Flexible Isosurface Extraction for Gradient-Based Mesh Optimization

TIANCHANG SHEN, NVIDIA, University of Toronto, Vector Institute, Canada
JACOB MUNKBERG, NVIDIA, Sweden
JON HASSELGREN, NVIDIA, Sweden
KANGXUE YIN, NVIDIA, Canada
ZIAN WANG, NVIDIA, University of Toronto, Vector Institute, Canada
WENZHENG CHEN, NVIDIA, University of Toronto, Vector Institute, Canada
ZAN GOJCIC, NVIDIA, Switzerland
SANJA FIDLER, NVIDIA, University of Toronto, Vector Institute, Canada
NICHOLAS SHARP*, NVIDIA, USA
JUN GAO*, NVIDIA, University of Toronto, Vector Institute, Canada

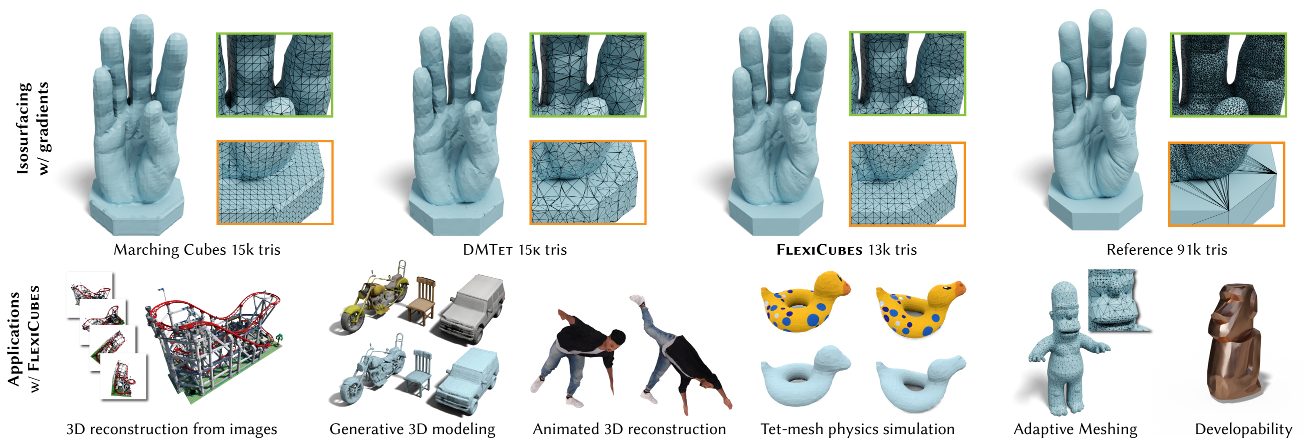

Fig. 1. We introduce FlexiCubes, a high quality isosurface representation specifically designed for gradient-based mesh optimization with respect to geometric, visual, or even physical objectives. We present a detailed quality evaluation and demonstrate that FlexiCubes improves the results in a range of applications.

This work considers gradient-based mesh optimization, where we iteratively optimize for a 3D surface mesh by representing it as the isosurface of a scalar field, an increasingly common paradigm in applications including photogrammetry, generative modeling, and inverse physics. Existing implementations adapt classic isosurface extraction algorithms like Marching Cubes or Dual Contouring; these techniques were designed to extract meshes from fixed, known fields, and in the optimization setting they lack the degrees of freedom to represent high-quality feature-preserving meshes, or suffer from numerical instabilities. We introduce FlexiCubes, an isosurface representation specifically designed for optimizing an unknown mesh with respect to geometric, visual, or even physical objectives. Our main insight is to introduce additional carefully-chosen parameters into the representation, which allow local *flexible* adjustments to the extracted mesh geometry and connectivity. These parameters are updated along with the underlying scalar field via automatic differentiation when optimizing for a downstream task. We base our extraction scheme on Dual Marching Cubes for improved topological properties, and present extensions to optionally generate tetrahedral and hierarchically-adaptive meshes. Extensive experiments validate FlexiCubes on both synthetic benchmarks and real-world applications, showing that it offers significant improvements in mesh quality and geometric fidelity.

CCS Concepts: • **Computing methodologies** → **Mesh geometry models**; **Shape representations**; **Reconstruction**.

Additional Key Words and Phrases: isosurface extraction, gradient-based mesh optimization, photogrammetry, generative models

ACM Reference Format:
Tianchang Shen, Jacob Munkberg, Jon Hasselgren, Kangxue Yin, Zian Wang, Wenzheng Chen, Zan Gojcic, Sanja Fidler, Nicholas Sharp, and Jun Gao. 2023. Flexible Isosurface Extraction for Gradient-Based Mesh Optimization. *ACM Trans. Graph.* 42, 4, Article 1 (August 2023), 16 pages. https://doi.org/10.1145/3592430

*Authors contributed equally.

Authors' addresses: Tianchang Shen, frshen@nvidia.com, NVIDIA, and University of Toronto, and Vector Institute, Canada; Jacob Munkberg, NVIDIA, Sweden; Jon Hasselgren, NVIDIA, Sweden; Kangxue Yin, NVIDIA, Canada; Zian Wang, NVIDIA, and University of Toronto, and Vector Institute, Canada; Wenzheng Chen, NVIDIA, and University of Toronto, and Vector Institute, Canada; Zan Gojcic, NVIDIA, Switzerland; Sanja Fidler, NVIDIA, and University of Toronto, and Vector Institute, Canada; Nicholas Sharp, NVIDIA, USA; Jun Gao, NVIDIA, and University of Toronto, and Vector Institute, Canada.

0730-0301/2023/8-ART1
https://doi.org/10.1145/3592430

1 INTRODUCTION

Surface meshes serve a ubiquitous role in the representation, transmission, and generation of 3D geometry across fields ranging from computer graphics to robotics. Among many other benefits, surface meshes offer concise yet accurate encodings of arbitrary surfaces, benefit from efficient hardware accelerated rendering, and support solving equations in physical simulation and geometry processing.

However, not all meshes are created equal—the properties above are often realized only on a *high quality* mesh. In fact, meshes which have an excessive number of elements, suffer from self-intersections and sliver elements, or poorly capture the underlying geometry, may be entirely unsuitable for downstream tasks. Generating a high-quality mesh of a particular shape is therefore very important, but far from trivial and often requires significant manual effort.

The recent explosion of algorithmic content creation and generative 3D modeling tools has led to increased demand for automatic mesh generation. Indeed, the task of producing a high-quality mesh, traditionally the domain of skilled technical artists and modelers, is increasingly tackled via automatic algorithmic pipelines. These are often based on differentiable mesh generation, i.e. parameterizing a space of 3D surface meshes and enabling their optimization for various objectives via gradient-based techniques. For example, applications such as inverse rendering [Hasselgren et al. 2022; Munkberg et al. 2022], structural optimization [Subedi et al. 2020], and generative 3D modeling [Gao et al. 2022; Lin et al. 2022] all leverage this basic building block. In a perfect world, such applications would simply perform naïve gradient descent with respect to some mesh representation to optimize their desired objectives. However, many obstacles have stood in the way of such a workflow, from the basic question of how to optimize over meshes of varying topology, to the lack of stability and robustness in existing formulations which lead to irreparably low-quality mesh outputs. In this work, we propose a new formulation that brings us closer towards this goal, significantly improving the ease and quality of differentiable mesh generation in a variety of downstream tasks.

Directly optimizing the vertex positions of a mesh easily falls victim to degeneracy and local minima unless very careful initialization, remeshing, and regularization are used [Liu et al. 2019; Nicolet et al. 2021; Wang et al. 2018]. As such, a common paradigm is to define and optimize a scalar field or a signed distance function (SDF) in space and then extract a triangle mesh approximating the level set of that function. The choice of scalar function representation and mesh extraction scheme greatly affects the performance of an overall optimization pipeline. A subtle but significant challenge of extracting a mesh from a scalar field is that the space of possible generated meshes may be restricted. As we will show later, the choice of the specific algorithm used to extract the triangle mesh directly dictates the properties of the generated shape.

Table 1. Taxonomy of isosurfacing methods. Grad means gradient-based based optimization is effective in practice, and Uniform means the resulting tessellations are generally uniform without sliver triangles.

Method	Grad.	Sharp Features	Uniform	Intersection-Free	2-Manifold
MC [Lorensen and Cline 1987]	✔	✖	✖	✔	✖
DC [Ju et al. 2002]	✖	✔	✖	✖	✖
NDC [Chen et al. 2022b]	✖	✔	✔	✔	✖
DMC [Nielson 2004] Centroid	✔	✖	✔	✔	✔
DMC [Schaefer et al. 2007] QEF	✖	✔	✔	✔	✔
Template Mesh [Liu et al. 2019]	✖	✖	✔	✖	✔
DMTet [Shen et al. 2021]	✔	✔	✖	✔	✔
FlexiCubes	✔	✔	✔	✖	✔

To capture these concerns, we identify two key properties that a mesh generation procedure should offer to enable easy, efficient, and high-quality optimization for downstream tasks:

(1) **Grad.** Differentiation with respect to the mesh is well-defined, and gradient-based optimization converges effectively in practice.
(2) **Flexible.** Mesh vertices can be individually and locally adjusted to fit surface features and find a high-quality mesh with a small number of elements.

However, these two properties are inherently in conflict. Increased flexibility provides more capacity to represent degenerate geometry and self-intersections, which hinder convergence in gradient-based optimization. As a result, existing techniques [Lorensen and Cline 1987; Remelli et al. 2020; Shen et al. 2021] usually neglect one of the two properties (Table 1). For example, the widely-used Marching Cubes procedure [Lorensen and Cline 1987] is not Flexible, because the vertices always lie along a fixed lattice and hence generated meshes can never align with non-axis-aligned sharp features (Figure 1). Generalized marching techniques can deform the underlying grid [Gao et al. 2020; Shen et al. 2021], but still do not allow the adjustment of individual vertices, leading to sliver elements and imperfect fits. On the other hand, Dual Contouring [Ju et al. 2002] is popular for its ability to capture sharp features, but lacks Grad.; the linear system used to position vertices leads to unstable and ineffective optimization. Section 2 and Table 1 categorize past work in detail.

In this work, we present a new technique called FlexiCubes, which satisfies both desired properties. Our insight is to adapt a particular Dual Marching Cubes formulation and introduce additional degrees of freedom to *flexibly* position each extracted vertex within its dual cell. We carefully constrain the formulation such that it still produces manifold and watertight meshes that are intersection-free in the vast majority of cases, enabling well-behaved differentiation (Grad.) with respect to the underlying mesh.

The most important property of this formulation is that gradient-based optimization of meshes succeeds consistently in practice. To assess this inherently empirical concern, we devote a significant part of this work to an extensive evaluation of FlexiCubes on several downstream tasks. Specifically, we demonstrate that our formulation offers significant benefits for various mesh generation applications, including inverse rendering, optimizing physical and geometric energies, and generative 3D modeling. The resulting meshes concisely capture the desired geometry at low element counts and are easily optimized via gradient descent. Moreover, we also propose extensions of FlexiCubes such as adaptively adjusting the resolution of the mesh via hierarchical refinement, and automatically tetrahedralizing the interior of the domain. Benchmarks and experiments show the value of this technique compared to past approaches, which we believe will serve as a valuable tool for high-quality mesh generation in many application areas.

2 RELATED WORK

In this section, we first provide a broad outline of related work before continuing with an in-depth analysis of the most relevant techniques in Section 3.

2.1 Isosurface Extraction

Traditional isosurfacing methods extract a polygonal mesh representing the level set of a scalar function, a problem that has been studied extensively across several fields. Here, we review particularly relevant work and refer the reader to the excellent survey of De Araújo et al. [2015] for a thorough overview. Following De Araújo et al. [2015] we divide isosurfacing methods into three categories and taxonomize the most commonly used ones in Table 1.

