# Displaced Signed Distance Fields for Additive Manufacturing

ALAN BRUNTON, Fraunhofer IGD, Germany
LUBNA ABU RMAILEH, Fraunhofer IGD, Germany and Norweg. Univ. of Sci. and Tech. NTNU, Norway

Fig. 1. Displaced signed distance fields provide an efficient way to address multiple challenges in additive manufacturing. Robust sign estimation allows 3D printing of open surfaces resulting directly from 3D scanning (left). Displaced signed distance fields allow the inclusion of meso-scale surface topography in the form of a displacement map (center) at virtually no extra cost, and the direct fabrication at device resolution of curved triangles (right). The same 80-primitive input (inset) allows the fabrication of a 3cm and a 5cm sphere without subdivision or further tessellation.

We propose displaced signed distance fields, an implicit shape representation to accurately, efficiently and robustly 3D-print finely detailed and smoothly curved surfaces at native device resolution. As the resolution and accuracy of 3D printers increase, accurate reproduction of such surfaces becomes increasingly realizable from a hardware perspective. However, representing such surfaces with polygonal meshes requires high polygon counts, resulting in excessive storage, transmission and processing costs. These costs increase with print size, and can become exorbitant for large prints. Our implicit formulation simultaneously allows the augmentation of low-polygon meshes with compact meso-scale topographic information, such as displacement maps, and the realization of curved polygons, while leveraging efficient, streaming-compatible, discrete voxel-wise algorithms. Critical for this is careful treatment of the input primitives, their voxel approximation and the displacement to the true surface. We further propose a robust sign estimation to allow for incomplete, non-manifold input, whether human-made for on-screen rendering or directly out of a scanning pipeline. Our framework is efficient both in terms of time and space. The running time is independent of the number of input polygons, the amount of displacement, and is constant per voxel. The storage costs grow sub-linearly with the number of voxels, making our approach suitable for large prints. We evaluate our approach for efficiency and robustness, and show its advantages over standard techniques.

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

Additional Key Words and Phrases: 3D printing, distance field, robust voxelization, displacement

**ACM Reference Format:**
Alan Brunton and Lubna Abu Rmaileh. 2021. Displaced Signed Distance Fields for Additive Manufacturing. *ACM Trans. Graph.* 40, 4, Article 179 (August 2021), 13 pages. https://doi.org/10.1145/3450626.3459827

Authors' addresses: Alan Brunton, alan.brunton@igd.fraunhofer.de Fraunhofer IGD, Darmstadt, Germany; Lubna Abu Rmaileh, lubna.abu.rmaileh@igd.fraunhofer.de Fraunhofer IGD, Darmstadt, Germany Norweg. Univ. of Sci. and Tech. NTNU, Gjøvik, Norway.


0730-0301/2021/8-ART179 $15.00
https://doi.org/10.1145/3450626.3459827

## 1 INTRODUCTION

This paper presents a unified framework for 3D printing highly detailed and smooth surfaces from compact representations by robustly and efficiently computing *displaced signed distance fields*, which represent the shape implicitly as the composition of a discrete signed distance field and a displacement field. The displacement field encodes the offset from a discrete voxel approximation to the true surface. We show how this formulation allows the use of efficient, streaming algorithms operating on discrete voxel grids while maintaining sub-voxel accuracy, and show how to use this framework to 3D print coarse tessellations with displacement maps encoding mesoscopic detail, curved surfaces, and incomplete or self-overlapping surfaces.

As the resolution and accuracy of 3D printers increase, accurate reproduction of finely detailed and smoothly curved surfaces becomes increasingly realizable from a hardware perspective. Representing surfaces with polygonal meshes to an accuracy on the order of modern 3D printers requires a large number of polygons, correspondingly increasing requirements for storage and bandwidth for transmission. Further, the same approximation accuracy requires different levels of tessellation for different print sizes.

With increasing use of graphical 3D printing in the entertainment sector, for films or 3D printing content from game engines, automated and efficient processing of graphical 3D content from these applications becomes increasingly important. Such models often reduce polygon counts by encoding mesoscopic geometric detail in displacement maps, topographic textures or height maps applied to a low-poly mesh. Thus, it is advantageous to directly support this type of input (Fig. 1, center) without computationally demanding tessellation, which can introduce further errors in the form of self-intersections and inverted surfaces. Since they are designed for on-screen display, not physical fabrication, such models are often composed of open, overlapping surfaces and decals. The mesh in Fig. 1 (center) is comprised of disjoint patches with small gaps in between. Thus, robust processing is key to automation.

Over half of parts produced by additive manufacturing are either functional parts or functional prototypes [Wohlers et al. 2020]. These are often designed and represented by parametric patches,

rather than polygonal meshes. The typical workflow still involves tessellating these surfaces into polygonal meshes, which in general introduces a discretization error. Tessellation into meshes of *curved* triangles is an alternative, which scales much better in terms of making the approximation error significantly less sensitive to the scale of the printed part. Existing approaches to fabrication with curved triangles are based on subdivision, again requiring different amounts of subdivision for different print sizes. This paper shows how displaced signed distance fields can realize curved triangle-based shapes to printer resolution without tessellation (Fig. 1, right).

Making a 3D copy of a real object involves scanning that object, the output of which often comes in the form of a point cloud, and typically there are portions of the surface which cannot be scanned, resulting in an incomplete surface (Fig. 1, left). Directly processing such data for 3D printing can also streamline workflows.

For applications in graphical 3D printing, including rapid prototyping and entertainment, the build volumes and resolutions of modern devices mean any method needs to scale up to nearly $10^{12}$ voxels [Mimaki 2017; Stratasys 2016]. This constrains us to algorithms that have complexity linear in the number of voxels, or constant time per voxel. More restrictively, it constrains us to so-called streaming compatible algorithms, which require a peak storage less than 1 byte per voxel for large prints. We define such algorithms formally in Section 3.

In this paper, our primary technical contribution is a unified implicit formulation and resulting algorithmic framework to robustly and efficiently

- produce highly detailed surfaces from low-polygon meshes with an attached displacement map,
- produce smooth surfaces from low-polygon meshes of higher order primitives.

We further propose a technique for representing surface structures with local feature size on the order of, and even below, the voxel size. Our formulation gives rise to algorithms with the following advantages:

- it combines the efficiency of discrete voxel distance computations, running in $O(1)$ time per voxel, with sub-voxel accuracy and bounded error w.r.t. the true signed distance;
- it is streaming compatible, requiring less than 1 byte per voxel for large prints;
- it supports unbounded displacement and performs no further tessellation or refinement;
- and it robustly handles incomplete, non-manifold and self-overlapping input.

In particular, if we consider as an alternative computing the true signed distance field (SDF), we would encounter the following problems. In the case of displacement maps or curved higher-order primitives, we would need to first tessellate to a desired tolerance (thus introducing some error anyways), along with the associated problems of potential self-intersection. Doing this on-the-fly within a streaming context would require bounded displacements. To compute the exact distance field w.r.t. the tessellation would need a bounding volume hierarchy (BVH) or similar space partitioning, resulting in running time logarithmic in the number of primitives, which would be large in the case of refined tessellation. Our displaced SDF provide an good approximation of the true SDF, and allows more efficient and robust processing.

## 2 RELATED WORK

### 2.1 Robust Voxelization

For sufficiently clean data, *i.e.* watertight and manifold, conversion to a solid voxel representation can be done very efficiently, in particular leveraging parallelism on the GPU [Eisemann and Decoret 2008; Schwarz and Seidel 2010].

The more challenging problem of converting surface data that does not represent the boundary of a volume, whether corrupt, incomplete, or intentionally designed so by an artist, has also been studied. Nooruddin and Turk [Nooruddin and Turk 2003] propose a “ray-stabbing" technique to convert a polygonal model to voxels. This technique is limited to a small voxel grid as it requires the full voxel grid in memory simultaneously.

The generalized winding number [Jacobson et al. 2013] provides a way to robustly determine whether a point lies inside or outside a curve of surface by extending the winding number to open curves and surfaces. This inherits the winding number’s natural handling of self-overlapping surfaces, behaves smoothly in holes, and degrades gracefully in the presence of noise or inconsisten surface orientation. Barill *et al.* [2018] provided a fast approximate evaluation for triangle soups and point clouds based on a multipole expansion, and showed some robust voxelization examples.

Schmidt [2019] describes converting an unsigned distance field into a SDF using a flood fill approach, starting at one or more voxels known to be outside the surface and stopping at inversions of the gradient. It is not clear that this technique can work for large holes, or is practical for large voxel grids. Another technique [Stevens and McKenna 2018] computes a SDF of a polygon mesh by generating multiple layered depth images [Shade et al. 1998], and using them to resolve tessellation inconsistencies to compute signed distances. Krayer and Müller [2019] propose a GPU-based approach based on ray-maps, which is both fast and robust, but does not appear to scale well to large voxel grids.

### 2.2 Surface Repair

There are many existing methods for surface repair, both those specifically targeting 3D printing and those targeting more general application areas. Most of these operate directly on a mesh. Such methods benefit from local operations and are therefore quite efficient. Difficulties arise in simultaneously handling holes, non-manifold edges and vertices, islands (disconnected pieces), and self-intersection. Most methods to convert arbitrary collections of polygons into surfaces do not robustly handle self-intersection [Guéziec et al. 2001; Hoppe et al. 1993; Kraevoy et al. 2003; Podolak and Rusinkiewicz 2005] in that they do not guarantee a watertight output. Methods that do offer guarantees typically fall into two categories: those that globally remesh the surface [Bischoff et al. 2005; Ju 2004] and those that remove parts of the mesh that are difficult to handle, either creating holes and refilling them [Attene 2010] or specifically removing parts causing self-intersections [Yamakawa and Shimada 2009]. In our application, global remeshing would be

acceptable, and our surface voxelization stage can be seen as an approximate resampling of the surface. However, we are interested in obtaining an implicit representation, which we can combine with mesoscopic detail information.

### 2.3 Surface Reconstruction and Meshing

Our approach is also related to surface reconstruction from point cloud data, although we do not propose our algorithm as a solution for surface reconstruction. We view our work as complementary, since the reconstructed mesh can also serve as an input to our framework. Berger *et al.* provide a comprehensive recent survey [2017]. We discuss the most closely related methods here.

