Differentiable Voxelization of Surface Representations

TOBIAS DJUREN, Technische Universität Berlin, Germany
UGO FINNENDAHL*, Technische Universität Berlin, Germany
MARKUS WORCHEL*, Technische Universität Berlin, Germany
HENDRIK MEYER, Technische Universität Berlin, Germany
MARC ALEXA, Technische Universität Berlin, Germany

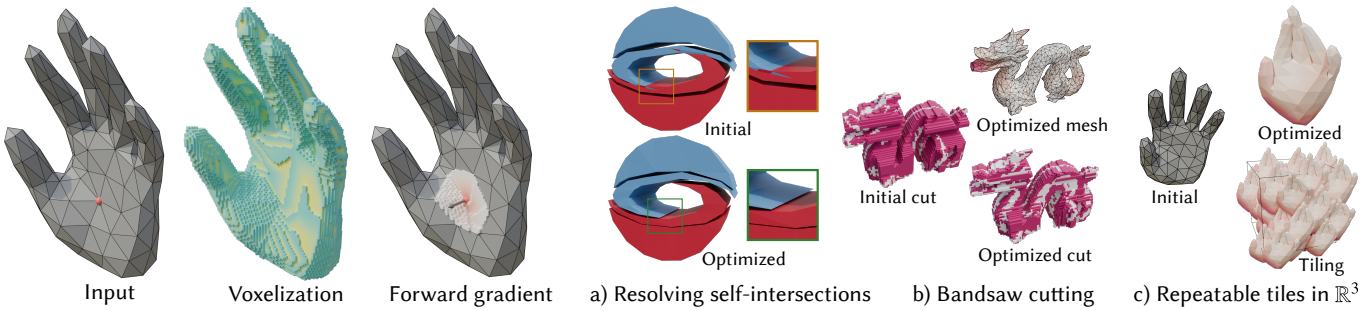

Fig. 1. We present an efficient technique for differentiating voxel values with respect to the surface that generates the voxel grid. This enables the solution of various problems that can easily be solved on regular voxel grids, such as intersection detection, for surface representations like triangle meshes. The technique can be readily integrated into gradient-based optimization frameworks like PyTorch [Paszke et al. 2019]. Moving the highlighted vertex (left) in the indicated direction changes the voxelization of all adjacent triangles (right). We discuss various applications, like a) the resolving of self-intersections, b) optimized cuts for manufacturing with bandsaws, and c) shape deformation to form repeatable tilings that fill the $\mathbb{R}^3$.

Different shape representations facilitate different computations. Surface representations, in particular meshes, are often used for modeling, whereas volume representations are useful for spatial queries such as intersection or containment. Optimizing a surface representation based on a volumetric properties by gradient descent requires the derivatives of the volume relative to its bounding surface. We derive this gradient for winding numbers and show that it can be efficiently computed for volumetric values sampled on a regular grid (voxel representation) and surface parameters based on vertex sets (triangle meshes). This enables an efficient solution for a variety of optimization problems. We demonstrate the practical use of this approach at the examples of deforming meshes to resolve intersections, being manufacturable by cutting with a bandsaw from three directions, and creating shapes that are close to tiling 3D space.

CCS Concepts: • **Computing methodologies** → **Shape modeling**.

ACM Reference Format:
Tobias Djuren, Ugo Finnendahl, Markus Worchel, Hendrik Meyer, and Marc Alexa. 2026. Differentiable Voxelization of Surface Representations. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers (SIGGRAPH Conference Papers '26), July 19–23, 2026, Los Angeles, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3799902.3811203

*Equal contribution (shared second author).

Authors' Contact Information: Tobias Djuren, Technische Universität Berlin, Berlin, Germany, t.djuren@tu-berlin.de; Ugo Finnendahl, Technische Universität Berlin, Berlin, Germany, finnendahl@tu-berlin.de; Markus Worchel, Technische Universität Berlin, Berlin, Germany, m.worchel@tu-berlin.de; Hendrik Meyer, Technische Universität Berlin, Berlin, Germany, hendrik.meyer@tu-berlin.de; Marc Alexa, Technische Universität Berlin, Berlin, Germany, marc.alexa@tu-berlin.de.

SIGGRAPH Conference Papers '26, Los Angeles, CA, USA

ACM ISBN 979-8-4007-2554-8/2026/07
https://doi.org/10.1145/3799902.3811203

1 Introduction

The shape of a solid object can be represented in two fundamentally different ways: (1) as the part of space $\Omega \subset \mathbb{R}^n$ covered by the object; or as the boundary $\partial\Omega$ enclosing its interior. We will refer to these representations as *volume* and *surface* representations. Each class comprises various instances, serving different purposes and application domains.

In many applications, the interior of the shape is not visible or irrelevant. Then surface representations suffice to describe the shape and avoid storing information that is irrelevant. The most relevant surface representations are polygon meshes [Botsch et al. 2010], dominating in the visual effects and games industry, parametric surfaces used in CAGD [Farin 2002], and subdivision surfaces [Warren and Weimer 2001] being a combination of the two.

Volumetric representations are useful when either the interior is heterogeneous or the space covered by the object, possibly also in relation to other shapes, is relevant. Examples are intersection queries [Amanatides and Woo 1987] or constructive solid geometry [Voelcker and Requicha 1977]. Clearly, the most relevant way to store information about the interior is a regular grid, so-called *voxel* representations. Voxel grids align well with hardware and, in particular, memory access, making algorithms operating on them very efficient in practice. This is likely part of the reason why they are used in compute-heavy domains, for example, in machine learning [Deng et al. 2021; Liu et al. 2020; Sun et al. 2022]. The information stored in the voxels varies: binary information about being interior

(occupancy), distance to the closest point on the surface (distance fields), or, as we suggest here, the winding number [Mobius 1865] – see Sec. 3 for details.

Modeling and optimization of shapes relies almost exclusively on defining an *energy function* E (similar to the *loss* used in machine learning) that takes on smaller values for better solutions to the problem. Our original motivation for this work was modeling geometry that can be cut out of a solid using a bandsaw. This requires the object to be the intersection of extrusions (see Sec. 4.3 for details).

While this condition can be easily checked for a volumetric representation, it is quite cumbersome for surfaces. Yet for exploring the space of deformation of an object, surface representations have shown to be very useful [Botsch and Sorkine 2008; Montagnat et al. 2001]. We believe this a common scenario: a shape is given in a surface representation, governed by parameters $\boldsymbol{\theta} \in \mathbb{R}^m$ (e.g. the vertex positions of a polygon mesh); we wish to find or approximate $\arg\min_{\boldsymbol{\theta}} E(\Omega(\boldsymbol{\theta}))$, where the energy E depends on the volumetric representation $\Omega(\boldsymbol{\theta})$ of the shape. Efficient optimization relies on the gradient of the energy E relative to the parameters $\boldsymbol{\theta}$, which by the chain rule is a product of two quantities:

$$\frac{\partial E(\Omega(\boldsymbol{\theta}))}{\partial \boldsymbol{\theta}} = \frac{\partial E(\Omega(\boldsymbol{\theta}))}{\partial\, \Omega} \cdot \frac{\partial\, \Omega}{\partial \boldsymbol{\theta}} \tag{1}$$

While the derivative(s) of the energy with respect to the volume representation will depend on the energy, the derivatives of the volume with respect to the parameters of the surface representation are a general problem, whose solution enables a range of methods and applications. In Sec. 3 we derive a general continuous solution for the winding number function, allowing the treatment of self-intersecting surfaces. For the most common case of triangle meshes, we show that this approach allows computing the derivatives for a voxel grid in a highly efficient way (Sec. 3.3).

We evaluate the implementation of this approach (Sec. 4) and then demonstrate its effectiveness and versatility in several applications: apart from the mentioned bandsaw geometry, we show how to resolve self-intersections and deform shapes so that they better tile space. Sec. 5 discusses directions for future work.

2 Related Work

Rasterization and Voxelization. Rasterization of solid primitives such as meshes is one of the fundamental blocks of image generation, and techniques go back to the early days of computer graphics [Appel 1968; Warnock 1969; Wylie et al. 1967]. Generalization from two-dimensional pixels to three-dimensional voxels was made early on [Lee and Requicha 1982], enabling a structured geometry representation that allows for easier computation of e.g. translucent geometry, Constructive Solid Geometry (CSG) or inter-object intersections, which is usually easier to calculate over volumes than boundaries. Due to the regular structure and independence of the voxels as in the 2D case, algorithms have been developed that take advantage of the highly parallelization possibilities of the GPU [Crassin and Green 2012; Eisemann and Décoret 2008; Schwarz and Seidel 2010]. The simplest form of rasterization/voxelization is point sampling the occupancy at the pixel/voxel centers. This leads to aliasing artifacts if frequencies higher than half the sampling rate are present. Convolving the signal with a filter reduces aliasing, enables the use of the divergence theorem to iterate over the boundary instead of the volume [Manson and Schaefer 2011] and allows differentiation [Sun et al. 2024].

For our primal voxelization, we took inspiration from Manson and Schaefer [2013], as we use a 3D generalization of the scanline approach that handles overlapping shapes inherently and is optimized for meshes.

Volumetric Representations in Gradient-based Optimizations. Volumetric shape representations, such as density [Chen and Zhang 2019; Mescheder et al. 2019] or signed distance functions [Jiang et al. 2020a; Park et al. 2019; Wang et al. 2021; Zhang et al. 2021] are ubiquitous in modern computer vision and machine learning pipelines. Conversion between representations is common, but the prevailing direction is from volumetric to explicit [Binninger et al. 2025; Liao et al. 2018; Remelli et al. 2020a; Shen et al. 2021, 2023]. This is partially motivated by the fact that some operations (e.g. rendering) or surface characteristics (e.g. developability) are more naturally expressed on the explicit representation. Accordingly, our approach follows the *inverse* direction by converting from explicit to volumetric representations in a fully differentiable manner. This is particularly useful for objectives that are more naturally expressed on the volume, such as resolving collisions during the optimization of articulated meshes [Hassan et al. 2019; Jiang et al. 2020b; Khirodkar et al. 2022; Xu et al. 2025] or for using volume-based neural networks on such representations [Maturana and Scherer 2015].

