
for each face $\mathcal{F}(v_a, v_b, v_c) \in \mathcal{F}_{\text{split}}$ **do**
  if $\mathcal{F} \in \mathcal{F}_{\text{mod}}$ **then continue**
  $e(v_a, v_b) \leftarrow \text{LongestEdge}(\mathcal{F})$
  $\mathcal{F}'(v_a, v_b, v_d) \leftarrow \text{AdjacentFace}(\mathcal{M}, e)$
  if $\mathcal{F}' \neq$ NULL **then** *// Non-boundary edge*
    if $\mathcal{F}' \in \mathcal{F}_{\text{mod}}$ **then continue**
    $t_c \leftarrow \text{Project}(v_c, e)$; $t_d \leftarrow \text{Project}(v_d, e)$
    $v_s \leftarrow (t_c + t_d)/2$
    $\mu \leftarrow \frac{\|v_s - v_a\|}{\|v_b - v_a\|}$
    if $\mu < \tau_{\text{pos}}$ **or** $\mu > 1 - \tau_{\text{pos}}$ **then continue**
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_s\}$
    $\mathcal{F} \leftarrow \mathcal{F} \setminus \{\mathcal{F}, \mathcal{F}'\}$
    $\mathcal{F}_{\text{new}} \leftarrow \{\mathcal{F}(v_a, v_s, v_c), \mathcal{F}(v_s, v_b, v_c),$
      $\mathcal{F}(v_a, v_s, v_d), \mathcal{F}(v_s, v_b, v_d)\}$
    $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{F}_{\text{new}}$
    $\mathcal{F}_{\text{mod}} \leftarrow \mathcal{F}_{\text{mod}} \cup \{\mathcal{F}, \mathcal{F}'\} \cup \mathcal{F}_{\text{new}}$
  else *// Boundary edge*
    $v_s \leftarrow (v_a + v_b)/2$; $\mu \leftarrow 0.5$
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_s\}$
    $\mathcal{F} \leftarrow \mathcal{F} \setminus \{\mathcal{F}\}$
    $\mathcal{F}_{\text{new}} \leftarrow \{\mathcal{F}(v_a, v_s, v_c), \mathcal{F}(v_s, v_b, v_c)\}$
    $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{F}_{\text{new}}$
    $\mathcal{F}_{\text{mod}} \leftarrow \mathcal{F}_{\text{mod}} \cup \{\mathcal{F}\} \cup \mathcal{F}_{\text{new}}$
  $(\mathcal{U}, \mathcal{U}^{\text{map}}) \leftarrow$ **HandleUVsForVertexSplit**$(\cdot)$
$(\mathcal{U}, \mathcal{U}^{\text{map}}) \leftarrow$ **CompactUVs**$(\cdot)$
return $\mathcal{M}'(\mathcal{V}', \mathcal{F}'), \mathcal{U}', \mathcal{U}^{\text{map}\prime}$

Algorithm 2 Adaptive Vertex Merging

Input: Mesh $\mathcal{M}(\mathcal{V}, \mathcal{F})$, UV coordinates $\mathcal{U}$, UV map $\mathcal{U}^{\text{map}}$, Face Render Counts $\mathcal{C}_{\text{render}}$, Hyperparameters $\tau_{\text{degen}}, \tau_{\text{area\_low}}$
Output: Updated Mesh $\mathcal{M}'(\mathcal{V}', \mathcal{F}')$, Updated UVs $\mathcal{U}'$

