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.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

© 2023 Copyright held by the owner/author(s).
0730-0301/2023/8-ART1
https://doi.org/10.1145/3592430

ACM Trans. Graph., Vol. 42, No. 4, Article 1. Publication date: August 2023.
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 naive 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.

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.

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 [Bloomental 1988; Bloomental 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 [Bloomental 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 un-
stable 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 underly-
ing 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 \( e \in V \) 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 Tetrahe-
dra [Doi and Koide 1991], and many generalizations. Mesh vertices
are extracted along grid edges where the linearly-interpolated scalar
function changes sign

\[
u_e = \frac{X_a \cdot s(X_b) - X_b \cdot s(X_a)}{s(X_b) - s(X_a)}.
\]

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

However, the mesh vertices resulting from marching extraction
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 de-
form [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 repre-
sentation with additional degrees of freedom to allow independent
repositioning of the vertices, as shown in Figure 4.
when optimizing with respect to the underlying function, differentiability is retained. More details about this experiment are provided in the Supplement.

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 control how dual vertices are split into triangles (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

\[ u_e = \frac{x_g \cdot s(x_g) - x_y \cdot s(x_y)}{s(x_g) - s(x_y)}, \]

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

\[ v_d = \frac{1}{|V_E|} \sum_{u_e \in V_E} u_e, \]

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 \( \alpha \in \mathbb{R}^8 \) associating a positive scalar with each cube edge. These weights adjust the location of the crossing point \( e \) along each edge, and Equation 3 then becomes

\[ u_e = \frac{s(x_j)\alpha_j x_i - s(x_i)\alpha_i x_j}{s(x_j) - s(x_i)}, \]

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 \( \beta \in \mathbb{R}_{>0}^{12} \), 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

\[ v_d = \frac{1}{\sum_{u_e \in V_E} \beta_e} \sum_{u_e \in V_E} \beta_e u_e. \]

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

Together these weights \( \alpha \in \mathbb{R}^8, \beta \in \mathbb{R}_{>0}^{12} \) 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].
We define a weight γ ∈ 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 \( \overline{d} \) (Figure 8). The location of this midpoint is computed as

\[
\overline{d} = \frac{yc1 yc2 (d^2 + d^2) + yc yc (d^2 + d^2)}{yc1 yc1 + yc2 yc2} \quad (7)
\]

with notation 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 \( \gamma \) parameters on the corresponding vertices. Intuitively, adjusting the \( \gamma \) weights smoothly interpolates between the geometries resulting from the two possible splits. Optimizing \( \gamma \) 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 \( \overline{d} \), but simply split each quadrilateral along whichever diagonal has larger product of \( \gamma \) 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.

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 adaptive 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.
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

$$\mathcal{L}_\text{dev} := \sum_{v \in V} \text{MAD}\{||v - u_e|_2 : u_e \in \mathcal{N}_v\}.$$  

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 $\mathcal{E}_g$ 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 $\text{sign}(s_a) \neq \text{sign}(s_b)$. Then the loss is given by

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

where $H$, $\sigma$ 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 differentially 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 values.
Flexible Isosurface Extraction for Gradient-Based Mesh Optimization

$\mathbf{MC}_{\text{SDF}}$  
$\mathbf{DC}_{\text{hermite}}$  
$\mathbf{NDC}_{\text{SDF}}$  
$\mathbf{MC}$  
$\mathbf{DMTet}$  
$\mathbf{FlexiCubes}$  
Reference

Fig. 13. Visual comparison of a set of iso-surfacing techniques. The three leftmost examples: Marching Cubes ($\mathbf{MC}_{\text{SDF}}$), 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 ($\mathbf{MC}_{\text{SDF}}$), DC ($\mathbf{DC}_{\text{hermite}}$) and NDC ($\mathbf{NDC}_{\text{SDF}}$). 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.

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 ($\text{IN}>5^\circ$) 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.

1https://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.