Spatial Decomposition. Methods in the first category obtain the isosurface through spatial decomposition, which divides the space into cells like cubes or tetrahedrons and creates polygons within the cells that contain the surface [Bloomenthal 1988; Bloomenthal et al. 1997]. Marching Cubes (MC) [Lorensen and Cline 1987] is the most representative method in this category. As originally presented, Marching Cubes suffers from topological ambiguities and struggles to represent sharp features. Subsequent work improves the look-up table which assigns polygon types to cubes [Chernyaev 1995; Hege et al. 1997; Lewiner et al. 2003; Montani et al. 1994; Nielson 2003; Scopigno 1994] or divides cubes into tetrahedra [Bloomenthal 1994] and uses the similar Marching Tetrahedra [Doi and Koide 1991] to extract the isosurface. To better capture sharp features, Dual Contouring (DC) [Ju et al. 2002] moved to a *dual* representation where mesh vertices are extracted per-cell, and proposed to estimate vertex position according to the local isosurface details. Dual Contouring was extended to adaptive meshing [Azernikov and Fischer 2005] and can output tetrahedral meshes. Another improved approach is Dual Marching Cubes (DMC) [Nielson 2004], which leverages the benefits from both Marching Cubes and Dual Contouring. Recently, Neural Marching Cubes [Chen and Zhang 2021] and Neural Dual Contouring (NDC) [Chen et al. 2022b] propose a data-driven approach to position the extracted mesh as a function of input field. Despite much progress in extraction from known scalar fields, applying isosurfacing methods to gradient-based mesh optimization remains challenging.

Surface Tracking. Methods in the second category utilize surface tracking and exploit the neighboring information between surface samples to extract the isosurface. Marching Triangles [Hilton et al. 1996, 1997], one of the first representative methods, iteratively triangulates the surface from an initial point under a Delaunay constraint. Following works aim to incorporate adaptivity [Akkouche and Galin 2001; Karkanis and Stewart 2001] or alignment to sharp features [McCormick and Fisher 2002]. However, gradient-based mesh optimization in the framework of surface tracking would require differentiating through the discrete, iterative update process, which is a non-trivial endeavor.

Shrink Wrapping. The methods from the third category rely on shrinking a spherical mesh [Van Overveld and Wyvill 2004], or inflating critical points [Stander and Hart 1995] to match the isosurface. By default, these methods apply only in limited topological cases and require manual selection of critical points [Bottino et al. 1996] to support arbitrary topology. Moreover, the differentiation through the shrinking process is also not straightforward and hence these methods are not well suited for gradient-based optimization.

2.2 Gradient-Based Mesh Optimization in ML

With recent advances in machine learning (ML), several works explore generating 3D meshes with neural networks, whose parameters are optimized via gradient-based optimization under some loss function. Early approaches seek to predefine the topology of the generated shape, such as a sphere [Chen et al. 2019; Hanocka et al. 2020; Kato et al. 2018; Wang et al. 2018], a union of primitives [Paschalidou et al. 2021; Tulsiani et al. 2017] or a set of segmented parts [Sung et al. 2017; Yin et al. 2020; Zhu et al. 2018]. However, they are limited in their ability to generalize to objects with complex topologies. To remedy this issue, AtlasNet [Groueix et al. 2018] represents a 3D shape as a collection of parametric surface elements, though it does not encode a coherent surface. Mesh R-CNN [Gkioxari et al. 2019] first predicts a coarse structure which is then refined to a surface mesh. Such a two-stage approach can generate meshes with different topologies, but since the second stage still relies on mesh deformation, topological errors from the first stage can not be rectified. PolyGen [Nash et al. 2020] autogressively generates mesh vertices and edges, but they are limited in requiring 3D ground truth data. CvxNet [Deng et al. 2019] and BSPNet [Chen et al. 2020] seek to use convex decomposition of the shape or binary planes for space partitioning, however extending them for various objectives defined on the meshes is non-trivial.

More recently, many works explore differentiable mesh reconstruction schemes, which extract an isosurface from an implicit function, often encoded via convolutional networks or implicit neural fields. Deep Marching Cubes [Liao et al. 2018] computes the expectation over possible topologies within a cube, which scales poorly with increasing grid resolution. MeshSDF [Remelli et al. 2020] proposes a specialized scheme for sampling gradients through mesh extraction, while Mehta et al. [2022] carefully formulates level set evolution in the neural context. DefTet [Gao et al. 2020] predicts a deformable tetrahedral grid to represent 3D objects. Most similar to our method is DMTet [Shen et al. 2021], which utilizes a differentiable Marching Tetrahedra layer to extract the mesh. An in-depth analysis of DMTet is provided in Section 3.

3 BACKGROUND AND MOTIVATION

Here, we first discuss common existing isosurface extraction schemes, to understand their shortcomings and motivate our proposed approach in Section 4.

Problem Statement. As outlined in Section 1, we seek a representation for differentiable mesh optimization, where the basic pipeline is to: i) define a scalar signed-distance function in space, ii) extract its 0-isosurface as a triangle mesh, iii) evaluate objective functions on that mesh, and iv) back-propagate gradients to the underlying scalar function. Several popular algorithms in widespread use for isosurface extraction still have significant issues in this differentiable setting. The main challenge is that the effectiveness of gradient-based optimization depends dramatically on the particular mechanism for isosurface extraction: restrictive parameterizations, numerically unstable expressions, and topological obstructions all lead to failures and artifacts when used in gradient-based optimization.

We emphasize that our FlexiCubes representation is *not* intended for isosurface extraction from fixed, known scalar fields, the primary case considered in past work. Instead, we particularly consider differentiable mesh optimization, where the underlying scalar field is an unknown and extraction is performed many times during gradient-based optimization. This setting offers new challenges and motivates a specialized approach.

Notation. All methods we consider extract an isosurface from a scalar function $s : \mathbb{R}^3 \rightarrow \mathbb{R}$, sampled at the vertices of a regular grid and interpolated within each cell. The function s may be discretized directly as values at grid vertices, or evaluated from an underlying neural network, *etc.*, the exact parameterization of s makes no difference for isosurface extraction. For clarity, the set X denotes the vertices of the grid with cells C, while $M = (V, F)$ denotes the resulting extracted mesh with vertices V and faces F. We implicitly overload $v \in V$ or $x \in X$ to refer to either a logical vertex, or that vertex's position in space *e.g.* $x \in \mathbb{R}^3$.

3.1 Marching Cubes & Tetrahedra

The most direct approach is to extract a mesh with vertices on the grid lattice, and one or more mesh faces within each grid cell, as in Marching Cubes [Lorensen and Cline 1987], Marching Tetrahedra [Doi and Koide 1991], and many generalizations. Mesh vertices are extracted along grid edges where the linearly-interpolated scalar function changes sign

$$u_e = \frac{x_a \cdot s(x_b) - x_b \cdot s(x_a)}{s(x_b) - s(x_a)}. \tag{1}$$

Liao et al. [2018]; Remelli et al. [2020] observe that this expression contains a singularity when $s(v_a) = s(v_b)$, which might obstruct differential optimization, although Shen et al. [2021] note that Equation 1 is never evaluated under the singular condition during extraction. The resulting mesh is always self-intersection-free and manifold.

However, the mesh vertices resulting from marching extraction can only lie along a sparse lattice of grid edges, by construction. This prevents the mesh from fitting to sharp features, and unavoidably creates poor-quality sliver triangles when the isosurface passes near a vertex. Recent methods propose schemes beyond naive auto-differentiation to compute improved gradients on the underlying scalar field [Mehta et al. 2022; Remelli et al. 2020], but this does not address the restricted output space for the mesh.

A promising remedy is to allow the underlying grid vertices to deform [Gao et al. 2020; Shen et al. 2021]. Although this generalization significantly improves performance, the extracted mesh vertices are still not able to move independently, leading to star-shaped skinny triangle artifacts as mesh vertices cluster around a degree of freedom on the grid. Our method takes inspiration from Shen et al. [2021] and also leverages grid deformation, but augments the representation with additional degrees of freedom to allow independent repositioning of the vertices, as shown in Figure 4.

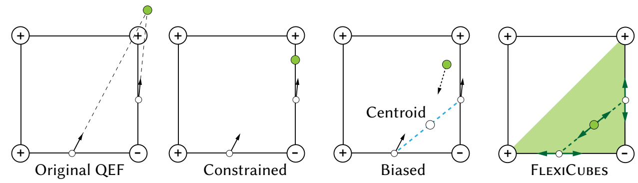

Fig. 2. **Grad. Issue in DC. Left:** When solving a quadratic error function (QEF) the resulting vertex is not guaranteed to be inside the cube. This leads to discrepancies between the geometry and the topological cases. In addition, there exists a singularity in QEF when the normals are coplanar. While there exist techniques to improve the stability of DC - constraining the solution space of QEF or biasing the QEF with regularization loss, they are not readily adaptable in optimization settings. The former (**Second**) zeros out the gradient in certain directions. The latter (**Third**) is hard to tune and having a strong regularization will downgrade the advantage of DC in flexibility. Our version (**Fourth**), FlexiCubes provides additional degrees of freedom, such that the dual vertex can be placed anywhere within the green triangle for this particular configuration.

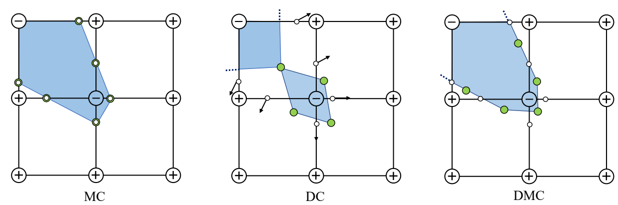

Fig. 3. **Comparison between MC, DC and DMC.** MC fails to capture sharp features. DC captures sharp features but can produce non-manifold vertices in some cases. DMC do not introduce non-manifold vertices, but has less freedom than DC in representing sharp features.

3.2 Dual Contouring

As the name suggests, Dual Contouring (DC) [Ju et al. 2002] moves to a *dual* representation, extracting mesh vertices that can be generally positioned within grid cells to better capture sharp geometric features. The position of each mesh vertex is computed by minimizing a local quadratic error function (QEF) depending on the local values and spatial gradients of the scalar function s

$$v_d = \underset{v_d}{\operatorname{argmin}} \sum_{u_e \in \mathcal{Z}_e} \nabla s(u_e) \cdot (v_d - u_e). \tag{2}$$

where $u_e \in \mathcal{Z}_e$ are the zero-crossings of the linearly-interpolated scalar function along the cell edges.

Dual Contouring excels at fitting sharp features when extracting a single mesh from a fixed scalar function, but several properties impede its use in differential optimization. Most importantly, Equation 2 does not guarantee that the extracted vertex lies inside the grid cell. In fact, co-planar gradient vectors $\nabla s(u_e)$ create degenerate configurations in which the vertex explodes to a distant location, leading to self-intersections and numerically unstable optimization when differentiating through the formulation. Explicitly constraining the vertex to lie in the cell zeros out the gradient, and regularizing Equation 2 enough to resolve the issue removes the ability to fit sharp features (Figure 2 & 4). Additionally, the resulting mesh

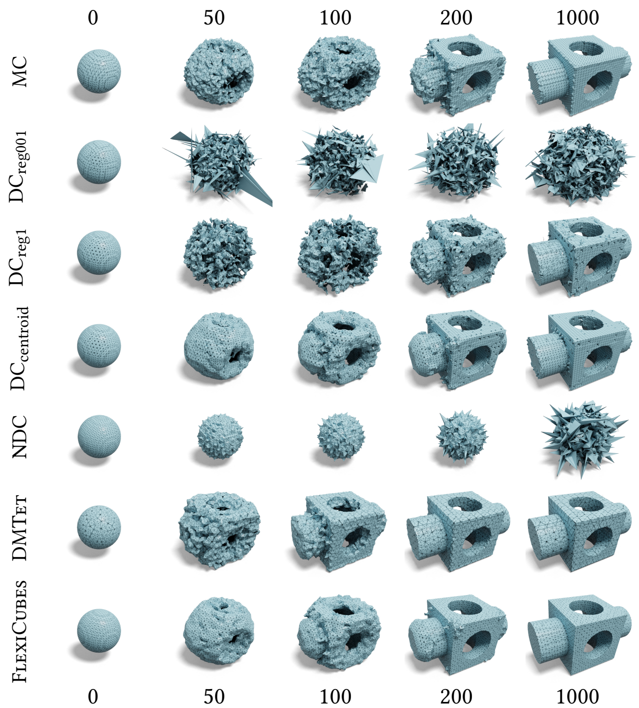

Fig. 4. **Comparison of different isosurfacing methods for mesh optimization.** Starting from a sphere initialization, we optimize the shapes towards the ground truth mesh using a set of isosurfacing methods. DMTet reconstructs sharp features but produces many sliver triangles. MC [Nielson 2003] fails to capture sharp features, NDC [Chen et al. 2022b] diverges during optimization, DC [Ju et al. 2002] converges with strong regularization, but suffer from artifacts, and FlexiCubes generates a high quality mesh with details retained. More details about this experiment are provided in the Supplement.

connectivity may be nonmanifold, and the output mesh contains non-planar quadrilaterals which introduce error as they are split in to triangles (Figure 3).