$\mathcal{F}_{\text{merge}} \leftarrow \emptyset$ *// Set of candidate faces*
$\mathcal{F}_{\text{mod}} \leftarrow \emptyset$ *// Set of modified faces*
for each face $\mathcal{F}_i \in \mathcal{F}$ **do**
  if $Area(\mathcal{F}_i) \leq \tau_{\text{area\_low}}$ **then**
    $\mathcal{D}_i \leftarrow Area(\mathcal{F}_i)/\text{LongestEdge}(\mathcal{F}_i)^2$
    if $\mathcal{C}_{\text{render}}(\mathcal{F}_i) = 0$ **or** $\mathcal{D}_i < \tau_{\text{degen}}$ **then**
      $\mathcal{F}_{\text{merge}} \leftarrow \mathcal{F}_{\text{merge}} \cup \{\mathcal{F}_i\}$
for each face $\mathcal{F}(v_i, v_j, v_m) \in \mathcal{F}_{\text{merge}}$ **do**
  if $\mathcal{F} \in \mathcal{F}_{\text{mod}}$ **then continue**
  if IsBoundaryFace$(\mathcal{F})$ **then**
    $e(v_i, v_m) \leftarrow \text{GetBoundaryEdge}(\mathcal{F})$
  else
    $e(v_i, v_m) \leftarrow \text{ShortestEdge}(\mathcal{F})$
  if $Degree(v_m) > Degree(v_i)$ **then**
    swap(v_i, v_m)
  $AdjFaces \leftarrow \text{GetAdjacentFaces}(\mathcal{M}, v_m)$
  if CheckConflict$(AdjFaces, \mathcal{F}_{\text{mod}})$ **then continue**
  for each face $\mathcal{F}_k \in AdjFaces$ **do**
    if $v_i \in \mathcal{F}_k$ **then** *// $\mathcal{F}_k$ contains both v_i and v_m*
      $\mathcal{F} \leftarrow \mathcal{F} \setminus \{\mathcal{F}_k\}$
    else *// $\mathcal{F}_k$ only contains v_m*
      $\mathcal{F}_k \leftarrow \text{ReplaceVertex}(\mathcal{F}_k, v_m, v_i)$
    $\mathcal{F}_{\text{mod}} \leftarrow \mathcal{F}_{\text{mod}} \cup \{\mathcal{F}_k\}$
  $\mathcal{V} \leftarrow \mathcal{V} \setminus \{v_m\}$
if $|\mathcal{F}_{\text{merge}}| > 0$ **then**
  $(\mathcal{U}, \mathcal{U}^{\text{map}}) \leftarrow$ **CompactUVs**$(\cdot)$
return $\mathcal{M}'(\mathcal{V}', \mathcal{F}'), \mathcal{U}', \mathcal{U}^{\text{map}\prime}$

Algorithm 3 Compact UV

Input: Mesh $\mathcal{M}(\mathcal{V}, \mathcal{F})$, UVs $\mathcal{U}$, UV map $\mathcal{U}^{\text{map}}$
Output: Compacted UVs $\mathcal{U}'$, Compacted UV map $\mathcal{U}^{\text{map}\prime}$

$S \leftarrow$ set of all UV indices referenced in $\mathcal{U}^{\text{map}}$
$\mathcal{U}' \leftarrow$ list of $\mathcal{U}[i]$ for $i \in S$, ordered by new index
Build $OldToNew$ mapping from old indices in S to new indices in $\mathcal{U}'$
for each face $\mathcal{F}_i \in \mathcal{F}$ **do**
  Update $\mathcal{U}^{\text{map}\prime}[\mathcal{F}_i]$ by replacing each old index with $OldToNew[\cdot]$
return $\mathcal{U}', \mathcal{U}^{\text{map}\prime}$

Algorithm 4 UV Interpolation for New Vertex

Input: Vertices $\mathcal{V}$, Faces $\mathcal{F}$, UVs $\mathcal{U}$, UV map $\mathcal{U}^{\text{map}}$, New vertex v_s, Split edge $e(v_a, v_b)$, Old faces $\mathcal{F}_{\text{old}}, \mathcal{F}'_{\text{old}}$, Relative pos μ, Seam threshold τ_{uv}
Output: Updated UVs $\mathcal{U}'$

$(u_a, u_b) \leftarrow \text{GetUVsForFace}(\mathcal{U}^{\text{map}}, \mathcal{F}_{\text{old}}, e)$
$u_s^{(1)} \leftarrow (1-\mu)u_a + \mu u_b$
if $\mathcal{F}'_{\text{old}} = \text{NULL}$ **then** *// Boundary edge*
  $idx_s \leftarrow \text{AppendToList}(\mathcal{U}, u_s^{(1)})$
  for each new face $\mathcal{F}_{\text{new}}$ containing v_s, assign idx_s to v_s in $\mathcal{F}_{\text{new}}$ and update $\mathcal{U}^{\text{map}}$
else *// Non-boundary edge*
  $(u'_a, u'_b) \leftarrow \text{GetUVsForFace}(\mathcal{U}^{\text{map}}, \mathcal{F}'_{\text{old}}, e)$
  $u_s^{(2)} \leftarrow (1-\mu)u'_a + \mu u'_b$
  if $\|u_s^{(1)} - u_s^{(2)}\| < \tau_{\text{uv}}$ **then** *// Check for UV seam*
    $u_s \leftarrow (u_s^{(1)} + u_s^{(2)})/2$ *// no seam*
    $idx_s \leftarrow \text{AppendToList}(\mathcal{U}, u_s)$
    for each new face $\mathcal{F}_{\text{new}}$ containing v_s, assign idx_s to v_s in $\mathcal{F}_{\text{new}}$ and update $\mathcal{U}^{\text{map}}$
  else *// Create a new UV seam*
    $idx_s^{(1)} \leftarrow \text{AppendToList}(\mathcal{U}, u_s^{(1)})$
    $idx_s^{(2)} \leftarrow \text{AppendToList}(\mathcal{U}, u_s^{(2)})$
    for each new face $\mathcal{F}_{\text{new}}$ containing v_s, assign $idx_s^{(1)}$ or $idx_s^{(2)}$ to v_s, and update $\mathcal{U}^{\text{map}}$
return $\mathcal{U}', \mathcal{U}^{\text{map}\prime}$

7.2. Loss Function

As described in Section 3.5, the overall optimization objective is formulated as a weighted sum of several loss terms. Below, we detail the main components of the loss function used in our framework.

