Flexible Isosurface Extraction for Gradient-Based Mesh Optimization Supplement

We start the supplement by providing more details on our method in Section 1, as well as further details of different isosurfacing methods in Section 2. Section 3 describes the experimental setting along with the baselines and depicts more qualitative results. Finally, in Section 4, we provide additional details and results for our applications.

1 DETAILS ON FLEXICUBES

1.1 Tetrahedral Mesh Extraction

In Section 4.5 of the main paper, we describe the ambiguity in connectivity when extending the tetrahedralization strategy proposed by Liang and Zhang [2014] to Dual Marching Cubes (DMC). The ambiguity arises when a cell contains multiple extracted mesh vertices. Connecting the incorrect vertices leads to intersections or holes in the extracted tetrahedral mesh. We propose two rules to address the ambiguity cases. Recall that the tetrahedra are formed in two cases:

(1) For each grid edge connecting two grid vertices with *different* signs, we first form a four-sided pyramid by connecting one of the grid vertices with four mesh vertices that correspond to the grid edge and then subdivide the pyramid into two tetrahedra. This case is uniquely determined in all DMC configurations (see bottom-left subfigure of Figure 10 in the main paper).
(2) For each grid edge connecting two grid vertices with the *same* sign, the tetrahedron is formed by the two grid vertices and two vertices in consecutive adjacent cells (referring to top-left subfigure in Figure 10 in the main paper). In this case, we first identify the face shared by the adjacent cells. We then identify an edge of the face with different signs and select the mesh vertex corresponding to the identified edge. Referring to Figure 7 in the main paper, in cases C6, C10, C12, and C15, it is obvious that the formed tetrahedra following this rule are always inside the primal faces. Note that the described rule can be implemented efficiently with a precomputed lookup table.

In most cases, these rules address the ambiguity and result in correct tetrahedralization of the interior volume. However, for case C18, the interior volume is not completely filled by the formed tetrahedra. Although this case rarely happens during optimization and does not obstruct the downstream application, it remains a limitation of our method.

1.2 Adaptive Mesh Resolution

In Section 4.6 of the main paper, we describe a constraint applied to the SDF on octree vertices to avoid cracks or non-manifold surfaces being produced by DMC. Here we provide more details on how this constraint is enforced. As a preliminary, Ju et al. [2002] propose a method to generate adaptive contours from an octree representation. Their method identifies the *minimal edges* of the octree, i.e., the edges which do not contain a finer edge of the adjacent cell, as well as four cells sharing each minimal edge with a recursive call. We refer readers to the original paper for more details about the algorithm. We repurpose this algorithm to identify the vertices whose SDF values we constrain. Specifically, for the four cells sharing a minimal edge, we check the four pairs of adjacent faces among these cells. If a finer face F_f is adjacent to a coarser face F_c, for every vertex v_f of F_f that is not a vertex of F_c, we compute and store the bilinear weights of v_f with respect to the vertices of F_c. During optimization, the SDF value of v_f is not optimized. Instead, it is directly interpolated using precomputed bilinear weights applied to the SDF values on F_c. Note that the constrained vertices only need to be re-identified when there is an update to the octree structure.

1.3 Addressing Ambiguity in DMC Configurations

In rare cases, the original Dual Marching Cubes algorithm can produce non-manifold meshes. We follow the solution described in Wenger [2013] to address the ambiguity of cases C16 and C19 in the DMC configurations. With this modification applied, the extracted surface of FlexiCubes in the uniform grid setting is always 2-manifold.

2 ANALYSIS

This section provides further details of the experiment shown in Figure 4 in the main paper, where we show the optimization process for a collection of isosurfacing algorithms.

In the analysis, we follow a similar experimental setup as Section 5.1 in the main paper. Specifically, we start by initializing the SDF to represent a sphere for all methods. In each iteration, we then extract the surface mesh from the SDF (defined on a grid). Finally, we render the reconstructed mesh from randomly sampled camera views (same for all methods) and compute the differences with the ground truth depth and silhouette image. We also compute the SDF loss, where we randomly sample 1000 points and evaluate their SDF values w.r.t the ground truth mesh, as well as the extracted mesh, and minimize the differences between two SDF values. We use L1 loss for the silhouette image, L2 norm for the depth image, and MSE loss for the SDF. The combinedd loss is back-propagated to the SDF through the differentiable isosurfacing layers, which we detail in the next paragraph. We use the same optimizer and learning rate for all methods. For FlexiCubes, we leverage the regularizers described in Section 4.7 of the main paper. We also leverage $\mathcal{L}_{\text{sign}}$ for all methods to remove floater and internal geometry. In this analysis, the grid resolution is set to 64 for the tetrahedral grid used by DMTet, and 48 for the voxel grid used by the other methods to roughly match the number of triangles in the output mesh.