Recent generalizations [Chen et al. 2022b] of Dual Contouring replace Equation 2 with a learned neural network, improving extraction quality from imperfect but *fixed* scalar functions. However, when optimizing with respect to the underlying function, differentiating through an additional neural network further complicates the optimization landscape and impedes convergence (Figure 4).

Our approach takes inspiration from these methods and the importance of positioning each vertex freely within a cell. However, rather than explicitly positioning the extracted vertex as a function solely of a scalar field, we introduce additional carefully-chosen degrees of freedom which are optimized to locally adjust the vertex position. We are able to resolve manifoldness by instead basing our scheme on the similar but lesser-known *Dual Marching Cubes*.

3.3 Dual Marching Cubes

Much like Dual Contouring, Dual Marching Cubes [Nielson 2004] extracts vertices positioned within grid cells. However, rather than extracting a mesh along the dual connectivity of the *grid*, it extracts a mesh along the dual connectivity of the mesh that *would be extracted by Marching Cubes*. This allows for manifold mesh outputs for all configurations, by emitting multiple mesh vertices within a single grid cell when needed. The extracted vertex locations are defined either as the minimizer of a QEF akin to Dual Contouring [Schaefer et al. 2007], or as a geometric function of the primal mesh geometry [Nielson 2004], such as the face centroid.

In general, Dual Marching Cubes improves the connectivity of the extracted mesh *vs.* Dual Contouring, but if a QEF is used for vertex positioning, it suffers from many of the same drawbacks as Dual Contouring. If vertices are positioned at the centroids of the primal mesh, then the formulation lacks the freedom to fit individual sharp features. In the subsequent text, whenever we refer to Dual Marching Cubes we mean the centroid approach, unless otherwise clarified.

Our approach builds on Dual Marching Cube extraction, but we introduce additional parameters for positioning vertices which generalize the centroid approach. Basing our method off a scheme which can emit correct topology even in difficult configurations is one key to our success.

4 METHOD

We propose the FlexiCubes representation for differentiable mesh optimization. The core of the method is a scalar function on a grid, from which we extract a triangle mesh via Dual Marching Cubes. Our main contribution is to introduce three additional sets of parameters, carefully chosen to add flexibility to the mesh representation while retaining robustness and ease of optimization:

- **Interpolation weights** $\alpha \in \mathbb{R}^8_{>0}, \beta \in \mathbb{R}^{12}_{>0}$ per grid cell, to position dual vertices in space (Section 4.2).
- **Splitting weights** $\gamma \in \mathbb{R}_{>0}$ per grid cell, to control how quadrilaterals are split into triangles (Section 4.3).
- **Deformation vectors** $\delta \in \mathbb{R}^3$ per vertex of the underlying grid for spatial alignment, as in Shen et al. [2021] (Section 4.4).

These parameters are optimized along with the scalar function s via auto-differentiation to fit a mesh to the desired objective. We also present extensions of the FlexiCubes representation to extract a tetrahedral mesh of the volume (Section 4.5) and represent hierarchical meshes with adaptive resolution (Section 4.6).

4.1 Dual Marching Cubes Mesh Extraction

We begin by extracting the connectivity of the Dual Marching Cubes mesh based on the value of the scalar function $s(x)$ at each grid vertex x, just as in Nielson [2004]; Schaefer et al. [2007]. The signs of $s(x)$ at cube corners determine the connectivity and adjacency relationships (Figure 7). Unlike ordinary Marching Cubes, which extracts vertices along grid edges, Dual Marching Cubes extracts a vertex for each primal face in the cell; typically a single vertex, but possibly up to four (Figure 7, case C13). Extracted vertices in adjacent cells are linked by edges to form the dual mesh, composed of quadrilateral faces (Figure 5). The resulting mesh is guaranteed to be manifold, although due to the additional degrees of freedom described below, it may rarely contain self-intersections; see Section 7.2.

4.2 Flexible Dual Vertex Positioning

Our method generalizes ordinary Dual Marching Cubes in how the extracted mesh vertex locations are computed. Recall that Marching Cubes primal vertices are located at scalar zero-crossings along grid cell edges

$$u_e = \frac{x_a \cdot s(x_b) - x_b \cdot s(x_a)}{s(x_b) - s(x_a)}, \tag{3}$$

and ordinary Dual Marching Cubes then defines the location of each extracted vertex to be the centroid of its primal face

$$v_d = \frac{1}{|V_E|} \sum_{u_e \in V_E} u_e, \tag{4}$$

where V_E is the set of crossings which are the primal face vertices.

To introduce additional flexibility into this representation, we first define a set of weights in each grid cell $\alpha \in \mathbb{R}^8_{>0}$ associating a positive scalar with each cube corner. These weights adjust the location of the crossing point c_e along each edge, and Equation 3 then becomes

$$u_e = \frac{s(x_i)\alpha_i x_j - s(x_j)\alpha_j x_i}{s(x_i)\alpha_i - s(x_j)\alpha_j}. \tag{5}$$

In our implementation, we apply a $\tanh(\cdot) + 1$ activation function to restrict $\alpha \in [0, 2]$, and do not observe any convergence problems due to degeneracy.

Likewise, rather than naively positioning the dual vertex at the centroid of the primal face, we introduce a set of weights in each cell $\beta \in \mathbb{R}^{12}_{>0}$, associating a positive scalar with each cube edge. These weights adjust the location of the dual vertex inside each face, and Equation 4 then becomes

$$v_d = \frac{1}{\sum_{u_e \in V_E} \beta_e} \sum_{u_e \in V_E} \beta_e u_e. \tag{6}$$

In practice, we again apply a $\tanh(\cdot) + 1$ activation to restrict the range of β, similar to α.

Together these weights $\alpha \in \mathbb{R}^8_{>0}, \beta \in \mathbb{R}^{12}_{>0}$ amount to 20 scalars per grid cell. In both cases, weights are defined independently per cell, *not* shared at adjacent corners or edges; independent weights offer more flexibility, and there is no continuity condition to maintain at adjacent elements in our dual setting.

Notice that both Equation 5 & 6 are intentionally parameterized as convex combinations, and thus the resulting extracted vertex position is necessarily within the convex hull of its grid cell vertices. Furthermore, when a convex cell emits multiple dual vertices (Figure 7), the corresponding primal faces in which the dual vertices are positioned are non-intersecting, which prevents nearly all self-intersections in the resulting mesh (see Section 7 and Supplement).

4.3 Flexible Quad Splitting

Dual Marching Cubes, and thus also FlexiCubes, extracts pure quadrilateral meshes with non-planar faces, which are typically split to triangles for processing in downstream applications. Simply splitting along an arbitrary diagonal can lead to significant artifacts in curved regions (Figure 8), and there is in general no single ideal policy to split non-planar quads to represent unknown geometry. Our next parameter is introduced to make the choice of split flexible, and optimize it as a continuous degree of freedom.

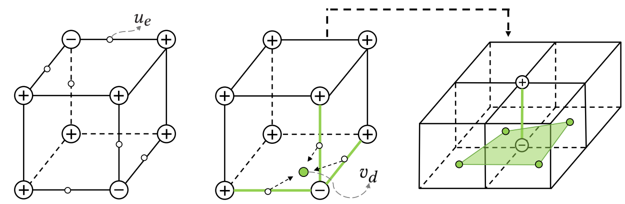

Fig. 5. Dual Marching Cubes first interpolate vertex along the edge to obtain u_e. The dual vertex v_d is computed via Equation 4. We connect four neighboring dual vertices to obtain a quadrilateral.

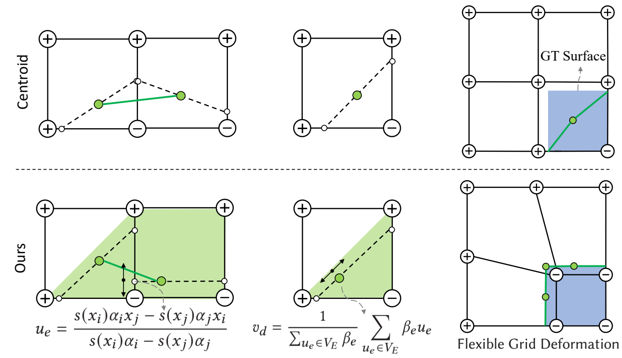

Fig. 6. Formulation of determining the position of dual vertex. The dual vertex in our formulation can be placed anywhere within the green region.

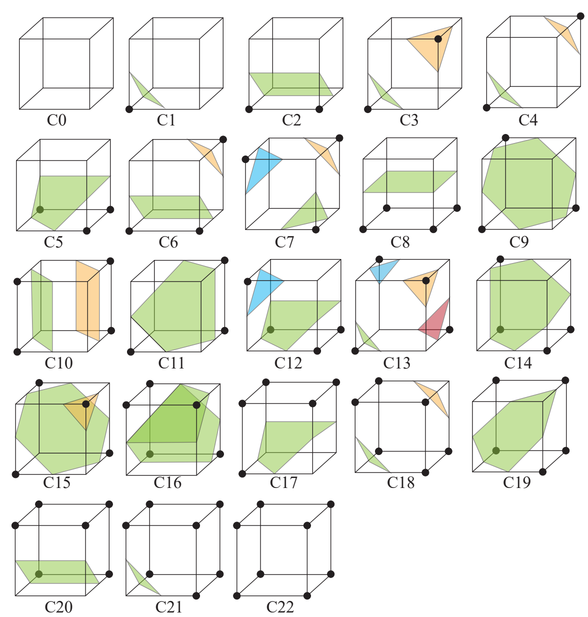

Fig. 7. All configurations for the Dual Marching Cubes surface, with rotation symmetric cases removed. Each colored polygon is a primal face, in which a single dual vertex is extracted as output. Marked vertices indicate negative signed distance values, $s(x) < 0$, and unmarked vertices indicate positive values. Figure adapted from Nielson [2004].

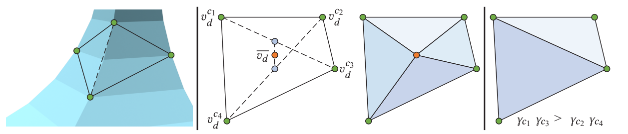

Fig. 8. **Left**: FlexiCubes generates non-planar quadrangles, which may need to be triangulated e.g. for rendering. Note that in this example, there is a clearly favourable triangulation, naturally aligning with the curvature of the reference surface. **Middle**: During optimization we use a differentiable strategy to select triangulation by tessellating each quad into four triangles with an interpolated midpoint. **Right**: During inference we tessellate each quad into two triangles along the dominant diagonal.