Photometric Loss Following the formulation in 3D Gaussian Splatting [23], our primary photometric loss is a combination of an $\mathcal{L}_1$ loss and a D-SSIM (Structural Dissimilarity) term, which balances absolute color accuracy with perceptual similarity:

$$\mathcal{L}_{\text{rgb}} = (1-\lambda_{\text{dssim}})\mathcal{L}_1(\hat{I}, I_{\text{gt}}) + \lambda_{\text{dssim}}\mathcal{L}_{\text{D-SSIM}}(\hat{I}, I_{\text{gt}}) \quad (6)$$

Here, $\hat{I}$ is the rendered RGB image and I_{gt} is the GT image. The $\mathcal{L}_1$ term provides a robust pixel-wise difference, while the D-SSIM term [52] adds a perceptually-driven gradient.

Depth Consistency Loss To ensure that the rendered surface geometry is consistent with the reference depth, we adopt a Pearson depth loss. This loss penalizes the discrepancy between the rendered depth map $\hat{D}$ and the reference depth D_{ref} obtained from Depth Anything 3 [27]:

$$\mathcal{L}_d = 1 - \frac{\text{Cov}(\hat{D}, D_{\text{ref}})}{\sigma_{\hat{D}}\sigma_{D_{\text{ref}}}} \quad (7)$$

where Cov denotes the covariance and σ denotes the standard deviation, both computed within the silhouette mask.

Table 6. Key hyperparameters used in experiments.

Parameter	Symbol	Value
Vertex learning rate	lr_{vertices}	0.0005
Feature learning rate	lr_{feature}	0.0025
Photometric loss weight	λ_{rgb}	1.0
D-SSIM loss weight	λ_{dssim}	0.2
Depth loss weight	λ_d	0.01
Silhouette loss weight	λ_m	0.005
Laplacian loss weight	λ_s	1000
Offset loss weight	λ_b	1000
Area threshold (split)	$\tau_{\text{area\_hi}}$	1.5× median
Area threshold (merge)	$\tau_{\text{area\_low}}$	0.5× median
Degeneracy threshold	τ_{degen}	0.05
Texture resolution	–	2560×2560

Silhouette Loss Additionally, a standard silhouette loss is employed to constrain the mesh geometry to the object's boundaries. This loss is formulated as a Binary Cross-Entropy (BCE) between the rendered alpha mask $\hat{M}$ and the ground-truth 2D mask M_{gt}:

$$\mathcal{L}_m = \text{BCE}(\hat{M}, M_{\text{gt}}) \quad (8)$$

Laplacian Smoothing Loss A Laplacian smoothing loss [10, 48] is further incorporated to promote a smooth and well-formed mesh topology. By minimizing the discrete Laplacian vector at each vertex v_i relative to its 1-ring neighbors $\mathcal{N}(i)$, this regularizer penalizes high-frequency noise and encourages uniform vertex distribution:

$$\mathcal{L}_s = \frac{1}{|\mathcal{V}|}\sum_{v_i \in \mathcal{V}} \left\| \left(v_i - \frac{1}{|\mathcal{N}(i)|}\sum_{v_j \in \mathcal{N}(i)} v_j \right) \right\|^2 \quad (9)$$

Bi-vertex Offset Loss Similar to FlexiCubes [41], ExMesh operates on a deformable grid where each vertex v_i has a deformation offset Δv_i. To prevent overly sharp or disjointed deformations between adjacent grid vertices, a bi-vertex offset regularization loss is introduced, which encourages the deformation field to remain locally smooth.

$$\mathcal{L}_b = \frac{1}{|\mathcal{V}|}\sum_{v_i \in \mathcal{V}} \left\| \left(\Delta v_i - \frac{1}{|\mathcal{N}(i)|}\sum_{v_j \in \mathcal{N}(i)} \Delta v_j \right) \right\|^2 \quad (10)$$

8. Implementation Details

8.1. Hyperparameters

Table 6 summarizes the main hyperparameters used in our framework. All values are selected based on validation performance and follow common practice in related works.

Figure 12. Visualization of Reconstructed Meshes with Vertex Color on DTU Dataset.