Winding Numbers. The winding number [Mobius 1865] has found many applications in geometry processing, including mesh booleans and CSG operations [Zhou et al. 2016], robust tetrahedral meshing [Hu et al. 2018; Jacobson et al. 2013], surface reconstruction from point clouds [Barill et al. 2018; Chen et al. 2024; He et al. 2024], and mesh repair for non-watertight inputs [Attene et al. 2013].

The method of Jacobson et al. [2013], provides a robust generalization of winding number that gracefully handles non-watertight meshes and triangle soups. Barill et al. [2018] accelerated this computation to $O(n \log n)$ using a hierarchical Barnes and Hut [1986] style algorithm, making winding number queries more practical for large meshes and multiple queries at once.

Most closely related to our work, Sun et al. [2024] introduced Gaussian-smoothed winding numbers with analytic derivatives for surface and sampling point optimization. They solve a boundary integral for each voxel separately, made practical only in 2D by approximating the surface using a Barnes and Hut [1986] style simplification of the curve. Sun et al. [2026] further used Gaussian-smoothed winding numbers in 3D, to guide the optimization of offset surfaces. In comparison, our key insight is that by using a box filter instead of a Gaussian, the regular grid structure of the sampling points can be exploited, leading to a highly efficient, differentiable voxelization algorithm without requiring acceleration data structures.

3 Differentiable Voxelization

Voxelization of a closed surface $\partial\Omega$ in $\mathbb{R}^n$ can generally be understood as point-sampling its winding number field $w_{\partial\Omega}(\mathbf{x}) \in \mathbb{R}$ on a regular

grid

$$w_{\partial\Omega}(\mathbf{x_i}), \tag{2}$$

where $\mathbf{x_i} \in \mathbb{R}^n$ is the center of the grid cell, or *voxel*, with multi-index $\mathbf{i} = (i_1, \ldots, i_n)_{1 \le i_k \le d_k}$. This perspective aligns with common intuition: if the surface is intersection-free, the winding number reduces to the indicator function

$$\chi_\Omega(\mathbf{x}) \coloneqq \begin{cases} 1, & \mathbf{x} \in \Omega \\ 0, & \mathbf{x} \notin \Omega \end{cases} \tag{3}$$

of the region $\Omega \subset \mathbb{R}^n$ bounded by the surface $\partial\Omega$. In this case, the value of each voxel indicates if its center $\mathbf{x_i}$ is inside or outside of the represented shape. In the following, we will omit the subscripts for brevity, such that $w(\mathbf{x})$ means $w_{\partial\Omega}(\mathbf{x})$.

Our goal is to differentiate the voxel representation with respect to parameters $\boldsymbol{\theta} \in \mathbb{R}^m$ that govern the surface $\partial\Omega$ (e.g., vertex positions of a mesh). While the winding number at each voxel naturally depends on the surface, and therefore the parameters $\boldsymbol{\theta}$, the particular difficulty is that the winding number field is piecewise constant: the derivative $\partial_{\boldsymbol{\theta}} w(\mathbf{x})$[1] is zero almost everywhere on $\mathbb{R}^n$, so the derivative at the voxel centers $\partial_{\boldsymbol{\theta}} w(\mathbf{x_i})$ is either zero or undefined. Instead, we consider the *pre-filtered* winding number

$$\tilde{w}(\mathbf{x}) = \int_{\mathbb{R}^n} k(\mathbf{x} - \mathbf{x}') w(\mathbf{x}') \, \mathrm{d}A(\mathbf{x}'), \tag{4}$$

where k is a filter kernel. For our particular choice of kernels, this corresponds to a smoothing of the winding number field, which leads to voxel values $\tilde{w}(\mathbf{x_i})$ that continuously vary with the surface. In Sec. 3.1, we will first derive an expression for the derivative $\partial_{\boldsymbol{\theta}} \tilde{w}(\mathbf{x})$ that is valid for general surfaces in arbitrary dimensions. Without loss of generality, in most of our derivations, we will consider just a single surface parameter $\theta \in \mathbb{R}$ for brevity.

The ability to differentiate the voxel representation of a surface enables gradient-based solutions of optimization problems

$$\boldsymbol{\theta}^\star = \arg\min_{\boldsymbol{\theta} \in \mathbb{R}^m} E(\mathbf{W}), \tag{5}$$

where E is some energy, or objective function, taking the voxel grid $\mathbf{W} \in \mathbb{R}^{d_1 \times \cdots \times d_n}$, with $\mathbf{W_i} = \tilde{w}(\mathbf{x_i})$, as input. Sec. 3.2 investigates the steps required for such optimizations, in particular how to compute the smoothed winding numbers $\tilde{w}(\mathbf{x_i})$ and their derivatives $\partial_{\boldsymbol{\theta}} \tilde{w}(\mathbf{x_i})$ in the general case.

We make the key observation that restricting the setting to one of the most common cases – piecewise linear surfaces with box filters – allows highly efficient voxelization and differentiation. The respective algorithms are introduced in Sec. 3.3.

3.1 Derivatives for General Surfaces in $\mathbb{R}^n$

For the 2D case ($n = 2$), Sun et al. [2024] differentiate Eq. (4), assuming that $\partial\Omega$ is a piecewise linear curve and k a Gaussian kernel, by locally decomposing the winding number into Heaviside step functions. We find that Reynold's transport theorem [Reynolds et al. 1903] enables a simple derivation for general surfaces in any dimension. This takes inspiration from differentiable rendering [Zhang et al. 2019], where similar integrals have been differentiated to optimize scene geometry using image-based objectives[2].

Consider the derivative of Eq. (4) for a surface parameter $\theta \in \mathbb{R}$:

$$\partial_\theta \tilde{w}(\mathbf{x}) = \partial_\theta \int_{\mathbb{R}^n} k(\mathbf{x} - \mathbf{x}') w(\mathbf{x}') \, \mathrm{d}A(\mathbf{x}'). \tag{6}$$

It is incorrect to simply "differentiate under the integral" because the integrand has discontinuities in $\mathbb{R}^n$ that move with θ: the winding number w is piecewise constant, and the constant regions are bounded by subsets of the surface $\partial\Omega$. In our case, the winding number field is also the *only* source of such discontinuities, even if the kernel is discontinuous, because θ only affects the surface and neither the kernel nor shape or position of the voxel grid.

The general idea is to partition the domain into those regions $D_i \subset \mathbb{R}^n$ with constant winding number w_i

$$= \sum_i \partial_\theta \int_{D_i(\theta)} k(\mathbf{x} - \mathbf{x}') w_i \, \mathrm{d}A(\mathbf{x}'). \tag{7}$$

Since the region boundaries are subsets of the surface, they depend on θ. Applying Reynold's transport theorem splits the differential integral into an *interior* and a *boundary* term:

$$= \sum_i \underbrace{\int_{D_i \setminus \partial D_i} \partial_\theta \left[k(\mathbf{x} - \mathbf{x}') w_i \right] \mathrm{d}A(\mathbf{x}')}_{\textbf{Interior}} + \underbrace{\int_{\partial D_i} \partial_\theta \mathbf{x}_b^\top \mathbf{n}_i(\mathbf{x}_b) \; k(\mathbf{x} - \mathbf{x}_b) w_i \, \mathrm{d}\ell(\mathbf{x}_b)}_{\textbf{Boundary}}. \tag{8}$$

The boundary integral is over region boundaries ∂D_i, where $\mathbf{n}_i$ is the outward-pointing normal and $\partial_\theta \mathbf{x}_b^\top \mathbf{n}_i$ measures the normal velocity of the boundary point $\mathbf{x}_b$ – its variation with θ in normal direction. The interior integrates the derivative of the integrand, which vanishes because the integrand is independent of θ. Therefore, the derivative simplifies to the boundary term:

$$= \sum_i \int_{\partial D_i} \partial_\theta \mathbf{x}_b^\top \mathbf{n}_i(\mathbf{x}_b) \; k(\mathbf{x} - \mathbf{x}_b) w_i \, \mathrm{d}\ell(\mathbf{x}_b). \tag{9}$$

Unfortunately, the region boundaries ∂D_i are unions of not necessarily connected subsets of $\partial\Omega$, which are difficult to construct explicitly if the surface has self-intersections. However, as we will show, this sum of integrals can be simplified to a single integral over the surface $\partial\Omega$ that is independent of winding numbers.

Special case: intersection-free surfaces. Before we continue with the general case of possibly self-intersecting surfaces, consider the simplified case without intersections for an intuition. We only distinguish two regions in this case: (1) the inner region $D_1 \coloneqq \Omega$ representing a solid shape bounded by its surface $\partial D_1 \coloneqq \partial\Omega$ where the normal $\mathbf{n}_1 \coloneqq \mathbf{n}$ aligns with the normal of the surface $\mathbf{n}$; and (2)

[1] We use $\partial_\theta f$ as shorthand notation for $\frac{\partial f}{\partial \boldsymbol{\theta}}$.

[2] The given setting has two key differences to differentiable rendering: (1) the interior term of the derivative vanishes, leaving only the boundary term; and (2) the discontinuity points are not object silhouettes (i.e., subsets of the surface), but the *entire* surface, which is trivial to sample.