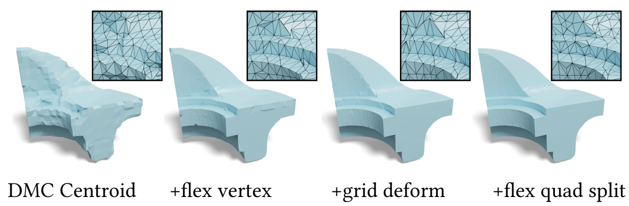

Fig. 9. Ablating the effect of parameters in FlexiCubes. Adding flexible dual vertex positioning, the mesh edges align much better with the sharp geometric features, which are further improved with grid deformation. With flexible quad splitting, FlexiCubes split the quadrangles along the diagonals that align with the features.

We define a weight $\gamma \in \mathbb{R}_{>0}$ in each grid cell, which is propagated to the emitted vertices in the extracted mesh. At optimization-time *only*, each quadrilateral mesh face is split into 4 triangles by inserting a midpoint vertex $\overline{v_d}$ (Figure 8). The location of this midpoint is computed as

$$\overline{v_d} = \frac{\gamma_{c_1}\gamma_{c_3}(v_d^{c_1} + v_d^{c_3})/2 + \gamma_{c_2}\gamma_{c_4}(v_d^{c_2} + v_d^{c_4})/2}{\gamma_{c_1}\gamma_{c_3} + \gamma_{c_2}\gamma_{c_4}} \tag{7}$$

with notation is as in Figure 8. This is a weighted combination of the midpoints of the two possible diagonals of the face, where the weights come from the γ parameters on the corresponding vertices. Intuitively, adjusting the γ weights smoothly interpolates between the geometries resulting from the two possible splits. Optimizing γ encourages the choice of split which fits the objective of interest. For final extraction when optimization is complete, we do *not* insert the midpoint vertex $\overline{v_d}$, but simply split each quadrilateral along whichever diagonal has larger product of γ values.

4.4 Flexible Grid Deformation

Inspired by DefTet [Gao et al. 2020] and DMTet [Shen et al. 2021], we furthermore allow the vertices of the underlying grid to deform according to displacements $\delta \in \mathbb{R}^3$ at each grid vertex. These deformations allow the grid to locally align with thin features, and give additional flexibility in positioning vertices. We limit the deformation to at most half of the grid spacing to ensure that grid cells never invert.

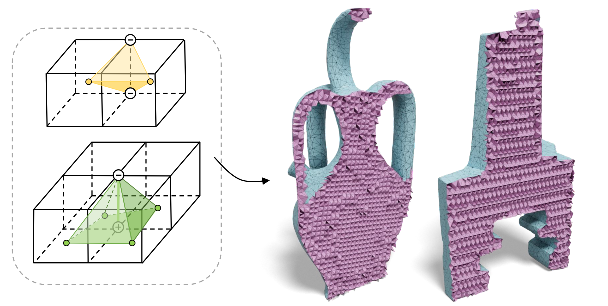

Fig. 10. We extend FlexiCubes to generate a tetrahedral mesh by dividing the interior volume illustrated *left*. Example outputs are shown *right*.

4.5 Tetrahedral Mesh Extraction

Many applications such as physical simulation and character animation require a tetrahedralization of the shape volume. We augment FlexiCubes to additionally output a tetrahedral mesh when desired, which exactly conforms to the boundary of the extracted surface and supports automatic differentiation in the same sense as our surface extraction.

Our approach adapts the strategy proposed by Liang and Zhang [2014] for Dual Contouring. The vertex set for the tetrahedral mesh is the union of the grid vertices, our extracted mesh vertices in cells, and the midpoint of any cell for which no surface vertex was extracted. We then emit tetrahedra as shown in Figure 10, *left*. For each grid edge connecting two grid vertices with the same sign, four tetrahedra are generated, each formed by the two grid vertices and two vertices in consecutive adjacent cells. For each grid edge connecting two grid vertices with different signs, two four-sided pyramids are generated, each formed by one grid vertex and a vertex from each adjacent cell. These pyramids are then split at the base as in Section 4.3 to yield two tetrahedra each. When working with Dual Marching Cubes connectivity, there is an additional complexity that a cell may contain multiple extracted mesh vertices, and we must choose the correct vertex when forming tetrahedra. In most cases, this choice can be read-off unambiguously from Figure 7; although rare difficult deformed configurations lead to small mesh defects–we detail these in the Supplement, and find that they do not obstruct downstream applications. The resulting meshes are visualized in Figure 10, *right*, and Figure 24 demonstrates an application of differentiable physical simulation.

4.6 Adaptive Mesh Resolution

We also augment FlexiCubes to leverage adapative hierarchical grids, and represent meshes which variably increased spatial resolution in areas of high geometric detail. The policy of where to refine the octree grid representation is application-specific, *e.g.* thresholds on local curvature in geometric fitting or visual error in inverse rendering; our representation is responsible for extracting hierarchically adaptive meshes while maintaining the key properties of flexibility and effective gradient-based optimization. Here we again mimic approaches designed for Dual Contouring [Ju et al. 2002; Schaefer et al. 2007], adapting them to our FlexiCubes extension of Dual Marching Cubes.

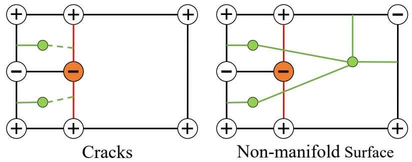

Fig. 11. If the sign of the SDF is inconsistent on the face shared by cubes at different levels (highlighted in red), it leads to cracks or non-manifold surfaces. Two such cases are shown here. As such, we restrict the SDF of these vertices (in orange) to be consistent with the face of the larger cube.

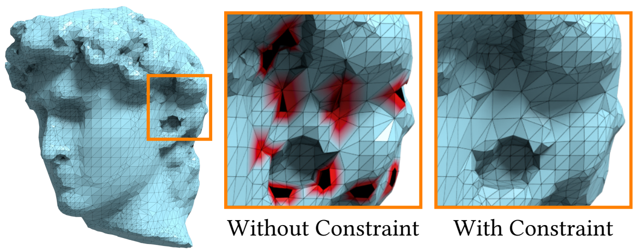

Fig. 12. Cracks and non-manifold vertices produced by running FlexiCubes with an octree representation can be largely resolved by enforcing the constraint on SDF of minimal vertices. For the unconstrained version there are 318 non-manifold vertices in the resulting mesh, and with constraints there is only a single non-manifold vertex remaining.

The approach is to locally refine our background grid into a hierarchical octree with varying resolution. Most of our algorithm applies unchanged on an octree, except for the challenge of connecting adjacent dual vertices to form quadrilateral meshes faces when they span different levels of the octree. On a general octree there may not exist any dual face connectivity which yields a closed manifold mesh (Figure 11); existing methods mitigate this problem by constraining the topology of the octree or signs of the implicit function [Ju et al. 2002; Schaefer et al. 2007]. However, these rules are not applicable in an optimization setting, where the topology is unknown and constantly changing. We adopt the approach shown in Figure 12; refined octree grid vertices adjacent to coarser cells always take their value as the interpolated value from coarse face vertices, guaranteeing consistency of signs. This projection yields nearly watertight adaptive meshes in our experiments. Here again the combination of all possible configurations from Figure 7 at adjacent octree nodes of different hierarchies leaves a small number of cases where the extracted mesh contains a hole. Nonetheless, the adaptively refined mesh yields significant improvements, as shown in Figure 14.

4.7 Regularizers

Our method is a general-purpose tool, which can be optimized according to application-specific objectives and regularizers, including geometric depth and SDF losses, image-space rendering losses, and mesh-quality regularizers. In the next section, we will detail several examples utilizing such terms. Here, we first propose two regularization terms which are specific to the internals of our representation.

Our over-parameterization of the location of each vertex, described in Section 4, is intentional and beneficial, allowing for properties such as the convex weighting in Section 4.2, and the bounded grid deformation in Section 4.4, as well as easing stochastic optimization. As such, we introduce two terms to regularize the internal representation, and encourage non-degenerate parameters which can easily "flex" to accommodate any local vertex movement. These regularizers are used for all examples shown in this work.

The first term penalizes the deviation of the distances between each dual vertex and the edge crossings which compose the face in which it sits

$$\mathcal{L}_{\text{dev}} := \sum_{v \in V} \text{MAD}\big[\{|v - u_e|_2 : u_e \in \mathcal{N}_v\}\big], \tag{8}$$

where $|\cdot|_2$ is Euclidean distance, MAD denotes the *mean absolute deviation* $\text{MAD}(Y) = \frac{1}{|Y|}\sum_{y \in Y} |y - \text{mean}(Y)|$, and $u_e \in \mathcal{N}_v$ are the edge crossings which bound the primal face for dual vertex v. This term regularizes the extracted connectivity, and encourages vertices to lie near the center of their cell so they have a margin in which to flex and adapt.

The second term discourages spurious geometry in regions of the shape which receive no supervision in the application objective, such as internal cavities. We follow Munkberg et al. [2022] and penalize sign changes of the implicit function on all grid edges. First, we let $\vec{\mathcal{E}}_g$ be the set of all pairs of scalar function values (s_a, s_b) at grid vertices (a, b) connected by an edge and with $sign(s_a) \neq sign(s_b)$. Then the loss is given by

$$\mathcal{L}_{\text{sign}} := \sum_{(s_a, s_b) \in \vec{\mathcal{E}}_g} H\big(\sigma(s_a), \text{sign}(s_b)\big)\big), \tag{9}$$

where H, σ are cross-entropy and sigmoid functions respectively.

5 EXPERIMENTS

In this section, we evaluate FlexiCubes in various mesh optimization tasks. First, we analyze the capacity of FlexiCubes in reconstructing 3D geometry under perfect 3D supervision defined on the surface and compare with other iso-surfacing techniques in Section 5.1. Next, we show that benefiting from differentiably extracting an explicit mesh, FlexiCubes can further optimize for various mesh-based regularization losses to improve the mesh quality for downstream applications.

5.1 Mesh Reconstruction

Motivation and Experimental settings. To evaluate the performance of optimizing 3D meshes using isosurfacing methods and avoid the inefficiency that could be introduced by imperfect objective functions, we experiment in an ideal setting where we define the objective functions directly on the geometric difference between the extracted mesh and a ground truth mesh. More specifically, in each iteration we reconstruct a mesh, render depth and silhouette images from a randomly sampled camera pose and compute the differences with a ground truth depth and silhouette images. 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