Poisson [Kazhdan et al. 2006] and screened Poisson [Kazhdan and Hoppe 2013] surface reconstruction solve for an indicator or characteristic function, which can be seen as equivalent to our sign function, by observing that its gradient at the surface should align with the surface normal. For efficiency, they discretize using an octree to obtain a sparse linear system and to simultaneously avoid computations away from the surface. Implicit moving least squares [Kolluri 2005], its variant based on robust estimation [Oeztireli et al. 2008], and others, follow the gradient of an implicit function to the surface. These techniques reduce computational effort by evaluating only in the vicinity of the surface. To apply displacement, we need to consider possibly large distances from the surface, and for this reason we must compute a signed distance, and therefore a sign, at all voxels in our bounding volume. Given this restriction, sophisticated regularization, such as solving a PDE, is too costly.

### 2.4 Deep Learning Methods

Recently a slew of new techniques propose to use deep neural network architectures to learn object-specific representations, which allow the approximate reconstruction of a SDF for that object. Erler *et al.* [2020] show robustness to severe noise levels by learning a global sign function and a local distance function, exploiting the generalizing power of local patches and the low complexity of sign information. Sign agnostic learning [Atzmon and Lipman 2020] provides indifference to errors in surface orientation, at the cost of limited detail preservation and robustness to thin structures. Including an Eikonal constraint in the loss improves reconstruction of sharp features and thin structures [Atzmon and Lipman 2021; Gropp et al. 2020]. Davies *et al.* [2021] propose to use an overfit neural network as a compact implicit shape representation. Sitzmann *et al.* [2020] also showed implicit shape encoding as an application. While this line of work present exciting results, these methods are too computationally intensive for our application. Even approaches using offline training [Erler et al. 2020] require a forward pass per signed distance evaluation, and shape representation/reconstruction approaches require training per object.

### 2.5 Displacement Mapping

Displacement mapping [Cook 1984; Cook et al. 1987] is a common technique in rendering, particularly ray tracing and ray marching, with efficient GPU implementations [Szirmay-Kalos and Umenhoffer 2006]. A height field texture provides the meso-scale information about the surface, specifying the distance along the surface normal of the tessellation to the true surface.

OpenFab [Vidimče et al. 2013] applies displacement maps by first tessellating the models to facets on the order of the printer resolution, displacing the resulting vertices, and then voxelizing. For bounded displacements, by paying close attention to the order in which facets are processed, such a tessellation approach can be done on-the-fly within a streaming framework, and efficient GPU implementations exist. In contrast, our approach can apply *unbounded* displacements within a streaming computation, without affecting performance, while replacing the geometric operations of tessellation with sampling and writing of displacement values. We show the performance of our framework under displacements of various magnitudes in Section 8.2.

A non-streaming tessellation approach can be implemented in widely available modeling software, *e.g.* Blender [2020], by first refining the tessellation using subdivision, then displacing vertices. Commercial 3D printing software also displaces the vertices of a densely tessellated mesh [Stratasys 2020]. This leads to high polygon counts and can introduce self-intersections and inverted surfaces, particularly for larger displacements, as shown in Section 8.2. Our approach does not suffer from these problems. While one could leverage techniques from geographic information systems to convert digital elevation maps to meshes [Garland and Heckbert 1995], this would only provide a trade-off between approximation quality and compactness, and would not address self-intersections.

### 2.6 Curved Triangles

Tessellating parametric surfaces to curved triangles instead of flat ones allows better approximation with fewer primitives. Point-normal (PN) triangles [Vlachos et al. 2001] are a subset of Bézier triangles [de Casteljau 1959; Farin 1986], in which surface normals specified at each vertex control how the triangle is curved (by determining the control points of the Bézier triangle). In finite element analysis (FEM), Lagrange polynomials are the standard high-order basis for shape functions [Zienkiewicz et al. 2013], and high-order Lagrange polynomials on triangles [Berrut and Trefethen 2004] allow for curved triangles. Like the Bézier basis on triangles, first order Lagrange basis is equal to a barycentric basis.

Techniques exist [Gohari et al. 2018; Starly et al. 2005] to directly slice NURBS surfaces. These are only practical for additive processes based on path following, *e.g.* fused filament fabrication (FFF), as they work by intersecting slicing planes with the NURBS surface.

### 2.7 Streaming Distance Field Computation

Recently, Brunton *et al.* [2018] proposed a technique for unsigned distance field computation, which runs in $O(1)$ time per voxel while being streaming compatible, meaning the full voxel grid does not need to be kept in memory; we reproduce their formal definition in Section 3. Their technique involves pre-computing a sparse surface voxelization, also constructed in $O(1)$ per voxel with limited memory, stored in ascending order of $z$ for each $xy$-column. This allows the unsigned distance to the surface voxels to be computed in constant time per voxel when processing slices in order, storing as little as a single slice in memory at once. Similar data structures

have also been used for slice thickness optimization [Alexa et al. 2017] and morphological operations [Martinez et al. 2015]. While we use this technique to compute unsigned distances and to identify the nearest surface voxel to a given voxel, it does not address sign estimation, sub-voxel accuracy or displacement. We further reduce the memory footprint during construction by using a hash map on the voxel coordinates, rather than storing dense slices. Our implicit formulation, given in Section 3, makes possible the use of such fast, streaming, voxel-wise techniques for robust and precise fabrication.

## 3 DISPLACED SIGNED DISTANCE FIELDS

Our framework exploits the relationship between implicit representations of a given approximate surface and the true shape. We consider how an incomplete approximation affects distance fields, signed and unsigned, w.r.t. the input and the true surface. We further consider that the approximation may be defined implicitly w.r.t. the true surface, and vice versa, via a displacement field.

In the context of additive manufacturing, we work with implicit representations sampled on a finite voxel grid partitioning the build volume of the device. Let $\mathcal{B} \subset \mathbb{R}^3$ denote the build volume of the 3D printer, and partition it into $W \times H \times D$ voxels of size $\delta_x, \delta_y, \delta_z$ along the $x$-, $y$- and $z$-axes of the build space, corresponding to the native resolution of the printer. We further define $\mathcal{C} \subset \mathcal{B}$ to be the set of voxel centers, and $V(\mathbf{u}) \subset \mathcal{B}$ to be the voxel containing $\mathbf{u} \in \mathcal{B}$. Typically, the shape we wish to manufacture fills only a fraction of the device's build volume, and we restrict $\mathcal{B}$ to be a padded bounding box of the input, reducing $W$, $H$ and $D$ accordingly.

The core of our method lies in composing implicit representations of the approximate surface with implicit representations of the true surface w.r.t. the approximation in a way that allows to robustly and efficiently reproduce finely detailed and curved surfaces at device resolution from a compact input. Fig. 2 shows these two scenarios and how the quantities defined below relate to realize them.

Given the resolution and build volumes of modern 3D printers, doing this without storing the full sampling grid is critical. Therefore, in this paper, we concentrate on algorithms that are *streaming compatible*, which we define as follows in the context of additive manufacturing [Brunton et al. 2018]:

*Definition 3.1.* Since the manufacturing process builds the object from bottom to top, we define streaming computation to proceed in ascending order of z. We further define that for a given print occupying a volume corresponding to a voxel grid of $W \times H \times D$ voxels, a streaming algorithm never exceeds $O(WH + N)$ storage, where $N \ll WHD$; *i.e.* a constant number of slices (2D array with constant $z$-coordinate) plus small auxiliary storage, relative to the size of the voxel grid.

Sections 4 and 7 give further details on how we robustly and efficiently represent shapes at device precision or better within a streaming compatible framework. However, the implicit formulation presented below is key to enabling this, and Section 3.1 discusses its advantages as an approximation versus computing the true SDF.

Given a shape $\mathcal{S} \subset \mathbb{R}^3$ s.t. $|\mathcal{S}| < \infty$ and its boundary $\partial\mathcal{S}$ is a closed manifold embedded in $\mathbb{R}^3$, let

$$d(\mathbf{u}) = \min_{\mathbf{v} \in \partial\mathcal{S}} \|\mathbf{u} - \mathbf{v}\|_2 \tag{1}$$

denote the unsigned distance of any point $\mathbf{u} \in \mathbb{R}^3$ to its nearest point in $\partial\mathcal{S}$ and

$$f(\mathbf{u}) = \begin{cases} -d(\mathbf{u}) & \mathbf{u} \in \mathcal{S} \\ d(\mathbf{u}) & \mathbf{u} \in \mathbb{R}^3 \backslash \mathcal{S} \end{cases} \tag{2}$$

the signed distance of $\mathbf{u}$ w.r.t. $\mathcal{S}$. That is, $d(\mathbf{u}) = |f(\mathbf{u})|$.

In practice, we consider an approximation $\widetilde{\partial\mathcal{S}}$ of the true surface $\partial\mathcal{S}$, which may differ in several ways. One possibility is an incomplete version or a point sampling of the true surface, $\widetilde{\partial\mathcal{S}} \subset \partial\mathcal{S}$. Now, our unsigned distance

$$\widetilde{d}(\mathbf{u}) = \min_{\mathbf{v} \in \widetilde{\partial\mathcal{S}}} \|\mathbf{u} - \mathbf{v}\|_2 \tag{3}$$

gives us only an upper bound, $\widetilde{d}(\mathbf{u}) \geq |f(\mathbf{u})|$, on the distance to the true surface. We can now compose the signed distance, $f$, to $\partial\mathcal{S}$, as

$$f(\mathbf{u}) = s(\mathbf{u})\widetilde{d}(\mathbf{u}) \tag{4}$$

where $s : \mathbb{R}^3 \mapsto [-1, 1]$ is a continuous sign function.