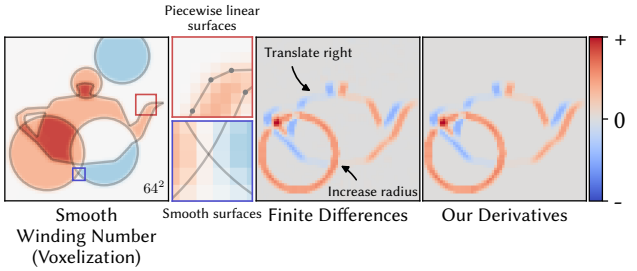

Fig. 2. We can differentiate voxelizations, i.e., point-sampled winding numbers, for arbitrary surfaces. This example shows piecewise linear surfaces and circles in 2D. Our derivatives match the finite difference reference. This example uses Monte Carlo integration and a box filter of diameter 3.

the outer region $D_2 \coloneqq \mathbb{R}^n \setminus \Omega$ with the same boundary $\partial D_2 \coloneqq \partial\Omega$ but *inverted* normal $\mathbf{n}_2 \coloneqq -\mathbf{n}$. Then Eq. (9) simplifies to

$$\partial_\theta \tilde{w}(\mathbf{x}) \overset{\partial\Omega \text{ without self-intersections}}{=} \int_{\partial\Omega} \partial_\theta \mathbf{x}_b^\top \mathbf{n}\, k(\mathbf{x}-\mathbf{x}_b)\, (\underbrace{w_1}_{:=1} - \underbrace{w_2}_{:=0})\, \mathrm{d}\ell(\mathbf{x}_b), \tag{10}$$

since the winding number reduces to the indicator function for intersection-free surfaces (see Eq. (3)).

General case. We now consider the general case of possibly self-intersecting surfaces. First note that the surface $\partial\Omega$ separates regions with piecewise constant winding number by definition[3], so let $\partial D_{i,j}$ denote the shared boundary between D_i and D_j. The union of shared boundaries is therefore simply the boundary itself

$$\partial D_i = \bigcup_{j \neq i} \partial D_{i,j}. \tag{11}$$

Splitting the integration domain in Eq. (9) into shared boundaries gives

$$\partial_\theta \tilde{w}(\mathbf{x}) = \sum_i \sum_{j \neq i} \int_{\partial D_{i,j}} \partial_\theta \mathbf{x}_b^\top \mathbf{n}_i(\mathbf{x}_b)\; k(\mathbf{x}-\mathbf{x}_b) w_i\; \mathrm{d}\ell(\mathbf{x}_b). \tag{12}$$

Note that the shared boundaries are a double cover of the surface $\partial\Omega$: Each part is considered twice, once as $\partial D_{i,j}$ with normal $\mathbf{n}_i$ and once as $\partial D_{j,i}$ with normal $\mathbf{n}_j = -\mathbf{n}_i$. Without loss of generality, assume that $\mathbf{n}_i$ aligns with the surface normal $\mathbf{n}$ on $D_{i,j}$, then we can write the integral as

$$= \sum_{\partial D_{i,j}} \int_{\partial D_{i,j}} \partial_\theta \mathbf{x}_b^\top \mathbf{n}(\mathbf{x}_b)\; k(\mathbf{x}-\mathbf{x}_b)(w_i - w_j)\; \mathrm{d}\ell(\mathbf{x}_b). \tag{13}$$

Since $w_i = w_j + 1$ by definition of the winding number, we have

$$\partial_\theta \tilde{w}(\mathbf{x}) = \int_{\partial\Omega} \partial_\theta \mathbf{x}_b^\top \mathbf{n}(\mathbf{x}_b)\; k(\mathbf{x}-\mathbf{x}_b)\; \mathrm{d}\ell(\mathbf{x}_b), \tag{14}$$

which is a surface integral independent of winding numbers. This result is the same as that for non-intersecting surfaces (c.f. Eq. (10)).

3.2 Optimizing with Integral Estimators

Computing the gradient $\partial_\theta E$ to solve an optimization problem involving the voxel grid, as in Eq. (5), boils down to three steps: (1) estimate the smoothed winding number field $\tilde{w}$ at the voxel centers $\mathbf{x_i}$ with Eq. (4), (2) evaluate the energy E to compute the *adjoint* winding number $\frac{\partial E}{\partial \tilde{w}(\mathbf{x_i})}$ for each voxel, and (3) estimate the gradient

$$\partial_\theta E = \sum_{\mathbf{i}} \frac{\partial E}{\partial \tilde{w}(\mathbf{x_i})} \partial_\theta \tilde{w}(\mathbf{x_i}) \tag{15}$$

as product of adjoints and winding number derivatives with Eq. (14).

Steps (1) and (3) are unique to differentiable voxelization, and both require estimating integrals as shown in Sec. 3.1. Since we have so far not made any assumption about the surface $\partial\Omega$, the kernel k, or the dimension, we will briefly discuss possible integration strategies for the general case (see Fig. 2 for results). For the most common case in practice – piecewise linear surfaces in 2D and 3D – we introduce a highly efficient closed-form algorithm in Sec. 3.3.

Smoothed winding number field. The smoothed winding number at voxel centers $\tilde{w}(\mathbf{x_i})$ can be evaluated by estimating the integral in Eq. (4). A straightforward approach would be quadrature on $\mathbb{R}^n$ – which, however, is prone to aliasing – or Monte Carlo integration. As a simple baseline, we have implemented a Monte Carlo estimator of Eq. (4) that uses stratified sampling with sub-voxel strata. The appeal of these direct approaches is that they can be trivially parallelized and use established rendering techniques because (1) samples in a voxel affect only their local neighborhood in the grid if the kernel k is (approximately) compact, and (2) the winding number can be computed by counting signed ray intersections with the surface, using a *single* ray in any direction.

The winding number computation can easily become a bottleneck in this approach. Sun et al. [2024] prove[4] that one can eliminate the winding numbers from the integral by transforming Eq. (4) into a surface integral using the divergence theorem:

$$\tilde{w}(\mathbf{x}) = \int_{\partial\Omega} \mathbf{F}_\mathbf{x}^\top(\mathbf{x}_b)\mathbf{n}(\mathbf{x}_b)\, \mathrm{d}\ell(\mathbf{x}_b), \tag{16}$$

where $\mathbf{F_x}$ is a vector field that satisfies $\nabla \cdot \mathbf{F_x}(\mathbf{x}') = k(\mathbf{x}-\mathbf{x}')$. It seems appealing to estimate this integral by simply sampling the boundary and assigning the respective sample contributions to all voxels where $\mathbf{F}_{\mathbf{x_i}}(\mathbf{x}_b) \neq 0$. However, this leads to visible "bar"-shaped correlation artifacts in the voxel grid, which is in line with what Sun et al. observe in preliminary quadrature experiments for the 3D case [Sun et al. 2024, Fig.14]. In 2D, their alternative is an artifact-free closed-form solution of Eq. (16) for Gaussian kernels. We derive a similar closed-form solution of Eq. (16) for piecewise linear surfaces in 2D and 3D (see Sec. 3.3). For the general case, we believe that more effective Monte Carlo estimators of Eq. (16) can be derived, but this is left for future work.

Differential integral. The gradient $\partial_\theta E$ is obtained by evaluating the product $\frac{\partial E}{\partial \tilde{w}(\mathbf{x_i})} \partial_\theta \tilde{w}(\mathbf{x_i})$ for each voxel center $\mathbf{x_i}$. This amounts to estimating the integral in Eq. (14) with the adjoint $\frac{\partial E}{\partial \tilde{w}(\mathbf{x_i})}$ moved inside of the integral. Similar to the winding number integral (Sec. 3.2), we have implemented a simple Monte Carlo baseline for the derivative of general kernels and shapes. The derivative is localized to the

[3] If the self-intersection has non-zero measure on the surface, the intersection creates a zero-volume region D_i with proper winding number.

[4] Their setting is the 2D case, but the proof generalizes to arbitrary dimensions.

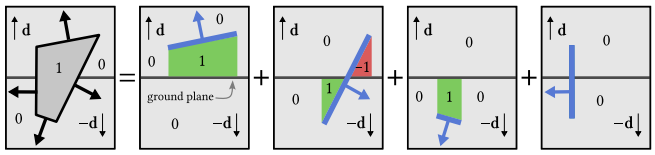

Fig. 3. The winding number field w of a surface depicted in the left-most box is the sum of the four patch-wise contributions w_i. Each w_i defines for every point in space how often the patch p_i (blue) is intersected in direction $\pm\mathbf{d}$. As the ray intersects the patch at most once, w_i is piecewise constant and either $-1, 1$ or 0 depending on the normal relative to the ground plane.

surface $\partial\Omega$ for (approximately) compact kernels k, so we "splat" contributions of surface samples into the voxel grid instead of solving the integral for each voxel separately.

3.3 Efficient Closed-Form Integration

While the evaluation in the general setting may require appropriately adjusted quadrature schemes, we notice that certain restrictions on the kernel and the surface representation simplify matters and lead to a very efficient algorithm – in particular for the practically relevant case of box filters and simplicial meshes. In this section, we will assume the 3D case. Given the explanation in 3D, the two-dimensional case or higher dimensions should be clear.

Winding numbers for piecewise surfaces. We begin by recalling and formalizing important established properties of the winding number [Grünbaum and Shephard 1990]: The winding number $w(\mathbf{x})$ at position $\mathbf{x}$ can be computed by summing the *'signed intersections'* between the surface $\partial\Omega$ and a single ray from $\mathbf{x}$ into an arbitrary direction $\mathbf{d_x}$: starting from zero, the winding number increases by one if $\mathbf{d}_\mathbf{x}^\mathsf{T}\mathbf{n} > 0$ and decreases by one if $\mathbf{d}_\mathbf{x}^\mathsf{T}\mathbf{n} < 0$, where $\mathbf{n}$ is the surface normal at the intersection. If we consider a division of the surface into patches p_i, these intersections can be computed patch-wise meaning the winding number at position $\mathbf{x}$ is the sum over all patch intersections:

$$w(\mathbf{x}) = \mathrm{sgnInt}(\mathbf{x}, \mathbf{d_x}, \partial\Omega) = \sum\nolimits_i \mathrm{sgnInt}(\mathbf{x}, \mathbf{d_x}, p_i) = \sum\nolimits_i w_i(\mathbf{x}, \mathbf{d_x}), \tag{17}$$

where $\mathrm{sgnInt}(\mathbf{x}, \mathbf{d_x}, \partial\Omega)$ is the sum of 'signed intersections' of the ray $\mathbf{x} + t\mathbf{d_x}$, $t \in \mathbb{R}$ and the surface $\partial\Omega$, which is divided into patches p_i, where $p_i \subset \partial\Omega$ is an arbitrary disjunct patch division ($p_i \cap p_j = \emptyset$ for $i \neq j$) of the surface $\partial\Omega = \bigcup_i p_i$. This results in a 'signed intersections' (or winding number contribution) field $w_i(\mathbf{x}, \mathbf{d_x})$ per patch. The idea is now to define a direction $\mathbf{d_x}$ per point $\mathbf{x}$ in space and a patch division that allows us to compute w_i efficiently.