2.1 Baselines

We use the official implementation of DMTet from the nvdiffrec codebase[1]. For Dual Contouring (DC) and Marching Cubes (MC),

Author's address:

[1] https://github.com/NVlabs/nvdiffrec

there are, to the best of our knowledge, no *differentiable* implementations available, so we implemented these methods in PyTorch to leverage the autograd functionalities. Specifically, we adapted the MC variant from Lorensen and Cline [1987], following the implementation of the Marching Tetrahedra function in DMTet, such that the zero-crossings on grid edges are computed in a differentiable manner. For DC, we utilize PyTorch's linear solver, `lstsq`[2], to solve the quadratic error function (QEF) given in Equation 2 (main paper). The gradient direction at any point, $\nabla s(u_e)$, is approximated by local differentiation over the SDF computed via trilinear interpolation. To avoid the solution exploding to a distant location when $\nabla s(v_e)$ are nearly coplanar, we followed the DC implementation in the NDC source code[3] and add a regularization term which biases the solution toward the centroid of associated zero-crossings V_E. Specifically, Equation 2 is regularized as:

$$v_d = \underset{v_d}{\operatorname{argmin}} \sum_{u_e \in \mathcal{Z}_e} \nabla s(u_e) \cdot (v_d - u_e) + \lambda |v_d - \overline{u_e}|, \qquad (1)$$

where $\overline{u_e} = \frac{1}{|V_E|} \sum_{u_e \in V_E} u_e$ is the centroid of the zero-crossing points and λ is the scalar weight that controls the strength of the regularizer. We ablate two DC versions with $\lambda = 1$ and $\lambda = 0.01$, denoted as DC_{reg1} and DC_{reg001} respectively. In addition, we compare with a DC variant which directly takes the centroid as the mesh vertex, denoted as $DC_{centroid}$. Please refer to Figure 2 in the main paper for illustrations of these regularized versions of DC. For NDC, we use a pretrained neural network provided by the authors to extract the isosurface. During optimization, we freeze the network parameters and only optimize the SDF values.

3 EXPERIMENTAL DETAILS

This section provides additional details for the experiment settings in Section 5 in the main paper.

3.1 Baselines

The implementation of the baseline methods used in this experiment is described in Section 5.1 of the main paper. The inputs to MC_{SDF} and NDC_{SDF} are the ground truth SDF values evaluated at grid vertices. Since the pretrained neural network in NDC_{SDF} is sensitive to the scale of the SDF inputs, we use the function[4] provided by the authors to compute the SDF. For $DC_{hermite}$, which requires gradients as input, we additionally compute the gradient of the SDF at zero-crossings using finite differences. The regularizer weight λ in Eqn. 1 is determined independently for each cube in an adaptive manner, following the DC implementation by Chen et al. [2022]. Specifically, we begin by solving Eqn. 1 with a small λ, and iteratively double the value of λ until the solution with the updated QEF falls inside the cube or λ reaches the limit. The initial λ is set to 0.01 and the limit is 10^6 in our experiment. This approach is very time-consuming to evaluate and hard to leverage in a general gradient-based mesh optimization framework. Therefore, we only adopt it in this main experiment (Section 5 in the main paper) but used fixed values of λ in the optimization process discussed in Section 2 (Figure 4 in the main paper) for completeness.

For the mesh reconstruction pipeline, we leverage the codebase of NVDIFFREC[5], and replace the image loss with depth and SDF losses described in the main paper. We also leverage the mask loss in NVDIFFREC, and the two regularization losses $\mathcal{L}_{\text{sign}}$ and $\mathcal{L}_{\text{dev}}$ in Eqnuation 8 and Equation 9 from the main paper. We use L1 loss for mask, L2 norm for depth and MSE loss for SDF. We scale the mask loss, depth loss, SDF loss and $\mathcal{L}_{\text{dev}}$ by 1, 10, 2000, and 1 respectively. The scale of $\mathcal{L}_{\text{sign}}$ decays from 0.2 to 0.01 linearly during training. We optimize each shape for 1000 iterations with a learning rate of 0.01.

For all isosurfacing methods, we use uniform grids in $[-1, 1]^3$. We center each object around the origin and scale it such that the longest side of its bounding-box equals 1.8. For NDC, the effective resolution of the grid is reduced by the padding of the network. Therefore, we increase the grid resolution for NDC by the padding size for a fair comparison with other methods.

3.2 Evaluation Metrics

We provide details on all the evaluation metrics we used in the main paper.

Chamfer Distance (CD). This metric measures the distance between two point clouds by nearest neighbor search. To measure the CD between meshes, we sample each mesh to get a point cloud of size 100,000. Note that for the NVDIFFREC NeRF synthetic dataset evaluation (Table 5 in the main paper), we use a different version of Chamfer Distance, computed only on visible triangles, using 2.5M points on meshes with another scale than in the main experiment, so the CD numbers from Table 5 cannot be directly compared against the CD scores reported in Section 5.

F1-score. The harmonic mean of precision and recall. To compute precision and recall, we sample each mesh into a point cloud of the same size as CD and search for the nearest neighbor points. When computing the precision, if the distance from a point on the predicted mesh to the GT point cloud is small enough (threshold = 0.003), we count it as a *true positive* point. Otherwise, it is counted as a *false positive* point. When computing the recall, if the distance from a point on the GT mesh to the predicted mesh is small enough (threshold = 0.003), we count it as a true positive point. Otherwise, we count it as a false negative point.

