Half-Space Power Diagrams and Discrete Surface Offsets

Zhen Chen, Daniele Panozzo, Jérémie Dumas

Abstract—We present an efficient, trivially parallelizable algorithm to compute offset surfaces of shapes discretized using a *dexel* data structure. Our algorithm is based on a two-stage sweeping procedure that is simple to implement and efficient, entirely avoiding volumetric distance field computations typical of existing methods. Our construction is based on properties of *half-space* power diagrams, where each seed is only visible by a half-space, which were never used before for the computation of surface offsets. The primary application of our method is interactive modeling for digital fabrication. Our technique enables a user to interactively process high-resolution models. It is also useful in a plethora of other geometry processing tasks requiring fast, approximate offsets, such as topology optimization, collision detection, and skeleton extraction. We present experimental timings, comparisons with previous approaches, and provide a reference implementation in the supplemental material.

Index Terms—Geometry Processing, Offset, Voronoi Diagram, Power Diagram, Dexels, Layered Depth Images.

1 Introduction

Morphological operations, such as dilation and erosion, have numerous applications: They can be used to regularize shapes [1], to ensure robust designs in topology optimization [2], to perform collision detection [3], or to compute image skeletons [4]. By combining these operations, it is possible to compute surface offsets. Offset surfaces are often used in digital fabrication applications [5], to generate support structures [6], to hollow object (Figure 1), to create molds, and to remove topological noise.

While offset surfaces can be computed exactly with Minkowski sums [7], these operations can be slow, especially on large models. Recent approaches [8] provide better results, but their performance is still insufficient for their use in interactive applications. Conversely, approximate algorithms which rely on a discrete re-sampling of the input volume, achieve efficiency by sacrificing accuracy in a controlled way [9, 10]. These methods are especially relevant in digital fabrication where resolutions are inherently limited by the machine fabrication tolerance, and using exact computation is unnecessary.

We propose a novel algorithm to compute offset surfaces on a solid object represented with a ray-based representation (ray-rep). A ray-rep, such as the *dexel buffer* [11], stores the intersections between a solid object and a set of parallel rays emanating from a uniform 2D grid: Each cell of the grid holds a list of *intervals* bounding the solid (Figure 2). Ray-reps are appealing in the context of digital fabrication, since they allow us to compute morphological operations directly at the resolution of the machine. CSG operations can be carried out directly in image space [12, 13], at a fraction of the cost of their counterparts on meshes, for which fast and robust implementations are notoriously difficult [14]. Another advantage is that implicit surfaces can be efficiently converted into ray-reps, avoiding explicit meshing. Examples in the literature can be found for Computer Numerical Control (CNC) milling applications [11], modeling for additive manufacturing [12], hollowing, or contouring [5]. Our offset algorithm exploits the ray-rep representation, and it is both fast and *accurate*, in the sense that it computes the *exact* offset at the resolution used by the input dexel structure. We demonstrate experimentally that our algorithm scales well with the offset radius, in addition to being embarrassingly parallel and thus can fully exploit multi-core, shared-memory architectures.

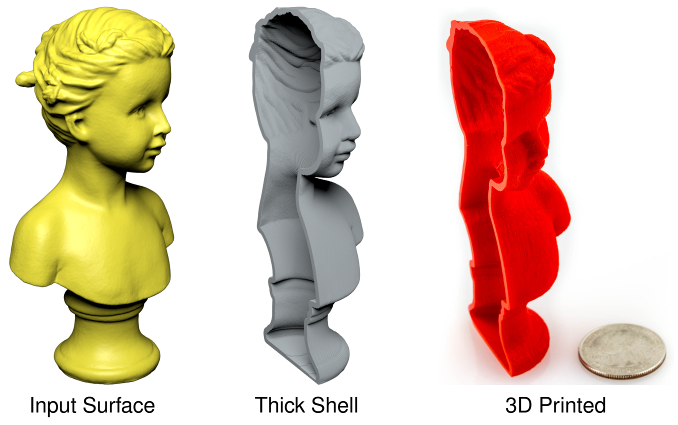

Fig. 1. 3D printed surface shell, computed with our approach. The shell is the Boolean difference of a dilation and an erosion of the input volume. The model uses a grid of 512^2 dexels, and a dilation radius of 4 dexels.

Contribution. Our algorithm relies on a novel approach for computing half-plane Voronoi diagrams [15]. We first describe this approach in 2D, demonstrating a simple and efficient sweeping algorithm to compute offsets in a 2D dexel structure. We then extend our sweeping procedure to 3D by leveraging the separability of the Euclidean distance [16].

In Section 4, we compare running times of our algorithm with existing approaches, and showcase applications of our approach in topological simplification and modeling for additive manufacturing.

2 Related Work

Exact Mesh Offsets. Dilated surfaces can be computed exactly for triangle meshes, by dilating every triangle and explicitly resolving the introduced self-intersections [17, 18]. A dilated mesh can also be obtained by computing the Minkowski sum of the input triangle mesh and a sphere of the desired radius [7]. This approach, which is implemented robustly in CGAL [19], generates exact results, but is slow for large models. A variant of this algorithm has been introduced in [20], where the arbitrary precision arithmetic is replaced by standard floating-point arithmetic. However, the performance of this method is still insufficient to achieve interactive runtimes. Offset surfaces can be used for shape optimization, e.g., to optimize weight distribution inside an object [21].

