QUADify: Extracting Meshes with Pixel-level Details and Materials from Images

Supplementary Material

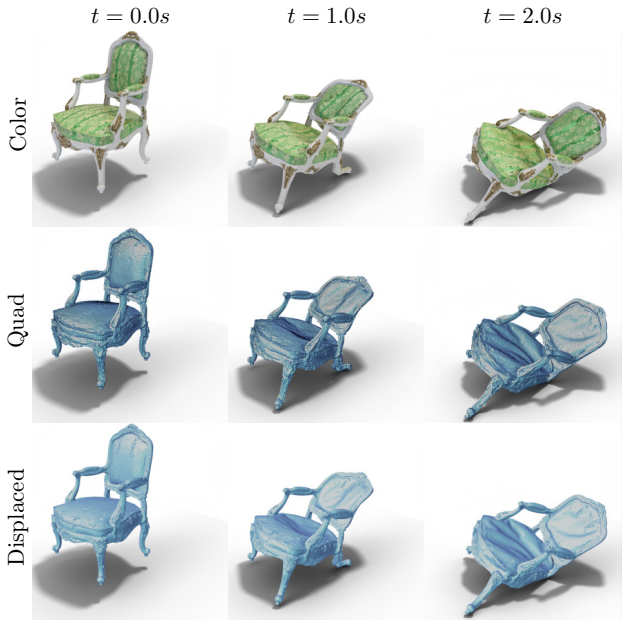


Figure 12. Soft-body physics simulation. We use the quad-dominant mesh of the CHAIR scene extracted from 100 images in the NeRF Synthetic [33] dataset in a soft-body physics simulation (top), simulated and rendered using Blender [13]. With minimal pre-processing, just removing disconnected internal geometry before running the simulation, we compute deformations on our extracted quad mesh (Quad). Subdivision and displacement can be applied after the simulation to recover pixel-level details, which would be infeasible to directly include in the physics simulation due to the increased geometric complexity (Displaced).

Acknowledgements We thank Seyedmorteza Sadat, Guilherme Haetinger, and Rajesh Sharma for many insightful discussions during the creation of this work.

7. Further Applications

7.1. Soft-Body Physics Simulation

A direct application of quad-dominant meshes is using them for simulation in a physics engine since their regular topology lends itself well to force propagation on the mesh. Particularly for soft-body physics, a regular arrangement of faces is crucial to achieve consistent deformations [43, 61]. Since our meshes consist of such regularly arranged isometric faces, they are particularly well-suited to soft-body physics simulation. In Fig. 12, we show the results of using one of our quad-dominant meshes in such a physics simulation performed in Blender [13]. Since the extracted mesh is lightweight, it can be efficiently simulated and deforms consistently. Furthermore, since we extract small-scale details using subdivision and displacement maps constrained to the surface with texture coordinates, these small-scale details also get consistently deformed. Crucially, the complex geometry created by subdivision and displacement does not need to be included in the simulation since it can be applied in a post-processing step after solving for the soft-body deformations. This enables efficient simulations while retaining the pixel-level details extracted through displacement directly on our quad-dominant meshes without requiring extensive mesh cleanup.

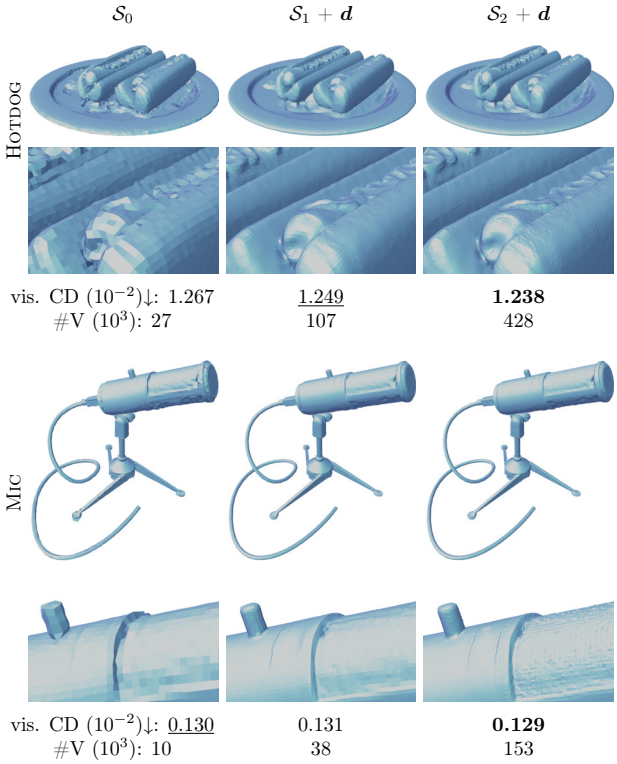


Figure 13. Offline Level-of-detail. We show the same extracted mesh of the HOTDOG and MIC scenes from the NeRF Synthetic dataset at three levels of details by varying the number of subdivision steps from none $\mathcal{S}_0$ to two $\mathcal{S}_2$. For each additional subdivision level, the number of vertices roughly quadruples while finer can be represented.

7.2. Offline Level-of-Detail Adjustment

Inspired by production workflows [7, 8], our method directly enables changing the level of detail of the extracted meshes offline after the reconstruction. Since the geometric complexity of a scene can considerably slow down production renderers and game engines, background objects in real-world scenes are frequently represented using fewer

vertices. Our method natively supports this workflow by varying the subdivision levels used for displacement.

We reconstruct the HOTDOG and MIC objects from the NeRF Synthetic dataset [33] as described in Sec. 5.1 using two subdivision levels during training. After the extraction, the same Catmull-Clark subdivision and displacement approach can also be applied in a 3D modeling package, such as Blender [13], recreating the displaced mesh. In Fig. 13, we show the same quad-dominant mesh at various subdivision levels with displacement. The number of vertices exponentially increases with the number of subdivision levels but simultaneously allows for representing more details using displacement. Since displacement is stored as a high-resolution 2D texture, which can be sampled at arbitrary mesh resolutions, our method enables adaptively changing the level of detail of the extracted mesh as a post-processing step after reconstruction.