Further, $\widetilde{\partial\mathcal{S}}$ may be locally displaced w.r.t. $\partial\mathcal{S}$. In particular, we take $\widetilde{\partial\mathcal{S}}$ to be a sparse voxel representation constructed from a set of input primitives $\mathcal{P}$, which may be points or triangles. More precisely $\widetilde{\partial\mathcal{S}} \subseteq C$ is the set of center points of those voxels. This introduces quantization error due to finite voxel resolution, which we model as the negative signed distance to the nearest point on a primitive $p \in \mathcal{P}$ of the center of a voxel, $\phi_{\mathcal{P}} : \widetilde{\partial\mathcal{S}} \mapsto \mathbb{R}$. The advantage of working from voxel centers is that it enables efficient distance field computation at low memory cost. Section 4 goes into more detail.

Additionally, the true surface may be displaced w.r.t. the input primitives, *e.g.* due to the use of a coarse tessellation to approximate a detailed or smooth surface. We model this displacement as an offset along the outward normal $\mathbf{n}_p$ of primitive $p$,

$$\mathbf{s} = \mathbf{v} + \phi_{\mathcal{S}}(\mathbf{v})\mathbf{n}_p(\mathbf{v}) \ : \ \mathbf{s} \in \partial\mathcal{S}, \ \mathbf{v} \in p \tag{5}$$

where $\phi_{\mathcal{S}} : \mathcal{P} \mapsto \mathbb{R}$ is a displacement map giving the signed distance of the true surface w.r.t. the primitives. The full displacement field $\phi : \widetilde{\partial\mathcal{S}} \mapsto \mathbb{R}$ from $\widetilde{\partial\mathcal{S}}$ to $\partial\mathcal{S}$ is combined as in Section 4.

Now we write our *displaced* SDF definition to be

$$\widetilde{f}(\mathbf{u}) = s(\mathbf{u})\widetilde{d}(\mathbf{u}) - \phi(\overline{\mathbf{u}}) \tag{6}$$

where

$$\overline{\mathbf{u}} = \arg\min_{\mathbf{v} \in \widetilde{\partial\mathcal{S}}} \|\mathbf{u} - \mathbf{v}\|_2 \tag{7}$$

is the nearest point on $\widetilde{\partial\mathcal{S}}$ to $\mathbf{u} \in \mathbb{R}^3$.

### 3.1 Approximation Power of $\widetilde{f}$

In general, $\widetilde{f} \neq f$, and is not a metric SDF satisfying the Eikonal equation due to the high frequency content and discontinuities potentially introduced by the term $\phi(\overline{\mathbf{u}})$ in (6). This section shows under what conditions it shares the same 0-level set as $f$,

$$\widetilde{f}(\mathbf{u}) = 0 \iff f(\mathbf{u}) = 0 \tag{8}$$

and gives bounds on the error of the 0-level of of $\widetilde{f}$, $\epsilon_0 = f(\mathbf{u})$ for $\mathbf{u}$ s.t. $\widetilde{f}(\mathbf{u}) = 0$. To simplify the analysis, we assume an isotropic voxel grid with voxel size $\delta = \delta_x = \delta_y = \delta_z$.

When $\phi = \phi_{\mathcal{P}}$ captures just the distance from the voxel centers to the primitives and $\mathcal{P} \subseteq \partial\mathcal{S}$, then (8) holds for most orientations

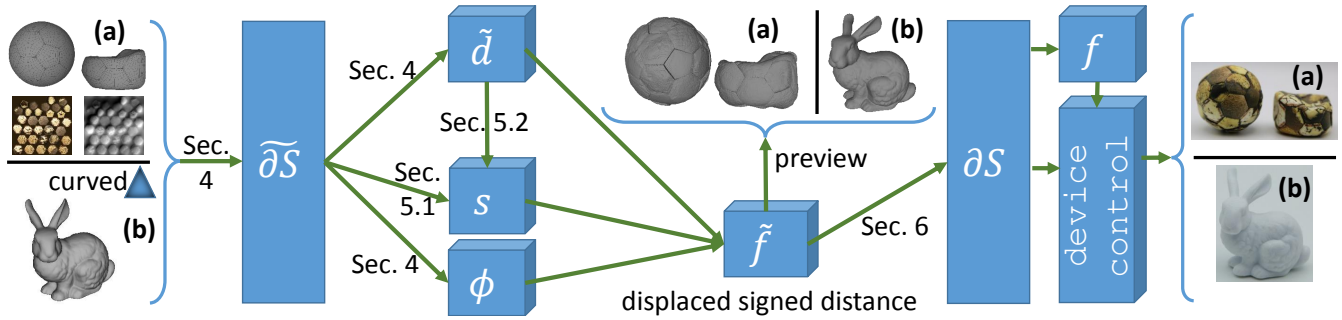



Fig. 2. An overview of our framework for two scenarios. In scenario (a), we are given a low-poly mesh with an albedo texture and a displacement map encoding mesoscopic surface detail. In scenario (b), we are given a polygonal mesh comprised of curved triangles, in this case point-normal triangles [Vlachos et al. 2001] with per-vertex normals. See Section 3 for definitions of the quantities in the boxes.

of planar surfaces and nearly holds for piecewise planar surfaces approximating smooth curved surfaces. For planar surfaces where the normal is nearly aligned with one axis, so that the voxelization is a single voxel thick, but not exactly aligned, we have a small error $\epsilon_0 \neq 0$. In Appendix A we show that surface orientations, which result in surface voxels on both positive and negative sides of the surface result in $\epsilon_0 = 0$, and otherwise $|\epsilon_0| < 0.029\delta$. This bound also holds when a planar surface is endowed with a non-zero displacement field $\phi_S$. In Appendix A we also analyze the error for voxels away from the surface.

For curved surfaces the analysis becomes more complex, but Fig. 3 shows empirically that the error of a surface extracted in this way remains near zero for low curvature regions and increases only to a small fraction of the voxel size for higher curvature regions.

We have shown that $\widetilde{f}$ provides a high quality approximation of the true SDF $f$, and in particular their 0-level sets are very close. And this approximation allows us to use efficient, streaming compatible algorithms operating on discrete grids. Computing the "true" SDF would require fine tessellation, followed by some kind of BVH to compute distances. This would introduce the potential for self-intersecting or inverted surfaces, complexity logarithmic in the number of primitives (after fine tessellation), and restrictions on the displacement magnitude. Displaced SDFs make a deliberate and bounded error in return for efficiency and flexibility.

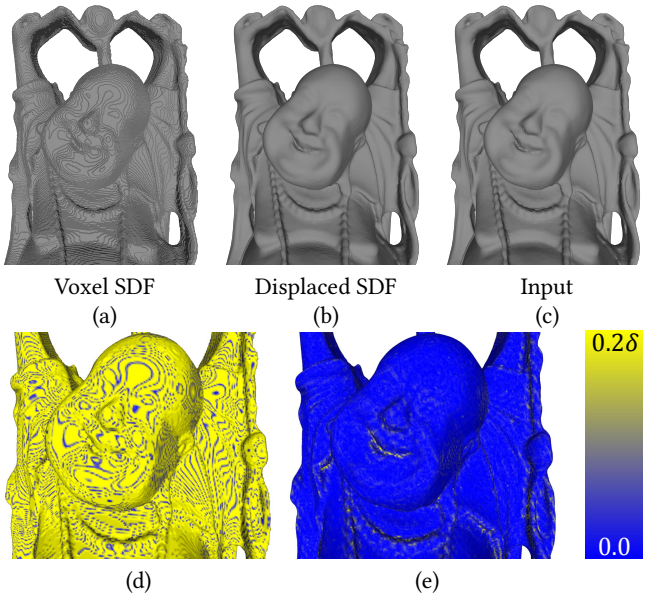



Fig. 3. Effect of including sub-voxel offset as displacement. The Happy Buddha model (c) voxelized and represented using (4) (a) and (6) (b) with $\phi$ computed as in (9). The absolute errors of (a) and (b) are visualized as a fraction of the voxel size in (d) and (e), respectively.

## 4 SUB-VOXEL BOUNDARY APPROXIMATION

We represent $\widetilde{\partial S}$ as a sparse set of voxels, per Brunton *et al.* [2018], allowing $O(1)$ per-voxel computation of $\widetilde{d}$, and the nearest point $\overline{\mathbf{u}} \in \widetilde{\partial S}\ \forall\, \mathbf{u} \in C$, which we reference with a globally unique index. A surface voxelization pre-process generates our sparse surface voxels $\widetilde{\partial S}$ from the input primitives $\mathcal{P}$, which may be points or triangles. Section 7.1 goes into more detail on this process. Important for the present discussion is that we compute a conservative 26-separating surface voxelization of triangles, registering every triangle-voxel intersection. For background and terminology on surface voxelization, please refer to Cohen-Or and Kaufman [1995].

Compensating for the quantization error introduced by working on a discrete grid is important because the fine scale detail provided by a displacement map is given relative to the primitives, and not the surface voxel centers. We do so by storing the signed distance of the voxel center relative to the primitives incident to that voxel

$$\phi_{\mathcal{P}}(\mathbf{u}) = \frac{1}{|\mathcal{P}(\mathbf{u})|} \sum_{p \in \mathcal{P}(\mathbf{u})} -\mathbf{n}_p^T \mathbf{c}_p(\mathbf{u}) \tag{9}$$

where $\mathbf{u} \in \widetilde{\partial S}$, $\mathcal{P}(\mathbf{u}) \subseteq \mathcal{P}$ is the set of primitives with non-empty intersection with $V(\mathbf{u})$, $\mathbf{n}_p$ is the normal of primitive $p$, and $\mathbf{c}_p(\mathbf{u})$ is the centroid of the intersection $p \cap V(\mathbf{u})$ of the primitive and the voxel relative to $\mathbf{u}$ (*i.e.* shifted so that the voxel center $\mathbf{u}$ is at the origin). We also average displacements from the primitives to the true surface, such that

$$\phi(\mathbf{u}) = \phi_{\mathcal{P}}(\mathbf{u}) + \frac{1}{|\mathcal{P}(\mathbf{u})|} \sum_{p \in \mathcal{P}(\mathbf{u})} \phi_{\mathcal{S}}(\mathbf{c}_p(\mathbf{u})) \tag{10}$$

which can be combined in a single summation over in the incident primitives. This allows us to compute $\phi(\mathbf{u})$ incrementally as each primitive is voxelized, independent of the order.