Edge Chamfer Distance (ECD) and Edge F-score (EF1). These metrics are used in prior works [Chen and Zhang 2021; Chen et al. 2022] for evaluating the reconstruction of sharp features (edge points). First, for each point in the sampled point cloud, we look at the dot products between its normal and the normal of its neighbor points. If the mean dot product is smaller than a threshold (0.2), the point is treated as an edge point. ECD and EF1 measure the Chamfer Distance and F1-score between edge point sets.

The percentage of inaccurate normals (IN$> 5^\circ$). For each point in the sampled point cloud, we store the normal of the face that the point was generated from. Given a predicted mesh and a GT mesh,

[2] https://pytorch.org/docs/stable/generated/torch.linalg.lstsq.html#torch.linalg.lstsq
[3] https://github.com/czq142857/NDC
[4] https://github.com/czq142857/NDC/blob/9054e20f55013d031af9e3a2c91f5cab75837bc4/data_preprocessing/get_groundtruth_NDC/SDFGen
[5] https://github.com/NVlabs/nvdiffrec

we search for nearest point pairs from one to another. We compute the angles between the normals stored on two points, and report the percentage of pairs having angles larger than 5 degrees.

Aspect Ratio (AR) and Radius Ratio (RR). AR and RR are different measures of triangle regularity, with a smaller value indicates better triangle quality. While there exist different definitions of AR and RR in the literature, we follow the definition in the PyVista[6] codebase in our evaluations.

Min and max angles. Given a triangle on the extracted mesh, we compute its three angles in degrees and select the max and min angles.

NV(%). The average percentage of non-manifold vertices.

NE(%). The average percentage of non-manifold edges.

SI(%). The average percentage of self-intersecting triangles.

SA<10°. The average percentage of triangles with the smallest angle <10°.

3.3 Adaptive Meshing

We provide experimental details for results demonstrated in Figure 14 in the main paper. Our goal is to reconstruct the target object with adaptive mesh resolution achieved by jointly optimizing mesh topology and the octree structure. We begin the optimization with a low-resolution, uniform voxel grid. During optimization, we keep a running average of the objective loss for each cell, computed by averaging the loss from all mesh vertices extracted from it. After the shape converges, we subdivide the cells in the octree with objective loss larger than a preset threshold of 0.04. Iterating this process, we obtain the adaptive mesh shown in Figure 14 without precomputed octree structure on GT geometry. Note that we apply the constraint described in Section 1.2 during optimization.

3.4 Additional Results

We include more visual examples in Figure 3, comparing all methods. Please zoom in the pdf to see the differences. Additional statistic are included in Table 1.

In Figure 1 we show how FlexiCubes robustly reconstructs the same object under different rotations. Note that the MC reconstruction quality deteriorates when features are not axis-aligned. We show the influence of the regularizer $\mathcal{L}_{\text{dev}}$ (Equation 8 in the main paper) in Figure 2.

4 APPLICATIONS

In this section, we provide a detailed description of the experimental setting and additional results for each application we did in the main paper.

4.1 Photogrammetry Through Differentiable Rendering

Our photogrammetry application is based on nvdiffrec, which jointly optimizes shape, materials, and lighting from image supervision [Hasselgren et al. 2022; Munkberg et al. 2022]. We followed the official codebase closely with minimal changes and only replaced the DMTet [2021] geometry extraction stage with FlexiCubes. We leverage the regularizer term, $\mathcal{L}_{\text{dev}}$, of Equation 8 from the main paper with a factor $\lambda_{dev} = 0.25$ in all experiments. The regularizer, $\mathcal{L}_{\text{sign}}$, (Equation 9, main paper) is already present with DMTet in nvdiffrec. Additionally, we scale the silhouette mask loss of nvdiffrec, with a factor, $\lambda_{mask} = 5.0$, to further emphasize geometry. The combined objective function is:

$$\mathcal{L}_{\text{total}} = \lambda_{dev}\mathcal{L}_{\text{dev}} + \mathcal{L}_{\text{sign}} + \lambda_{mask}\mathcal{L}_{\text{mask}}. \quad (2)$$

[6]https://docs.pyvista.org/api/core/_autosummary/pyvista.DataSet.compute_cell_quality.html

Table 1. Quantitative results on Mesh Reconstruction. CD: Chamfer distance (1e-5), F1: F1 score. ECD, edge chamfer distance (1e-2). EF1: edge F1. NV: Non-mainfold vertices, NE: Non-manifold edges, SI: self-intersection. IN>5°: normal angle difference > 5°. SA: small angle. # V: number of vertices. #F: number of triangles.