We make two choices: (1) we set $\mathbf{d_x}$ to be constant; and (2) assume that each surface patch has at most one intersection with any ray in the direction $\mathbf{d}_x$. This is true, importantly, for triangles[5]. Given the observation that now w_i changes along a ray in direction $\mathbf{d_x}$ only at an intersection with the surface patch p_i from -1 or 1 to 0 and is 0 everywhere else, we see that w_i is piecewise constant and either $-1, 1$ or 0. A complete example is depicted in Fig. 3. For efficiency reasons, which we will describe later, we actually define two different directions $\mathbf{d}$ and $-\mathbf{d}$, pointing in opposite directions, divided by a *ground plane* such that the direction in each half-space is orthogonal to the ground plane and points away from it.

Box filtering on voxel grids. So far, we considered the winding number and not the *filtered* winding number. If we restrict the kernel to a box filter aligned with the voxels, the sample values for the voxels taken at their centers become:

$$\tilde{w}(\mathbf{x_i}) = \int_{\mathbb{R}^n} \mathrm{box}(\mathbf{x_i} - \mathbf{x}')w(\mathbf{x}')\,\mathrm{d}A(\mathbf{x}') = \frac{1}{|\square_{\mathbf{x_i}}|}\int_{\square_{\mathbf{x_i}}} w(\mathbf{x}')\,\mathrm{d}A(\mathbf{x}'), \tag{18}$$

where $|\square_{\mathbf{x_i}}|$ is the volume of the voxel (i.e., of the box filter) and

$$\square_{\mathbf{x_i}} = \left\{\mathbf{y} \mid \mathbf{x}_{\mathbf{i},j} - \frac{\mathbf{l}_j}{2} \leq \mathbf{y}_j < \mathbf{x}_{\mathbf{i},j} + \frac{\mathbf{l}_j}{2}, j \in [1, \ldots, n]\right\}, \tag{19}$$

where $(\cdot)_j$ is the vector entry in dimension j and $\mathbf{l}$ is the vector with the side lengths of the voxel. So $\tilde{w}$ evaluated at the voxel center is just the *average* winding number inside the voxel.

If a patch does not cross the ground plane, the corresponding w_i takes only two values: 0 and either 1 or -1. In this case, we can interpret w_i as the winding number field of the oriented *shadow surface* of the patch. The shadow surface bounds the *shadow solid*, which is the region between the patch and its 'shadow' – the (orthogonal) orthographic projection onto the ground plane (c.f. colored area in Fig. 3). The orientation of the shadow surface is consistent with the patch normal and determines the sign of the shadow solid volume. Substituting Eq. (17) into Eq. (18) shows that we can compute the average winding number inside a voxel by summing over all average winding numbers of all shadow surfaces. This is simply the signed volume of the intersection of the shadow solid with each voxel, divided by the volume of the voxel.

Computation for a triangle mesh. Assuming the surface is a triangle mesh, the general idea is to compute the intersection of each triangle with all voxels and re-triangulate the resulting polygons. This results in new sub-triangles that are fully contained within a voxel. If the ground plane is aligned with the voxel boundaries, no sub-triangle will cross it, making the shadow solid perspective applicable. As the shadow solid of each sub-triangle is a truncated prism that covers only voxels within a column of the grid, the signed volume of the voxel intersections can be easily computed:

(1) Choose a ground plane such that $\mathbf{d}$ is aligned with one of the grid axes and represent all points in this coordinate system. W.l.o.g., set $(x, y, 0)$ as the ground plane and $\mathbf{d} = (0, 0, 1)$.
(2) For each triangle, compute the intersection with the voxel grid. This gives convex polygons with degrees between 3 and 9 that are each fully contained in a single voxel (i, j, k). Each polygon is further triangulated into sub-triangles by fixing a vertex and walking over the edges not incident on that vertex. This results in up to 7 sub-triangles per polygon.
(3) The non-zero contribution of each sub-triangle needs to be distributed to the voxel (i, j, k) and all the voxels below, down to the ground plane at $(i, j, 0)$.
(3.1) The contribution to voxel (i, j, k) is the signed volume of the shadow solid, minus the contribution to the voxels below. The signed shadow solid volume is the signed area of the

[5]Apart from degenerate cases, which can be ignored because their contribution will later become zero.

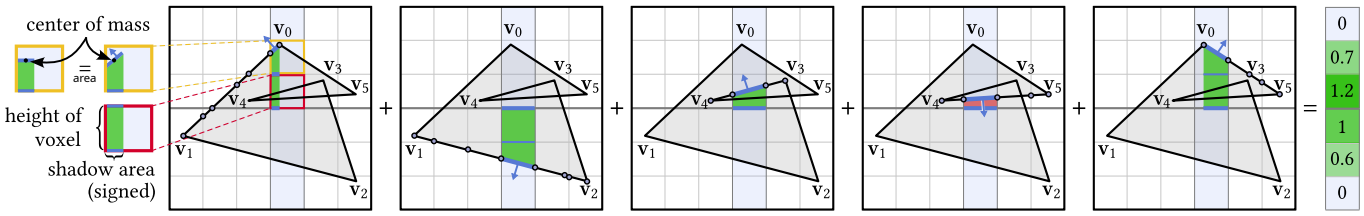

Fig. 4. Given a polygon, the integrated winding numbers in each voxel are updated whenever a simplex intersects a voxel above the voxel. The volume of the intersection of the signed shadow solid volume (green for positive and red for negative contribution) and a voxel (small boxes) can be computed as described in Sec. 3.3 and added to the value of each voxel, resulting in an update of all voxel values 'below' the simplex segment. Therefore, the integrated winding number of all voxels in the highlighted grid column can be computed as the sum of 'shadow' contributions of the five blue segments.

shadow of the patch on the ground plane – the signed area of the triangle ignoring the third coordinate – times the average height of the triangle over the ground plane – the third coordinate of the centroid of the sub-triangle.

(3.2) The contribution to each voxel below (i, j, k) is the signed shadow area times the side length of each voxel in direction $\mathbf{d}$. This can be done efficiently for all voxels in a process very similar to the scan-conversion of primitives.

Fig. 4 visualizes this process in 2D. This algorithm is a special case of the scanline algorithm of Manson and Schaefer [2013] extended to 3D, who show that this method can also be extended to other filters. A pseudo-code with a ground plane at the bottom of the voxel grid is provided in the supplementary material.

Acceleration. In practice, we use a ground plane at the center of the voxel grid, which halves the average distance of voxels to the ground plane and reduces the number of write operations for meshes where most triangles are near the ground plane. Alternatively, it is possible to first gather the downward contributions of each voxel and then distribute them downward in a second pass.

Derivatives. Given the grid-sized box filter, by Eq. (14), the derivative of the winding number at the voxel centers is

$$\partial_\theta \tilde{w}(\mathbf{x_i}) = \int_{\partial\Omega} \partial_\theta \mathbf{x}_b^\top \mathbf{n}(\mathbf{x}_b)\ \mathrm{box}(\mathbf{x}-\mathbf{x}_b)\ \mathrm{d}\ell(\mathbf{x}_b) \tag{20}$$

$$= \frac{1}{|\square_{\mathbf{x_i}}|} \int_{\partial\Omega\cap\square_{\mathbf{x_i}}} \partial_\theta \mathbf{x}_b^\top \mathbf{n}(\mathbf{x}_b)\ \mathrm{d}\ell(\mathbf{x}_b). \tag{21}$$

For a piecewise linear mesh where all triangles t_i are entirely within the voxel ($t_i = t_i \cap \square_{\mathbf{x}}$), this reduces to

$$\partial_\theta \tilde{w}(\mathbf{x_i}) = \frac{1}{|\square_{\mathbf{x_i}}|} \sum_{t_i} \sum_{\mathbf{v}_j \in t_i} \frac{1}{3} \partial_\theta \mathbf{v}_j^\top\ \mathbf{n}_{t_i}\ \mathrm{Area}(t_i), \tag{22}$$

where $\mathbf{v}_j$ are the three vertices of the triangle t_i. This is because the normal vector $\mathbf{n}_{t_i}$ is constant over the triangle and the derivative of the surface over a triangle with respect to the vertices ($\partial_{\mathbf{v}}\mathbf{x}_b$) is a linear function, going from 1 at the respective vertex to 0 at the opposite edge, making it integrable by taking the center of mass of the triangle times its area. If we now assume arbitrary triangles, possibly spanning multiple voxels, we can divide each triangle into sub-triangles $s_j \in (t_i \cap \square_x)$, as described above. According to Eq. (22), we can compute the derivative explicitly by

$$\partial_\theta \tilde{w}(\mathbf{x_i}) = \frac{1}{|\square_{\mathbf{x_i}}|} \sum_{(\partial\Omega \ni t_i)\cap\square_{\mathbf{x_i}}} \sum_{s_j \in (t_i\cap\square_{\mathbf{x_i}})} \sum_{\mathbf{w}_k \in s_j} \frac{1}{3} \partial_\theta \mathbf{w}_k^\top \mathbf{n}_{t_i}\ \mathrm{Area}(s_j). \tag{23}$$

The three vertices $\mathbf{w}_k$ of the sub-triangle s_j are expressed as linear combinations of the original triangle t_i vertices ($\mathbf{v}_j$ in Eq. (22)), ensuring the dependence on θ. Therefore, the derivatives can be computed using the same structure as the voxelization algorithm, without the downward contribution, if we represent the triangle-voxel intersections in barycentric coordinates. This can be seen by comparing the voxelization pseudocode Alg. 1 with the differentiation in Alg. 2 and Alg. 3 in the supplementary material.