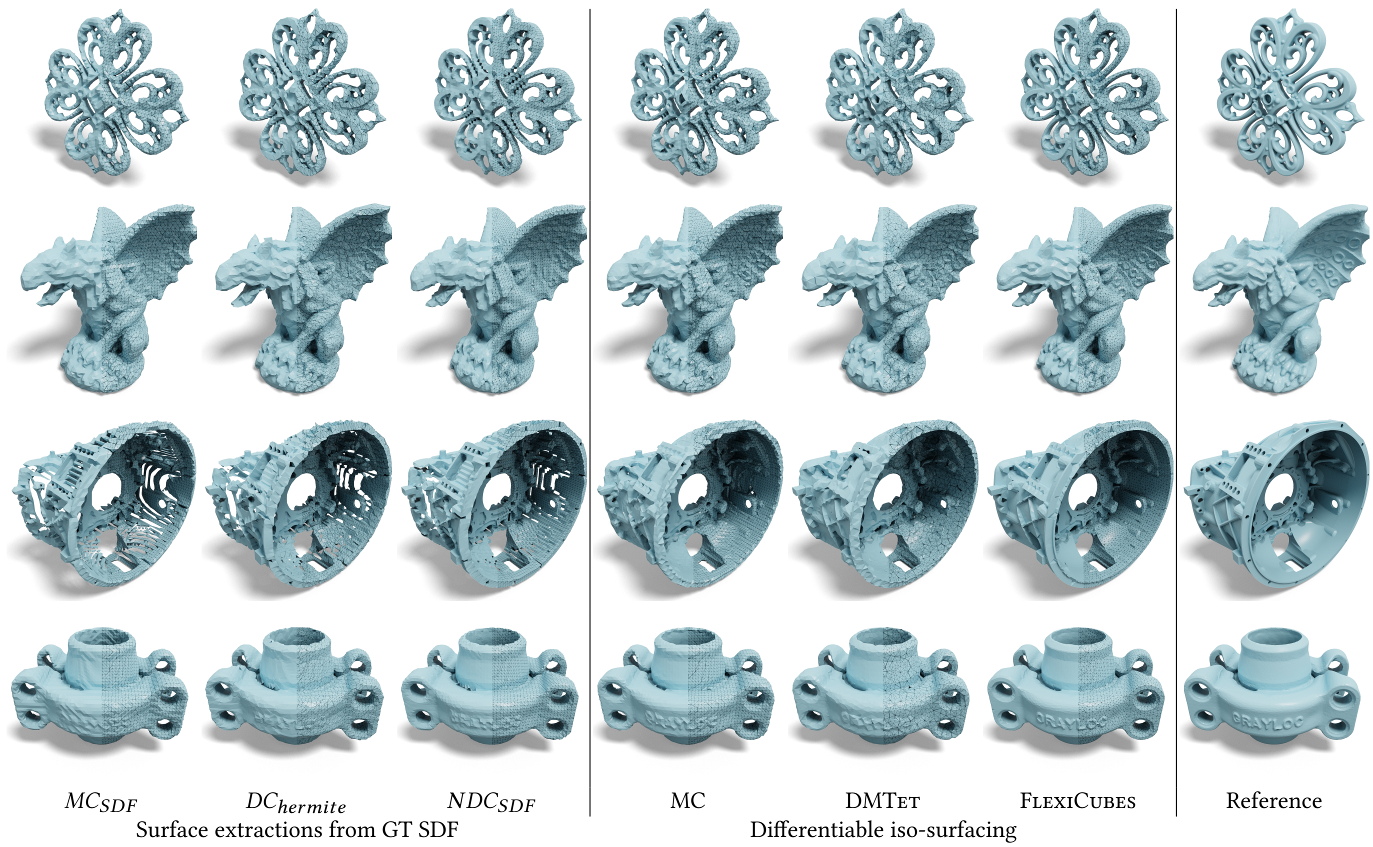

Fig. 13. Visual comparison of a set of iso-surfacing techniques. The three leftmost examples: Marching Cubes (MC_{SDF}), Dual Contouring, Neural Dual Contouring, use surface extraction from the ground truth SDF. The next three examples: Marching Cubes, Deep Marching Tetrahedra, and FlexiCubes, use differentiable 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.

values. Please refer to the Supplement for details of the objective functions and their weighting factors.

Dataset. We use the dataset collected by Myles et al. [2014], which contains 3D shapes from the AIM@Shape database and popular assets from other community repositories. This shape collection has a great diversity in geometric features and topology complexities, ranging from noisy scanned surfaces to highly-detailed CAD models. Following Chen and Zhang [2021], we remove the non-watertight and very skinny (e.g. wires) shapes, which are not suitable for isosurfacing methods to reconstruct. In total, we use 79 different shapes in our evaluation.

Baselines. As shown in Figure 13, we compare FlexiCubes with different methods split into two categories. FlexiCubes is grouped with the *differentiable* isosurfacing algorithms, MC and DMTet, which provides the most direct comparisons. We reconstruct meshes through optimization with objective functions mentioned above. Note that the resolution of tetrahedral grid used by DMTet is not directly comparable with voxel grids used by our method, as the number of vertices are different under the same resolution. Thus, we additionally report DMTet at different resolution to match the triangle counts. The resolution of DMTet is specified in the brackets.

In the other category we group the *non-differentiable* isosurfacing methods. To ensure a fair comparison, we use the *ground truth* SDF field and extract the mesh using vanilla MC (MC_{SDF}), DC ($DC_{hermite}$) and NDC (NDC_{SDF}). For DC we complement the ground truth SDF with normal vectors computed using finite differences, and for NDC, we use a pretrained model provided by the authors[1].

Evaluation Metrics. We evaluate the reconstructed meshes in terms of reconstruction accuracy and intrinsic quality of the reconstructed mesh. For the former, we follow NDC [Chen et al. 2022b] and compute Chamfer Distance (CD), F-Score (F1), Edge Chamfer Distance (EDC), Edge F-score (EF1), and the percentage of Inaccurate Normals (IN> 5°) w.r.t to the ground truth mesh. For the latter, we compute triangle aspect ratios, radius ratios, and min and max angles. A detailed description of the evaluation metrics is provided in the Supplement.

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

Table 2. Quantitative results on Mesh Reconstruction. We report the following metrics: IN> 5°: normal angle difference > 5°, CD: Chamfer Distance, F1: F1 score, ECD: Edge Chamfer Distance, EF1: Edge F1 Score. #V: number of vertices, #F: number of faces.

32^3	IN>5°(%)↓	CD(10^{-5}) ↓	F1 ↑	ECD (10^{-2}) ↓	EF1 ↑	#V	#T
MC_{SDF}	85.60	22.65	0.28	5.56	0.08	2387	4771
$DC_{hermite}$	74.43	17.15	0.38	4.82	0.11	2360	4775
NDC_{SDF}	72.60	17.61	0.42	3.55	0.13	1877	3801
MC	66.61	9.11	0.54	2.60	0.13	2573	5146
DMTet(32)	66.22	11.56	0.52	3.64	0.17	1691	3387
DMTet(40)	61.21	8.35	0.58	3.64	0.20	2626	5259
FlexiCubes	**50.52**	**7.01**	**0.64**	**2.11**	**0.26**	2400	4800
64^3	IN>5°(%)↓	CD(10^{-5}) ↓	F1 ↑	ECD (10^{-2}) ↓	EF1 ↑	#V	#T
MC_{SDF}	80.67	6.84	0.55	2.55	0.14	9881	19783
$DC_{hermite}$	63.34	5.90	0.61	3.80	0.23	9828	19769
NDC_{SDF}	55.22	6.16	0.57	1.22	0.26	9828	19769
MC	52.37	6.33	0.66	1.25	0.25	10459	20801
DMTet(64)	50.20	7.50	0.66	3.77	0.28	6783	13566
DMTet(80)	48.66	5.17	0.66	3.59	0.29	10385	20784
FlexiCubes	**34.87**	**4.87**	**0.70**	**0.71**	**0.43**	9916	19843
128^3	IN>5°(%)↓	CD(10^{-5}) ↓	F1 ↑	ECD (10^{-2}) ↓	EF1 ↑	#V	#T
MC_{SDF}	77.23	4.72	0.68	1.13	0.33	40164	80374
$DC_{hermite}$	53.98	4.59	0.69	3.82	0.40	40128	80360
NDC_{SDF}	43.20	5.04	0.65	0.79	0.43	40129	80361
MC	42.56	4.51	0.72	1.32	0.44	38645	77212
DMTet(128)	48.86	4.98	0.74	1.50	0.39	23535	47001
FlexiCubes	**30.57**	**4.31**	**0.71**	**0.42**	**0.51**	38923	77845

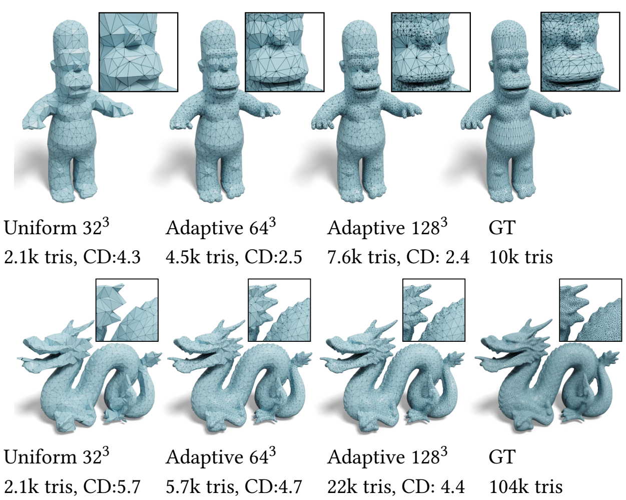

Fig. 14. Adaptive meshing with FlexiCubes optimizing mesh topology and the octree structure jointly. CD denotes Chamfer distance (10^{-5}).

.

Table 3. Quantitative results on shape reconstruction ablating different formulations of the dual vertex.

64^3	IN>5°(%) ↓	CD(10^{-5}) ↓	F1 ↑	ECD (10^{-2}) ↓	EF1 ↑	#V	#T
$DMC_{centroid}$[Nielson 2004]	53.02	5.85	0.65	2.60	0.19	10240	20488
+ flex vertex (Section 4.2)	40.88	5.34	0.68	0.99	0.37	10022	20055
+ grid deform (Section 4.4)	39.46	5.01	0.69	0.98	0.41	9879	19766
+ flex quad split (Section 4.3)	**34.87**	**4.87**	**0.70**	**0.71**	**0.43**	9916	19843

Results. The quantitative results of reconstruction quality are provided in Table 2 with qualitative examples depicted in Figure 13. Figure 15 shows the quantitative results of intrinsic mesh quality.

Table 4. Quantitative results on mesh reconstruction with equilateral triangle regularizer. Adding regularizer for DMTet and MC significantly impacts geometric metrics (IN>5°(%), CD), while FlexiCubes only sacrifices a bit.

64^3	IN>5°(%) ↓	CD(10^{-5}) ↓	Aspect Ratio > 4 (%) ↓	Radius Ratio > 4 (%) ↓	Min Angle < 10 (%) ↓
MC	52.37	6.33	11.71	11.71	11.84
DMTet(80)	48.66	5.17	17.31	16.68	17.83
FlexiCubes	34.87	4.87	2.93	4.49	2.04
MC + Reg.	50.16	8.56	11.46	11.43	11.62
DMTet(80) + Reg.	67.65	6.69	0.29	0.46	0.26
FlexiCubes + Reg.	41.05	5.46	0.39	0.69	0.24

Methods that extract the mesh as a post-processing step fail to achieve competitive performance in terms of reconstruction quality, highlighting the importance of end-to-end optimization that mitigates the discretization errors introduced in post-processing. When compared with other methods that use differentiable iso-surfacing for mesh reconstruction (MC and DMTet), FlexiCubes extracts meshes that align significantly better with ground truth geometry, while maintaining superior mesh quality which is on par with the best performing NDC method.

We further ablate each component we introduced in FlexiCubes, and provide quantitative results in Table 3 with qualitative examples in Figure 9. In the Supplement we also include reconstructions of the same object under different rotations.

5.2 Mesh Optimization with Regularizations

Our FlexiCubes representation is flexible enough that objectives and regularizers which depend on the extracted mesh itself can be directly evaluated with automatic differentiation and incorporated into gradient-based optimization. Some surface-based regularizers such as surface area may be easily expressed directly as functions of the underlying scalar field, while others, especially those which depend on the mesh discretization itself, have no direct equivalent. This same simple strategy does not succeed with more rigid representations like Marching Cubes, because the extracted mesh does not have the degrees of freedom to adapt to arbitrary objectives. We provide two examples of mesh regularizations below.

Equilateral Edge Length. In many applications, such as physics simulation, generating equilateral triangles is preferable over thin triangles. We penalize the variance of the edge lengths on the extracted mesh in this regularization. In particular, we compute the average edge length $\bar{e} = \frac{1}{|\mathcal{E}|}\sum_{e\in\mathcal{E}}|e|_2$, where $\mathcal{E}$ denotes the set of all the edges in the extracted mesh. The regularization is computed as: $R_{\text{edge}} = \frac{1}{|T|}\sum_{e\in t, t\in T}(|e-\bar{e}|^2)$, where T is the set of extracted triangles. We combine this regularization with the reconstruction loss mentioned in Section 5.1. We first run the optimization to reconstruct an input mesh without the regularization term for 1000 steps, then we further run 300 steps using both the reconstruction loss and the regularization loss, with the regularization weight progressively increasing from 0 to 100. Adding equilateral triangle regularization allows FlexiCubes to generate more uniform triangles with a slight degradation in the reconstruction quality. We compare FlexiCubes with MC and DMTet, and provide qualitative results in Figure 16. The quantitative comparison in Table 4 shows that both our method and DMTet can gain a significant improvement in triangle quality after adding the regularization, as measured by percentages of triangles having Aspect Ratio > 4, Radius Ratio > 4, or Min Angle < 10.