32^3	CD↓	F1 ↑	ECD ↓	EF1 ↑	#V	#T	NV(%) ↓	NE(%) ↓	SI(%) ↓	IN>5°(%) ↓	SA<10°(%) ↓
MC_{SDF}	22.65	0.28	5.56	0.08	2387	4771	0.0	0.0	0.0	85.6	24.7
$DC_{hermite}$	17.15	0.38	4.82	0.11	2360	4775	0.131	0.349	1.882	74.43	9.59
NDC_{SDF}	17.61	0.42	3.55	0.13	1877	3801	0.155	0.378	0.232	72.60	0.77
MC	9.11	0.54	2.60	0.13	2573	5146	0.0	0.0	0.0	66.61	11.74
DMTet(32)	11.56	0.52	3.64	0.17	1691	3387	0.0	0.0	0.0	66.22	12.32
DMTet(40)	8.35	0.58	3.64	0.20	2626	5259	0.0	0.0	0.0	61.21	12.72
FlexiCubes	7.01	0.64	2.11	0.26	2400	4800	0.0	0.0	0.715	50.52	2.99
64^3	CD↓	F1 ↑	ECD ↓	EF1 ↑	#V	#T	NV(%) ↓	NE(%) ↓	SI(%) ↓	IN>5°(%) ↓	SA<10°(%) ↓
MC_{SDF}	6.84	0.55	2.55	0.14	9881	19783	0.0	0.0	0.0	80.67	24.32
$DC_{hermite}$	5.90	0.61	3.80	0.23	9828	19769	0.043	0.139	1.483	63.34	8.67
NDC_{SDF}	6.16	0.57	1.22	0.26	9828	19769	0.043	0.140	0.130	55.22	0.48
MC	6.33	0.66	1.25	0.25	10459	20801	0.0	0.0	0.0	52.37	11.90
DMTet(64)	7.50	0.66	3.77	0.28	6783	13566	0.0	0.0	0.0	50.20	14.41
DMTet(80)	5.17	0.66	3.59	0.29	10385	20784	0.0	0.0	0.0	48.66	17.88
FlexiCubes	4.87	0.70	0.71	0.43	9916	19843	0.0	0.0	0.103	34.87	1.97
128^3	CD↓	F1 ↑	ECD ↓	EF1 ↑	#V	#T	NV(%) ↓	NE(%) ↓	SI(%) ↓	IN>5°(%) ↓	SA<10°(%) ↓
MC_{SDF}	4.72	0.68	1.13	0.33	40164	80374	0.0	0.0	0.0	77.23	24.25
$DC_{hermite}$	4.59	0.69	3.82	0.40	40128	80360	0.017	0.069	1.280	53.98	7.95
NDC_{SDF}	5.04	0.65	0.79	0.43	40129	80361	0.017	0.036	0.084	43.2	0.37
MC	4.51	0.72	1.32	0.44	38645	77212	0.0	0.0	0.0	42.56	12.46
DMTet(128)	4.98	0.74	1.50	0.39	23535	47001	0.0	0.0	0.0	48.86	23.52
FlexiCubes	4.31	0.71	0.42	0.51	38923	77845	0.0	0.0	0.017	30.57	0.79

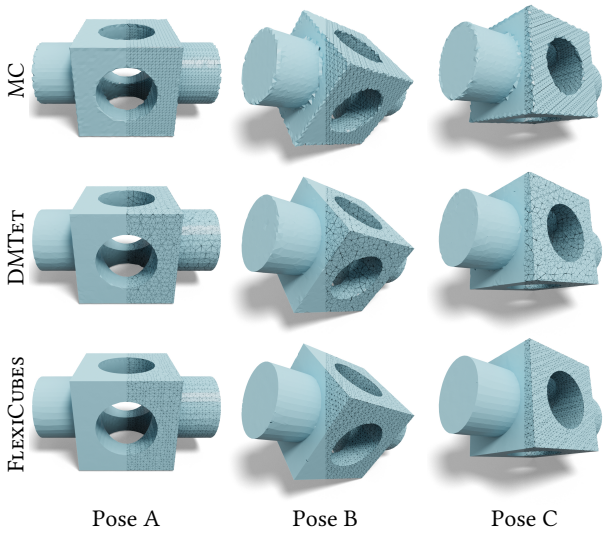
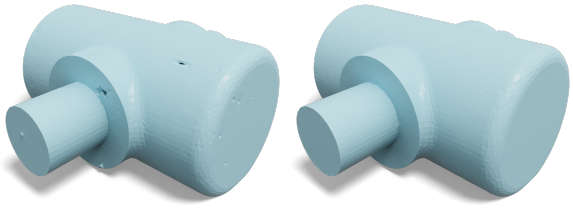


Fig. 1. We reconstruct the same model with three different poses. Marching Cubes performs well when features are axis-aligned, but show stair step artifacts in the other poses. DMTet performs more robustly, but produces many sliver triangles. FlexiCubes has more uniform triangles and no stair step artifacts.



$\mathcal{L}_{\text{dev}}$ off $\mathcal{L}_{\text{dev}}$ on

Fig. 2. Influence of the regularizer $\mathcal{L}_{\text{dev}}$. The reconstructed shape with $\mathcal{L}_{\text{dev}}$ has less visible seams compared to the result without $\mathcal{L}_{\text{dev}}$ applied.