<table>
<thead>
<tr>
<th>Method</th>
<th>IN&gt;5° (%)</th>
<th>CD (10⁻³)</th>
<th>F1</th>
<th>ECD (10⁻²)</th>
<th>EF1</th>
<th>#V</th>
<th>#F</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC</td>
<td>85.69</td>
<td>22.65</td>
<td>0.28</td>
<td>5.56</td>
<td>0.08</td>
<td>2387</td>
<td>4771</td>
</tr>
<tr>
<td>DChermite</td>
<td>74.43</td>
<td>17.15</td>
<td>0.38</td>
<td>4.82</td>
<td>0.11</td>
<td>2360</td>
<td>4775</td>
</tr>
<tr>
<td>NDC</td>
<td>72.69</td>
<td>17.64</td>
<td>0.41</td>
<td>3.55</td>
<td>0.13</td>
<td>1877</td>
<td>3801</td>
</tr>
<tr>
<td>FlexiCubes</td>
<td>67.52</td>
<td>7.01</td>
<td>0.64</td>
<td>2.31</td>
<td>0.26</td>
<td>2400</td>
<td>4800</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Method</th>
<th>IN&gt;5° (%)</th>
<th>CD</th>
<th>GP5 °</th>
<th>Aspect Ratio</th>
<th>Radius Ratio</th>
<th>Min Angle</th>
<th>Max Angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC</td>
<td>50.16</td>
<td>8.56</td>
<td>11.46</td>
<td>11.43</td>
<td>11.62</td>
<td>DMTet(S)</td>
<td>48.66</td>
</tr>
<tr>
<td>DChermite</td>
<td>67.65</td>
<td>6.60</td>
<td>11.32</td>
<td>11.49</td>
<td>11.92</td>
<td></td>
<td>4.40</td>
</tr>
<tr>
<td>NDC + Reg.</td>
<td>50.16</td>
<td>8.56</td>
<td>11.46</td>
<td>11.43</td>
<td>11.62</td>
<td></td>
<td>4.40</td>
</tr>
</tbody>
</table>

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 \( \bar{e} = \frac{1}{|E|} \sum_{e \in E} |e| \), where \( E \) denotes the set of all the edges in the extracted mesh. The regularization is computed as:

\[
R_{edge} = \frac{1}{T} \sum_{e \in E} (|e - \bar{e}|^2)
\]

This combination of regularizer in combination with the mesh 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.
Flexible Isosurface Extraction for Gradient-Based Mesh Optimization

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.

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.

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
Table 5. View interpolation results (PSNR) for \textsc{nvdiffrec} reconstructions of the NeRF synthetic dataset, using either \textsc{DMTet} or \textsc{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.

<table>
<thead>
<tr>
<th></th>
<th>Chair</th>
<th>Drums</th>
<th>Ficus</th>
<th>Hotdog</th>
<th>Lego</th>
<th>Mats</th>
<th>Mic</th>
<th>Ship</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textsc{DMTet}</td>
<td>31.8</td>
<td>24.6</td>
<td>30.9</td>
<td>33.2</td>
<td>29.0</td>
<td>27.0</td>
<td>30.7</td>
<td>26.0</td>
</tr>
<tr>
<td>\textsc{FlexiCubes}</td>
<td>31.8</td>
<td>24.7</td>
<td>30.9</td>
<td>33.4</td>
<td>28.8</td>
<td>26.7</td>
<td>30.8</td>
<td>25.9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Chair</th>
<th>Drums</th>
<th>Ficus</th>
<th>Hotdog</th>
<th>Lego</th>
<th>Mats</th>
<th>Mic</th>
<th>Ship</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textsc{CD (10^{-2})}</td>
<td>4.51</td>
<td>3.98</td>
<td>0.30</td>
<td>2.67</td>
<td>2.41</td>
<td>0.41</td>
<td>1.20</td>
<td>55.8</td>
</tr>
<tr>
<td>\textsc{FlexiCubes}</td>
<td>0.45</td>
<td>2.27</td>
<td>0.37</td>
<td>1.44</td>
<td>1.60</td>
<td>0.53</td>
<td>1.51</td>
<td>10.5</td>
</tr>
</tbody>
</table>

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

Table 6. Quantitative FID scores for a 3D generative modeling application. \textsc{FlexiCubes} can be applied as a 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 \textsc{DMTet} with \textsc{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 \textsc{FlexiCubes}. The training procedure, dataset (we use ShapeNet [Chang et al. 2015]) and other hyperparameters of GET3D are kept unchanged.

<table>
<thead>
<tr>
<th>Isosurfacing Method</th>
<th>Motorbike</th>
<th>Chair</th>
<th>Car</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textsc{DMTet} [Shen et al. 2021]</td>
<td>48.90</td>
<td>22.41</td>
<td>10.60</td>
</tr>
<tr>
<td>\textsc{FlexiCubes}</td>
<td>44.87</td>
<td>17.51</td>
<td>9.55</td>
</tr>
</tbody>
</table>

Qualitative comparisons and quantitative results are provided in Figure 23 and Table 6, respectively. \textsc{FlexiCubes} achieves better FID scores across all categories, demonstrating the higher capacity in generating 3D models. Qualitatively, the shapes generated using the \textsc{FlexiCubes} version of GET3D are of significantly higher quality, with more details and fewer sliver triangles.
Flexible Isosurface Extraction for Gradient-Based Mesh Optimization

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.