Box filters of different sizes. The box filter can be larger than a grid cell. The Monte Carlo implementation Sec. 3.2 handles this out-of-the-box (see Fig. 2). For the closed-form integration, one can apply another discrete filter to the already box-filtered winding number, as computed above. If the second filter is again a box filter, the composition of voxelization and discrete convolution is equivalent to using a single box filter with integer-scaled support.

4 Evaluation and Applications

We implemented the algorithms using C++ with Python bindings for PyTorch [Paszke et al. 2019]. We will first start by investigating the performance of our approach and comparing it with another differentiable voxelization technique. Afterwards, we demonstrate the versatility of our differentiable voxelization with applications for self-intersection resolving (Sec. 4.2), optimization for bandsaw cutting (Sec. 4.3), and space tiling using shapes in $\mathbb{R}^3$ (Sec. 4.4).

4.1 Evaluation

Runtime. Computation time is a limiting factor for many volumetric representations, as the number of voxels grows cubically with the number of voxels in each dimension. We measure the computation time of our implementation in 32-bit floating-point precision on a 12th Gen Intel Core i9-12900K, for both the voxelization and the backward-mode gradient computation. We tested multiple meshes with different numbers of faces on grids between 16^3 up to 1024^3 voxels, by averaging the execution time over 30 runs per mesh and grid size (Fig 5). Even though the implementation is not fully optimized (for example, it is not parallelized), the computation times stay below a second even for meshes with more than 100, 000 faces on 1024^3

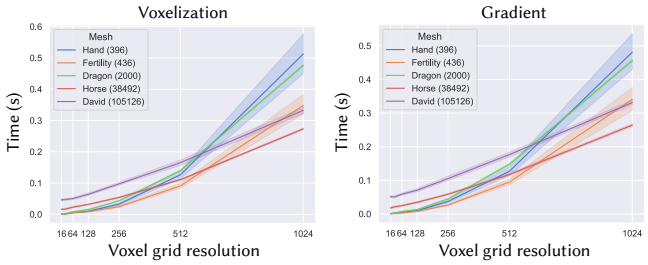

Fig. 5. For high grid resolutions (up to 1024 voxels in each dimension), the runtime stays below a second even for meshes over 100,000 faces for voxelization as well as the gradient computation. All timings are averaged over 30 repetitions and were computed on an Intel i9-12900K CPU. The number of faces is shown in parentheses.

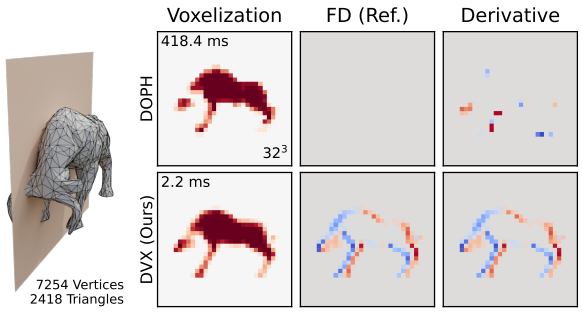

Fig. 6. Luo et al. [2024] propose differentiable voxelization by automatically differentiating through a (non-smoothed) winding number (DOPH). Since the winding number is piecewise constant, the derivatives of their voxelization are zero if computed numerically with finite differences (FD), and their automatic derivatives are only spuriously non-zero very close to the surface. In contrast, our technique (DVX) is based on the smoothed winding number, which leads to expected numerical derivatives that match our results exactly.

voxels. This makes it practical for optimizations with many gradient descent iterations. Interestingly, the overhead of the David mesh for low-resolution grids disappears for larger grid sizes. The reason for this is likely because the number of intersections needed to be computed for low-resolution meshes increases cubically, whereas for the David mesh it remains constant, since the triangles are typically contained within a voxel. It is very likely that, for even finer voxel grids, the David mesh will perform worse than simple meshes again.

Preliminary experiments with a naive GPU implementation indicate significant potential for performance improvements by exploiting parallelism (speedup of 3.6× for voxelization and 14.5× for gradients, for meshes with more than 100,000 faces on 1024^3 grids). We leave an optimized GPU implementation for future work.

Comparison with DOPH. Differentiable voxelization of surfaces has been attempted in previous work by Luo et al. [2024]: They point-sample the *non-smoothed* winding number $w(\mathbf{x_i})$ and use trilinear interpolation to get a smooth result. The derivative is then obtained with automatic differentiation. This approach corresponds

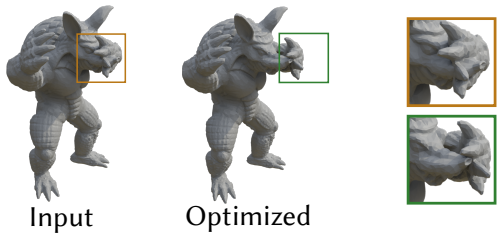

Fig. 7. Differentiable voxelization allows resolving self-intersections by an ARAP deformation of the input shape.

to "Discretize-then-Differentiate" [Bangaru et al. 2021], which is known to produce incorrect derivatives: Since the winding number is piecewise constant, the derivative of their voxelization is zero almost everywhere. Our approach (Sec. 3) is based on point-sampling the smoothed winding number field $\tilde{w}(\mathbf{x_i})$, which leads to the expected derivatives (see Fig. 6).

4.2 Self-Intersection Resolving

Given a winding number, it is easy to detect self-intersections where multiple parts of the same surface overlap: these are all positions where the winding number is larger than one. Self-intersections may appear due to insufficient sampling of parametric models or as a result of many geometric optimization methods that do not penalize self-intersections such as ARAP surface deformation [Sorkine and Alexa 2007]. While Sun et al. [2024] showed that their method is suitable to resolve self-intersections for planar curves, our method can be used to resolve self-intersections for surfaces in $\mathbb{R}^3$.

Starting with a voxelization $\mathbf{W} \in \mathbb{R}^{d\times d\times d}$ of resolution d of a triangle mesh $(\mathbf{V}, \mathbf{F})$, we use an energy functional that penalizes voxels with a value higher than one: $E_{\text{sect}} = \sum_{\mathbf{W}_{ijk}>1} \mathbf{W}_{ijk}^2$. Due to numerical errors, it is advisable to filter values $\mathbf{W}_{ijk} > 1 + \epsilon$ instead (for small $\epsilon = 0.001$) as some values of $\mathbf{W}$ can be larger than one.

Optimizing E_{sect} using gradient descent resolves self-intersections in the input mesh. To prioritize rigid deformations of the input, we regularize the self-intersection energy using the as-rigid-as-possible (ARAP) energy $E = E_{\text{sect}} + \alpha E_{\text{arap}}$ [Sorkine and Alexa 2007] as it is often used as regularization for surface deformation [Bouaziz et al. 2023; Dinh et al. 2025; Huang et al. 2021; liu and Jacobson 2019]:

$$E_{\text{arap}}(\bar{\mathbf{V}}) = \min_{\mathbf{R}_i \in SO(3)} \sum_{i\in\mathbf{V}}^{n} \sum_{(j,k)\in\mathcal{N}(i)} w_{jk} ||\mathbf{e}_{jk} - \mathbf{R}_i \bar{\mathbf{e}}_{jk}||^2 \qquad (24)$$

where $\mathcal{N}(i)$ are all spokes and rims half edges $\mathbf{e}_{jk} = \mathbf{v}_k - \mathbf{v}_j$ for vertex i in the meshes [Chao et al. 2010]. Note that this energy is fully differentiable, and optimal rotations $\mathbf{R}_i$ between vertex stars can be found by singular value decompositions (for further details, we refer to Sorkine and Alexa [2007]).

We minimize the regularized self-intersection energy with $\alpha = 0.005$ by gradient descent using the Adam method [Kingma and Ba 2017] and a learning rate of 10^{-2}. The optimization is iterated until all voxels with $\mathbf{W}_{ijk} > 1$ are resolved (Fig. 7).

This experiment demonstrates that differentiable physical simulations that require cumbersome collision detection [Cleac'h et al.

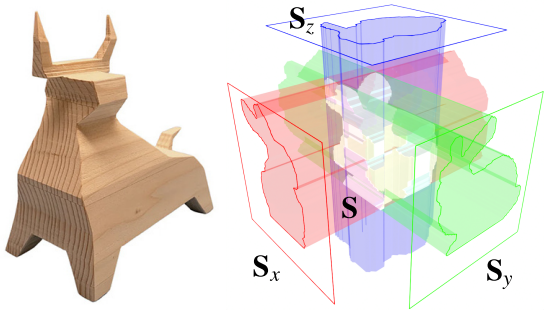

Fig. 8. Left: Wooden figure that was created by cutting along handmade cut silhouettes [Gauthier 2019] (used with permission). Right: The cut out surface is defined as the intersection of extruded silhouettes.

2023; Qiao et al. 2020] could benefit from our fast differential collision detection and resolution. Note that self-intersections are resolved only approximately. There may be intersections in half-empty voxels that are not penalized by E_{sect}, as the energy only integrates the winding number over the entire voxel. Due to volume inversions, the winding number may also be negative. This could hide self-intersections in the optimization objective.

4.3 Shape Deformation for Bandsaw Cutting

Cutting objects from wooden blocks using orthogonal cutting directions restricts the space of shapes that can be created by sawing. For example, it is not possible to create certain types of concavities or surfaces with normals that are not orthogonal to at least one cut direction. Every *cut out* surface is defined by a set of silhouettes, one for each cut direction. Gauthier [2019] offers such handmade cut silhouettes that result in beautiful wooden figures as depicted in Fig. 8 (left).

A naive way to cut a given object is to directly use its silhouettes in x, y and z direction for sawing. This often leads to shapes that strongly differ from their initial shape due to material that is not removed but does not belong to the input shape (Fig. 10, top row). Our idea is to find small deformations of the input that make the shape more cutable but also preserve its overall shape.