8. Experimental Details

8.1. Baselines

Inverse Rendering for Meshes. In the experiments in Sec. 5.1, we compare our method with Nvdiffrec, Nvdiffrecmc, and FlexiCubes, which were all re-trained from their respective available public source code. However, we do not compare to NeRF-based techniques [1, 33], since these reconstruct density fields, not meshes. Similarly, Neural Surface techniques, such as NeuS and Neuralangelo, can only extract a mesh in a post-processing step, usually with excessive geometry resolution [26, 39, 55].

Mesh reconstruction. For the experiments in Sec. 5.2, we compare reconstructions from Marching Cubes (MC), DMTet [47], the concurrent work FlexiCubes [48] and our method. Since there did not exist a *differentiable* implementation of MC during this work to the best of our knowledge, we use the forward implementation of kaolin[1] implementing the method described by Lorensen et al. [29] and add a custom backward pass of Equ. 1 in PyTorch. For DMTet and FlexiCubes, we use their respective available source code.

Since the vertex count of the extracted meshes bounds the amount of detail that can be extracted for the reference objects, we match the mesh resolution between all methods and report the average number of vertices over all objects in Tab. 3 and Tab. 6. However, since each iso-surfacing method extracts a different number of vertices at a fixed voxel grid resolution, we vary the grid resolutions for each method to match the vertex counts.

The dataset collected by Myles et al. [36] contains 116 objects ranging from CAD models to photogrammetry scans. For evaluation, we use the same subset of 79 watertight and non-skinny objects[2] as FlexiCubes [48] provided by the original authors. We center each mesh such that its centroid is at the coordinate origin and its bounding box is in $[-0.9, 0.9]^3$ following NDC [12] and scale the voxel grid to $[-1, 1]^3$.

For supervision, we again follow FlexiCubes and render depth and mask images from randomly sampled poses around each object. Since some reference objects contain internal geometry, we use the SDF loss proposed by FlexiCubes, sampling 1000 random points and supervising the predicted SDF with the reference.

8.2. Evaluation Metrics

Inverse Rendering for Meshes. We evaluate the visual quality of the surface reconstructions by comparing the Peak Signal to Noise Ratio (PSNR), Structural Similarity Index Measure (SSIM), and Learned Perceptual Image Patch Similarity (LPIPS) as implemented by TorchMetrics [37] on the 200 test images for each scene in the NeRF Synthetic dataset [33]. We report per-scene view interpolation metrics in Tab. 4 and show examples in Fig. 18.

Mesh reconstruction. As the key focus of our method is extracting high-quality meshes, we compare the extracted meshes in terms of extrinsic (accuracy of the reconstruction) and intrinsic (mesh quality) metrics: We report the following commonly used extrinsic metrics:

- visible Chamfer Distance (vis. CD): This metric measures the distance between two point clouds by searching for nearest neighbors. We measure the visible CD shown in Tab. 2 and Tab. 5 between the predicted and ground meshes by scaling each mesh to fit the $[-10, 10]^3$ bounding box [18] and sampling $2.5 \cdot 10^6$ points on the triangles visible in at least one test view following Nvdiffrec [35] and FlexiCubes [48]. The visible CD is computed as the mean symmetric squared L_2 distance between the two point clouds.
- Chamfer distance (CD): In Tab. 3 and Tab. 6, we report a second Chamfer Distance, distinct from the previously described visible CD to report consistent results with related work [12, 48]. Therefore, the numbers between both Chamfer Distances are not directly comparable due to different scaling of the objects. For CD, sample 10^5 points on all mesh faces and compute the mean symmetric squared L_2 distance between the point clouds.
- F-score (F1): Following NDC [12], we compute precision and recall on the same point clouds as in the CD computation above. We define a point as a *true positive* if the distance of a point on the predicted mesh is close enough to one on the ground truth mesh (threshold = 0.005), else

[1] https://github.com/NVIDIAGameWorks/kaolin/blob/master/kaolin/ops/conversions/voxelgrid.py

[2] https://github.com/nv-tlabs/FlexiCubes/issues/4

it is a *false positive*. If a ground truth point is close enough to a predicted one (threshold = 0.005), it is defined as a *true positive*, otherwise a *false negative*. The F-Score is then the harmonic mean of precision and recall. We use the publicly available implementation of NDC to compute these metrics[3].

Since our method focuses on extracting meshes with high topological quality, we also evaluate the intrinsic quality of our meshes using the following metrics:

- Minimal and maximal face angle: We compute the angle distributions in Fig. 9 by computing the respective angles per face for each method.
- Radius Ratio (RR) and Aspect Ratio (AR): Both metrics measure the distortion of faces from an ideal isometric quad or triangle face, where lower is better. We follow the definition of PyVista [51] to compute both metrics for all faces. In Tab. 3, Tab. 6, and Fig. 9, we report the outlier percentage of each metric, where all faces with values greater than 4 are commonly considered outliers [48].

9. Additional Results

9.1. Per-scene Results

Inverse Rendering for Meshes. In Tab. 4 we provide per-scene breakdowns of the view interpolation results we report in Sec. 5.1 and visually show the reconstructions in Fig. 18. We consistently observe on-par results with Nvdiffrec and FlexiCubes for all scenes except FICUS, where our method struggles to reconstruct the individual leaves in the training images and instead predicts a single larger blob. We attribute this failure to recover high-frequency geometry due to our method using an MLP to represent the SDF instead of a randomly initialized per-vertex SDF as used by all related works. We initialize the SDF with a sphere and learn the object's shape by continuously deforming this sphere. This provides a continuous surface throughout the optimization, which is required for our re-meshing formulation, as it rearranges the edges of the Marching Cubes (MC) mesh to form quad faces. However, this comes at the cost of the MLP SDF getting stuck in local minima, especially in concave regions, and having difficulty in reconstructing disconnected geometry.

In Tab. 5, we provide a per-scene breakdown of the visible Chamfer distances shown in Sec. 5.1. Here, we see the advantage of using subdivision and displacement to recover small details. Our method extracts more accurate meshes than all baselines across scenes except on FICUS and SHIP, which are challenging due to the mentioned limitations of the MLP SDF and limited shading model [35] respectively. Still, on average, our method offers state-of-the-art geometric accuracy on the Quad mesh with an even more significant improvement when using displacement.