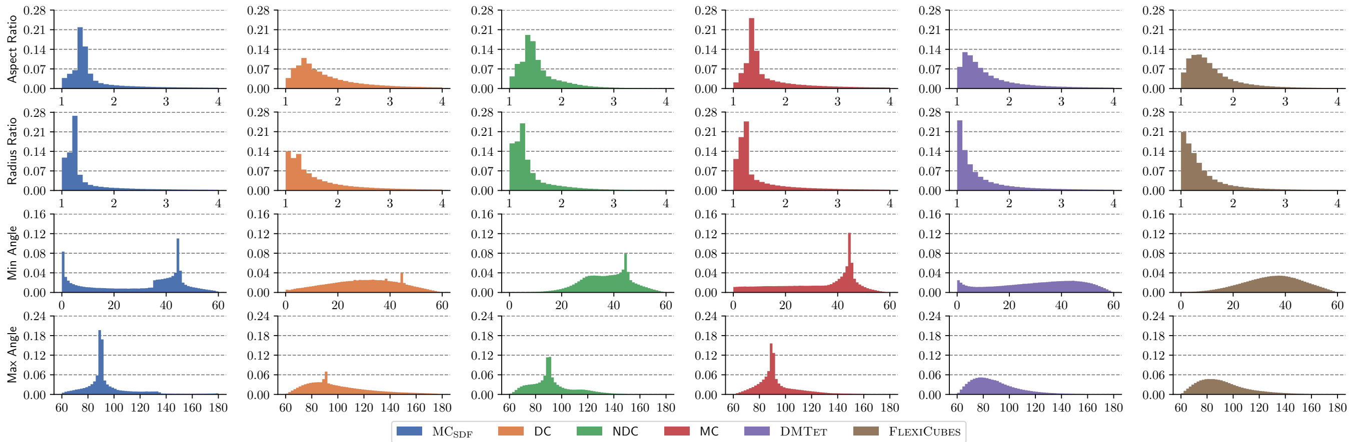

Fig. 15. Quantitative comparison of the intrinsic quality of extracted meshes. FlexiCubes extracts high-quality meshes with a significantly smaller amount of sliver triangles (row 3) and more equilateral sides (row 1). The smooth distributions of Min and Max Angle (rows 3 and 4) show that FlexiCubes can adjust the triangles to better fit the geometry in slightly non-planar regions.

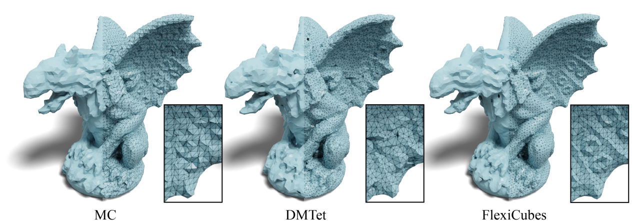

Fig. 16. Adding equilateral edge regularization to enhance triangle quality. Comparing with results in Figure 13, which does not have equilateral edge regularization loss, FlexiCubes only has slight degradation of the reconstruction quality, while MC and DMTet lose details in the local geometry. Please zoom in to see the mesh details.

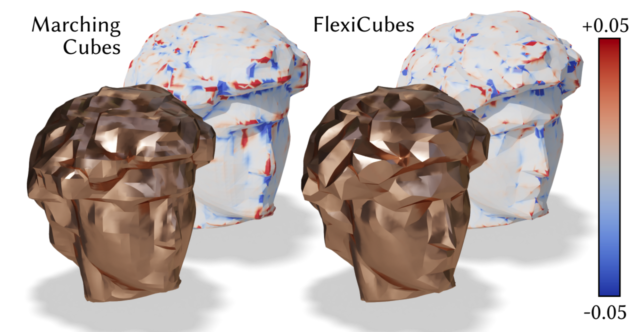

Fig. 17. To demonstrate mesh-based regularizers, we combine synthetic photogrammetry with a *developability* term [Stein et al. 2018], which encourages fabricability from panels. FlexiCubes has the freedom to fit the shape and satisfy the regularizer, whereas Marching Cubes does not. *Top:* Gaussian curvature in radians, *Bottom:* stylized render to show panelization.

However, our method has a significantly smaller drop in geometric quality, as measured by the first two metrics in Table 4, thanks to the flexibility of our surface extraction formulation.

Developability. As a more complex mesh-based term, we consider the *developability* energy of Stein et al. [2018, Equation 4], which amounts to penalizing the smallest eigenvalue of the covariance matrix of face normals about each vertex. Developability is a geometric measure penalizes stretching of the surface relative to a flat sheet, but does not penalize bending in a single direction; it has applications to manufacturing from sheets of material like sheet metal or plywood. Although developability could in-principle be quantified directly on an implicit function, it has a significant relationship to discrete mesh connectivity, as discussed by Stein et al. [2018]. Figure 17 shows the result of incorporating this term into a synthetic reconstruction problem. Attempting to do the same with Marching Cubes is much less successful, failing to preserve shape features and achieve the desired style.

6 APPLICATIONS

6.1 Photogrammetry Through Differentiable Rendering

The differentiable isosurfacing technique DMTet [2021] is at the core of the recent work, nvdiffrec, which jointly optimizes shape, materials, and lighting from images [Hasselgren et al. 2022; Munkberg et al. 2022]. By simply replacing DMTet with FlexiCubes in the topology optimization step, leaving the remainder of the pipeline unmodified, we observe improved geometry reconstructions at equal triangle count, which is illustrated in Figure 20. We also report nvdiffrec result with DMTet vs. FlexiCubes on the NeRF synthetic dataset [Mildenhall et al. 2020]. View interpolation scores and Chamfer distances are shown in Table 5. We show additional results on datasets of real-world photographs in Figure 19. In general, FlexiCubes produces fewer sliver triangles as can be observed in the visual examples (Figure 18) and the min angle histogram. Additionally, the nicer triangulation of FlexiCubes leads to easier creation

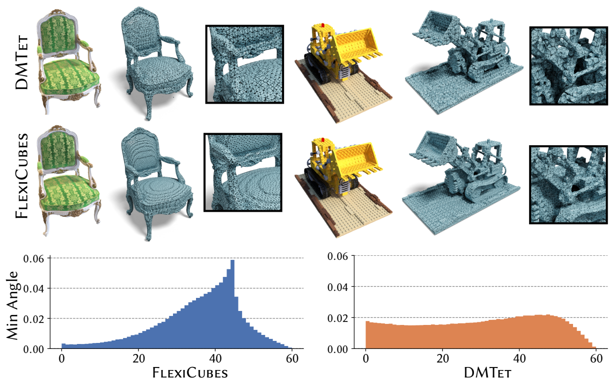

Fig. 18. Visualization of nvdiffrec reconstructions for two scenes in the NeRF synthetic dataset. We compare DMTet and FlexiCubes for the topology extraction step. We note fewer sliver triangles for FlexiCubes. We illustate this by including min angle histogram for nvdiffrec reconstructions for all eight scenes in the NeRF synthetic dataset. Fewer triangles with small angles means less sliver triangles for FlexiCubes.

Table 5. View interpolation results (PSNR) for nvdiffrec reconstructions of the NeRF synthetic dataset, using either DMTet or FlexiCubes for the topology step. The image metric scores are arithmetic means over all test images. We also include Chamfer distances (CD) computed on visible triangles (the set of triangles visible in at least one test view) using 2.5 M point. Lower scores indicate better geometric fidelity.

PSNR (dB) ↑	Chair	Drums	Ficus	Hotdog	Lego	Mats	Mic	Ship
DMTet	31.8	24.6	30.9	33.2	29.0	27.0	30.7	26.0
FlexiCubes	31.8	24.7	30.9	33.4	28.8	26.7	30.8	25.9
CD (10^{-2}) ↓	Chair	Drums	Ficus	Hotdog	Lego	Mats	Mic	Ship
DMTet	4.51	3.98	0.30	2.67	2.41	0.41	1.20	55.8
FlexiCubes	0.45	2.27	0.37	1.44	1.60	0.53	1.51	10.5

of unique texture coordinates (UV unwrapping) and improved UV layouts when running the extracted meshes through an off-the-shelf unwrapping tool [Young 2021]. Figure 21 illustrates this property.

6.2 Mesh Simplification of Animated Objects

We show the benefit of mesh optimization using the explicit mesh representation from FlexiCubes in an animated meshing task, illustrated in Figure 22. Given a known animated skeleton and images of the target animated object, we leverage nvdiffrec to extract a concise mesh which accurately represents the object throughout the animation. Here, we use an animation sequence from RenderPeople [2020].

Rather than fitting a single mesh in a reference pose, FlexiCubes allows us to differentiably skin and deform the mesh via off-the-shelf skinning tools, and simultaneously optimize with respect to the entire animated sequence. This is in contrast to neural volumetric [Mildenhall et al. 2020] or implicit surface representations [Wang et al. 2021], where a geometry deformation system either needs to be redesigned for the specific neural representation, e.g., D-NeRF [Pumarola et al. 2020], or where skinning is applied only *after* mesh optimization, without end-to-end mesh optimization and gradient flow through the deformation.

As a baseline, we use FlexiCubes and optimize using images from randomized cameras viewing only the mesh in its T-pose. The mesh is then re-skinned in a post-processing step. To illustrate the benefit of optimizing over the animation, we combine FlexiCubes with the differentiable mesh skinning approach of Hasselgren et al. [2021], re-skin the mesh in each training iteration, and optimize for image loss using images rendered from randomized cameras and animation frames. As shown in the bottom part of Figure 22, optimizing for the appearance over the entire animation helps re-distribute triangle density to avoid mesh stretching. Note that the differentiable skinning approach from Hasselgren et al. deforms a template mesh with fixed topology, while the FlexiCubes version presented here additionally optimizes topology, hence provides a more flexible approach to mesh simplification of animated assets.

6.3 3D Mesh Generation

Generation of 3D meshes, typically with the goal of facilitating 3D content creation, is an important task for computer graphics and vision, and benefits industries such as gaming and social platforms. Recent 3D generative models [Chan et al. 2022; Gao et al. 2022; Gu et al. 2022; Schwarz et al. 2022; Zhou et al. 2021] differentiably render a 3D representation into 2D images, and combine with a classic generative adversarial framework [Karras et al. 2019, 2020] to synthesize 3D content using only 2D image supervision. The recent state-of-the-art GET3D [Gao et al. 2022] directly synthesizes high-quality textured 3D meshes, enabled by the differentiable iso-surfacing module DMTet [Shen et al. 2021].

In this application, we demonstrate that FlexiCubes can serve as a plug-and-play differentiable mesh extraction module in a 3D generative model, and produce significantly improved mesh quality. Specifically, we use GET3D [Gao et al. 2022] and replace DMTet with FlexiCubes in the mesh extraction step. We only modify the last layer of the 3D generator in GET3D to additionally generate 21 weights for every cube in FlexiCubes. The training procedure, dataset (we use ShapeNet [Chang et al. 2015]) and other hyperparameters of GET3D are kept unchanged.

Table 6. Quantitative FID scores for a 3D generative modeling application. FlexiCubes can be applied as a differentiable mesh extraction module to GET3D [Gao et al. 2022], producing significantly improved synthesis quality.

Isosurfacing Method	Motorbike	Chair	Car
DMTet [Shen et al. 2021]	48.90	22.41	10.60
FlexiCubes	**44.87**	**17.51**	**9.55**

Qualitative comparisons and quantitative results are provided in Figure 23 and Table 6, respectively. FlexiCubes achieves better FID scores across all categories, demonstrating the higher capacity in generating 3D models. Qualitatively, the shapes generated using the FlexiCubes version of GET3D are of significantly higher quality, with more details and fewer sliver triangles.