In the case where $\mathcal{P} \subseteq \partial\mathcal{S}$, *i.e.* the flat input primitives are at least a subset of the true surface, we have $\phi(\mathbf{u}) = \phi_{\mathcal{P}}(\mathbf{u})$. Fig. 3 shows how this improves the precision of the voxel representation. Fig. 3a,b show iso-surfaces extracted from $s(\mathbf{u})\widetilde{d}(\mathbf{u})$ and $\widetilde{f}(\mathbf{u})$, respectively, with $\phi(\overline{\mathbf{u}}) = \phi_{\mathcal{P}}(\overline{\mathbf{u}})$. Fig. 3a shows clear quantization error, whereas (b) closely matches the input (c). The respective errors $\epsilon_0$ are visualized in (d) and (e).

Human-created models and scaled versions of scanned objects will in general include structures with local feature size below the device resolution. While such features are not directly printable, we wish to preserve them since techniques exist to make them printable (see *e.g.* Reinhard [2017] and the references therein). Averaging surface normals when multiple primitives intersect the same voxel is numerically sensitive, and fails to properly model sharp edges or corners, crevices, or tubular structures. One could compute $\widetilde{f}$ at a higher resolution, and downsample for printing, but this would correspondingly increase the computation time, without solving the problem for structures with feature size below the resolution of $\widetilde{f}$.

Instead, we propose to represent sub-voxel structures using a weighted sum of signed distance gradients. Informally, we determine whether each face of the voxel is "looking" out or in from the part of the surface contained within the voxel. Our approach is similar 2D vector glyph rendering [Nehab and Hoppe 2008; Qin et al. 2006], where multiple primitives are stored in a grid cell. One key difference is the fixed storage length of our representation.

Formally, to model how the SDF will change as we move from the voxel center along a given direction, we compute weighted averages of gradients of the SDFs defined by the primitives. Each primitive defines a linear SDF w.r.t. its plane, with its gradient defined by $\mathbf{n}_p$. For a given direction $\omega \in S^2$, we define

$$\partial_\omega f(\mathbf{u}) = \frac{1}{\sum_{p\in\mathcal{P}} w_p(\mathbf{u},\omega)} \sum_{p\in\mathcal{P}} \mathbf{n}_p^T \omega w_p(\mathbf{u},\omega) \tag{11}$$

where the weights are defined by

$$w_p(\mathbf{u},\omega) = a_p(\mathbf{u}) \exp\left(-(\mathbf{c}_p(\mathbf{u})^T\omega - \delta_\omega/2)^2/\sigma^2\right) \tag{12}$$

where $a_p(\mathbf{u})$ is the area of $p \cap V(\mathbf{u})$, and $\delta_\omega = [\delta_x\ \delta_y\ \delta_z]\omega$. The term $\mathbf{c}_p(\mathbf{u})^T\omega - \delta_\omega/2$ gives the distance of the centroid from the boundary of the voxel along the direction $\omega$ from its center, and $\sigma$ is controls the fall-off rate of the weight as this term increases. We set $\sigma = 0.1\left\|[\delta_x\ \delta_y\ \delta_z]^T\right\|_2$. We treat point primitives as having a constant area. Barill *et al.* [2018] recommend non-uniform areas based tangent plane Voronoi cells, but we use this only for primitives within a single voxel, not to influence queries at a distance.

This approach to representing the geometry inside a voxel has the advantage that it disentangles geometric precision within the voxel ($\phi_{\mathcal{P}}$) and sign information for other voxels ($\{\partial_{\omega_i} f\}$), which reflects the notion that if the surface intersect a voxel, the sign of that voxel is not unique. Further, this representation can be updated incrementally as primitives are processed, is compact, has fixed-length storage, and allows efficient sign evaluation.

## 5 ESTIMATING SIGN

Having constructed our boundary approximation $\widetilde{\partial\mathcal{S}}$ as in Section 4, we now turn to the problem of computing the sign $s(\mathbf{u})$ for all $\mathbf{u} \in C$. We proceed in two steps: initializing from the nearest boundary voxel $\overline{\mathbf{u}}$, then regularizing based on the distance to boundary $\widetilde{d}(\mathbf{u})$.

### 5.1 Initialization

For voxel centers $\mathbf{u} \in (C \setminus \widetilde{\partial\mathcal{S}})$ away from the boundary we obtain a sign value from the boundary as

$$s(\mathbf{u}) = \text{sign}\left((\mathbf{u}-\overline{\mathbf{u}})^T \sum_{i=1}^{N_\omega} \omega_i \mathbf{1}_{\mathbb{R}^+}\left((\mathbf{u}-\overline{\mathbf{u}})^T\omega_i\right)\partial_{\omega_i} f(\overline{\mathbf{u}})\right) \tag{13}$$

where per (7) $\overline{\mathbf{u}}$ is the nearest surface voxel to $\mathbf{u}$, $\mathbf{1}_{\mathbb{R}^+}$ is an indicator function equal to 1 when its argument is $> 0$ and 0 otherwise, and $N_\omega$ denotes the number of directions along which the gradients are projected. According to (13), we sum up the projected gradients $\omega_i \partial_{\omega_i} f(\overline{\mathbf{u}})$, for which $\omega_i \cdot (\mathbf{u}-\overline{\mathbf{u}}) > 0$. If the scalar product of the sum with $\mathbf{u}-\overline{\mathbf{u}}$ is negative then $s(\mathbf{u}) = -1$, otherwise $s(\mathbf{u}) = 1$. For voxel centers $\mathbf{v} \in \widetilde{\partial\mathcal{S}}$ intersected by $\mathcal{P}$, we initialize $s(\mathbf{v}) = 0$. Fig. 4 shows the quantities for sub-voxel structures (in 2D).

This approach allows surface voxels containing thin walls or tubular structures to export different sign values in opposite directions, while for voxels containing a single primitive it behaves equivalently to $\text{sign}(\mathbf{n}_p^T(\mathbf{u}-\overline{\mathbf{u}}))$. In practice, we use $N_\omega = 6$ with $\omega_1$ and $\omega_2$ being the negative and positive $x$-axes, $\omega_3$ and $\omega_4$ the negative and positive $y$-axes, and $\omega_5$ and $\omega_6$ the negative and positive $z$-axes, respectively. This greatly simplifies the evaluation of (11) and (13).

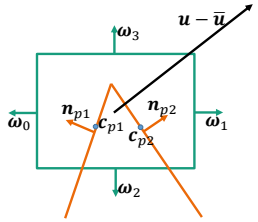


Fig. 4. Quantities for addressing sub-voxel structures.

Fig. 5 shows how this preserves geometric structures at or below the size of a voxel. The input (a) contains a glasses frame, which is sliced separately at the same scale and orientation in (e) and (f) with a subset of slices shown. Due to the scale it is thin enough to intersect single voxels and cause problems in resolving the sign for adjacent voxels. The evaluation of $s$ according to (13) preserves the correct sign for voxels not intersected by the primitives (e), and this is refined after applying sub-voxel displacement (f). This shows that (13) successfully models complex geometry with local feature size below the voxel size everywhere. Section 9 describes some limits on how far below the voxel size we can take this, and how complex the sub-voxel geometry can be.

Fig. 6c and 7 show how the binary sign estimation based on the nearest surface voxel (13) extends sign information from incomplete surfaces to complete them. Note, however, the discontinuous distance fields and jagged isosurfaces that it creates.

### 5.2 Regularization

While the sign evaluation given in (13) is robust to sub-voxel structures, for open surfaces it will simply continue the surface linearly from the nearest surface voxel. This results in jagged surfaces, as in

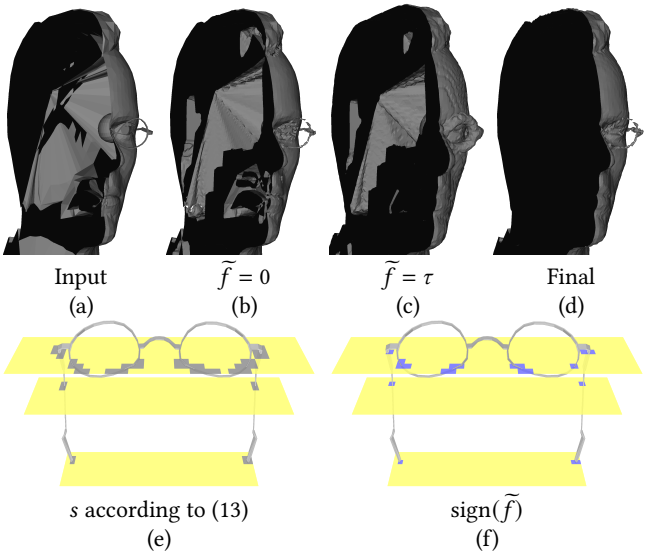



Fig. 5. Preservation of sub-voxel structures and hidden surface removal. The input (a) contains many overlapping parts and structures on the order of or below the voxel size, such as the glasses frame, which is shown separately with slice information in (e) and (f), where yellow is positive, blue negative and grey 0.

Figs. 6c and 7. We wish to regularize $s$ so that (4) is smooth and has equally spaced iso-surfaces.

We experimented with a number of different regularization techniques. For example, we found that Monte Carlo estimation of the Laplace equation $\Delta s(\mathbf{u}) = 0;\ \mathbf{u} \notin \widetilde{\partial\mathcal{S}}$, similar to [Sawhney and Crane 2020], to be effective in removing low-frequency discontinuities, but converged far too slowly to sufficiently remove high-frequency noise. Instead, we settled on the following efficient regularization.