We use a triangle mesh with vertices $\mathbf{V}$ and face indices $\mathbf{F}$ as input and compute the voxelized shape $\mathbf{W} \in \mathbb{R}^{d\times d\times d}$. We generate rasterized filled silhouettes $\mathbf{S}_x, \mathbf{S}_y$ and $\mathbf{S}_z \in \mathbb{R}^{d\times d}$ using the mellowmax operator [Asadi and Littman 2017], specifically $(\mathbf{S}_x)_{ij} = 1/w \ln(1/d \sum_k \exp(w\mathbf{W}_{kij}))$. For $w \to \infty$ the mellowmax converges to the maximum of the elements. We use $w = 1000$. The gradient of the mellowmax is the softmax. If multiple voxels have the same value, then they contribute equally to the gradient.

The voxelized cut shape $\mathbf{S} \in \mathbb{R}^{d\times d\times d}$ is then given by product of the extruded silhouettes, i.e., $\mathbf{S}_{ijk} = (\mathbf{S}_x)_{jk}\ (\mathbf{S}_y)_{ik}\ (\mathbf{S}_z)_{ij}$ (Fig. 8, right). The cut energy that measures the squared voxelized volume differences between the cut shape and the voxelized input is: $E_{cut} = \frac{1}{d^3} \sum_{ijk} (\mathbf{S}_{ijk} - \mathbf{W}_{ijk})^2$.

In Fig. 9 we visualize the steps to compute the energy function E_{cut}. Note that $\mathbf{W}$ contains winding numbers that are greater than one for self intersections. This intentionally penalizes self intersections that require additional volume. The final energy is the cut energy regularized by the as-rigid-as-possible energy $E = E_{cut} + \alpha E_{arap}$. The parameter α weights between the additional volume and the deformation cost. We minimize the energy with $\alpha = 0.05$ by gradient descent using the Adam method [Kingma and Ba 2017] with a learning rate of 10^{-3}.

We limit the resolution of the voxelization to $d = 128$. When cutting a wooden block of 10cm^3, this gives a resolution of $< 0.8\text{mm}^3$, which is below the precision of many standard bandsaws.

In Fig. 10 we show various meshes optimized for improved bandsaw cutting. These deformations preserve the overall meaning of the shapes, but substantially improve the cut object when compared to the naive cutting strategy without deformation. We demonstrate the 'cutability' of the generated surfaces by cutting various examples out of wood with a bandsaw and out of polystyrene foam with a foam cutter (Fig. 11).

To quantify the improved volume difference, we applied the optimization to the TOSCA data set [Bronstein et al. 2008]. For each optimized mesh, we compute the relative volume improvement between initial voxelized cut shape and cut shape for the voxelized optimized shape. We also compute the Hausdorff and mean surface distance between the initial and deformed triangle meshes using METRO [Cignoni et al. 1998] to quantify the deformation (Tab. 1 in the supplementary material).

4.4 Space-filling Shapes in $\mathbb{R}^3$

Inspired by the tessellations of M.C. Escher (see Kaplan and Salesin [2000] for comparable examples), we can use differentiable voxelization to generate space-filling shapes in $\mathbb{R}^3$. Given a set of shapes, the goal is to find a deformation such that the voxelized shapes fill the entire space without overlapping. Escher created his tilings by drawing textures for various known tilings such that the textures connect continuously with adjacent tiles. The textures contained figures of one or more animals that are repeated within the texture. Based on this idea, we optimize repeated shapes in a cube such that shapes at the boundary of the cube continuously connect to shapes on the opposite boundary. All shapes inside the cube should be identical, up to translation and rotation, and all repetitions (*instances*) of a shape should be similar.

For practical applications, this can be seen as instance of a packing problem. The goal is to find a small deformation for a given shape that maximizes the number of shapes that fit into a domain.

Outputs of the optimization are the deformed vertex positions for each shape, as well as a translation and rotation (parametrized by yaw, pitch and roll) for each instance of a shape. For initialization, we randomly position each shape multiple times inside the cube such that each mean of vertices is inside $[-\frac{1}{3}, \frac{1}{3}]^3$. We refer to this cube as *tiling cube*, which can later be repeated to fill $\mathbb{R}^3$. This initialization might cause some shapes to stick out of the tiling cube. The idea is that everything outside should enter the cube on the opposite site to create a space-filling that is a continuous transitions between opposite sides of the cube. To achieve this, we voxelize the entire $[-1, 1]^3$ cube for each mesh j using the differential voxelizer on a voxel grid $\mathbf{C}_j \in \mathbb{R}^{d\times d\times d}$. This $[-1, 1]^3$ cube is split in $3^3 = 27$ smaller

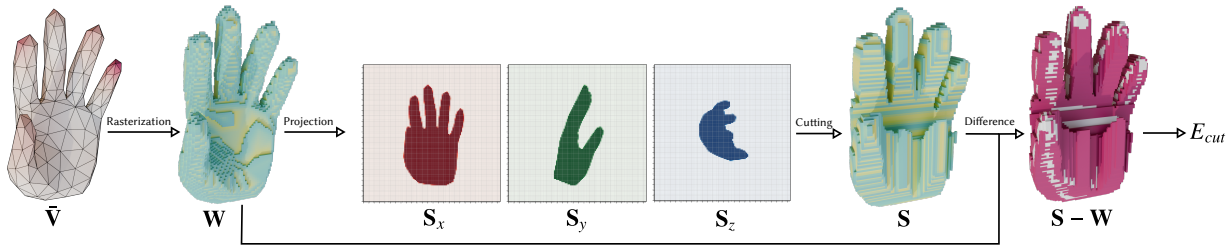

Fig. 9. The energy E_{cut} measures the volume difference between the deformed mesh $\bar{\mathbf{V}}$ and the deformed cut out shape S. Larger vertex distances of the deformed mesh $\bar{\mathbf{V}}$ to $\mathbf{V}$ are shown in red (left). $\mathbf{W}$ is the discretized shape of $\bar{\mathbf{V}}$ where yellow voxels have high volume and green voxels have low volume. Voxels with volume lower than 10^{-3} are hidden. $\mathbf{W}$ is projected to the silhouette shapes $\mathbf{S}_x, \mathbf{S}_y$ and $\mathbf{S}_z$ that are used to generate the cut shape S. For visualization of the volume difference $(\mathbf{S} - \mathbf{W})$, we highlight all voxels in red that are part of the shape $\mathbf{S}_{ijk} > 0.5$ but not in $\mathbf{W}_{ijk} > 0.5$.

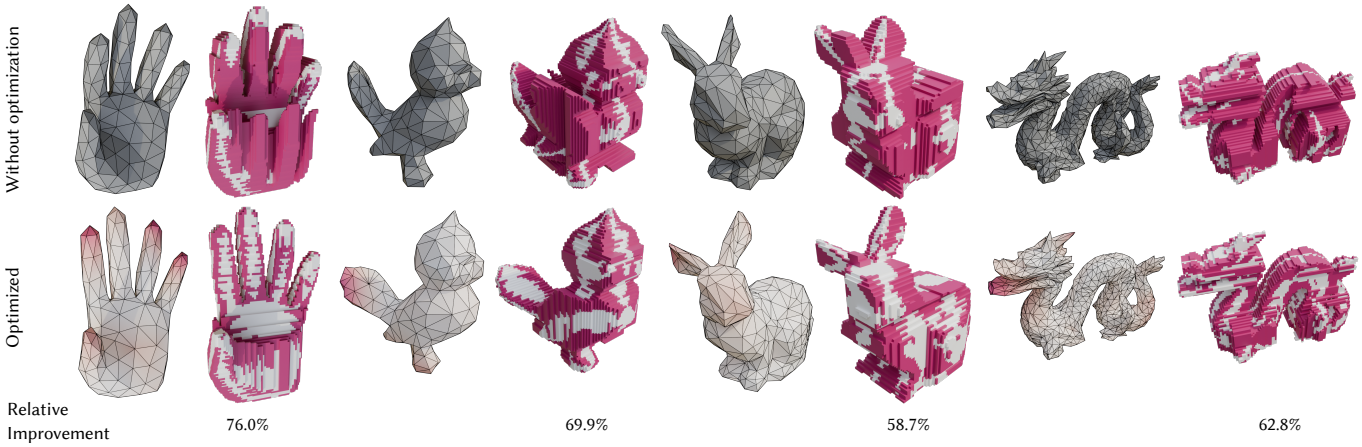

Fig. 10. When optimized (bottom row), much more material that does not belong to the actual shape can be removed by cutting, when compared to the cut-out shape for meshes without optimization (top row). All voxels of $\mathbf{S}_{ijk} > 0.5$ that do not belong to $\mathbf{W}_{ijk} > 0.5$ are highlighted in red. The color of the optimized triangle meshes shows the difference to the initial meshes. Gray voxels are part of the voxelization of the triangle meshes and red voxels are additional voxels that are not removed by cutting. The relative improvements denote how much unnecessary material of the initial meshes can be removed with the optimization $(1 - E_{cut}(\bar{\mathbf{V}})/E_{cut}(\mathbf{V}))$. Minimal deformations, such as straightening the fingers of the hand or adjusting the tail and ears of Tweety and Bunny, significantly improve the shapes' cutability.

Fig. 11. We test the 'cutability' of optimized cut shapes by cutting various optimized shapes using a bandsaw (left top and bottom) and a polystyrene foam cutter (bottom right). The references for the cut shapes are at the top right.

cubes $\bar{\mathbf{C}}_{ji} \in \mathbb{R}^{d/3 \times d/3 \times d/3}$ of side length $\frac{1}{3}$ (Fig. 12, left). The sum of all splitted cubes $\sum_i \bar{\mathbf{C}}_{\mathbf{ji}}$ gives a repeatable tiling for each shape. We optimized the differentiable energy that measures the coverage of the space by summation over all shapes and splitted cubes $\sum_{ij} \bar{\mathbf{C}}_{\mathbf{ji}}$:

$$E_{\text{fill}} = \sum_{klm} \left(\left(\sum_{ij} \bar{\mathbf{C}}_{ji} \right)_{klm} - 1 \right)^2.$$

This energy becomes zero when each voxel in the tiling cube is fully covered by one shape without self intersections. For the final energy, we again use the ARAP energy E_{arap} as regularization: $E = E_{\text{fill}} + \alpha \sum_j E_{\text{arap}}(\bar{\mathbf{V}}_j)$.