[3] https://github.com/czq142857/NDC

PSNR (dB) ↑

	Chair	Drums	Ficus	Hotdog	Lego	Mats.	Mic	Ship	Avg.
Nvdiffrec	31.98	24.56	30.66	33.06	29.03	26.56	**30.71**	25.95	**29.06**
Nvdiffrecmc	28.93	23.02	25.54	30.64	26.41	25.58	28.93	23.42	26.56
FlexiCubes	**32.02**	**24.67**	**31.16**	**33.24**	**29.99**	**27.28**	27.26	**26.02**	28.95
Our	30.95	24.40	25.56	32.63	28.30	26.62	30.07	25.15	27.96

SSIM ↑

	Chair	Drums	Ficus	Hotdog	Lego	Mats.	Mic	Ship	Avg.
Nvdiffrec	0.969	0.921	0.969	0.972	0.946	0.931	**0.976**	**0.827**	**0.939**
Nvdiffrecmc	0.946	0.905	0.938	0.944	0.912	0.910	0.962	0.786	0.913
FlexiCubes	**0.970**	**0.923**	**0.971**	**0.973**	**0.953**	**0.940**	0.936	0.822	0.936
Our	0.963	0.922	0.947	0.969	0.941	0.938	0.974	0.812	0.933

LPIPS ↓

	Chair	Drums	Ficus	Hotdog	Lego	Mats.	Mic	Ship	Avg.
Nvdiffrec	0.043	0.090	0.047	0.062	0.062	0.088	**0.040**	**0.192**	**0.078**
Nvdiffrecmc	0.075	0.111	0.077	0.111	0.108	0.110	0.062	0.263	0.115
FlexiCubes	**0.041**	0.089	**0.046**	**0.059**	**0.056**	**0.079**	0.087	0.202	0.082
Our	0.047	**0.088**	0.070	0.061	0.067	0.083	0.040	0.209	0.083

Table 4. View interpolation. Image quality metrics on all 200 test images in the NeRF Synthetic dataset. Results for Nvdiffrec [35], Nvdiffrecmc [20] and FlexiCubes [47] are computed by running the respective public source code.

vis. CD↓(10^{-2})	Chair	Drums	Ficus	Hotdog	Lego	Mats.	Mic	Ship	Avg.
Nvdiffrec	2.95	1.95	0.64	2.36	2.60	0.59	0.12	50.88	7.76
Nvdiffrecmc	2.75	2.05	1.80	2.09	4.54	**0.33**	0.14	30.04	5.47
FlexiCubes	7.79	2.04	**0.11**	2.31	1.60	0.40	**0.11**	**11.76**	3.26
Our Quad	1.97	0.70	4.64	1.27	**1.25**	0.37	0.13	15.42	3.22
Our Disp.	**1.86**	**0.64**	2.68	**1.24**	1.32	0.38	0.13	15.47	**2.96**

#V ↓(10^3)	Chair	Drums	Ficus	Hotdog	Lego	Mats.	Mic	Ship	Avg.
Nvdiffrec	50	33	**19**	28	57	32	11	92	40
Nvdiffrecmc	50	**30**	20	29	**53**	27	11	113	42
FlexiCubes	49	52	28	34	82	53	17	101	52
Our Quad	**33**	30	26	**27**	55	**23**	**10**	**56**	**32**
Our Disp.	530	484	398	428	886	361	153	888	516

Table 5. Geometric quality by Chamfer scores and vertex counts on the NeRF synthetic dataset. Results for Nvdiffrec are re-run based on publicly available code. Note that our method can primarily reconstruct opaque geometry larger than the re-meshing edge length s, causing our method to under-perform on the FICUS scene due to the small-scale leaf geometry present.

Mesh reconstruction. In Tab. 6, we supplement the evaluation in Sec. 5.2 by additionally comparing across mesh resolutions. In Fig. 19, we show additional qualitative comparisons. We see a similar pattern across scales where our quad mesh consistently improves on MC and DMTet regarding reconstruction accuracy but is slightly behind FlexiCubes. However, when adding a single level of subdivision with displacement, our method reconstructs the objects more accurately than all baselines.

The representational power of displacement is shown by our results at the lowest resolution (Our Disp.), equivalent to 32^3 in terms of number of vertices, which gives lower CD than either MC or DMTet at 64^3 and 80^3 voxel resolution respectively and is only slightly less accurate than FlexiCubes. This shows that our quad-dominant geometry with displacement extracts significantly more accurate geometry than either related work.

Regarding intrinsic metrics, our quad-dominant meshes

32^3	G	CD(10^{-5})↓	F1↑	AR>4(%)↓	RR>4(%)↓	#V(10^3)
MC	32	10.49	0.52	12.15	12.08	2.62
DMTet	40	9.08	0.58	14.37	14.04	2.76
FlexiCubes	32	6.52	0.65	8.16	9.71	3.08
Our Quad	55	8.02	0.56	**0.00**	**0.38**	2.62
Our Disp.	55	**5.00**	**0.69**	0.55	2.84	10.42
64^3	G	CD(10^{-5})↓	F1↑	AR>4(%)↓	RR>4(%)↓	#V(10^3)
MC	64	5.22	0.66	12.02	12.01	10.34
DMTet	80	5.23	0.67	16.89	16.32	10.92
FlexiCubes	64	4.87	0.69	6.69	8.26	11.87
Our Quad	110	4.96	0.67	**0.00**	**0.24**	10.66
Our Disp.	110	**4.63**	**0.70**	0.22	0.88	42.75

Table 6. Mesh reconstruction. Quantitative results on the selected four objects from the Myles [36] dataset. We report the following metrics: G: Voxel grid resolution, CD: Chamfer Distance, F1: F1 Score, AR: face Aspect Ratio, RR: face Radius Ratio, #V: Number of vertices

use more regular faces and almost eliminate outliers in Aspect Ratio (AR) and Radius Ratio (RR) across scales. Similar to the results shown in Tab. 6, the displaced mesh (Our Disp.) introduces slightly more irregular faces due to displacing the subdivided geometry.

9.2. Real-world Results

Due to space constraints, we could not show further results on real-world scenes from the Tanks and Temples dataset [24]. We qualitative results in Fig. 14, showing that our method can also extract high-quality meshes from large-scale real-world objects.