Laplace regularization is equivalent to imposing the condition $s(\mathbf{u}) = \frac{1}{|B(\mathbf{u},r)|}\int_{\mathbf{v}\in B(\mathbf{u},r)} s(\mathbf{v})d\mathbf{v}$, where $B(\mathbf{u}, r)$ denotes the ball centered on $\mathbf{u}$ of radius $r$. The distance field $\widetilde{d}$ gives us the maximum radius we can use without encountering $\widetilde{\partial\mathcal{S}}$. Rather than impose the mean value property over the full 3D ball $B(\mathbf{u}, \widetilde{d}(\mathbf{u}))$, we do so over the 1D axis aligned balls $B_x(\mathbf{u}, \widetilde{d}(\mathbf{u}))$, $B_y(\mathbf{u}, \widetilde{d}(\mathbf{u}))$ and $B_z(\mathbf{u}, \widetilde{d}(\mathbf{u}))$, where *e.g.*

$$B_x(\mathbf{u}, r) = \{\mathbf{u} + [x\ 0\ 0]^T\ :\ -r < x < r\}. \tag{14}$$

This amounts to applying a sequence of adaptive 1D mean filters along the $x$-, $y$- and $z$-axes. We implement this using 1D integral images, allowing $O(1)$ computation at an additional storage of only $O(\max\{W, H, D\})$. We experimented with iterating each 1D mean filter to approximate Gaussian filters, but this did not add much benefit for the additional cost. Figs. 6 and 7 show the effect of this regularization compared to the unregularized sign function (13) for open surfaces. Fig. 8 shows how the regularization performs for further degraded input.

While this does not provide the same level of regularization as PDE-based methods [Barill et al. 2018; Kazhdan et al. 2006; Kazhdan and Hoppe 2013; Sawhney and Crane 2020] or neural architectures [Atzmon and Lipman 2020, 2021; Erler et al. 2020; Gropp et al. 2020], it is less computationally intensive to evaluate at every voxel, which we require to allow for arbitrarily large displacements. We found this regularization to work well when the hole diameter is below the local feature size. In Section 8.1, we describe how introducing an additional minimum threshold for surface extraction can help with larger holes and missing surface data. Appendix F shows examples of the limits of this approach.

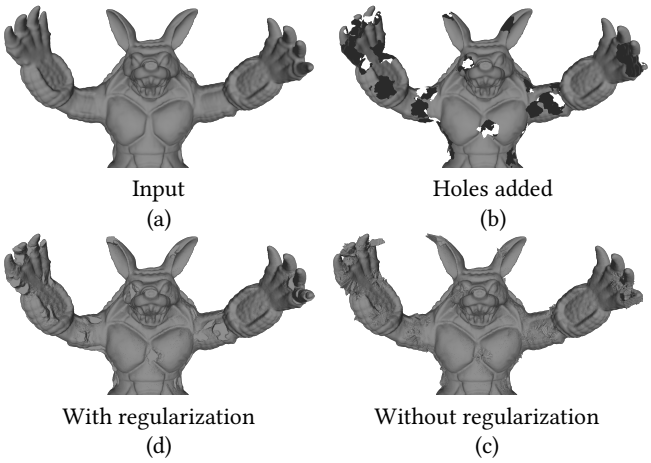



Fig. 6. The Armadillo model (a) with holes added (b). Computing $\widetilde{f}$ with $s$ computed as in (13) results in jagged completion (c), whereas applying a sequence of 1D box filters results in a smoother completion (d).

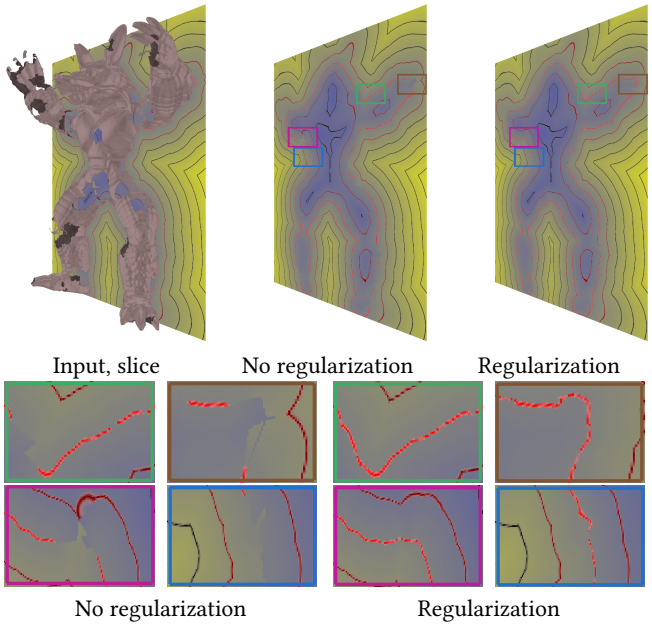



Fig. 7. The Armadillo with holes as in Fig. 6b with slices of $\widetilde{f}$ without and with regularization. The boxes highlight the effect of regularization.

## 6 SURFACE EXTRACTION

We can directly use the displaced SDF slices, *e.g.* for a preview. Figs. 3, 5, 6 and 8 show iso-surfaces extracted by marching cubes

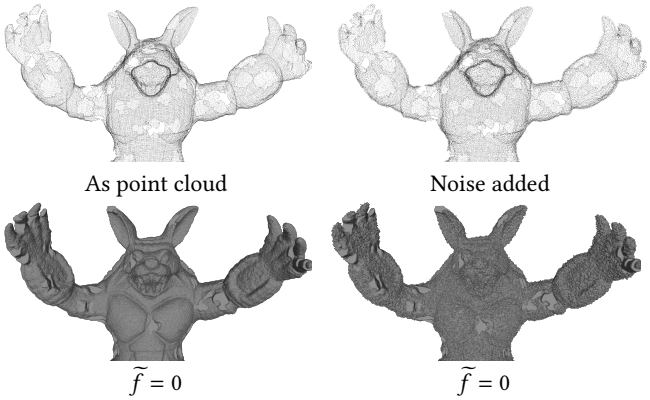



Fig. 8. Further corruptions of the Armadillo model. On top of the holes in Fig. 6b, we remove facet information to create a point cloud (left) and add noise (right). The resulting 0-level sets of $\widetilde{f}$ are shown underneath.

[Lorensen and Cline. 1987] directly from slices of $\widetilde{f}$. However, to pair the displaced SDF directly with a device, without generating an intermediate high-poly mesh via marching cubes or other meshing algorithm, we instead extract a sparse set of surface voxels.

From our displaced SDF (6), we extract the same ordered-column surface representation [Brunton et al. 2018], which allows subsequent efficient slice-wise evaluation of the unsigned distance to the nearest surface point. In Appendix B, we show how this can be modified for direct slice-wise evaluation of the SDF. Formally, we seek to extract the true surface as the 0 level set of (6),

$$\partial\mathcal{S} = \{\mathbf{u} \text{ s.t. } \widetilde{f}(\mathbf{u}) = 0\}. \tag{15}$$

We extract voxels containing $\partial\mathcal{S}$ to be those with $\widetilde{f} \leq 0$ that have at least one neighboring voxel satisfying $\widetilde{f} > 0$. We use a 6-neighborhood, which gives us a 6-separating voxel surface. This ensures a thin and sparse surface.

To address self-overlapping surfaces, we apply a simple floodfill-based hidden surface removal, which simultaneously identifies contiguous regions marked as outside the object according to (6) and (15), and labels those regions reaching the exterior of $\mathcal{B}$, as *exterior* regions. A lightweight post-process performs a pass over the surface voxels and discards those not bordering on any exterior regions.

At self-intersections themselves, it can occur that no voxel in the immediate neighborhood has $\widetilde{f}(\mathbf{u}) \leq 0$, due to the discrete nature of the distance computation and arbitrary tie-breaking in determining $\overline{\mathbf{u}}$. To address this, we extract the surface at a small positive iso-level

$$\partial\mathcal{S}_\tau = \{\mathbf{u} \text{ s.t. } \widetilde{f}(\mathbf{u}) = \tau > 0\}. \tag{16}$$

We found using $\tau = \max\{\delta_x, \delta_y, \delta_z\}$ to be sufficient even for surfaces with many small multiply-overlapping components. To subsequently recover $\partial\mathcal{S}$, we store $\widetilde{f}(\mathbf{u})$ with the extracted surface voxels, which we can then use to displace the SDF. Fig. 5b–d shows iso-surfaces $\widetilde{f} = 0$, $\widetilde{f} = \tau$ and the re-displaced iso-surface after hidden surface removal, respectively.

## 7 IMPLEMENTATION

We implement our framework in standard C++14, using `tbb` [Intel 2020] for multi-threading. Algorithms that are streaming compatible according to definition 3.1 are scalable w.r.t. build size at high resolution, but also very limited in the quantities they can store per voxel and the number of dense slices of voxels they can store at once. In our implementation, we store three 4-byte quantities in dense slices of voxels: $\widetilde{d}$, $\widetilde{f}$ and an index to look-up the nearest surface voxel. The storage for $\widetilde{f}$ is initially used to store $s$ for filtering before composition (6), and following extraction of $\partial\mathcal{S}_\tau$ the storage for $\widetilde{d}$ is re-purposed for the floodfill-based hidden surface removal. We processes small chunks of slices at a time; 16 is a typical chunk size, though this is set based on the number of processing cores.

### 7.1 Sparse Surface Voxelization

To ensure we capture even geometric features below the voxel resolution, we perform a conservative, 26-separating surface voxelization, registering every primitive-voxel intersection. This proceeds chunk wise, incrementally computing the surface voxels for small range along the $z$-axis. We do not store all voxels in this range, rather only those intersected by a primitive, using a look-up based on a hash-map of the voxel coordinates. Details on the surface voxelization algorithm are given in Appendix C.

We batch primitive-voxel intersections, defined by voxel coordinates $\mathbf{u}$, primitive normal $\mathbf{n}_p$, $\mathbf{c}_p(\mathbf{u})$, $a_p(\mathbf{u})$, along with quantities interpolated between vertices such as texture coordinates or color, together and upload the values to OpenGL textures. A fragment shader runs on the batch and computes the displacement $\phi$, per Sections 7.2 or 7.3, and performs additional processing, *e.g.*, surface color assignment. This is the only part of our implementation that uses a GPU.

We then write the results to a sparse raster. For each slice within the chunk, we use `std::unordered_map` to compute a map $(x, y) \mapsto \Sigma$, where $\Sigma$ denotes a surface voxel element comprising the attributes needed for a surface voxel

$$\Sigma = \left(\phi, \{\partial_{\omega_n} f \ : \ n = 1, \ldots, 6\}, \text{attribs}\right) \tag{17}$$

where attribs indicates other attributes that may be of interest, such as color. For each primitive $p$ that intersects a voxel, the map retrieves the corresponding $\Sigma$ and updates $\partial_{\omega_n} f$ per (11).