We leave the second pass of NVDIFFREC (which performs light, material, and shape optimization with fixed topology) unmodified.

We show shaded results and mesh illustrations in Figure 4 for the entire NeRF dataset.

4.2 Mesh Simplification of Animated Objects

We extend our photogrammetry application (from Section 4.1) by adding support for mesh based, synthetic, datasets with skinned animations. In practice, we use the skinning provided by Universal Scene Description (USD) [2016] and source our animated meshes from RenderPeople [2020]. We achieve mesh simplification by using a coarse FLEXICUBES grid (32^3) and optimize geometry using image supervision. In this application, we assume that the skeleton animation and reference skinning weights are known, and the task at hand is to learn a simplified mesh, and retarget the animation using the same skeleton, without manual adjustments.

Mesh simplification using only the T-pose is straightforward and can be done in NVDIFFREC out of the box, with either DMTET or FLEXICUBES for the topology extraction step. In each iteration, we pick a random viewpoint and optimize the parameters using photometric loss. We then re-skin the reconstructed, simplified mesh in a post-processing step as follows: for each vertex in the simplified mesh, we find the k-nearest neighbors ($k = 10$) in the reference mesh and use inverse distance weighting to compute skinning weights for the simplified vertex. More formally, given a vertex, $\mathbf{v}_{\text{lod}}$, in the simplified mesh and the k-nearest neighbors, $\mathbf{v}^i_{\text{ref}}$, and their skinning weights, w^i_{ref}, from the reference mesh, the skinning weight, w_{lod}, for the simplified mesh is computed as:

$$\begin{aligned} d\left(\mathbf{v}^i_{\text{ref}}, \mathbf{v}_{\text{lod}}\right) &= \frac{1}{\max(10^{-3}, \|\mathbf{v}^i_{\text{ref}} - \mathbf{v}_{\text{lod}}\|_2)} \\ w_{\text{lod}} &= \frac{\sum_i d\left(\mathbf{v}^i_{\text{ref}}, \mathbf{v}_{\text{lod}}\right) w^i_{\text{ref}}}{\sum_i d\left(\mathbf{v}^i_{\text{ref}}, \mathbf{v}_{\text{lod}}\right)}. \end{aligned} \tag{3}$$

Performing end-to-end mesh simplification over the animation is more challenging. We first optimize for 300 iterations using the T-pose, as described above, to get a reasonable initial guess for geometry. We then enable the animation and optimize for random viewpoints and random animation frames. For this to work, our pipeline must support animation of a mesh with changing topology in a consistent way with gradients propagating back to the topology representation. In each iteration we re-skin the FLEXICUBES mesh using a differentiable version of the same k-nearest neighbor method outlined in Equation 3, and animate the re-skinned mesh using the differentiable skinning approach of Hasselgren et al. [2021]. Note that, while we do not explicitly optimize skinning weights, each operation must be differentiable to enable end-to-end training. An interesting avenue for future work is to also include optimization of the skinning weights, but that would require a consistent parametrization, as vertices can be added or removed during optimization as the topology evolve. This is similar to the texture parameterization problem in NVDIFFREC [2022] work, which is solved by using 3D texturing (encoded in a MLP) during the topology optimization phase.

4.3 3D Mesh Generation

In the application of mesh generation, we adopt the recent state-of-the-art 3D generative model GET3D [Gao et al. 2022], and show that FLEXICUBES as a plug-and-play differentiable mesh extraction module can produce significantly improved mesh quality.

GET3D [Gao et al. 2022] is a learning-based model trained on 2D images, and can directly synthesize high-quality textured 3D meshes at inference time. The framework combines classic generative adversarial networks [Karras et al. 2019, 2020], differentiable iso-surfacing [Shen et al. 2021] and differentiable rasterization-based rendering [Laine et al. 2020]. Given a sampled noise vector from a Gaussian distribution, the generator of GET3D predicts a signed distance field. Then, a mesh is extracted by DMTet [Shen et al. 2021], and a differentiable renderer renders one RGB image and one 2D silhouette, which are fed into 2D discriminators to classify whether they are real or fake. The differentiable iso-surfacing module enables the ability to directly generate meshes, and also largely affects the quality of the produced meshes.

In this application, we replace the DMTet module with FLEXICUBES. In particular, we modify the last layer of the 3D generator in GET3D to output the SDF and deformation for each vertex, and additionally generate 21 weights for every cube defined in FLEXICUBES, including the 8 vertex weights, 12 edge weights, and the remaining 1 parameter for quad splitting. FLEXICUBES is adopted to differentiably extract meshes, and the remaining architecture, training procedure and other hyperparameters of GET3D are kept unchanged following the official released implementation[7]. We follow GET3D and train the 3D generative model on ShapeNet [Chang et al. 2015] Car, Chair and Motorbike categories using the same dataset rendering pipeline and train/validation/test split released by the official implementation. Note that the architecture changes happen only at the the last layer of the generator while the rest of the backbone remains the same. Thus, the computational overhead of the modification is relatively negligible. We include more qualitative visualization in Figure 5.