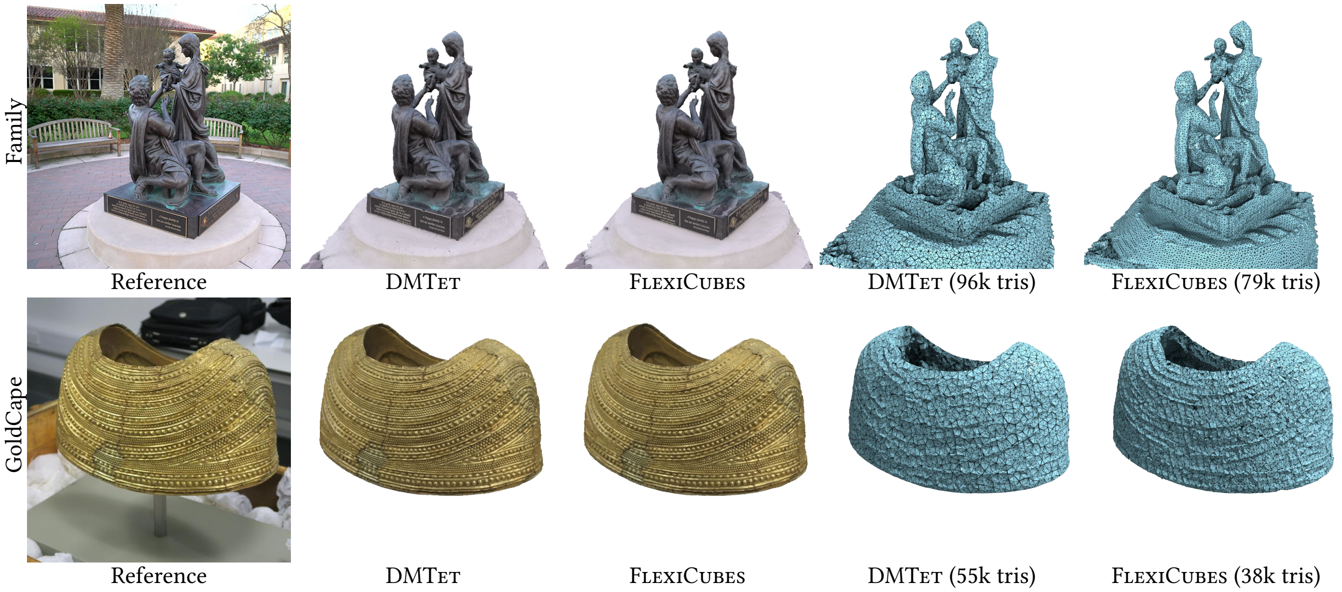

Fig. 19. We compare FlexiCubes and DMTet on real world photographic datasets using nvdiffrecmc for the Family dataset from Tanks&Temples [Knapitsch et al. 2017] and nvdiffrec for the GoldCape dataset [Boss et al. 2021]. FlexiCubes offers more uniform tessellation and more faithfully captures small geometric details (e.g. the grooves in the GoldCape scene). PSNR view interpolation validation scores are 28.49 / 28.47 dB (DMTet / FlexiCubes) for the Family scene and 24.44 / 24.56 dB for the GoldCape scene.

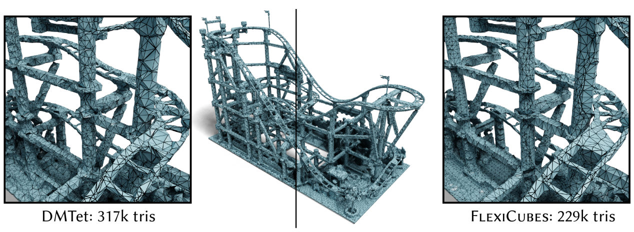

Fig. 20. Extracting a 3D model of the Roller scene from LDraw resources [Lasser 2022] using nvdiffrecmc [2022] unmodified (DMTet) and nvdiffrecmc with FlexiCubes for the topology extraction step. We note more uniform triangulation and higher detail.

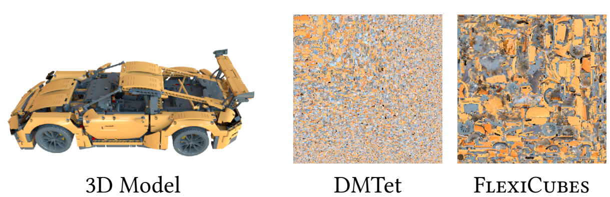

Fig. 21. Extracting a 3D model of the Porsche scene from LDraw resources [Lasser 2022] using nvdiffrecmc [2022]. We visualize the diffuse texture, and note that FlexiCubes simplifies the UV unwrapping step (performed using xatlas [2021]). The resulting textures have larger regions, which improves texture filtering.

6.4 Differentiable Physics Simulation

To leverage FlexiCubes's ability to differentiably extract tetrahedral meshes, we combine it with differentiable physics simulation [Jatavallabhula et al. 2021] and a differentiable rendering pipeline [Laine et al. 2020] to jointly recover 3D shapes and physical parameters from multi-view videos. Given a video sequence of an object deforming, we aim to recover a tetrahedral mesh of the rest pose as well as material parameters which reproduce the motion under simulation. In particular, we focus on FEM simulation with neo-Hookean elasticity to model elastic objects. After extracting the tetrahedral mesh from FlexiCubes, we feed it into GradSim [Jatavallabhula et al. 2021] to obtain deformed shapes at different time steps, these shapes are then differentiably rendered into multi-view images. We optimize both the 3D geometry and the physical density of the 3D shape in two-stage manner as in past work. See Figure 24 and the Supplement for more details. The optimized physical parameters and 3D geometry with texture are close to the ground truth.

7 DISCUSSION

7.1 Performance

Introducing additional degrees of freedom into the extraction representation incurs a moderate increase in runtime and memory usage. However, in many applications, the cost of mesh extraction is often small compared to the overall computation, and the ability to work with more concise extracted meshes may ultimately reduce the memory requirements of the overall pipeline. Concretely, we show a performance benchmark of different isosurfacing methods in Table 7. FlexiCubes is indeed slower and more memory-intensive than DMTet, and significantly more so than ordinary Marching

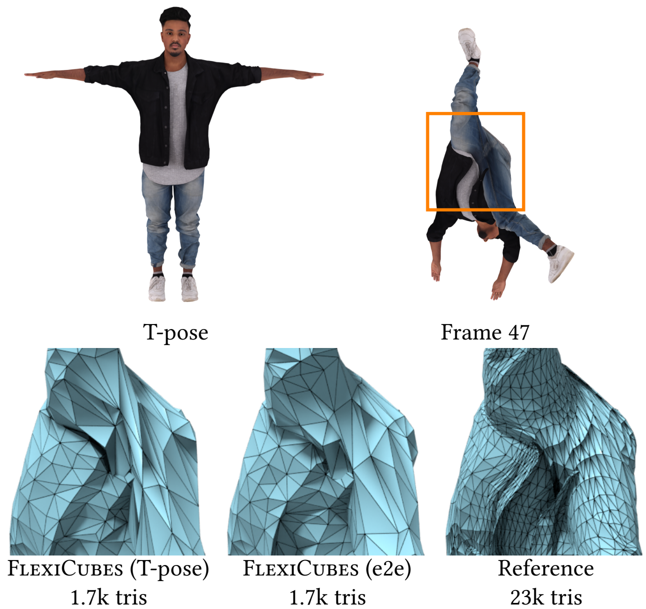

Fig. 22. Mesh simplification of a skinned animation, showing the benefits of end-to-end optimization using an explicit, differentiable, mesh representation. We learn the topology and appearance of a simplified mesh through image supervision using nvdiffrec. The top row shows the reference mesh T-pose and an animated frame. The bottom row shows a baseline of FlexiCubes optimized for the T-pose and re-skinned in a post pass, FlexiCubes with end-to-end optimization, where we re-skin the mesh in each optimization step and use a randomized viewpoint and animation frame, and the reference.

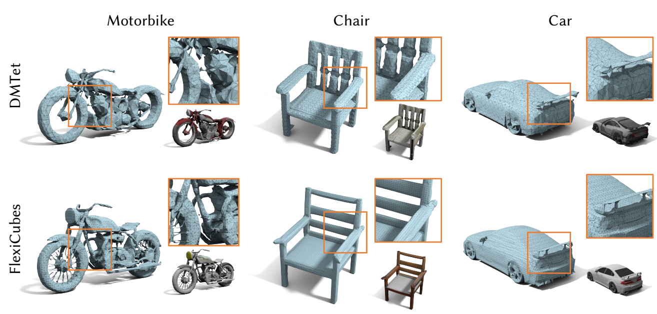

Fig. 23. Qualitative results for 3D generative modeling with meshes in GET3D [Gao et al. 2022]. FlexiCubes produces significantly improved mesh quality with detailed thin structures and more uniform surfaces.

Cubes, but all of these costs are small compared to the downstream task, which we benchmark in Table 8. The maximum grid resolution is not constrained by isosurface extraction, but rather by other components of the applications, such as rendering or neural network evaluations. We consistently choose the highest resolution that can be supported by high-end GPUs for all of our applications.

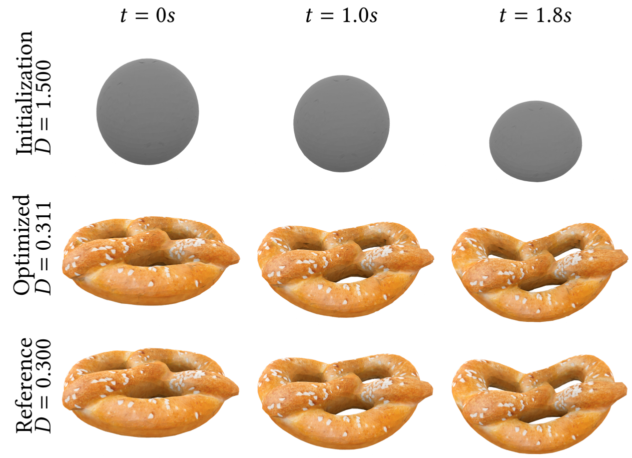

Fig. 24. We optimize shape, texture, and physical properties of an object from multi-view videos by feeding the extracted tetrahedral mesh from FlexiCubes into differentiable physics simulation [Jatavallabhula et al. 2021] and differentiable rendering [Laine et al. 2020] pipeline. In the bottom row (Reference shape), we set the mass density $D = 0.300$ and fix the two ending points of an object, letting it drop and deform under gravity. We show the output at three timesteps (three columns). We initialize an SDF with a sphere geometry and randomly-guessed mass density ($D = 1.500$, top row), and optimize in a two-stage manner. In the first stage, we utilize the beginning frame to optimize the shape and texture. In the second stage, we extract a tetrahedral mesh (Section 4.5) using FlexiCubes and apply differentiable physics simulation pipeline to optimize the physical parameters. The optimized physical parameters ($D = 0.311$, middle row) are close to the ground truth ($D = 0.300$, bottom row).

Table 7. Quantitative comparison of the performance of isosurfacing operations. FlexiCubes may significantly increase time and memory costs compared to simpler extractors, however these costs are still generally small in the context of downstream applications (see Table 8).

64^3	Forward Time (ms)	Backward Time (ms)	Memory (MB)
MC	2.28	0.43	12.05
DMTet	2.33	1.38	22.44
$DMC_{centroid}$ [Nielson 2004]	4.97	1.69	25.08
FlexiCubes	8.93	7.32	116.56
128^3	Forward Time (ms)	Backward Time (ms)	Memory (MB)
MC	5.08	0.58	72.85
DMTet	6.94	1.39	168.27
$DMC_{centroid}$ [Nielson 2004]	7.34	1.74	150.75
FlexiCubes	14.06	9.53	816.17

7.2 Limitations

Self-intersections. Although our approach generally produces high-quality meshes with improved element shapes in practice, and our core algorithm guarantees manifoldness, we do not guarantee non-self-intersecting output. Intersections arise because our flexible dual representation (Section 4) allows the extracted vertices to move into intersecting configurations; we found that strictly constraining the motions to non-intersecting configurations unacceptably worsened the expressivity and ease of optimization of the method. At a grid resolution of 64^3, in our experiment with 79 3D shapes in Table 2, we observed self intersections on 0.10% of the triangles, and we

Table 8. Quantitative comparison of the performance of various applications using two isosurfacing methods: DMTet and FlexiCubes. Note that nvdiffrecmc stores the per-vertex parameters in memory, whereas GET3D uses MLPs to predict these parameters. As a result, the introduction of more parameters leads to a larger increase in memory usage for nvdiffrecmc. FlexiCubes may even lower the memory requirements of the overall application, because fewer triangles are needed to represent the same geometry.