### 7.2 Displacement Mapping

Displacement maps are topographic textures that specify a height map relative to the tessellation geometry along the direction of the surface normal vector. In contrast to bump maps or normal maps used for shading, a displacement map alters the topography of the object, adding detail to the tessellation. Adding detail by refining the tessellation results in high polygon counts, increasing the storage, transmission and processing costs. A displacement map is typically a much more efficient representation for geometric detail. In this section, we describe our implementation of applying a displacement map to a low-poly mesh that is then printed with high surface detail.

Our surface representation stores displacement, along with other attributes such as RGBA values, with each surface voxel; a unique index per surface voxel allows access to these attributes. Using the

method of Brunton *et al.* [2018], we transfer this index to the discrete Voronoi cell of each surface voxel.

So far we have described $\phi$ in infinite precision. In practice, displacement maps are often given in fixed precision formats, with as few as 8-bits per pixel being typical. We interpret such displacement maps as $\hat{\phi}(\mathbf{t}) \in [0,1]$, $\mathbf{t} \in \mathbb{R}^2$, and apply a per-object displacement scale $\kappa$ to convert this value into a metric distance. Specifically,

$$\phi_{\mathcal{S}}(\mathbf{v}) = \kappa\hat{\phi}(\mathbf{t}(\mathbf{v})) \;:\; \mathbf{v} \in \mathcal{P} \tag{18}$$

where $\mathbf{t}(\mathbf{v})$ are the texture coordinates interpolated at $\mathbf{v}$ within a primitive, and $\kappa > 0$ means $\partial\mathcal{S}$ lies within a dilation by $\kappa$ of $\widetilde{\partial\mathcal{S}}$. To account for this dilation, we add a padding of sufficient voxels along the x-, y-, and z-axes. To avoid aliasing when sampling $\hat{\phi}$, which is typically a high-frequency texture, we apply mipmapping.

### 7.3 Curved Surfaces

We implement curved triangles by computing the distance along the normal of the flat triangle to the curved triangle during surface voxelization, and store this as the displacement value of the surface voxel created by the surface voxelization of the flat triangle. For PN triangles, we convert them to general Bézier triangles [Vlachos et al. 2001], and evaluate the position of the Bézier triangle using the de Casteljau algorithm [de Casteljau 1959].

Let $\mathbf{u} \in \widetilde{\partial\mathcal{S}}$ and $\mathbf{n}_p$ be the normal of a primitive $p$ intersecting the $V(\mathbf{u})$. We have

$$\phi_{\mathcal{S}}(\mathbf{v}) = \mathbf{n}_p^T(\mathbf{q}(\mathbf{v}) - \mathbf{v}) \tag{19}$$

for $\mathbf{v} \in p$, where $\mathbf{q}(\mathbf{v})$ is the point on the curved primitive constructed from $\mathbf{v}$. Combining with (9) gives

$$\phi(\mathbf{u}) = \frac{1}{|\mathcal{P}(\mathbf{u})|} \sum_{p \in \mathcal{P}(\mathbf{u})} \mathbf{n}_p^T(\mathbf{q}(\mathbf{c}_p(\mathbf{u})) - \mathbf{u}). \tag{20}$$

## 8 APPLICATIONS AND EVALUATION

In this section we apply our displaced SDFs to different tasks in additive manufacturing, and evaluate its performance. To evaluate the robustness and performance of our algorithm, we used a subset of 50 models from the Thingi10K [Zhou and Jacobson 2016], chosen to include large, intermediate and small triangle counts, and the Armadillo, Happy Buddha and Asian Dragon from the Stanford scanning repository [Stanford Computer Graphics Laboratory 2014]. To evaluate displacement mapping, we used a different set of four texture-mapped models as described in Section 8.2.

We created printed examples by pairing our framework with the implementation provided by the Cuttlefish SDK [IGD 2020] for color gamut mapping, separation, halftoning, and printer-specific output. We printed examples on a Mimaki 3DUJ-553 [Mimaki 2017].

### 8.1 Robust voxelization and fabrication

In this section, we evaluate the robustness of our approach to corrupted data. We show some visual results for real-world data in Figs. 1a, 9 and 10. Fig. 1a shows an open surface as a result of a scan, which our method is able to voxelize for printing. To obtain a near-constant thickness, we set a minimum level set value, −0.8 cm, and extract a surface at that iso-level in addition to the 0-level. Fig. 9 shows the result of printing a multi-view stereo reconstruction [Waechter et al. 2014] of a statue of Copernicus. Due to the many occluding surfaces, the scan has many holes, small and large with complex boundaries, and some floating triangles. The bottom of the model is completely open, since it is the ground surrounding the statue. Our approach preserves the detail of the textured surfaces and completes the model without introducing extraneous surface features. Fig. 10 shows a point cloud [Rodrigues et al. 2014] and the 3D printed result. The model includes colors and normals at each point. Again, the model is completely open on one side, and to print with a near-constant thickness, we use a second iso-level of −0.8 cm.

Fig. 9. This scan of a statue of Copernicus has many holes (boundaries highlighted in red), and is completely open on the bottom. The print on the right is neither missing structures nor introducing extraneous structures.

Fig. 10. A 3D printed point cloud.

Figs. 6 and 8 show the robustness of our method w.r.t. incomplete input and noise, with a ground truth for reference in Fig. 6a. Fig. shows robustness w.r.t. self-intersection and geometric features beyond the voxel resolution. We evaluated robustness quantitatively by corrupting models in our test set with different numbers and sizes of holes, removing facet information to create point clouds, and adding Gaussian noise along the surface normal. Fig. 11 shows

the Hausdorff distance for different numbers of holes, for an average hole radius of 0.02 (left), and for different hole sizes for a 100 holes. We see that the error increases gradually as the number of holes increase, but that the variance also increases. A similar behavior can be observed for increasing hole size. Additional results can be found in Appendix E. All lengths (radii, Hausdorff distance) are given as a fraction of the model's bounding box diagonal.

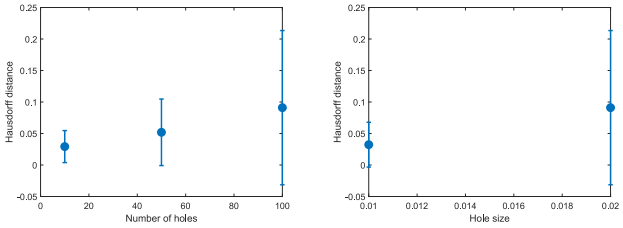



Fig. 11. Left: Hausdorff distance for different numbers of holes with an average hole radius of 0.02. Right: Hausdorff distance for different holes radii for 100 holes. All lengths are given as a fraction of the model's bounding box diagonal.

## 8.2 Displacement mapping

In this section, we compare the displacement mapping performance of our approach to that of vertex displacement, which was performed using Blender [2020]. A displacement map can be applied to a model in Blender using the `Subdivision Surface` and `Displace` modifiers to subdivide the surface and displace the vertices, respectively. We compare this method to ours in terms of geometric compactness, and processing efficiency. We also verify that with increasing subdivision, vertex displacement converges toward our displaced result. We use a test set of four models with displacement maps, shown in Fig. 14 and in Appendix D.

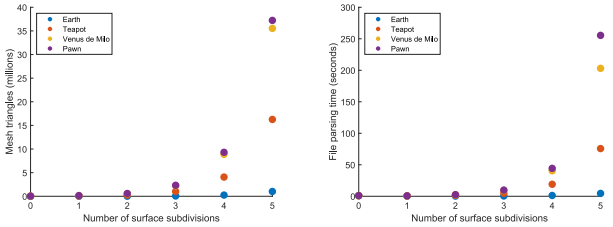
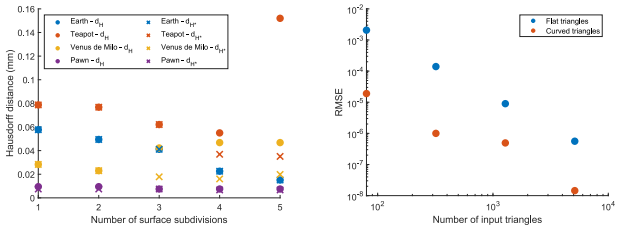



Fig. 12. Left: Increasing the number of surface subdivisions exponentially increases the number of triangles and mesh file size in different models. Right: Setup time, including reading from disk, parsing, and placing the model in the printer's build space, with increasing surface subdivisions for different models. The 0-level subdivision refers to the original tessellation with a displacement map applied to using our method.

Fig. 12 (left) shows the exponential increase in the number of triangles for increasing subdivision levels. Fig. S6 in Supplemental Appendix D shows the corresponding exponential increase in file size. In Fig. 12 (right), we compare the time required to load and parse the input mesh, and position it in the printer's build space for different subdivision levels, where level 0 indicates the original mesh with displacement map. For higher subdivision levels the time is dominated by file parsing. This is clearly dependent on the file format; we used the OBJ format, since it supports displacement maps, and is more widely supported by 3D printing software than other formats. Fig. 17a shows the processing time of our method for different levels of subdivisions. Again, 0 indicates the original mesh with displacement map. We see that the processing time for a fixed number of voxels does not change significantly (note the compressed range on the vertical axis). We choose this measure since the different subdivision levels result in different numbers of voxels to process (the object becomes bigger).

To compare the level of detail obtained using vertex displacement vs. our method, we look at the Hausdorff distance $d_H$ between our method and the different levels of subdivision. Fig. 13a shows both $d_H$ and the one-sided variant $d_{H*}(D,T)$, measuring the distance of $D$ to $T$, such that $d_H(D,T) = \max(d_{H*}(D,T), d_{H*}(T,D))$, where $D$ denotes the surface generated using the original tessellation with displacement map, and $T$ denotes the surface generated by subdivision. We can see that in general both decrease as the tessellated versions are refined. Self-intersections created by vertex displacement result in inside-out surfaces in some locations, which causes $d_H$ to increase for higher subdivision levels. However, $d_{H*}$ continues to decrease showing that with higher subdivision, the vertex displaced models approach the shape of applying displacement using our method.