10. Implementation

We trained our method on either a single NVIDIA A6000 48GB in Sec. 5.1 or a single NVIDIA RTX 4090 GPU 24GB for Sec 5.2. Each scene in Sec. 5.1 can be reconstructed in a few hours with meaningful results in the first few steps, as shown in the supplemental video. Our method gives meaningful results within the first minutes (≈300 steps, see suppl. video) In Sec. 5.1 a run to convergence (6k steps) for a single object takes 14.5h. In Sec. 5.2 each object takes 19 min in total (1k steps), with the difference mainly due to different voxel resolutions (110^3 vs. 181^3) and number of steps detailed in Sec. 11.1.

Re-meshing size. For quad-dominant re-meshing, we choose the minimal quad size s possible at $s = \sqrt{2}\frac{D}{G-1}$, where D is the size of the bounding box and G is the voxel grid resolution used by Marching Cubes (MC). This choice comes at the cost of slightly worse re-meshing quality than InstantMeshes [22], who choose a larger re-meshing size but improves coupling of the MC mesh to the quad-dominant one. This choice also results in the maximal quad-dominant mesh resolution at a fixed voxel grid resolution, allowing us to extract the maximum detail possible.

Triangulation. Since we use Nvdiffrast [25] for differentiable rasterization, our method needs to convert the quad-dominant mesh to a triangle mesh for rendering. There are two possible ways to split a quad face into two triangles, and the chosen split direction can significantly impact the surface location, particularly at low geometry resolutions. So we use an adaptive triangulation rule: We query the o-MLP for each possible split direction and choose the one that best aligns with the orientation field at the center of each face.

10.1. Optimization

Inverse Rendering for Meshes. Our method uses an MLP SDF as described in Sec. 5.4 instead of a per-voxel SDF. To regularize the training of the SDF, we progressively enable encoding levels for both the SDF and materials. Specifically, we initialize training the SDF using 5 hash encoding levels for 1000 steps and enable an additional level every 500 steps until all 10 levels are enabled. Similarly, we initially enable 3 levels for the material encoding and enable an additional level every 500 steps until 16 levels are reached. We only apply subdivision and displacement after 3000 steps once the overall shape has converged, and we enable another level every 1500 steps until 2 levels are reached. This schedule is visualized in Fig. 15 (a). The same configuration is used for all scenes, except the FICUS, where we saw improved results by enabling more SDF hash levels earlier, as shown in Fig. 15 (b). In total, we optimize each scene for 6000 steps.

Experimentally, we found that applying subdivision on the quad-dominant mesh during the entire optimization process resulted in worse convergence across scenes, as the surface optimization would fail to recover concave regions in the objects as described in Sec. 4.3. By directly rendering the un-subdivided mesh $\mathcal{S}_0$ and enabling subdivision after 3000 steps and the final level $\mathcal{S}_2$ after 1500 further steps.

Following Nvdiffrec, we warm up the optimization by linearly increasing the learning rate to 0.03 over 200 steps and then exponentially decaying it over 5000 steps to 0.003 as visualized in Fig. 15.

Mesh reconstruction. We use a hash-encoded MLP SDF described in Sec. 4 with progressively increasing hash encoding depth starting at 5 levels for the first 200 steps and then enabling another level every 100 steps until the maximum of 10 levels is reached. In total, we run shape optimization for 1000 steps with a learning rate of 0.01 for each method. When displacement is used, we train the reconstruction for 500 steps without it and enable one level of subdivision and displacement for the last 500 steps. We visualize this training schedule in Fig. 15 (c).

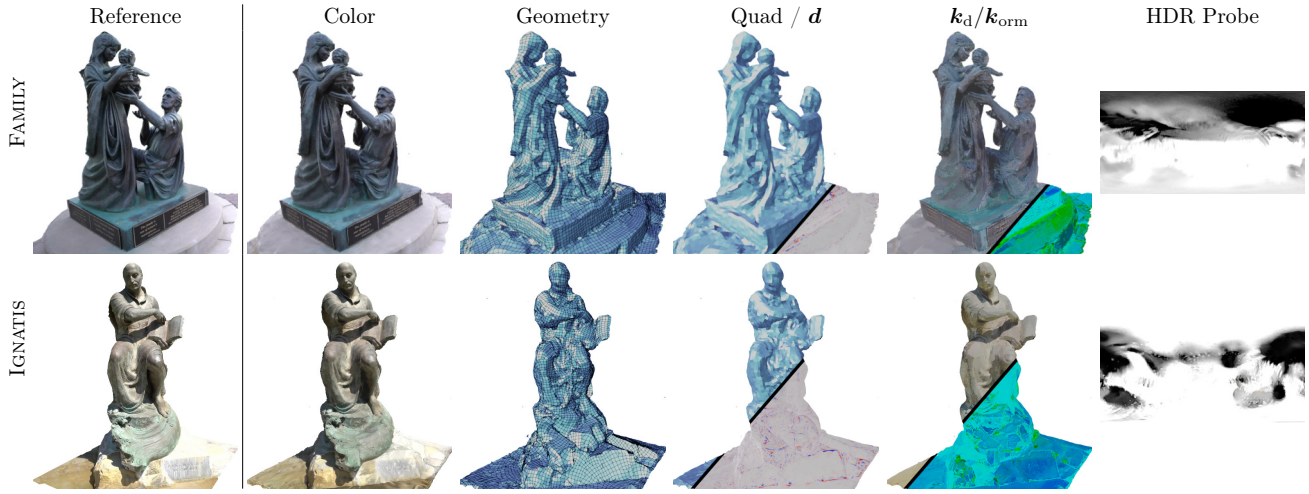
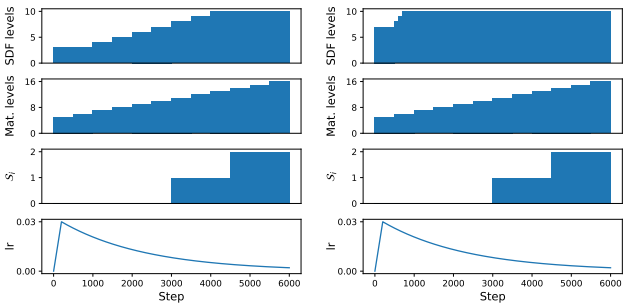
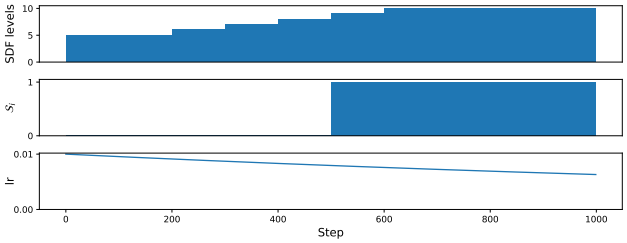


Figure 14. Reconstruction on two scenes from the Tanks and Temples dataset [24]. We show large-scale scenes with extracted quad-dominant geometry, displacement, materials, and lighting.