8.2. Baselines

All baselines are trained using their official implementations and recommended settings. For methods that do not produce an explicit mesh, surfaces are extracted using the authors' prescribed procedures (e.g., TSDF fusion or Marching Cubes) to ensure a fair comparison. Note that consistent with our method, we apply object masks during the evaluation of all baselines to filter out background noise and focus on the reconstruction quality of the target object.

NeRF-based (Implicit) Methods. **VolSDF** [57] represents the surface as a neural SDF and defines the volume density using Laplace's cumulative distribution function. **NeuS** [51] also represents the surface as a neural SDF, but introduces an unbiased S-density function for volume rendering. **Neuralangelo** [47] scales implicit surface reconstruction to large scenes by encoding the SDF in multi-resolution hash grids and employs a coarse-to-fine optimization schedule with geometric regularization.

GS-based (Explicit) Methods. **SuGaR** [46] introduces a regularization term to encourage 3D Gaussians to align with surfaces and extracts a coarse mesh via Poisson reconstruction. **2DGS** [18] represents the scene using explicit 2D Gaussian disks, providing stronger geometric constraints. The final mesh is extracted using TSDF fusion. **PGSR** [6] uses planar primitives derived from 3D Gaussians, introduces an unbiased depth rendering method, and enforces geometric consistency through single-view and multi-view regularization. **Triangle Splatting** [16] proposes a differentiable renderer that directly optimizes an unstructured triangle soup, enabling end-to-end optimization of vertex positions and colors. **GOF** [58] derives a continuous opacity field from optimized Gaussians, enabling surface extraction by identifying a level-set and using Marching Tetrahedra. **QGS** [61] extends 3DGS by introducing second-order quadric primitives, allowing each primitive to represent more complex local geometry.

Mesh-driven (Hybrid) Methods. **Nvdiffrec** [33] jointly optimizes topology, materials, and lighting from images. It uses a differentiable marching tetrahedra (DMTet) layer to extract a mesh from an intermediate volumetric SDF grid inside the training loop. **FlexiCubes** [41] also optimizes an SDF grid, and introduces a differentiable isosurface extraction layer based on Dual Marching Cubes, with learnable parameters for vertex positioning and quad splitting. **IMLS-Splatting** [55] uses a point cloud as the core representation and introduces a differentiable Implicit Moving-Least Squares (IMLS) algorithm to generate a sparse SDF grid for meshing. **GeoSVR** [26] is based on a sparse voxel representation, which is optimized directly with monocular depth constraints and surface regularization.

Table 7. Efficiency comparison on DTU dataset. We report total reconstruction time and peak GPU memory usage.

Method	Time ↓	Memory ↓
VolSDF [57]	>12h	14GB
NeuS [51]	>12h	7GB
Neuralangelo [47]	>12h	16GB
SuGaR [46]	1h	23GB
2DGS [18]	11m	5GB
TriSplat [16]	15m	23GB
GOF [58]	1h	9GB
PGSR [6]	30m	3.5GB
QGS [61]	48m	3.5GB
Nvdiffrec [33]	>1h	20GB
IMLS-Splatting [55]	15m	20GB
GeoSVR [26]	49m	14GB
Ours	15m	4.5GB

9. Additional Results

9.1. Additional Results on DTU Dataset

Figure 12 shows meshes with vertex color reconstructed by our method on the DTU [20] dataset, demonstrating high-fidelity geometry and realistic surface appearance. In addition, Figures 15–18 present a comprehensive visual comparison between our approach and several recent baselines, including NeuS [51], Neuralangelo [47], PGSR [6], IMLS-Splatting [55], and GeoSVR [26]. ExMesh achieves more accurate and complete surface reconstruction, with fewer artifacts and better preservation of fine details, highlighting its robustness and effectiveness on real-world objects.

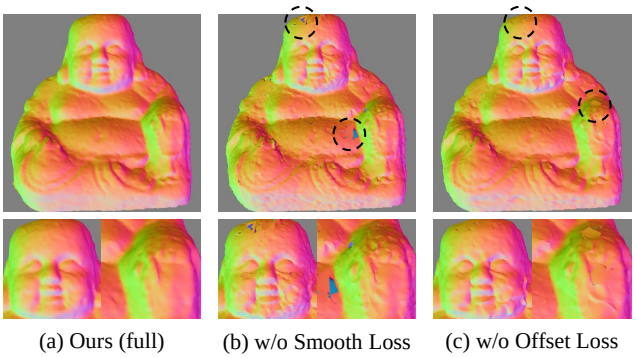
9.2. Efficiency Analysis

We compare the efficiency of our method with recent baselines in terms of total reconstruction time and peak GPU memory usage on the DTU dataset [20]. As summarized in Table 7, our method achieves competitive or superior efficiency, requiring significantly less training time and memory compared to most approaches. These results demonstrate the practical advantages of our framework for scalable and resource-efficient surface reconstruction.