(a) (b)

Fig. 13. Hausdorff distance between a low poly mesh with displacement and high-poly meshes subdivided and displaced using the same displacement map in Blender (a), and root mean squared error of a unit sphere represented using displaced SDF generated from flat and curved triangles (b).

Fig. 14 gives a visual comparison between the two methods for the teapot model, as does Fig. S1 in Appendix D for the earth model. The left and middle prints use vertex displacement after 1 and 5 subdivisions, respectively. The prints on the right were displaced using our method with the same displacement map. Our method attains the quality of 5 subvisions with a smaller file size than 1 subdivision. Figs. 1 (center) and 15 (left) show the benefits of displacement mapping in terms of visual realism.

## 8.3 Direct fabrication of curved surfaces

Fig. 16 shows a chess piece printed using flat and curved PN-triangles. The tessellation is no longer visible in the case of curved triangles. Fig. 1c shows how the same curved triangle tessellation scales to different print sizes while maintain precision to the native resolution of the device. No additional tessellation or subdivision is applied to the 5cm print (right) versus the 3cm print. Fig. 13b shows the

5.48 MB 1382.75 MB 4.25 MB

Fig. 14. A printed 9-cm teapot with 1 surface subdivision (left), 5 surface subdivisions (center), and displacement map applied using our method (right). Under each image we give the file size used to encode the geometry, including the displacement map for our method.

Fig. 15. A 13-cm teapot printed with a dishcloth displacement map (left) and without (right).

root mean squared error (RMSE) of a unit sphere represented using our displaced SDF generated from flat and curved (PN) triangles. The RMSE is measure on vertices extracted on the 0-set of $\widetilde{f}$ using marching cubes. The same number of curved triangles produces about two orders of magnitude lower error compare to flat triangles.

Fig. 16. A chess piece printed with flat (left) and PN-triangles (right).

## 8.4 Performance

Our algorithm has asymptotic theoretical running time of $O(1)$ per voxel, or linear in the number of voxels. In this section, we show experimentally its independence w.r.t. the number of primitives $|\mathcal{P}|$, its run-time in practice as the print size changes, and its low storage requirements, $< 1$ byte per voxel for moderate print sizes, decreasing for larger prints. We conducted all performance evaluations, including those for Section 8.2, on a laptop PC with an Intel i9 processor with 8 cores and 16 MB of cache, and 16 GB of memory.

To generate the data for Figs. 17b-d, we ran each of the 53 models from our test set at two print sizes chosen randomly from {3 cm, 5 cm, 10 cm, 15 cm, 20 cm}, where the print size is measured along the model's longest axis.

We see from Fig. 17a that the running time is independent of the number of primitives. Fig. 17b,c further shows that our implementation runs in constant time per voxel (linear in the number of voxels), and independent of the number of primitives. Each figure shows the running time, including surface voxelization, distance computation, sign estimation, evaluation of (6), surface voxel extraction and hidden surface removal. Over all 106 trials, the mean running time per million voxels was 0.0832 s.

Fig. 17d shows the peak storage per voxel of our framework for our test set. Blue points show individual trials, while the orange dots show the average peak storage for each print size. The peak

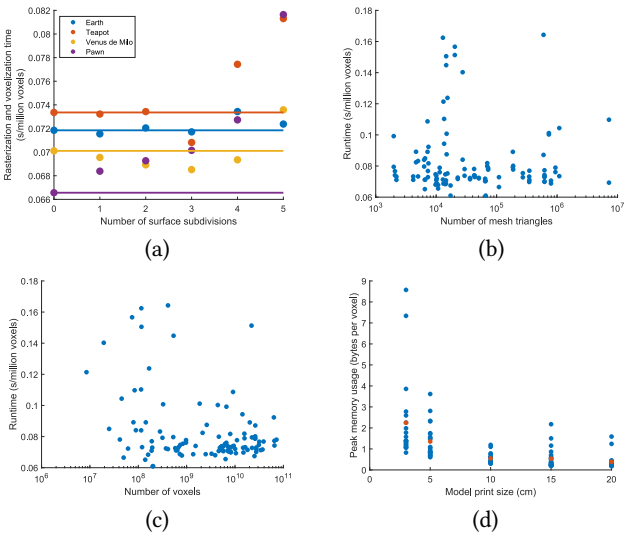



Fig. 17. The running time is constant with respect to the number of mesh triangles (a,b), and linear in the number of voxels, or constant for a fixed number of voxels (c). The per-voxel storage requirements decrease with increasing print size (d). Note the logarithmic horizontal scale in (a-c).

storage per voxel is computed as

$$B_{\text{voxel}} = (\max_t B_{\text{total}}(t))/(WHD) \tag{21}$$

where $t$ is wall time during program execution, $B_{\text{total}}(t)$ is total storage in bytes allocated for our implementation at time $t$, and $W$, $H$ and $D$ are the number of voxels along the $x$-, $y$- and $z$-axes, respectively. The total storage $B_{\text{total}}(t)$ is the sum at time $t$ of storage allocated for

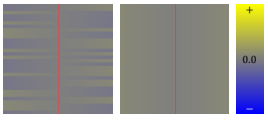
- temporary storage for surface voxelization (including overhead for hash maps),
- sparse surface voxel storage (including overhead and, *e.g.*, RGBA values not specific to our approach), and
- dense voxel storage for a chunk of slices.

We compute $B_{\text{total}}(t)$ at the end of processing of each $C$ slices in both the surface voxelization process and the computation of $\widetilde{f}$. This account misses some small 1D temporary stores $O(\max\{W, H\})$, which has a minimal effect on $B_{\text{voxel}}$.

## 9 LIMITATIONS

Our approach assumes an oriented surface as input, and does not model errors or noise in normals (triangle vertex order). When such problems occur on a small scale, our approach handles them with only small artifacts, but will fail when large portions of a surface are flipped, or there is significant noise in point normals. Appendix F shows examples of the limits of our regularization scheme, w.r.t. flipped triangles and large holes.

Our modeling of sub-voxel structures is theoretically limited to structures of non-zero wall thickness. The inset shows how for wall thickness 0, (13) results in arbitrary sign on each side, with the result that $\widetilde{f}$ (left) has a arbitrary error vs. $f$ (right). We observed that for walls as thin as $10^{-7}\delta$ the sign is correctly assigned, whereas for a wall of thickness $10^{-8}\delta$ sign errors occur. Walls of 0 thickness are possible in artist-created models, so this is something to address in future work. Appendix A provides more analysis. Note that this discussion considers computing sign via (13) with no regularization.




The floodfill post-process for self-overlapping surfaces means internal voids are not possible. This is a relatively minor problem in the context of additive manufacturing. For processes that allow for internal voids, such as FFF/FDM, voxel-compatible techniques exist [Tricard et al. 2020] to generate internal support structures. For polyjet processes used in graphical 3D printing, internal voids are anyways filled with support material, and voxel-based hollowing or lattice generation to save build material is likely advantageous.

While our approach avoids the pitfall of self-intersecting surfaces that can arise with vertex displacement, it does not validate that the combination of macroscopic and mesoscopic geometry constitute a physically realizable shape. A large negative displace on a thin wall can cause the wall to locally disappear, and a positive displacement in a crevice can fill it in. Providing checks and feedback for such invalid input is an avenue for future work.

Using a single displacement value per surface voxel creates a problem for curved triangles when the corresponding flat triangles induce large dihedral angles across the edges between them. This results in a small bump along the edge.

## 10 CONCLUSION

We have presented an efficient framework based on displaced signed distance fields for robust fabrication of finely detail and curved surfaces from compact representations. Our framework allows the augmentation of low-polygon tessellations with displacement maps for meso-scopic surface detail, and the direct fabrication of curved triangle surfaces. In both cases it does this without subdivision and the ensuing risk of creating self-intersecting surfaces. Our algorithms run constant time per voxel. We have verified the robustness, quality and performance qualitatively and quantitatively.

Looking forward, it would be interesting to combine our displaced SDF with neural implicit representations such as Davies *et al.* [2021]. Currently, displacement maps will scale with the object as it is scale, since the texture parametrization is fixed. Allowing the displacement signal to have a fixed scale would allow the mesoscopic information to maintain its frequency as the object changes in size. Including support for other types of curved triangles, *e.g.* Lagrange simplices, would simplify interfacing 3D printing with FEM analysis.

## ACKNOWLEDGMENTS

We thank Philipp Urban, Marco Dennstädt, Johann Reinhard and Mostafa Morsy for helpful discussions; Marketiger BV for printing; and the anonymous reviewers for insightful feedback. Lubna Abu Rmaileh was funded by EU Horizon 2020 ITN project ApPEARS no. 814158. Further information on the models and textures used in this paper can be found in the supplemental material.

## REFERENCES


M. Alexa, K. Hildebrand, and S. Lefebvre. 2017. Optimal Discrete Slicing. *ACM TOG* 36, 1, Article 12 (Jan. 2017), 16 pages. https://doi.org/10.1145/2999536

M. Attene. 2010. A lightweight approach to repairing digitized polygon meshes. *The Visual Computer* 26, 11 (2010), 1393–1406.

M. Atzmon and Y. Lipman. 2020. SAL: Sign Agnostic Learning of Shapes from Raw Data. In *Proc. CVPR*.

M. Atzmon and Y. Lipman. 2021. SALD: Sign Agnostic Learning with Derivatives. In *Proc. ICLR*.

G. Barill, N. Dickson, R. Schmidt, D.I.W. Levin, and A. Jacobson. 2018. Fast Winding Numbers for Soups and Clouds. *ACM TOG (Proc. SIGGRAPH)* (2018).

M. Berger, A. Tagliasacchi, L.M. Seversky, P. Alliez, G. Guennebaud, J.A. Levineand A. Sharf, and C.T. Silva. 2017. A Survey of Surface Reconstruction from Point Clouds. *Computer Graphics Forum* 36, 1 (2017), 301–329. https://doi.org/10.1111/cgf.12802

J.-P. Berrut and L.N. Trefethen. 2004. Barycentric Lagrange Interpolation. *SIAM Rev.* 46, 3 (2004), 501–517.