Note that in practice E_{arap} only needs to be computed for each class of shapes once in each iteration. We minimize the regularized energy by gradient descent with $\alpha = 0.005$ using the Adam method [Kingma and Ba 2017] and a learning rate of 10^{-3}.

We use a resolution of $d = 192$. When split into multiple cubes, it creates a tileable cube with a resolution of $d/3 = 64$. The stopping criterion is reached when the change in all vertex positions is less

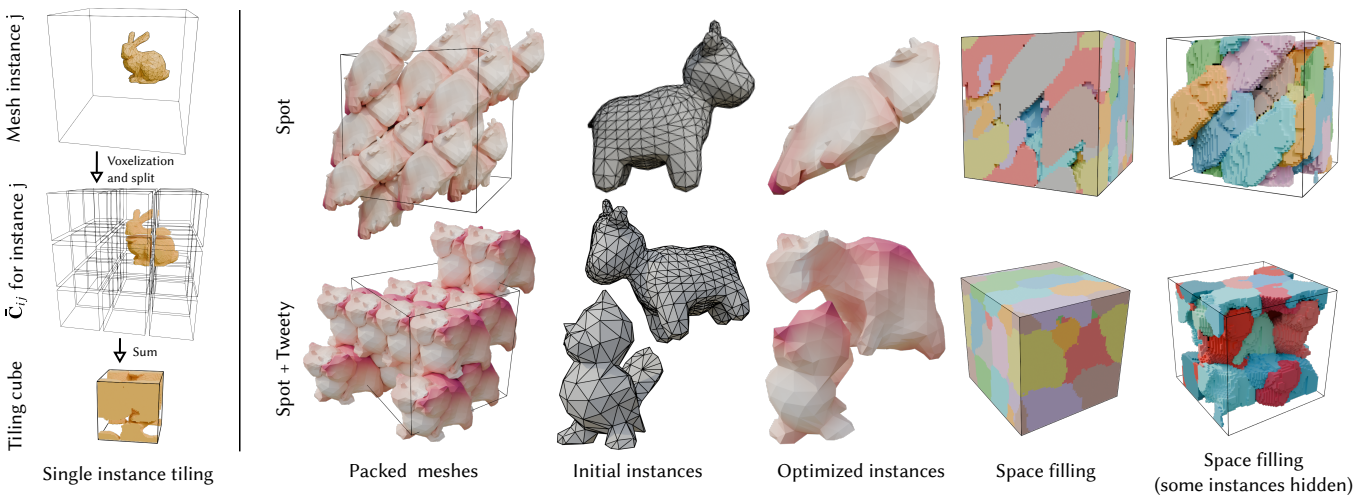

Fig. 12. Differentiable voxelization can be used to optimize tilings in $\mathbb{R}^3$ to periodically fill the space. Left: Tiling procedure for a single shape instance. Right: Each row belongs to one optimized packing. The packed meshes (first column) can be repeated along all three axis to fill the space. The packed meshes are composed of one or more deformed meshes that are all similar (within a class of animals). When voxelized and repeated in all dimensions, the packing fills $\mathbb{R}^3$ (third column). The last column shows the interior of the space filling voxels by hiding some instances (for the space filling for multiple animal types with some hidden instances, all Tweety birds are shaded in reds and all Spots are shaded in blues).

than 10^{-3}. We optimize a tileable cube filling with 27 Spots (Fig. 12, right side and top row) and 27 hands (Fig. 1). We also experimented with two meshes, Tweety and Spot, to optimize a tileable cube filled with 14 Tweety birds and 14 Spots (Fig. 12 right side and bottom row). Although the meshes do not fill the entire volume and may not be intersection-free, the experiments show near tileable results.

5 Discussion and Future Work

There are operations for which the regular grid structure significantly simplifies the computation. Nevertheless, surface representations are often the more prevalent representation in comparison to volumetric grids, and many methods specifically rely on triangular meshes, including the classical ARAP energy. This work transfers the simplicity of many problems that can be solved on voxel grids to common explicit surface representations. Optimizing cut shapes (Sec. 4.3) or space-fillings in $\mathbb{R}^3$ are non-trivial and challenging problems and we are unaware of any other methods apart from voxelization that could handle these problems.

Differentiable voxelization opens many possibilities for applications that benefit from voxel grids, which we could not cover in the application section (Sect. 4). This includes (1) applications in CSG, as intersections, differences, and unions of shapes are particularly simple to compute on grids. Morphological operators (2) can also be easily formulated on grids. Differentiable voxelization provides a method to differentiate through these operations back to a triangular mesh. This might inspire further applications for thin shell modeling. Many applications in machine learning (3) benefit from grid structures. This opens the possibility of directly comparing shapes in $\mathbb{R}^3$ in a simple differentiable way, instead of relying on image-based losses for shapes in $\mathbb{R}^3$ [Yuan et al. 2024].

Interestingly, the inverse problem of generating a surface mesh from a voxel representation has been solved many times using differentiable algorithms [Remelli et al. 2020b]. However, our direction has not received much attention. Closing this loop efficiently enables the generation of self-supervised optimization to improve algorithms anywhere within the loop.

Besides applications that could benefit from differentiable voxelization, we see several possibilities to decrease computation time by parallelization. Although this work focuses mainly on triangular meshes, this approach can be further extended and evaluated for more surface representations or different smoothing filters that are fast to integrate, such as polynomials [Manson and Schaefer 2013].

Acknowledgments

The authors would like to thank Maximilian Kohlbrenner for the helpful discussions on winding numbers and for sharing source code, and the anonymous reviewers for their valuable comments. This work was funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Grant agreement No. 101055448, ERC Advanced Grand EMERGE).

References

John Amanatides and Andrew Woo. 1987. A Fast Voxel Traversal Algorithm for Ray Tracing. In *EG 1987-Technical Papers*. Eurographics Association. doi:10.2312/egtp.19871000

Arthur Appel. 1968. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference* (Atlantic City, New Jersey) *(AFIPS '68 (Spring))*. Association for Computing Machinery, New York, NY, USA, 37–45. doi:10.1145/1468075.1468082

Kavosh Asadi and Michael L. Littman. 2017. An Alternative Softmax Operator for Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 243–252.

Marco Attene, Marcel Campen, and Leif Kobbelt. 2013. Polygon mesh repairing: An application perspective. *ACM Comput. Surv.* 45, 2, Article 15 (March 2013), 33 pages. doi:10.1145/2431211.2431214

Sai Praveen Bangaru, Jesse Michel, Kevin Mu, Gilbert Bernstein, Tzu-Mao Li, and Jonathan Ragan-Kelley. 2021. Systematically differentiating parametric discontinuities. *ACM Trans. Graph.* 40, 4, Article 107 (July 2021), 18 pages. doi:10.1145/3450626.3459775

Gavin Barill, Neil G. Dickson, Ryan Schmidt, David I. W. Levin, and Alec Jacobson. 2018. Fast winding numbers for soups and clouds. *ACM Trans. Graph.* 37, 4, Article 43 (July 2018), 12 pages. doi:10.1145/3197517.3201337

Josh Barnes and Piet Hut. 1986. A Hierarchical 0 Nlogn Force Calculation Algorithm. *Nature* 324 (1986), 446–449. doi:10.1038/324446a0

Alexandre Binninger, Ruben Wiersma, Philipp Herholz, and Olga Sorkine-Hornung. 2025. TetWeave: Isosurface Extraction using On-The-Fly Delaunay Tetrahedral Grids for Gradient-Based Mesh Optimization. *ACM Transactions on Graphics (TOG)* 44, 4 (2025), 1–19.

Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. 2010. *Polygon mesh processing*. CRC press.

Mario Botsch and Olga Sorkine. 2008. On Linear Variational Surface Deformation Methods. *IEEE Transactions on Visualization and Computer Graphics* 14, 1 (2008), 213–230. doi:10.1109/TVCG.2007.1054

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. 2023. *Projective Dynamics: Fusing Constraint Projections for Fast Simulation* (1 ed.). Association for Computing Machinery, New York, NY, USA.

Alexander M Bronstein, Michael M Bronstein, and Ron Kimmel. 2008. *Numerical geometry of non-rigid shapes*. Springer Science & Business Media.

Isaac Chao, Ulrich Pinkall, Patrick Sanan, and Peter Schröder. 2010. A simple geometric model for elastic deformations. *ACM Trans. Graph.* 29, 4, Article 38 (July 2010), 6 pages. doi:10.1145/1778765.1778775

Hanyu Chen, Bailey Miller, and Ioannis Gkioulekas. 2024. 3D Reconstruction with Fast Dipole Sums. *ACM Trans. Graph.* 43, 6, Article 192 (Nov. 2024), 19 pages. doi:10.1145/3687914

Zhiqin Chen and Hao Zhang. 2019. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 5939–5948.

P. Cignoni, C. Rocchini, and R. Scopigno. 1998. Metro: Measuring Error on Simplified Surfaces. *Computer Graphics Forum* 17, 2 (1998), 167–174. doi:10.1111/1467-8659.00236

Simon Cleac'h, Hong-Xing Yu, Michelle Guo, Taylor Howell, Ruohan Gao, Jiajun Wu, Zac Manchester, and Mac Schwager. 2023. Differentiable Physics Simulation of Dynamics-Augmented Neural Objects. *IEEE Robotics and Automation Letters* PP (05 2023), 1–8. doi:10.1109/LRA.2023.3257707

Cyril Crassin and Simon Green. 2012. Octree-Based Sparse Voxelization Using The GPU Hardware Rasterizer. In *OpenGL Insights*, Patrick Cozzi and Christophe Riccio (Eds.). CRC Press. doi:10.1201/b12288

Jiajun Deng, Shaoshuai Shi, Peiwei Li, Wengang Zhou, Yanyong Zhang, and Houqiang Li. 2021. Voxel R-CNN: Towards High Performance Voxel-based 3D Object Detection. *Proceedings of the AAAI Conference on Artificial Intelligence* 35, 2 (May 2021), 1201–1209. doi:10.1609/aaai.v35i2.16207

Nam Anh Dinh, Itai Lang, Hyunwoo Kim, Oded Stein, and Rana Hanocka. 2025. Geometry in Style: 3D Stylization via Surface Normal Deformation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 28456–28467.