10. Additional Ablations

10.1. Loss Function

To further investigate the impact of regularization terms in our framework, we conduct ablation studies on the Laplacian smoothing loss ($\mathcal{L}_s$) and the bi-vertex offset regularization loss ($\mathcal{L}_b$) on the DTU dataset. As illustrated in Figure 13, removing $\mathcal{L}_s$ leads to increased high-frequency noise and jagged artifacts, especially along thin structures and boundaries, indicating its importance for suppressing local fluctuations and maintaining mesh uniformity. In contrast, omitting $\mathcal{L}_b$ results in discontinuities and abrupt changes in the deformation field, causing locally stretched or folded regions and compromising surface regularity.



(a) Ours (full) (b) w/o Smooth Loss (c) w/o Offset Loss

Figure 13. Ablation study on the Laplacian smoothing loss ($\mathcal{L}_s$) and bi-vertex offset loss ($\mathcal{L}_b$) on DTU dataset.

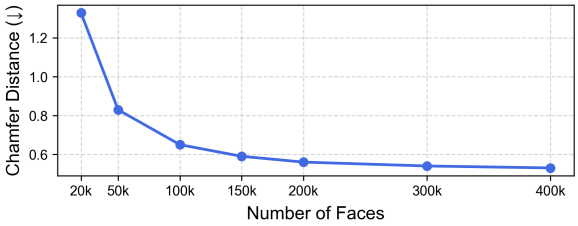


Figure 14. Analysis of the relationship between geometric accuracy and the final mesh face count on the DTU dataset.

10.2. Effect of Face Count on Performance

Finally, we analyze the impact of face count on geometric accuracy. As shown in Figure 14, geometric accuracy improves significantly as face count increases to 100k, but saturates beyond 200k. This indicates that further increasing face count yields little benefit once the mesh is sufficiently complex. Moreover, excessive faces significantly rendering time, making 150k-250k an optimal balance between quality and efficiency.

11. Discussion

While our method achieves strong reconstruction quality and efficiency, there remain practical limitations. First, the framework depends on careful tuning of multiple hyperparameters, such as loss weights and topology update thresholds, which could be time-consuming and may require expert knowledge. In addition, the efficiency of UV unwrapping poses a bottleneck for large meshes, as current tools like xatlas are CPU-based and lack CUDA acceleration. When the face count exceeds 200,000, UV unwrapping becomes significantly slower, limiting the scalability of our approach for scene-level reconstruction. Addressing these challenges, including developing automated parameter selection strategies and exploring GPU-accelerated UV unwrapping, will be an important direction for future work.