S. Bischoff, D. Pavic, and L. Kobbelt. 2005. Automatic restoration of polygonal models. *ACM TOG* 24, 4 (2005).

Blender Online Community. 2020. *Blender - a 3D modelling and rendering package*. http://www.blender.org

A. Brunton, C. A. Arikan, T. M. Tanksale, and P. Urban. 2018. 3D Printing Spatially Varying Color and Translucency. *ACM TOG (Proc. SIGGRAPH)* 37, 4 (2018), 157:1–157:13.

D. Cohen-Or and A. Kaufman. 1995. Fundamentals of Surface Voxelization. *Graphical Models and Image Processing* 57, 6 (November 1995), 453–461.

R.L. Cook. 1984. Shade Trees. In *Proc. SIGGRAPH 1984*. 223–231.

R.L. Cook, L. Carpenter, and E. Catmull. 1987. The Reyes Rendering Architecture. In *Proc. SIGGRAPH 1987*. 95–102.

T. Davies, D. Nowrouzezahrai, and A. Jacobson. 2021. On the Effectiveness of Weight-Encoded Neural Implicit 3D Shapes. arXiv:2009.09808 [cs.GR]

P. de Casteljau. 1959. *Outillage méthodes calcul*. Technical Report.

E. Eisemann and X. Decoret. 2008. Single-pass gpu solid voxelization and applications. In *Proc. of Graphics Interface (GI)*. 73–80.

P. Erler, P. Guerrero, S. Ohrhallinger, N.J. Mitra, and M. Wimmer. 2020. Points2Surf Learning Implicit Surfaces from Point Clouds. In *Proc. ECCV*. 108–124.

G. Farin. 1986. Triangular Bernstein-Bézier patches. *Computer Aided Geometric Design* 3, 2 (1986), 83–127. https://doi.org/10.1016/0167-8396(86)90016-6

M. Garland and P.S. Heckbert. 1995. *Fast polygonal approximation of terrains and height fields*. Technical Report CMU-CS-95-181.

H. Gohari, A. Barari, and H. Kishawy. 2018. An efficient methodology for slicing NURBS surfaces using multi-step methods. *International Journal of Advanced Manufacturing Technology* 95 (2018), 3111–3125.

A. Gropp, L. Yariv, N. Haim, M. Atzmon, and Y. Lipman. 2020. Implicit Geometric Regularization for Learning Shapes. In *Proc. ICML*.

A. Guéziec, G. Taubin, F. Lazarus, and B. Hom. 2001. Cutting and stitching: converting sets of polygons into manifold surfaces. *IEEE TVCG* 7, 2 (2001).

H. Hoppe, T. Derose, T. Duchamp, J. McDonald, and W. Stuetzle. 1993. Mesh optimization. In *Proc. ACM SIGGRAPH*.

Fraunhofer IGD. 2020. Cuttlefish SDK. https://www.cuttlefish.de/.

Intel. 2020. Intel Threading Building Blocks. https://software.intel.com/content/www/us/en/develop/tools/threading-building-blocks.html.

A. Jacobson, L. Kavan, and O. Sorkine-Hornung. 2013. Robust inside-outside segmentation using generalized winding numbers. *ACM TOG (Proc. SIGGRAPH)* 32, 4 (2013).

T. Ju. 2004. Robust repair of polygonal models. *ACM TOG* 23, 3 (2004).

M. Kazhdan, M. Bolitho, and H. Hoppe. 2006. Poisson Surface Reconstruction. In *Proc. SGP*.

M. Kazhdan and H. Hoppe. 2013. Screened Poisson surface reconstruction. *ACM TOG* 32, 3 (2013).

R. Kolluri. 2005. Provably Good Moving Least Squares. In *Symposium on Discrete Algorithms*.

V. Kraevoy, A. Sheffer, and C. Gotsman. 2003. Matchmaker: constructing constrained texture maps. *ACM TOG* 22, 3 (2003).

B. Krayer and S. Müller. 2019. Generating signed distance fields on the GPU with raymaps. *The Visual Computer* 35 (2019), 961–971.

W.E. Lorensen and H.E. Cline. 1987. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *SIGGRAPH Comput. Graph.* 21, 4 (Aug. 1987), 163–169. https://doi.org/10.1145/37402.37422

J. Martinez, S. Hornus, F. Claux, and Sylvain Lefebvre. 2015. Chained segment offsetting for ray-based solid representations. *Computers & Graphics* 46 (2015), 36–47.

Mimaki. 2017. 3DUJ-553. https://mimaki.com/product/3d/3d-inkjet/3duj-553/.

D. Nehab and H. Hoppe. 2008. Random-Access Rendering of General Vector Graphics. *ACM TOG* 27, 5, Article 135 (Dec. 2008), 10 pages. https://doi.org/10.1145/1409060.1409088

F.S. Nooruddin and G. Turk. 2003. Simplication and Repair of Polygonal Models Using Volumetric Techniques. *IEEE TVCG* 9, 2 (2003), 191–205.

A.C. Oeztireli, G. Guenneband, and M. Gross. 2008. Feature Preserving Point Set Surfaces based on Non-Linear Kernel Regression. *Computer Graphics Forum (Proc. Eurographics)* (2008).

J. Podolak and S. Rusinkiewicz. 2005. Atomic volumes for mesh completion. In *Proc. SGP*.

Z. Qin, M.D. McCool, and C.S. Kaplan. 2006. Real-Time Texture-Mapped Vector Glyphs. In *Proc. I3D 2006*. 125–132. https://doi.org/10.1145/1111411.1111433

J. Reinhard. 2017. *Discrete Medial Axis Transform and Applications for 3D Printing*. Bachelor Thesis. Technische Universität Darmstadt.

J. Rodrigues, M. Gazziro, N. Goncalves, O. Neto, Y. Fernandes, A. Gimenes, C. Alegre, and R. Assis. 2014. *The 12 prophets dataset*. Technical Report ICMC-USP-400. ICMC, University of Sao Paulo. 1–9 pages. www.aleijadinho3d.icmc.usp.br

R. Sawhney and K. Crane. 2020. Monte Carlo Geometry Processing: A Grid-Free Approach to PDE-Based Methods on Volumetric Domains. *ACM TOG (Proc. SIGGRAPH)* 39, 4 (2020).

M.-P. Schmidt. 2019. Additive Manufacturing of A 3D Part. Patent Application US20190134915A1. https://patents.google.com/patent/US20190134915A1/

M. Schwarz and H.-P. Seidel. 2010. Fast Parallel Surface and Solid Voxelization on GPUs. *ACM Transaction on Graphics (Proc. SIGGRAPH Asia)* 29, 6 (2010).

J. Shade, S. Gortler, L.-W. He, and R. Szeliski. 1998. Layered Depth Images. In *Proc. SIGGRAPH 1998 (SIGGRAPH '98)*. 231–242. https://doi.org/10.1145/280814.280882

V. Sitzmann, J.N.P. Martel, A.W. Bergman, D.B. Lindell, and G. Wetzstein. 2020. Implicit Neural Representations with Periodic Activation Functions. In *Proc. NeurIPS*.

Stanford Computer Graphics Laboratory. 2014. The Stanford 3D Scanning Repository. https://graphics.stanford.edu/data/3Dscanrep/.

B. Starly, A. Lau, W. Sun, W. Lau, and T. Bradbury. 2005. Direct slicing of STEP based NURBS models for layered manufacturing. *Computer Aided Design* 37 (2005), 387–397.

J.P. Stevens and D.J. McKenna. 2018. Preparing a polygon mesh for printing. Patent US10137646B2. https://patents.google.com/patent/US10137646B2/

Stratasys. 2016. J750. http://www.stratasys.com/3d-printers/production-series/stratasys-j750.

Stratasys. 2020. Design for Additive Manufacturing with PolyJet. https://my.stratasys.com/SupportCenter/HTML5UserGuides/Design_DFAM_Guide_July_2020/Responsive%20HTML5/index.html#t=DOC-01103_x_Design-PJ-AM-Guide-HTML%2FDfAM_Guide-Chapter%2FDfAM_Guide-Chapter.htm%23TOC_Additional_Resourcesbc-1&rhtocid=_1_0.

L. Szirmay-Kalos and T. Umenhoffer. 2006. Displacement Mapping on the GPU–State of the Art. *Computer Graphics Forum* 25, 3 (2006), 1–24.

T. Tricard, F. Claux, and S. Lefebvre. 2020. Ribbed Support Vaults for 3D Printing of Hollowed Objects. *Computer Graphics Forum* 39, 1 (2020), 147–159. https://doi.org/10.1111/cgf.13750

K. Vidimče, S.-P. Wang, J. Ragan-Kelley, and W. Matusik. 2013. OpenFab: A Programmable Pipeline for Multi-Material Fabrication. *ACM TOG (Proc. SIGGRAPH)* 32, 4 (2013).

A. Vlachos, J. Peters, C. Boyd, and J.L. Mitchell. 2001. Curved PN Triangles. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics (I3D '01)*. Association for Computing Machinery, New York, NY, USA, 159–166. https://doi.org/10.1145/364338.364387

M. Waechter, N. Moehrle, and M. Goesele. 2014. Let There Be Color! Large-Scale Texturing of 3D Reconstructions. In *Proc. ECCV*. 836–850.

T. Wohlers, I. Campbell, O. Diegel, R. Huff, and J. Kowen. 2020. *Wohlers Report 2020: 3D Printing and Additive Manufacturing Global State of the Industry*. Wohlers Associates, Inc.

S. Yamakawa and K. Shimada. 2009. Removing self intersections of a triangular mesh by edge-swapping, edge hammering, and face lifting. In *Proc. IMR*.

Q. Zhou and A. Jacobson. 2016. Thingi10K: A Dataset of 10,000 3D-Printing Models. arXiv:2009.09808 [cs.GR]

O.C. Zienkiewicz, R.L. Taylor, and J.Z. Zhu. 2013. Chapter 6 - Shape Functions, Derivatives, and Integration. In *The Finite Element Method: its Basis and Fundamentals* (seventh edition ed.), O.C. Zienkiewicz, R.L. Taylor, and J.Z. Zhu (Eds.). Butterworth-Heinemann, Oxford, 151 – 209. https://doi.org/10.1016/B978-1-85617-633-0.00006-X