Elmar Eisemann and Xavier Décoret. 2008. Single-pass GPU solid voxelization for real-time applications. In *Proceedings of Graphics Interface 2008* (Windsor, Ontario, Canada) *(GI '08)*. Canadian Information Processing Society, CAN, 73–80.

Gerald E Farin. 2002. *Curves and surfaces for CAGD: a practical guide*. Morgan Kaufmann.

Pierre-Luc Gauthier. 2019. *Brikologik*. https://brikologik.com/en/collections/all

Branko Grünbaum and G C Shephard. 1990. Rotation and winding numbers for planar polygons and curves. *Trans. Am. Math. Soc.* 322, 1 (1990), 169–187.

Mohamed Hassan, Vasileios Choutas, Dimitrios Tzionas, and Michael J Black. 2019. Resolving 3D human pose ambiguities with 3D scene constraints. In *Proceedings of the IEEE/CVF international conference on computer vision*. 2282–2292.

Xin He, Chenlei Lv, Pengdi Huang, and Hui Huang. 2024. WindPoly: Polygonal Mesh Reconstruction via Winding Numbers. In *Computer Vision – ECCV 2024: 18th European Conference, Milan, Italy, September 29–October 4, 2024, Proceedings, Part XLVII* (Milan, Italy). Springer-Verlag, Berlin, Heidelberg, 294–311. doi:10.1007/978-3-031-72970-6_17

Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2018. Tetrahedral meshing in the wild. *ACM Trans. Graph.* 37, 4, Article 60 (July 2018), 14 pages. doi:10.1145/3197517.3201353

Qixing Huang, Xiangru Huang, Bo Sun, Zaiwei Zhang, and Junfeng Jiang. 2021. ARAPReg: An As-Rigid-As Possible Regularization Loss for Learning Deformable Shape Generators. 5795–5805. doi:10.1109/ICCV48922.2021.00576

Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. 2013. Robust inside-outside segmentation using generalized winding numbers. *ACM Trans. Graph.* 32, 4, Article 33 (July 2013), 12 pages. doi:10.1145/2461912.2461916

Wen Jiang, Nikos Kolotouros, Georgios Pavlakos, Xiaowei Zhou, and Kostas Daniilidis. 2020b. Coherent reconstruction of multiple humans from a single image. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 5579–5588.

Yue Jiang, Dantong Ji, Zhizhong Han, and Matthias Zwicker. 2020a. SDFDiff: Differentiable Rendering of Signed Distance Fields for 3D Shape Optimization. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 1248–1258. doi:10.1109/CVPR42600.2020.00133

Craig S. Kaplan and David H. Salesin. 2000. Escherization. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. ACM Press/Addison-Wesley Publishing Co., USA, 499–510. doi:10.1145/344779.345022

Rawal Khirodkar, Shashank Tripathi, and Kris Kitani. 2022. Occluded human mesh recovery. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 1715–1725.

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG]

Yong Tsui Lee and Aristides A. G. Requicha. 1982. Algorithms for computing the volume and other integral properties of solids. II. A family of algorithms based on representation conversion and cellular approximation. *Commun. ACM* 25, 9 (Sept. 1982), 642–650. doi:10.1145/358628.358648

Yiyi Liao, Simon Donne, and Andreas Geiger. 2018. Deep marching cubes: Learning explicit surface representations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2916–2925.

Hsueh-Ti Derek liu and Alec Jacobson. 2019. Cubic stylization. *ACM Trans. Graph.* 38, 6, Article 197 (Nov. 2019), 10 pages. doi:10.1145/3355089.3356495

Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. 2020. Neural Sparse Voxel Fields. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 15651–15663. https://proceedings.neurips.cc/paper_files/paper/2020/file/b4b758962f17808746e9bb832a6fa4b8-Paper.pdf

Yihao Luo, Yikai Wang, Zhengrui Xiang, Yuliang Xiu, Guang Yang, and ChoonHwai Yap. 2024. Differentiable Voxelization and Mesh Morphing. *arXiv preprint arXiv:2407.11272* (2024).

J. Manson and S. Schaefer. 2011. Wavelet Rasterization. *Computer Graphics Forum* 30, 2 (2011), 395–404. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2011.01887.x doi:10.1111/j.1467-8659.2011.01887.x

Josiah Manson and Scott Schaefer. 2013. Analytic Rasterization of Curves with Polynomial Filters. *Computer Graphics Forum* 32, 2pt4 (2013), 499–507. doi:10.1111/cgf.12070

Daniel Maturana and Sebastian Scherer. 2015. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. Ieee, 922–928.

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. 2019. Occupancy Networks: Learning 3D Reconstruction in Function Space. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 4455–4465. doi:10.1109/CVPR.2019.00459

A Mobius. 1865. Über die Bestimmung des Inhaltes eines Polyëders. *Mathematisch-Physische Klasse* 17 (1865), 31–68.

J. Montagnat, H. Delingette, and N. Ayache. 2001. A review of deformable surfaces: topology, geometry and deformation. *Image and Vision Computing* 19, 14 (2001), 1023–1040. doi:10.1016/S0262-8856(01)00064-6

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 165–174. doi:10.1109/CVPR.2019.00025

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 8024–8035.

Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. 2020. Scalable differentiable physics for learning and control. In *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*. JMLR.org, Article 727, 10 pages.

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

Edoardo Remelli, Artem Lukoianov, Stephan R. Richter, Benoît Guillard, Timur Bagautdinov, Pierre Baque, and Pascal Fua. 2020b. MeshSDF: differentiable iso-surface extraction. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '20)*. Curran Associates Inc., Red

Hook, NY, USA, Article 1884, 11 pages.
Osborne Reynolds, Arthur William Brightmore, and William Henry Moorby. 1903. *The sub-mechanics of the universe*. Vol. 3. University Press.
Michael Schwarz and Hans-Peter Seidel. 2010. Fast parallel surface and solid voxelization on GPUs. *ACM Trans. Graph.* 29, 6, Article 179 (Dec. 2010), 10 pages. doi:10.1145/1882261.1866201
Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. 2021. Deep Marching Tetrahedra: a Hybrid Representation for High-Resolution 3D Shape Synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*.
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 Transactions on Graphics (TOG)* 42, 4 (2023), 1–16.
Olga Sorkine and Marc Alexa. 2007. As-rigid-as-possible surface modeling. In *Symposium on Geometry processing*, Vol. 4. 109–116.
Cheng Sun, Min Sun, and Hwann-Tzong Chen. 2022. Direct Voxel Grid Optimization: Super-Fast Convergence for Radiance Fields Reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 5459–5469.
Haoran Sun, Jingkai Wang, Hujun Bao, and Jin Huang. 2024. GauWN: Gaussian-smoothed Winding Number and its Derivatives. In *SIGGRAPH Asia 2024 Conference Papers* (Tokyo, Japan) *(SA '24)*. Association for Computing Machinery, New York, NY, USA, Article 89, 9 pages. doi:10.1145/3680528.3687569
Haoran Sun, Shuang Wu, Hujun Bao, and Jin Huang. 2026. Variational Mesh Offsetting by Smoothed Winding Number. *IEEE Transactions on Visualization and Computer Graphics* 32, 2 (2026), 1668–1681. doi:10.1109/TVCG.2025.3637845
H.B. Voelcker and A.A.G. Requicha. 1977. Geometric Modeling of Mechanical Parts and Processes. *Computer* 10, 12 (1977), 48–57. doi:10.1109/C-M.1977.217601
Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. 2021. NeuS: learning neural implicit surfaces by volume rendering for multi-view reconstruction. In *Proceedings of the 35th International Conference on Neural Information Processing Systems (NIPS '21)*. Curran Associates Inc., Red Hook, NY, USA, Article 2081, 13 pages.
John Edward Warnock. 1969. *A hidden surface algorithm for computer generated halftone pictures*. Ph. D. Dissertation. AAI6919002.
Joe Warren and Henrik Weimer. 2001. *Subdivision methods for geometric design: A constructive approach*. Elsevier.
Chris Wylie, Gordon Romney, David Evans, and Alan Erdahl. 1967. Half-tone perspective drawings by computer. In *Proceedings of the November 14-16, 1967, Fall Joint Computer Conference* (Anaheim, California) *(AFIPS '67 (Fall))*. Association for Computing Machinery, New York, NY, USA, 49–58. doi:10.1145/1465611.1465619
Hao Xu, Yinqiao Wang, Niloy J. Mitra, Shuaicheng Liu, Pheng-Ann Heng, and Chi-Wing Fu. 2025. Hand-Shadow Poser. *ACM Trans. Graph.* 44, 4, Article 153 (July 2025), 16 pages. doi:10.1145/3730836
Haocheng Yuan, Adrien Bousseau, Hao Pan, Quancheng Zhang, Niloy J. Mitra, and Changjian Li. 2024. DiffCSG: Differentiable CSG via Rasterization. In *SIGGRAPH Asia 2024 Conference Papers* (Tokyo, Japan) *(SA '24)*. Association for Computing Machinery, New York, NY, USA, Article 9, 10 pages. doi:10.1145/3680528.3687608
Cheng Zhang, Lifan Wu, Changxi Zheng, Ioannis Gkioulekas, Ravi Ramamoorthi, and Shuang Zhao. 2019. A differential theory of radiative transfer. 38, 6 (2019), 227.
Jingyang Zhang, Yao Yao, and Long Quan. 2021. Learning Signed Distance Field for Multi-view Surface Reconstruction. 6505–6514. doi:10.1109/ICCV48922.2021.00646
Qingnan Zhou, Eitan Grinspun, Denis Zorin, and Alec Jacobson. 2016. Mesh arrangements for solid geometry. *ACM Trans. Graph.* 35, 4, Article 39 (July 2016), 15 pages. doi:10.1145/2897824.2925901