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).

<table>
<thead>
<tr>
<th>Method</th>
<th>Forward Time (ms)</th>
<th>Backward Time (ms)</th>
<th>Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC</td>
<td>2.28</td>
<td>0.43</td>
<td>12.05</td>
</tr>
<tr>
<td>DMTet</td>
<td>2.33</td>
<td>1.38</td>
<td>22.44</td>
</tr>
<tr>
<td>DMC_{centroid} [Nielson 2004]</td>
<td>4.97</td>
<td>1.69</td>
<td>25.08</td>
</tr>
<tr>
<td>FlexiCubes</td>
<td>8.93</td>
<td>7.32</td>
<td>116.56</td>
</tr>
</tbody>
</table>

Table 8. Forward and backward time and memory usage for DMTet and DMC_{centroid} [Nielson 2004] for 128³ shapes.

<table>
<thead>
<tr>
<th>Method</th>
<th>Forward Time (ms)</th>
<th>Backward Time (ms)</th>
<th>Memory (Mb)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC</td>
<td>5.08</td>
<td>8.58</td>
<td>72.85</td>
</tr>
<tr>
<td>DMTet</td>
<td>6.94</td>
<td>1.39</td>
<td>168.27</td>
</tr>
<tr>
<td>DMC_{centroid} [Nielson 2004]</td>
<td>7.34</td>
<td>1.74</td>
<td>150.75</td>
</tr>
<tr>
<td>FlexiCubes</td>
<td>14.06</td>
<td>9.53</td>
<td>816.17</td>
</tr>
</tbody>
</table>

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 64³, 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.

<table>
<thead>
<tr>
<th>Applications</th>
<th>nvdiffrecmc</th>
<th>GET3D</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsoSurface</td>
<td>DMTet</td>
<td>FlexiCubes</td>
</tr>
<tr>
<td>Time per iter. (ms)</td>
<td>307</td>
<td>315</td>
</tr>
<tr>
<td>Memory (GiB)</td>
<td>13.1</td>
<td>15.3</td>
</tr>
<tr>
<td>DMTet</td>
<td>FlexiCubes</td>
<td></td>
</tr>
<tr>
<td>Time per iter. (ms)</td>
<td>510</td>
<td>610</td>
</tr>
<tr>
<td>Memory (GiB)</td>
<td>11.6</td>
<td>11.1</td>
</tr>
</tbody>
</table>

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 non-manifold 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

gradfun: Differentiable simulation for system identification and visuo-motor control.


Tasso Karkanis and A James Stewart. Curvature-dependent triangulation of implicit surfaces.


Tero Karras, Samuli Laine, Mikko Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of StyleGAN.


Arno Knipatsch, Jarsik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction.

Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular Primitives for High-Performance Differentiable Rendering.


Thomas Lewiner, Hélio Lopes, Antônio Wilson Vieira, and Geovan Tavares. Efficient implementation of marching cubes’ cases with topological guarantees.


Shichen Liu, Tianye Li, WeiKai Chen, and Hao Li. Soft rasterizer: A Differentiable Renderer for Image-based 3D Rendering.


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

Claudio Montani, Riccardo Scatena, and Roberto Scopigno. Discretized marching cubes.


Ashish Myles, Nico Pietroni, and Denis Zorin. Robust field-aligned global parametrization.


Baptiste Nicolet, Alex Jacobson, and Wenzel Jakob. Large steps in Inverse Rendering of Geometry.


Gregory M. Nielson. Dual Marching Cubes.


ArXiv, 2022.

Roberto Scopigno. A modified look-up table for implicit disambiguation of marching cubes.


Barton T Stander and John C Hart. Interactive re-polygonization of sloppy implicit curves.


Learning shape abstractions by assembling volumetric primitives.
In Computer Vision and Pattern Recognition (CVPR), 2017.

Kees Van Overveld and Brian Wyvill. Shrinkwrap: An efficient adaptive algorithm for triangulating an iso-surface.


Chenyang Zhu, Kai Xu, Siddhartha Chaudhuri, Renjiao Yi, and Hao Zhang. SCORES: Shape composition with recursive substructure priors.