(a) View interpolation (b) View interpolation on Ficus



(c) Mesh reconstruction

Figure 15. Training schedules for the View interpolation and Mesh reconstruction experiments in Sec. 5.1 and Sec. 5.2 respectively.

10.2. Losses

To supervise our method, we follow a similar approach to Nvdiffrec [35] and mainly use an image and mask loss to train our pipeline and use similar losses throughout this work. In this section, we provide details for each loss.

Image loss. Since colors are represented in linear space in our physically-based shading pipeline, a loss function robust to unbounded radiance values is required. Therefore, we tone map the rendered images I_θ using a logarithmic mapping and the sRGB transfer function $\Gamma(x)$ [50] before comparing it to the ground-truth images I_{gt}. After tone-mapping each image, we compute the coverage and texture loss, following Nvdiffrec [35]:

$$\begin{aligned} \Gamma(x) &= \begin{cases} 12.92x & x \leq 0.0031308 \\ (1+a)x^{1/2.4} - a & x > 0.0031308 \end{cases} \\ C(x) &= \Gamma(\log(1+x)) \\ \mathcal{L}_{\mathrm{tex}} &= \| I^{\mathrm{a}}_{\mathrm{gt}} \left(C(I^{\mathrm{a}}_{\mathrm{gt}} I^{\mathrm{rgb}}_{\theta}) - C(I^{\mathrm{a}}_{\mathrm{gt}} I^{\mathrm{rgb}}_{\mathrm{gt}}) \right) \|_1 \\ \mathcal{L}_{\mathrm{mask}} &= \frac{1}{|I_{\mathrm{gt}}|} \| I^{\mathrm{a}}_{\theta} - I^{\mathrm{a}}_{\mathrm{gt}} \|_2^2 \end{aligned} \tag{13}$$

To separate the texture and coverage losses, we denote the color channels of the rendered and ground-truth image respectively, by $I^{\mathrm{rgb}}_{\theta}, I^{\mathrm{rgb}}_{\mathrm{gt}}$ with their associated alpha channels $I^{\mathrm{a}}_{\theta}, I^{\mathrm{a}}_{\mathrm{gt}}$.

Light regularizer. Since real-world datasets contain mostly neutral white lighting, we follow Nvdiffrec [35] and NeRFactor [60] and regularize the lighting towards white:

$$\mathcal{L}_{\mathrm{light}} = \frac{1}{3} \sum_{i=1}^{3} |\bar{c}_i - c_i| \tag{14}$$

Where $c_i, \bar{c}_i$ are the RGB channel intensities of the light cube map and average channel intensity, respectively.

Occlusion regularizer. As the shading model Nvdiffrec uses does not support shadows, they modulate the outgoing radiance by an occlusion mask stored in the first channel of the $\boldsymbol{k}_{\mathrm{orm}}$ map. Since this effect is not physically accurate, it

is regularized to be small.

$$\mathcal{L}_{\text{occ}} = \frac{1}{|\boldsymbol{k}_{\text{orm}}|}\|\boldsymbol{k}^{\text{o}}_{\text{orm}}\|^2 \tag{15}$$

Material regularizer. Similar to NeRFactor [60], Nvdiffrec regularizes the predicted Albedo to be as smooth as possible by enforcing the albedo at a surface location $\mathbf{x}_{\text{surf}}$ to be similar at over small perturbations $\epsilon \sim \mathcal{N}(0, \sigma I)$:

$$\mathcal{L}_{\text{mat}} = \sum_{\mathbf{x}_{\text{surf}}} |\boldsymbol{k}_{\text{d}}(\mathbf{x}_{\text{surf}}) - \boldsymbol{k}_{\text{d}}(\mathbf{x}_{\text{surf}} + \epsilon)| \tag{16}$$

Different from Nvdiffrec, during second stage optimization, we perform albedo regularization using perturbations ϵ in image space, unlike Nvdiffrec, which does so in uv-space. This difference is particularly pronounced when the automatic uv-unwrapping of the first stage mesh contains many disconnected uv-islands, on the boundary of which adjacent faces on the mesh might not be adjacent in uv-space.

Laplacian regularizer. In the second stage, when the topology is fixed, a Laplacian regularizer is used to regularize vertex movements. Given the uniformly weighted vertex differential $\delta_i = \mathbf{v}_i - \frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)}\mathbf{v}_j$, where $\mathcal{N}(i)$ are the 1-ring neighbors of vertex $\mathbf{v}_i$, Nvdiffrec use the following regularizer:

$$\mathcal{L}_{\delta} = \frac{1}{n}\sum_{i=1}^{n}\|\delta_i - \delta'_i\|_2^2 \tag{17}$$

Where δ_i, δ'_i are the Laplacians of the vertex $\mathbf{v}_i$ in the currently optimized mesh, and the fixed input mesh (output from the first stage), respectively.

Displacement regularizer. To encourage the overall surface shape to be represented using the quad-dominant mesh and only represent small shading details too small for the quad-dominant mesh as displacement, we regularize the displacement magnitude:

$$\mathcal{L}_{\text{disp}} = \frac{1}{|\mathcal{S}_l|}\sum_{\mathbf{v}\in\mathcal{S}_l}\|\mathbf{d}(\mathbf{v})\|_1 \tag{18}$$

Op loss. For self-learning the orientation field, he wrapped version of the van-Mises distribution introduced in Sec. 4.1 can be efficiently computed by applying the double angle formula repeatedly ($\cos(2x) = 2\cos^2(x) - 1$):

$$\begin{aligned}\mathcal{L}_{\mathbf{o}}(\hat{\mathbf{o}}, \mathbf{o}) &= 1 - \exp(\cos(4\theta) - 1) \\ &= 1 - \exp(8(\cos^4\theta - \cos^2\theta))\end{aligned} \tag{19}$$

Given two unit vectors $\mathbf{a}, \mathbf{b}$, the equivalence $\cos\theta = \langle \mathbf{a}, \mathbf{b}\rangle$ allows for efficiently computing the powers of $\cos\theta$ in the above equation.