4.4 Physics Simulation

FlexiCubes enables extracting tetrahedral meshes with well-defined topology, which can be directly utilized in physical simulation. It can be further combined with differentiable physical simulation frameworks [Hu et al. 2020; Jatavallabhula et al. 2021] and differentiable rendering pipelines [Chen et al. 2019; Laine et al. 2020; Munkberg et al. 2022] to optimize shape, material and physical parameters from multi-view videos. We have shown two examples in teaser and Figure 24 in the main paper. In this subsection, we talk about the details of our physical simulation experiments.

Ground Truth Generation. We first prepare multi-view ground truth videos with FlexiCubes. Given a surface mesh, we apply FlexiCubes to learn the shape with 3D supervision, as well as employing texture field [Müller et al. 2022; Munkberg et al. 2022] to learn the texture map from multi-view 2D images (Section 6.1 in the main paper). FlexiCubes supports directly exporting a tetrahedral mesh. We then send it to the physical simulation framework. Here we adopt a low-res grid (32^3 resolution) to extract the tetrahedral mesh such that the the physical simulation can be more efficient. We choose to use GradSim [Jatavallabhula et al. 2021], a differentiable physics simulation framework which has shown both forward simulation and derivatives w.r.t. the physical parameters in the backward pass. We focus on FEM simulation and use neo-Hookean elasticity to model elastic objects during the simulation. During the forward simulation, we fix the two ending points of an object, letting it drop and deform under gravity. We choose time step as $\frac{1}{8192}s$ and set the mass density $D = 0.5$, For the Lamé parameters, λ, μ, which control the element's resistance to shearing and volumetric strains, we set $\mu = 1000$, $\lambda = 1000$, and a damping coefficients $d = 1.5$ for the example in the teaser. For Figure 24 in the main paper, we choose $D = 0.3$, $\mu = 1000$, $\lambda = 1000, d = 1.5$. With the deformed meshes, we then employ the differentiable rendering pipeline [Laine et al. 2020] to render them into images and composite into videos. We generate videos with 512 views, where we randomly set circular camera positions around the object. Note here we do not render images at all the time steps, but instead instead, we choose video fps as 64.

Training. Given multi-view video sequences, we then optimize shape, texture and physical materials from the video input only. As a challenging task, it is extremely hard to jointly optimize geometry and physical parameters together. In practice, we find FEM simulation is quite unstable, *e.g.*, joint optimization always has NaN values and crashed in the training. Therefore, following recent work [Anonymous 2023] (one anonymous ICLR submission 2023 at the time of our submission), we optimize the shape, texture, and physical parameters in a two-stage manner.

First, we apply the beginning frame of the video to optimize the shape and texture only. We assume the object doesn't deform in the first frame. Therefore, it is equivalent to a rigid-body mesh reconstruction (Section 6.1 in the main paper). We use the same losses but slightly tune the weights (we set $\lambda_{dev} = 1.0$ and $\lambda_{mask} = 10.0$). The first-stage optimization allows us to start from an descent shape to make the physical parameter optimization easier.

[7]https://github.com/nv-tlabs/GET3D

In the second stage, we start from the initial guess of the mass density (we initialize it as $D = 1.5$) and apply GradSim [Jatavallabhula et al. 2021] to compute the gradients of the video loss w.r.t. to the mass density. At each iteration, we send the tetrahedral mesh to GradSim and execute forward simulation to deform the shape. Then, we render the deformed shape into images and compare them with the ground truth images. We use the same loss as in the first stage and backpropagate it to the mass dentity. We use a different learning rate schedule here. At the beginning, we use the learning rate 0.1 and decays it 10 times smaller every 50 iterations). The whole optimization converges in 200 iterations.

It is worth mentioning that when extracting the tetrahedral mesh, some tetrahedra can have tiny volume, which could lead to unstable physical simulation due to numerical issue. Therefore, we conduct a tetrahedra filtering process after tetrahedral mesh extraction. Specifically, we check the volume of each tetrahedron and remove the one whose volume is less than a threshold ($2e^{-7}$ for a shape normalized in $(-0.45, 0.45)$). We find this step significantly improves the stability of the physics simulation, though at the cost of introducing some slits of the shape, as shown in Figure 6. We hope this can be further addressed by designing new regularization terms, which we leave for future work. In Figure 6, we provide more details of the physics simulation examples in the teaser and Figure24.

REFERENCES

Anonymous. PAC-NeRF: Physics Augmented Continuum Neural Radiance Fields for Geometry-Agnostic System Identification. *ICLR Sumbission*, 2023.

Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

Wenzheng Chen, Jun Gao, Huan Ling, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to Predict 3D Objects with an Interpolation-based Differentiable Renderer. In *Advances In Neural Information Processing Systems*, 2019.

Zhiqin Chen and Hao Zhang. Neural Marching Cubes. *ACM Trans. Graph.*, 40(6), 2021.

Zhiqin Chen, Andrea Tagliasacchi, Thomas Funkhouser, and Hao Zhang. Neural Dual Contouring. *ACM Trans. Graph.*, 41(4), 2022.