Applications (96^3)	nvdiffrecmc		GET3D	
Isosurface	DMTet	FlexiCubes	DMTet	FlexiCubes
Time per iter. (ms)	307	315	510	610
Memory(GiB)	13.1	15.3	11.6	11.1

note that this is lower than Dual Contouring variants (DC: 1.48%, NDC: 0.13%). Our optional extensions to tetrahedral and hierarchical meshing have slightly weaker guarantees, occasionally containing small cavities or nonmanifold elements in ambiguous cases arising from the Dual Marching Cubes topology. In our evaluation, we find that these small imperfections are not detrimental for downstream applications, but note that additional consideration may be required if a watertight mesh is imperative for a given application.

Continuity. More fundamentally, although we consider differentiable mesh extraction, our method is actually not even globally continuous. When the isosurface slips over a grid vertex, the mesh jumps discontinuously, a property we inherit from Dual Contouring and Dual Marching Cubes. Fortunately, because we apply our extraction in stochastic optimization settings, such as stochastic gradient descent with Adam, small local discontinuities do not obstruct optimization in practice. For this reason, we focus on our analysis and experiments on the property of effective optimization in downstream applications (Figure 4), rather than on formal notions of differentiability or smoothness.

7.3 Future Work

Looking forward, one opportunity to advance this approach is to integrate volumetric rendering with mesh-based representations for improved gradient approximation on visual tasks [Chen et al. 2022a]. Furthermore, 4D spatiotemporal meshing has important applications in dynamic geometry representation and optimization [Park et al. 2021]. Very directly, we also hope to integrate adaptive hierarchical mesh extraction (Section 4.6) into generative modeling applications. More broadly, in our experiments, we have found FlexiCubes to be a powerful tool for mesh optimization in visual computing, and we are eager to continue to build on top of it both in our own work and across the larger community.

ACKNOWLEDGMENTS

The authors are grateful to Aaron Lefohn and David I.W. Levin for their support and discussions during this research, as well as the anonymous reviewers for their valuable comments and feedback.

REFERENCES

Samir Akkouche and Eric Galin. Adaptive implicit surface polygonization using marching triangles. In *Computer Graphics Forum*, volume 20:2, pages 67–80, 2001.

Sergei Azernikov and Anath Fischer. Anisotropic meshing of implicit surfaces. In *International Conference on Shape Modeling and Applications 2005 (SMI'05)*, pages 94–103. IEEE, 2005.

Jules Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5(4):341–355, 1988.

Jules Bloomenthal. An implicit surface polygonizer. *Graphics gems*, 4:324–350, 1994.

Jules Bloomenthal, Chandrajit Bajaj, Jim Blinn, Marie-Paule Cani, Brian Wyvill, Alyn Rockwood, and Geoff Wyvill. *Introduction to implicit surfaces.* Morgan Kaufmann, 1997.

Mark Boss, Raphael Braun, Varun Jampani, Jonathan T. Barron, Ce Liu, and Hendrik P.A. Lensch. NeRD: Neural Reflectance Decomposition from Image Collections. In *IEEE International Conference on Computer Vision (ICCV)*, 2021.

Andrea Bottino, Wim Nuij, and Kees Van Overveld. How to shrinkwrap through a critical point: an algorithm for the adaptive triangulation of iso-surfaces with arbitrary topology. In *Proc. Implicit Surfaces*, volume 96, pages 53–72, 1996.

Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. Efficient Geometry-aware 3D Generative Adversarial Networks. In *CVPR*, 2022.

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, and Hao Zhang. Bsp-net: Generating compact meshes via binary space partitioning. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. *arXiv preprint arXiv:2208.00277*, 2022a.

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

Evgeni Chernyaev. Marching Cubes 33: Construction of topologically correct isosurfaces. Technical report, Institute for High Energy Physics, Moscow, 1995.

Bruno Rodrigues De Araújo, Daniel S Lopes, Pauline Jepp, Joaquim A Jorge, and Brian Wyvill. A survey on implicit surface polygonization. *ACM Computing Surveys (CSUR)*, 47(4):1–39, 2015.

Boyang Deng, Kyle Genova, Soroosh Yazdani, Sofien Bouaziz, Geoffrey Hinton, and Andrea Tagliasacchi. Cvxnets: Learnable convex decomposition. *arXiv preprint arXiv:1909.05736*, 2019.

Akio Doi and Akio Koide. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. *IEICE Transactions on Information and Systems*, 74(1): 214–224, 1991.

Jun Gao, Wenzheng Chen, Tommy Xiang, Clement Fuji Tsang, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Learning Deformable Tetrahedral Meshes for 3D Reconstruction. In *Advances In Neural Information Processing Systems*, 2020.

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.

Georgia Gkioxari, Jitendra Malik, and Justin Johnson. Mesh R-CNN. *ICCV 2019*, 2019.

Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan Russell, and Mathieu Aubry. AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Jiatao Gu, Lingjie Liu, Peng Wang, and Christian Theobalt. Stylenerf: A style-based 3d aware generator for high-resolution image synthesis. In *International Conference on Learning Representations*, 2022.

Rana Hanocka, Gal Metzer, Raja Giryes, and Daniel Cohen-Or. Point2Mesh: A Self-Prior for Deformable Meshes. *ACM Trans. Graph.*, 39(4), 2020.

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.

Hans-Christian Hege, Martin Seebass, Detlev Stalling, and Malte Zöckler. A Generalized Marching Cubes Algorithm Based On Non-Binary Classifications. *ZIB Preprint SC-97-05*, 1997.

Adrian Hilton, Andrew J Stoddart, John Illingworth, and Terry Windeatt. Marching triangles: range image fusion for complex object modelling. In *Proceedings of 3rd IEEE international conference on image processing*, volume 2, pages 381–384. IEEE, 1996.

Adrian Hilton, John Illingworth, et al. Marching triangles: Delaunay implicit surface triangulation. *University of Surrey*, 1997.

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.

Tasso Karkanis and A James Stewart. Curvature-dependent triangulation of implicit surfaces. *IEEE Computer Graphics and Applications*, 21(2):60–69, 2001.

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.

Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3907–3916, 2018.

Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Trans. Graph.*, 36(4):78:1–78:13, 2017.

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.

Gerald Lasser. LDraw.org, 2022. URL https://www.ldraw.org/.

Thomas Lewiner, Hélio Lopes, Antônio Wilson Vieira, and Geovan Tavares. Efficient implementation of marching cubes' cases with topological guarantees. *Journal of graphics tools*, 8(2):1–15, 2003.

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.

Yiyi Liao, Simon Donné, and Andreas Geiger. Deep Marching Cubes: Learning Explicit Surface Representations. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2916–2925, 2018.

Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. Magic3d: High-resolution text-to-3d content creation. *arXiv preprint arXiv:2211.10440*, 2022.

Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning. In *International Conference on Computer Vision (ICCV)*, pages 7707–7716, 2019.

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

Neil H McCormick and Robert B Fisher. Edge-constrained marching triangles. In *Proceedings. First International Symposium on 3D Data Processing Visualization and Transmission*, pages 348–351. IEEE, 2002.

Ishit Mehta, Manmohan Chandraker, and Ravi Ramamoorthi. A Level Set Theory for Neural Implicit Evolution Under Explicit Flows. In *ECCV 2022, Proceedings, Part II*, page 711–729, 2022.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*, 2020.

Claudio Montani, Riccardo Scateni, and Roberto Scopigno. Discretized marching cubes. In *Proceedings Visualization'94*, pages 281–287. IEEE, 1994.

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.

Ashish Myles, Nico Pietroni, and Denis Zorin. Robust field-aligned global parametrization. *ACM Transactions on Graphics (TOG)*, 33(4):1–14, 2014.

Charlie Nash, Yaroslav Ganin, S. M. Ali Eslami, and Peter W. Battaglia. Polygen: An autoregressive generative model of 3d meshes. *ICML*, 2020.

Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. Large Steps in Inverse Rendering of Geometry. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 40(6), 2021.

Gregory M. Nielson. On Marching Cubes. *IEEE Transactions on visualization and computer graphics*, 9(3):283–297, 2003.

Gregory M Nielson. Dual Marching Cubes. In *IEEE visualization 2004*, pages 489–496. IEEE, 2004.

Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M Seitz. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *arXiv preprint arXiv:2106.13228*, 2021.

Despoina Paschalidou, Angelos Katharopoulos, Andreas Geiger, and Sanja Fidler. Neural parts: Learning expressive 3d shape abstractions with invertible neural networks. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. *arXiv preprint arXiv:2011.13961*, 2020.

Edoardo Remelli, Artem Lukoianov, Stephan Richter, Benoît Guillard, Timur Bagautdinov, Pierre Baque, and Pascal Fua. Meshsdf: Differentiable iso-surface extraction. *Advances in Neural Information Processing Systems*, 33:22468–22478, 2020.

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

Scott Schaefer, Tao Ju, and Joe Warren. Manifold dual contouring. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):610–619, 2007.

Katja Schwarz, Axel Sauer, Michael Niemeyer, Yiyi Liao, and Andreas Geiger. Voxgraf: Fast 3d-aware image synthesis with sparse voxel grids. *ARXIV*, 2022.

Roberto Scopigno. A modified look-up table for implicit disambiguation of marching cubes. *The visual computer*, 10:353–355, 1994.

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.

Barton T Stander and John C Hart. Interactive re-polygonization of blobby implicit curves. In *Proc. Western Computer Graphics Symposium*, volume 5, 1995.

Oded Stein, Eitan Grinspun, and Keenan Crane. Developability of triangle meshes. *ACM Trans. Graph.*, 37(4), 2018.

Subodh C Subedi, Chaman Singh Verma, and Krishnan Suresh. A review of methods for the geometric post-processing of topology optimized models. *Journal of Computing and Information Science in Engineering*, 20(6), 2020.

Minhyuk Sung, Hao Su, Vladimir G Kim, Siddhartha Chaudhuri, and Leonidas Guibas. Complementme: Weakly-supervised component suggestions for 3d modeling. *ACM Transactions on Graphics (TOG)*, 36(6):1–12, 2017.

Shubham Tulsiani, Hao Su, Leonidas J. Guibas, Alexei A. Efros, and Jitendra Malik. Learning shape abstractions by assembling volumetric primitives. In *Computer Vision and Pattern Regognition (CVPR)*, 2017.

Kees Van Overveld and Brian Wyvill. Shrinkwrap: An efficient adaptive algorithm for triangulating an iso-surface. *The visual computer*, 20(6):362–379, 2004.

Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proceedings of the European conference on computer vision (ECCV)*, pages 52–67, 2018.

Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 27171–27183, 2021.

Kangxue Yin, Zhiqin Chen, Siddhartha Chaudhuri, Matthew Fisher, Vladimir Kim, and Hao Zhang. Coalesce: Component assembly by learning to synthesize connections. In *Proc. of 3DV*, 2020.

Jonathan Young. xatlas, 2021. https://github.com/jpcy/xatlas.

Peng Zhou, Lingxi Xie, Bingbing Ni, and Qi Tian. Cips-3d: A 3d-aware generator of gans based on conditionally-independent pixel synthesis. *arXiv preprint arXiv:2110.09788*, 2021.

Chenyang Zhu, Kai Xu, Siddhartha Chaudhuri, Renjiao Yi, and Hao Zhang. SCORES: Shape composition with recursive substructure priors. *ACM Transactions on Graphics*, 37(6):Article 211, 2018.