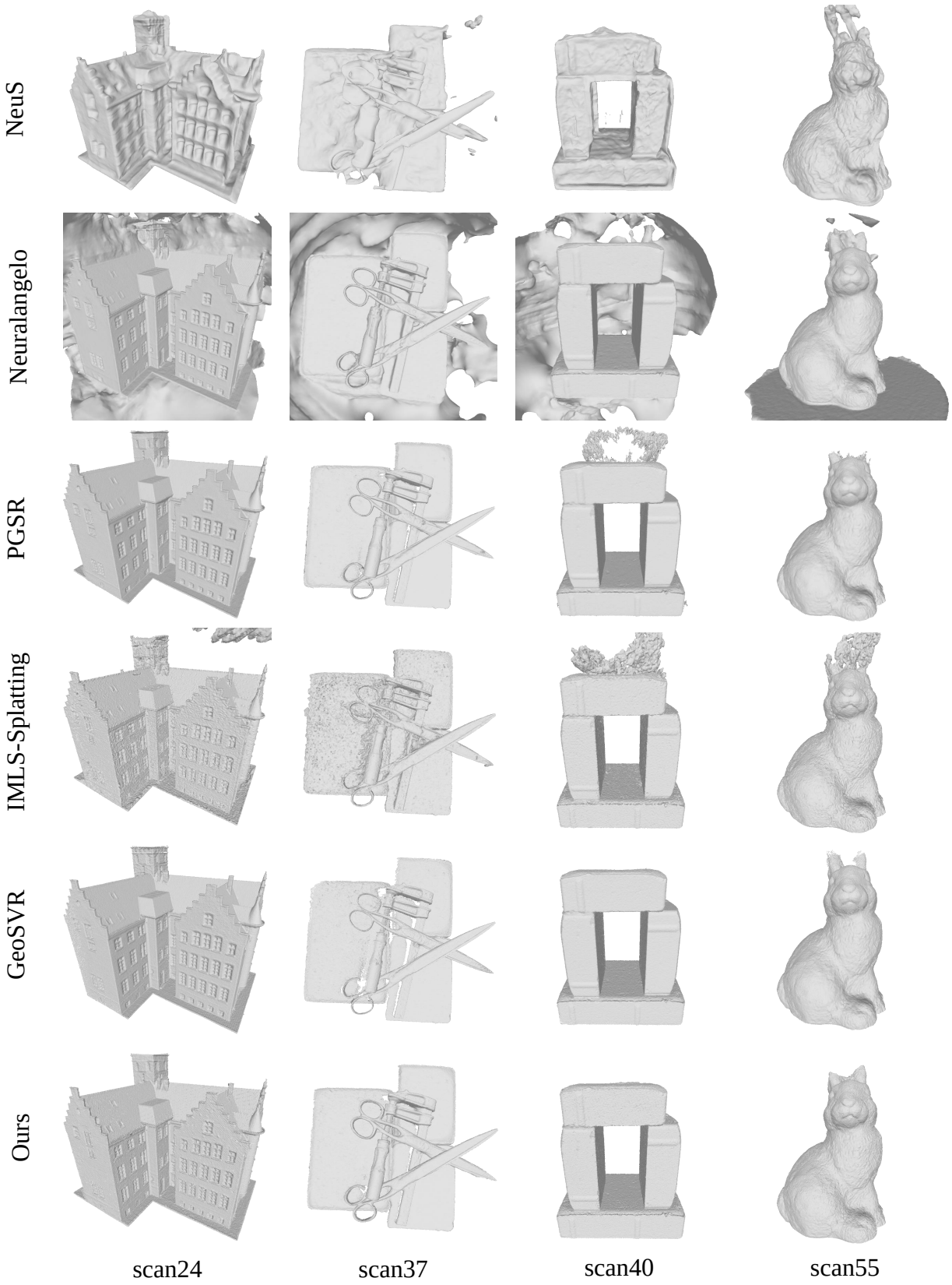


Figure 15. Qualitative geometric comparison on DTU dataset (part 1).