Jun Gao, Tianchang Shen, Zian Wang, Wenzheng Chen, Kangxue Yin, Daiqing Li, Or Litany, Zan Gojcic, and Sanja Fidler. GET3D: A Generative Model of High Quality 3D Textured Shapes Learned from Images. In *Advances In Neural Information Processing Systems*, 2022.

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

Jon Hasselgren, Nikolai Hofmann, and Jacob Munkberg. Shape, Light, and Material Decomposition from Images using Monte Carlo Rendering and Denoising. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *ICLR*, 2020.

Krishna Murthy Jatavallabhula, Miles Macklin, Florian Golemo, Vikram Voleti, Linda Petrini, Martin Weiss, Breandan Considine, Jerome Parent-Levesque, Kevin Xie, Kenny Erleben, Liam Paull, Florian Shkurti, Derek Nowrouzezahrai, and Sanja Fidler. gradSim: Differentiable simulation for system identification and visuomotor control. *International Conference on Learning Representations (ICLR)*, 2021.

Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual Contouring of Hermite Data. *ACM Trans. Graph.*, 21(3):339–346, 2002.

Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019.

Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of StyleGAN. In *Proc. CVPR*, 2020.

Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular Primitives for High-Performance Differentiable Rendering. *ACM Transactions on Graphics*, 39(6), 2020.

Xinghua Liang and Yongjie Zhang. An octree-based dual contouring method for triangular and tetrahedral mesh generation with guaranteed angle range. *Engineering with Computers*, 30(2):211–222, 2014.

William E. Lorensen and Harvey E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, 1987.

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

Jacob Munkberg, Jon Hasselgren, Tianchang Shen, Jun Gao, Wenzheng Chen, Alex Evans, Thomas Mueller, and Sanja Fidler. Extracting Triangular 3D Models, Materials, and Lighting From Images. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8270–8280, 2022.

Pixar Animation Studios. Universal Scene Description Website, 2016. http://www.openusd.org.

RenderPeople. Renderpeople, 2020. https://renderpeople.com/3d-people/.

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 (NeurIPS)*, 2021.

Rephael Wenger. *Isosurfaces: geometry, topology, and algorithms.* CRC Press, 2013.

Fig. 3. Visual comparison of a set of iso-surfacing techniques. The three leftmost examples: Marching Cubes (MC_{SDF}), Dual Contouring, Neural Dual Contouring, use surface extraction from the GT SDF. The next three examples: Marching Cubes, Deep Marching Tetrahedra, and FlexiCubes, use differentiable iso-surfacing. The grid resolution is 64^3 for all methods except DMTet, which uses 80^3 tetrahedral grid to match the triangle count in output meshes.

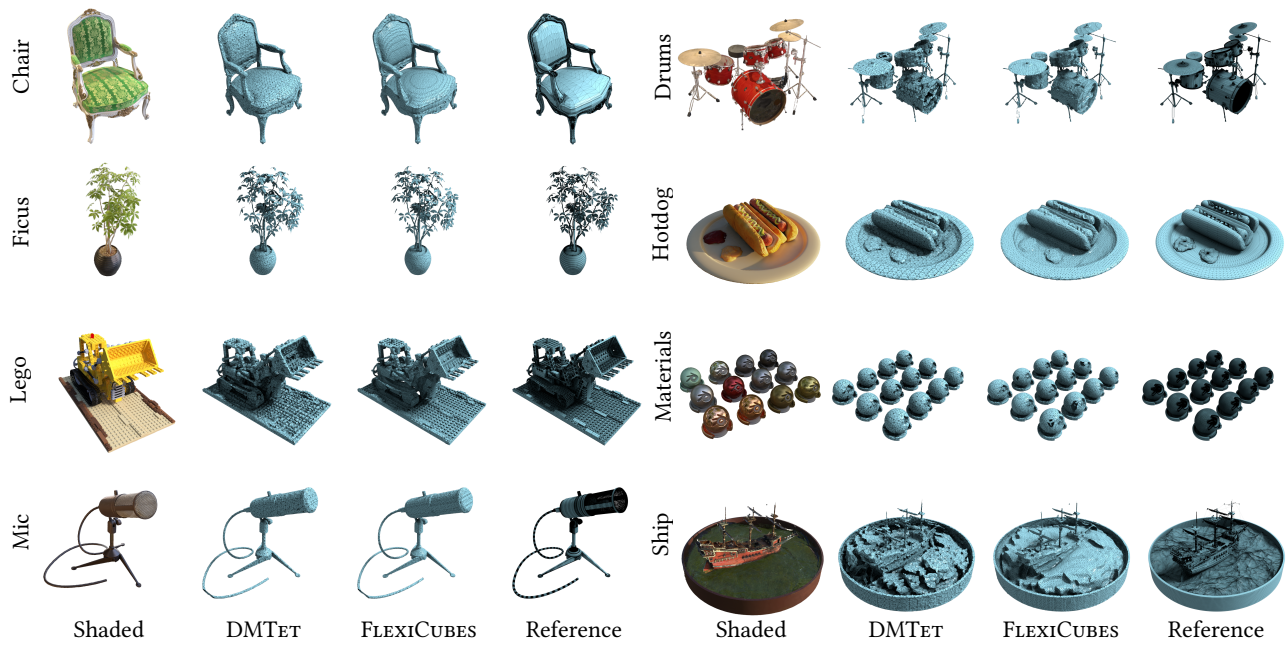


Fig. 4. Geometry breakdown for all scenes of the original NeRF dataset. We compare to the original nvdiffrec implementation using DMTet. Note that FlexiCubes offers more uniform tessellation, while being better capturing small details, as seen in the Lego scene.