λ_{mask}	1	λ_{tex}	1
λ_{light}	$5 \cdot 10^{-4}$	λ_{occ}	10^{-3}
λ_{mat}	$3 \cdot 10^{-2}$	λ_{disp}	$5 \cdot 10^{-3}$
λ_{op}	10^{-2}	λ_{δ}	$3 \cdot 10^{4}$
λ_{depth}	10	λ_{M}	1
λ_{eik}	1	λ_{sdf}	$2 \cdot 10^{3}$

Table 7. Loss weights.

Inverse Rendering losses. For the inverse rendering experiments in Sec. 5.2, we follow Nvdiffrec and run the optimization for two stages: In both stages, we optimize for an accurate reconstruction of the training images using the following losses:

$$\begin{aligned}\mathcal{L}_{\text{recon}} =& \lambda_{\text{mask}}\mathcal{L}_{\text{mask}} + \lambda_{\text{tex}}\mathcal{L}_{\text{tex}} + \lambda_{\text{light}}\mathcal{L}_{\text{light}} + \\ & \lambda_{\text{occ}}\mathcal{L}_{\text{occ}} + \lambda_{\text{mat}}\mathcal{L}_{\text{mat}} + \lambda_{\text{disp}}\mathcal{L}_{\text{disp}}\end{aligned} \tag{20}$$

Weights for all losses are given in Tab. 7. In the first stage, we optimize the shape, topology, materials, and lighting using the reconstruction loss combined with our orientation and position loss:

$$\mathcal{L}_{\text{first}} = \mathcal{L}_{\text{recon}} + \lambda_{\text{op}}\mathcal{L}_{\text{op}} \tag{21}$$

Once the reconstruction has converged, we freeze the topology, re-parametrize the model to use 2D textures to represent materials instead of the material MLP, and continue optimizing in a second stage to remove texture seams:

$$\mathcal{L}_{\text{second}} = \mathcal{L}_{\text{recon}} + \lambda_{\delta}\mathcal{L}_{\delta} \tag{22}$$

Eikonal Loss. To enforce that the MLP SDF is a valid distance, we use an eikonal loss, regularizing the gradient magnitude towards 1. This enforces the SDF to satisfy the eikonal equation $\|\nabla s(\mathbf{x})\| = 1$ almost everywhere:

$$\mathcal{L}_{\text{eik}} = \frac{1}{N}\sum_{i=1}^{N}(\|\nabla s(\mathbf{x}_i)\|_2 - 1)^2 \tag{23}$$

Mesh reconstruction losses. For the mesh reconstruction experiments in Sec. 5.2, we eliminate potential inconsistencies when using images for reconstruction and supervise the mesh extraction using depth, mask images following FlexiCubes [48]. Since we are supervising training using depth D_θ, D_{gt} and mask images M_θ, M_{gt} rendered form the reference objects, we replace the image losses with the ones an L_2 loss comparing depth $\mathcal{L}_{\text{depth}} = \frac{1}{|D_{\text{gt}}|}\|M_{\text{gt}}D_\theta - M_{\text{gt}}D_{\text{gt}}\|$ and an L_1 loss on the rendered mask $\mathcal{L}_M = \frac{1}{|M_{\text{gt}}|}\|M_\theta - M_{\text{gt}}\|_1$. Additionally, the predicted SDF $s_\theta(\mathbf{x})$ is supervised at 1000 randomly sampled points with the ground truth SDF $s_{\text{gt}}(\mathbf{x})$ every iteration with

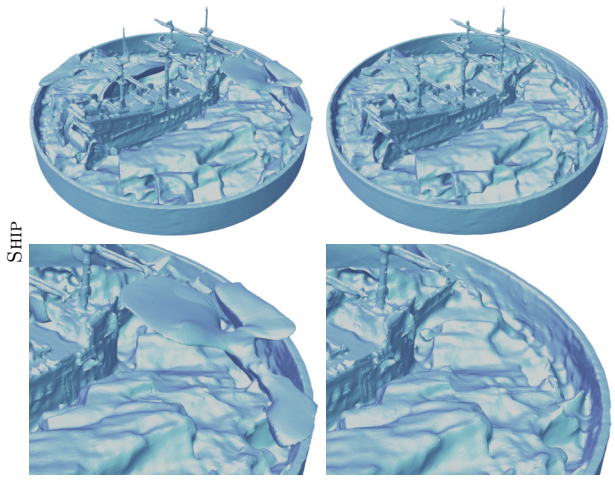


Figure 16. Transparency in secondstage optimization as proposed by Nvdiffrec. Enabling the prediction of transparent Albedo causes the reconstruction to predict transparent blobs (left), decreasing reconstruction accuracy.

$\mathcal{L}_{\text{sdf}} = \frac{1}{N}\sum_{i=1}^{N}(s_{\text{gt}}(\mathbf{x}_i) - s_\theta(\mathbf{x}_i))^2$. This gives the following loss:

$$\begin{aligned}\mathcal{L}_{\text{mesh}} = &\lambda_M\mathcal{L}_M + \lambda_{\text{depth}}\mathcal{L}_{\text{depth}} + \lambda_{\text{sdf}}\mathcal{L}_{\text{sdf}} \\ &\lambda_{\text{disp}}\mathcal{L}_{\text{disp}} + \lambda_{\text{eik}}\mathcal{L}_{\text{eik}} + \lambda_{\text{op}}\mathcal{L}_{\text{op}}\end{aligned} \tag{24}$$

We do not perform second stage optimization in this experiment since we do not reconstruct materials and lighting.