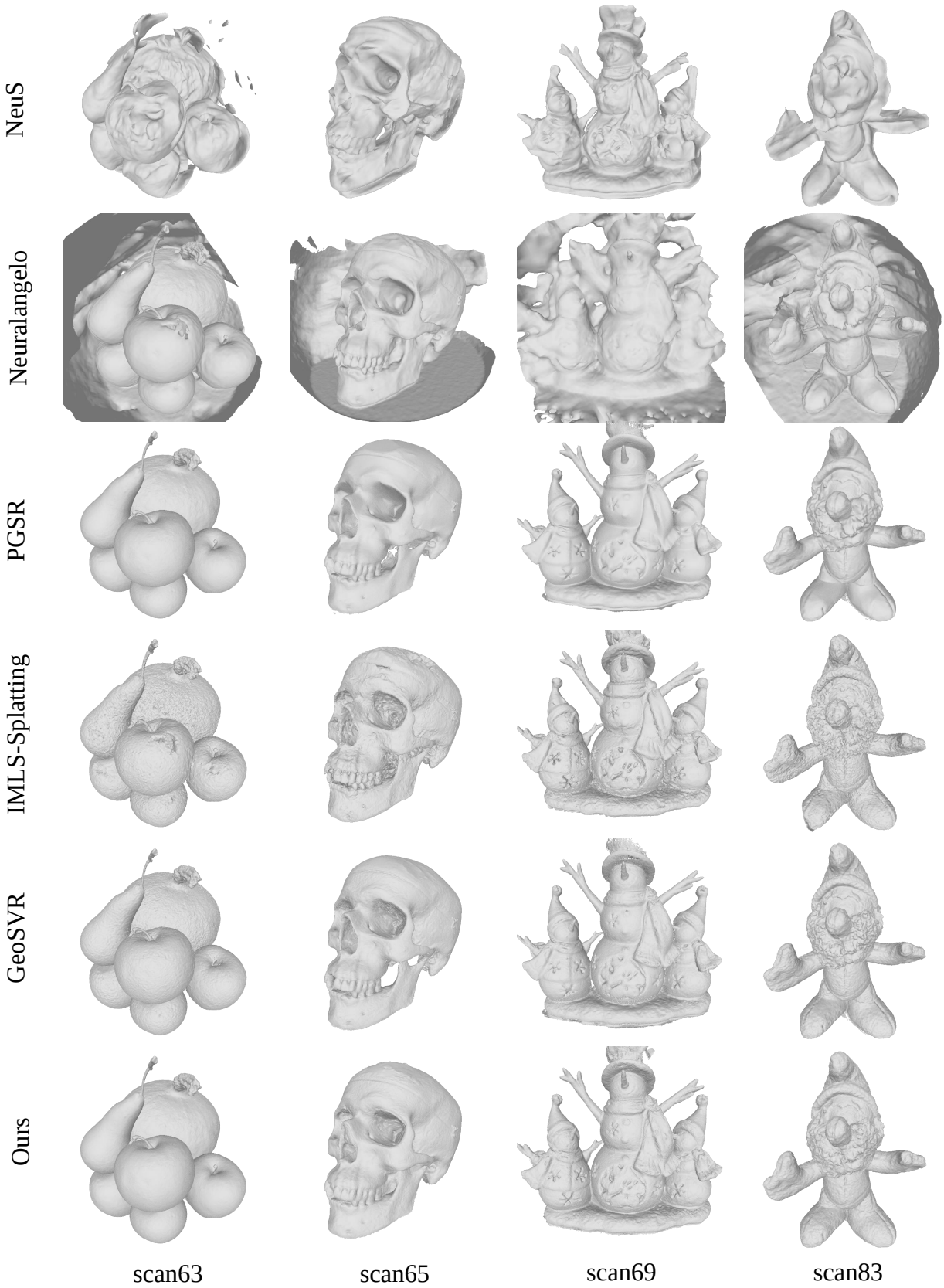


Figure 16. Qualitative geometric comparison on DTU dataset (part 2).

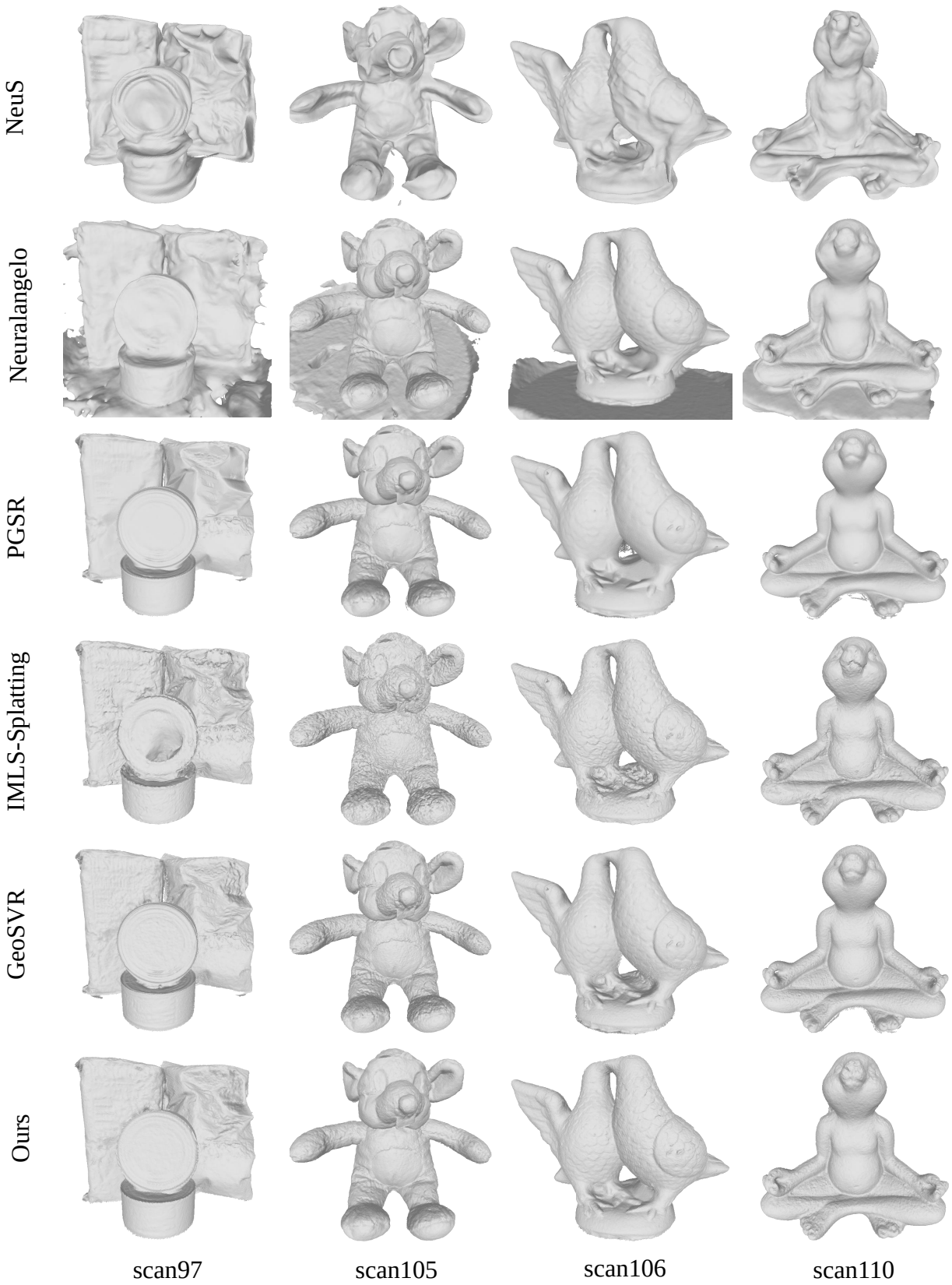


Figure 17. Qualitative geometric comparison on DTU dataset (part 3).