Fig. 5. Qualitative textured mesh generation combining FlexiCubes with GET3D [Gao et al. 2022].

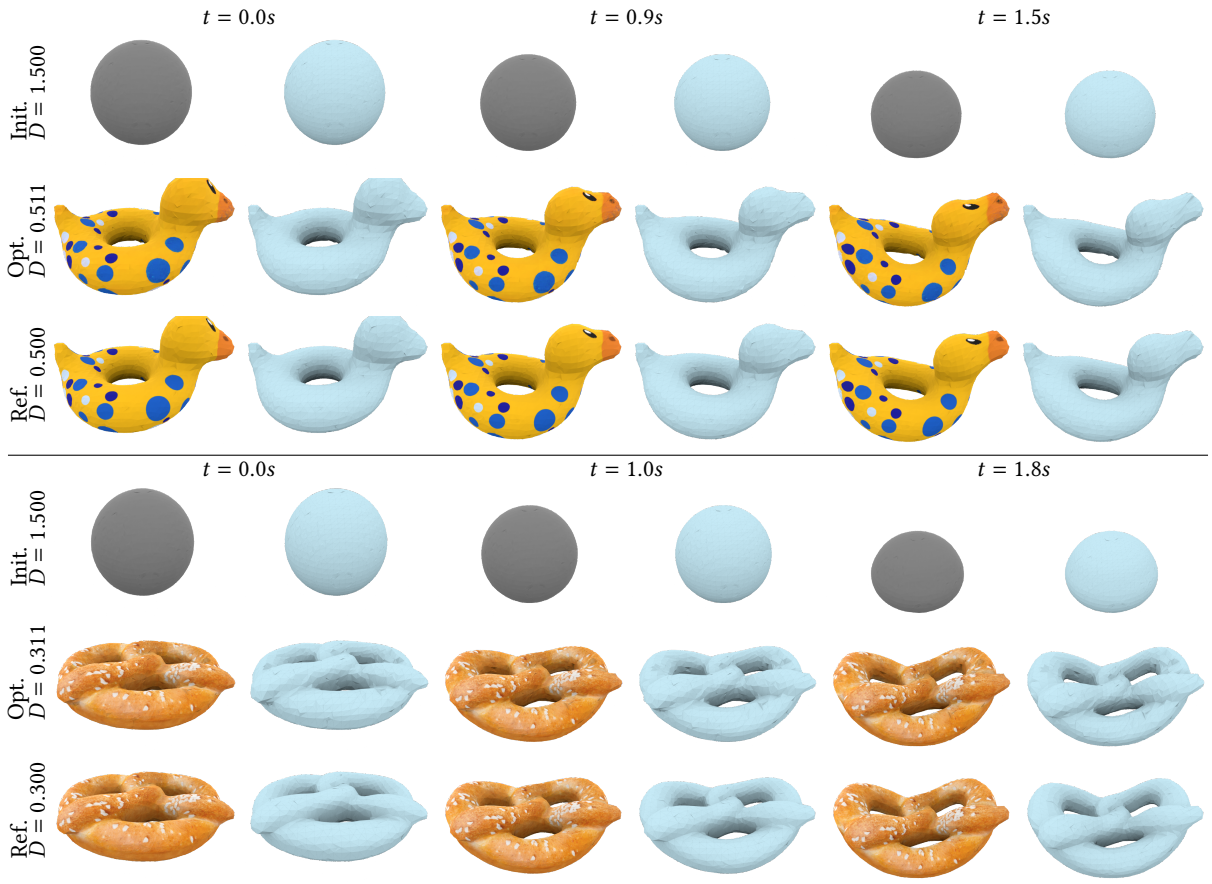


Fig. 6. Geometry details of the tetrahedral meshes in the physical simulation experiment. FlexiCubes generates tetrahedral meshes with well-defined topology which can be directly utilized in physical simulation. We further bridge it with differentiable physical simulation and differentiable rendering frameworks to optimize shape, texture, and physical parameters from multi-view videos. In the two examples, we optimize the mass density D of the object and get very close parameters after converging. We also show the wireframe of the deformed objects. To apply the extracted tetrahedral meshes in physical simulation, we delete tetrahedrons with tiny volume. This results in some spiky parts of the shape, *e.g.*, the shape has some black slits. We find without deleting small tetrahedrons leads to unstable simulation results. This problem can be potentially addressed by designing new regularization terms, which we leave for future work.