10.3. Second Stage optimization

Nvdiffrec supports rendering a transparent Albedo texture $\boldsymbol{k}_{\text{d}}$ during second-stage optimization, where mesh the topology is frozen and only vertex positions, lighting, and materials are optimized. Allowing for partially transparent materials improves view interpolation quality, especially on scenes with small geometric features, such as FICUS. Also, since the shading model used in Nvdiffrec does not model refractive surfaces explicitly, these are represented by semi-transparent geometry.

However, this transparent geometry causes blob-like artifacts, as shown in Fig. 16, which lower chamfer scores. To give a fair comparison with Nvdiffrec, Nvdiffrecmc, and FlexiCubes, we show results throughout this work using transparency as proposed by Nvdiffrec. In particular, the results in Fig. 18 allow using transparent Albedo textures.

10.4. Differentiable Subdivision Algorithm

The Algorithm proposed by Dupuy et al. [17] as presented only supports manifold meshes. However, our method can extract non-manifold vertices [28] during optimization, which must also be subdivided consistently to compute correct gradients on all vertices. Therefore, we modify the algorithm by Dupuy et al. [17] to freeze non-manifold vertices, shown in Fig. 17 and in Alg. 1. We use $\mathcal{S}_d = (\mathcal{H}_d, \mathcal{V}_d)$ to denote the subdivided mesh at the d-th level with $\mathcal{H}_d, \mathcal{V}_d$ denoting the half-edges and vertices respectively.

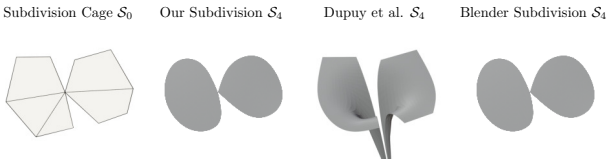


Figure 17. The method by Dupuy et al. [17] only subdivides manifold meshes. Their method introduces spurious geometry when subdividing non-manifold vertices $\mathcal{S}_0$.

Algorithm 1 Modified VERTEXPOINTS algorithm by [17] freezing non-manifold vertices.

procedure NONMANIFOLDVERTEXPOINT($\mathcal{S}_d$: Input mesh)
  $\mathcal{V}_{d+1} \leftarrow$ VERTEXPOINTS($\mathcal{S}_d$) ▷ Alg. 4 [17]
  $n_0, n_1, \ldots, n_{|\mathcal{V}_{d+1}|} = 0$
  $e_0, e_1, \ldots, e_{|\mathcal{V}_{d+1}|} = 0$
  for $h \in \mathcal{H}_d$ **do**
    $n \leftarrow$ VALENCE($\mathcal{S}_d, h$) ▷ Alg. 6 [17]
    $v \leftarrow$ VERT(h) ▷ Vertex of h
    $n_v \leftarrow n_v + n$ ▷ Valence to vertices
    $e_v \leftarrow e_v + 1$ ▷ Inc. vertex edges
  end for
  for $v \in \mathcal{V}_{d+1}$ **do**
    if $e_v \neq n_v$ **then**
      $\mathcal{V}_{d+1}[v] \leftarrow \mathcal{V}_d[v]$ ▷ Freeze non-manifold
    end if
  end for
  return $\mathcal{V}_{d+1}$
end procedure

11. Scene Credits

Spot and Bob models (CC-0) by Keenan Kane. The NeRD dataset [39] (moldGoldCape and ethiopianHead) (CC-NC-SA 4.0). TanksAndTemple Dataset by Knapitsch et al. [24]. We thank the blendswap[4] users for the models from the NeRF Synthetic dataset [33]: 1DInc (chair, CC-0), bryanajones (drums, CC-BY), Herberhold (ficus, CC-0), erickfree (hotdog, CC-0) Heinzelnisse (lego, CC-BY-NC), elbrujodelatribu (materials, CC-0), uped.de (mic, CC-0), gregzaal (ship, CC-BY-SA). The environment maps from Poly Heaven (CC0) are shipped with Blender [13]. Models in the dataset by Mylet et al. [36] are collected from the AIM@Shape database and other popular repositories.