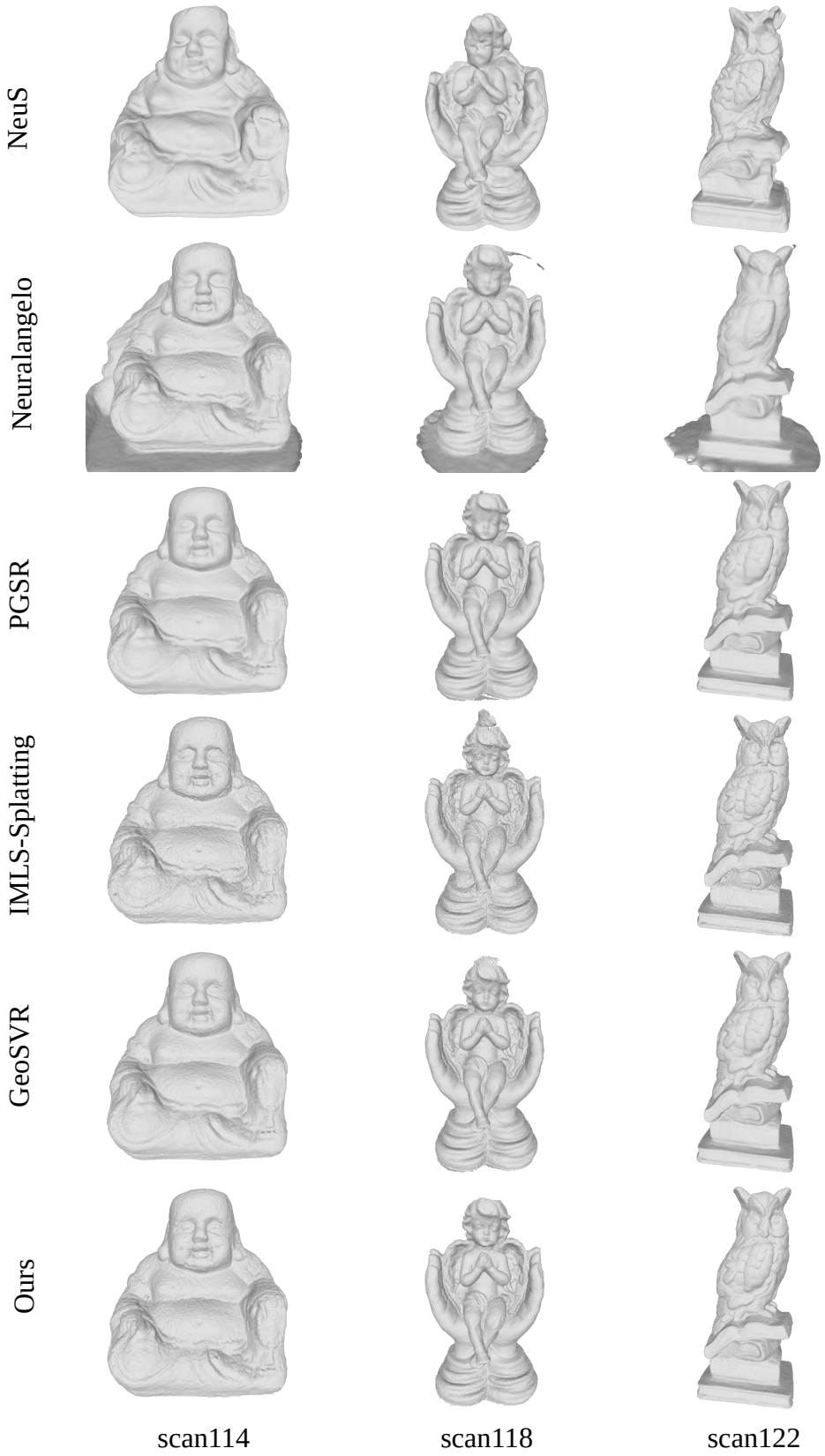


Figure 18. Qualitative geometric comparison on DTU dataset (part 4).