[4] https://blendswap.com/

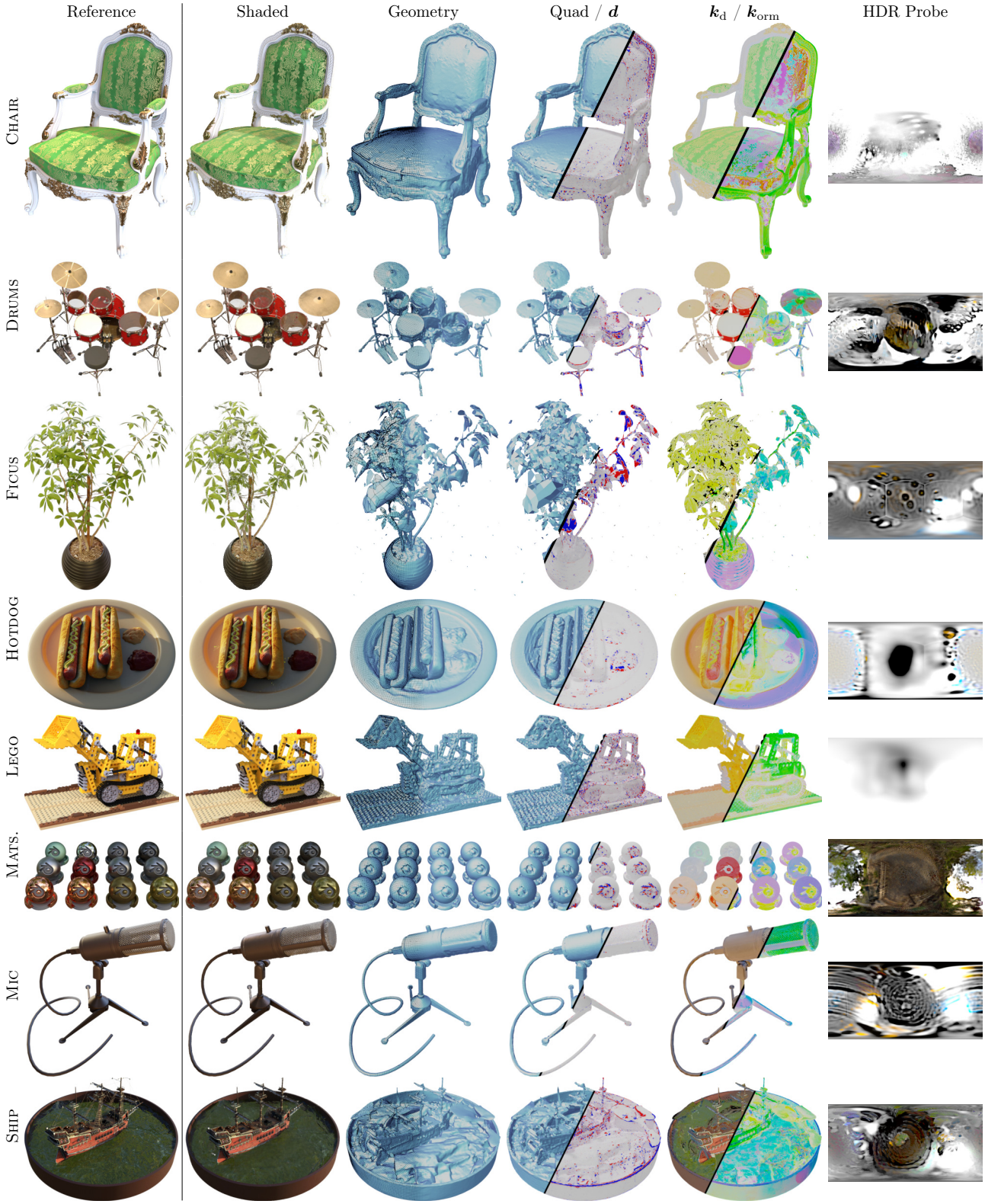


Figure 18. Mesh extraction results on the NeRF Synthetic dataset. We show our shaded models, the displaced geometry, quad-dominant meshes with the displacement magnitude, materials, and lighting. Quad-dominant meshes capture the base geometry, while high-frequency details are represented by displacement on the subdivided mesh. Note that transparent textures represent fine structures in the shaded images better.

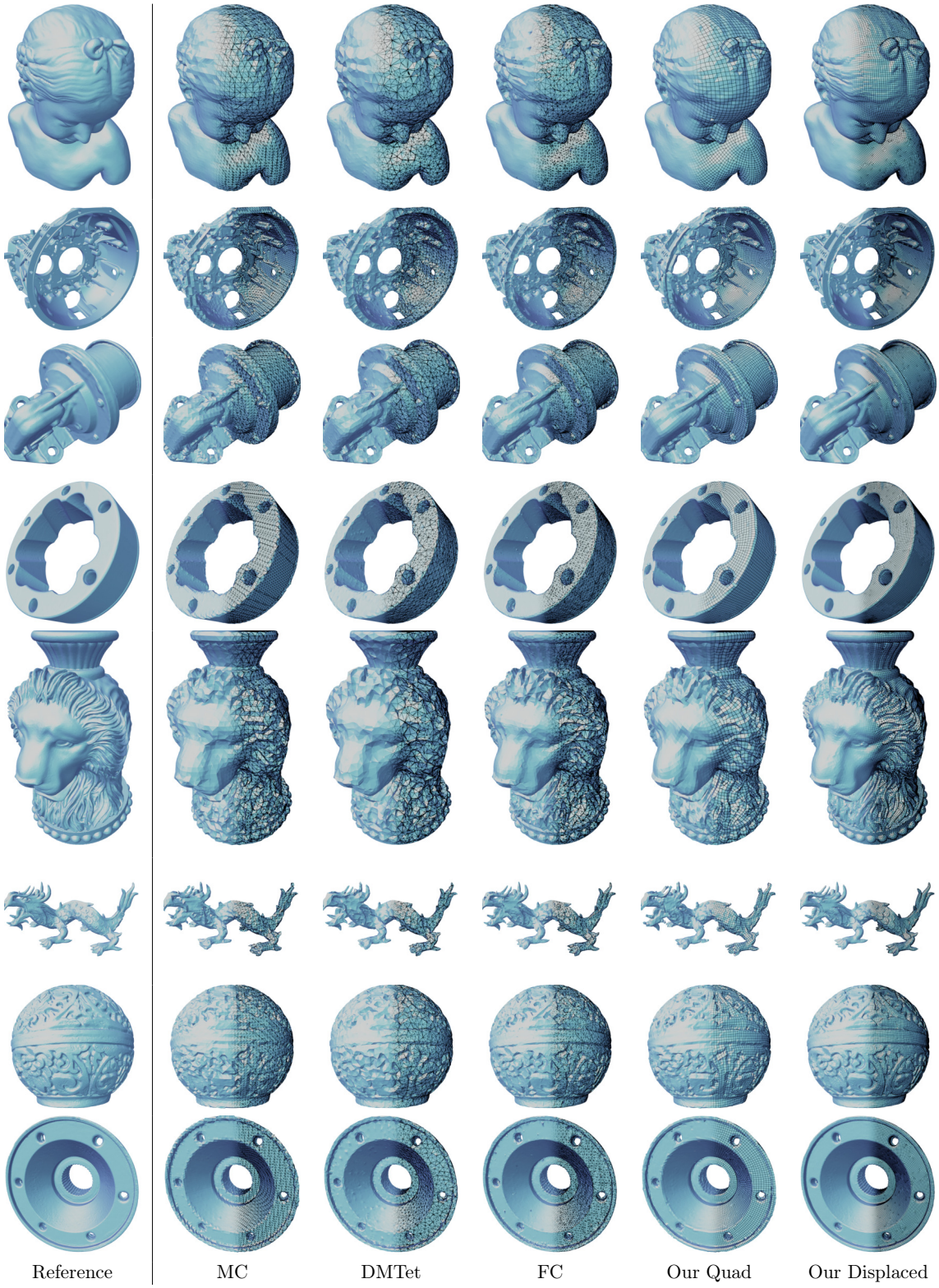


Figure 19. Visual comparison of different mesh extraction techniques. Our displaced and quad-dominant meshes, Marching Cubes (MC), and DMTet [47] are generated by directly supervising predicted geometry with depth and mask losses on the ground truth. Our quad-dominant meshes show clear alignment of the quad faces to object features and extraction of surface details visible in the displaced surfaces.