Resampled Offsets. To avoid the explicit computation of the Minkoswki sum, which is a challenging and time-consuming operation, other methods resample the offset surface by computing the isosurface of the signed-distance function of the original surface [22, 23]. Other methods relying on adaptive sampling include [24, 25], but they are known to have a large memory overhead when a high accuracy is required. Meng *et al.* [8] distributes and optimizes the position of sites at a specified distance from the original surface, and produces a triangle from a restricted Delaunay triangulation of the sites. Calderon *et al.* [26] introduced a framework for performing morphology operations directly on point sets.

Voronoi Diagrams and Signed-Distance Transforms. Centroidal tessellations of Voronoi and power diagrams have important application in geometry processing and remeshing [27, 28]. Our algorithm relies on these techniques, in particular on the implicit computation of Voronoi and power diagrams of points and segments.

Fortune [29] introduces a sweep line algorithm to position the Voronoi vertices (intersection of 3 bisectors) and create a 2D Voronoi diagram of point sets. By relying on a slight modification of the Voronoi diagram definition, called half-space Voronoi diagrams [15], we will see in Section 3 how to compute the Voronoi diagram of a set of parallel segments directly, using two sweeps instead of one, and how it leads to the efficient computation of a discrete offset surface. The extension of our method to 3D requires the computation of power diagrams [30] with a weight associated to each seed. Our algorithm implicitly relies on the Voronoi and power diagrams of line segments: while those could be approximated by sampling each segments with multiple points [31], we opted for an alternative solution which is more efficient and simpler to implement.

Finally, our 3 dimensional, two-stage sweeping algorithm bears some similarity to existing signed-distance transform approaches [16, 32], as it leverages the separability of the Euclidean distance to compute the resulting offset by sweeping in two orthogonal directions. However, as we operate on a dexel structure, we never need to store a full 3D distance field in memory. For a more complete review of distance transform algorithms, the reader is referred to the survey [33].

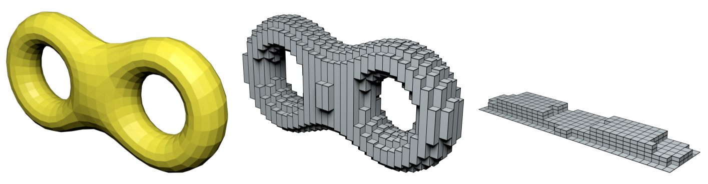

Fig. 2. A dexel data structure constructed from a triangle mesh. The input surface (a) intersects with evenly spaced parallel rays (b), and is stored compactly as a 2D grid of events in the dexel buffer (c). This can be used to perform efficient CSG operations in modeling software [12].

Medial Axis and Skeleton. Medial axis and shape skeletons are widely used in geometry processing applications, such as shape deformation, analysis, and classification [34]. One popular approach is "thinning" [35, 36], which exploits an erosion operator to reduce a shape to its skeleton. Our algorithm provides a practical and efficient way of defining such an operator. A shape can be approximated by a union of balls centered on its medial axis, and Voronoi diagrams are a common way of computing candidate centers for these balls [37]. These algorithms are know to be sensitive to small scale surface details [38], and special care needs to taken to ensure robustness [39, 40].

Ray-Based Representations and Offsets. A ray-based representation of a shape is obtained by computing the intersection of a set of rays with the given shape. In most applications, the cast rays are parallel and sampled on a uniform 2D grid. They are typically stored in a structure called *dexel buffer*. A dexel buffer is simply a 2D array, where each cell contains a list of *intersection* events $(z^{\vdash}, z^{\dashv})$, each one representing one intersection with the solid (Figure 2). To the best of our knowledge, the term dexel (for *depth pixel*) can be traced back to [11], which introduced the dexel buffer to compute the results of CSG operations to ease NC milling path-planning. Similar data structures have been described in different contexts over the years. *Layered Depth Images* (LDI) [41] are used to achieve efficient image-based rendering. The A-buffer [42, 43] was used to achieve order-independent transparency. It is worth mentioning that, while the construction algorithm is different, the underlying data structures used in all these algorithms remain extremely similar. The G-buffer [44] stores a normal in every pixel for further image-processing and CNC milling applications, augmenting ray-reps with normal information. In the context of additive manufacturing, *Layered Depth Normal Images* have been proposed as an alternative way to discretize 3D models [45, 13].

Ray-based data structures offer an intermediate representation between usual boundary representations (such as triangle meshes), and volumetric representations (such as a full or sparse 3D voxel grids). While both 3D images and dexel buffers suffer from uniform discretization errors across the volume, a dexel buffer is cheaper to store compared to

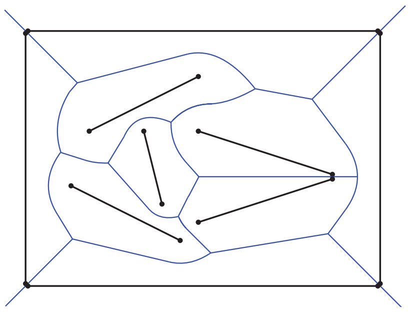

Fig. 3. Voronoi diagram of line segments. The bisectors are defined by second-order piecewise polynomial curves. Note that the Voronoi cell of a single seed can be comprised of multiple connected components. Illustration from [49].

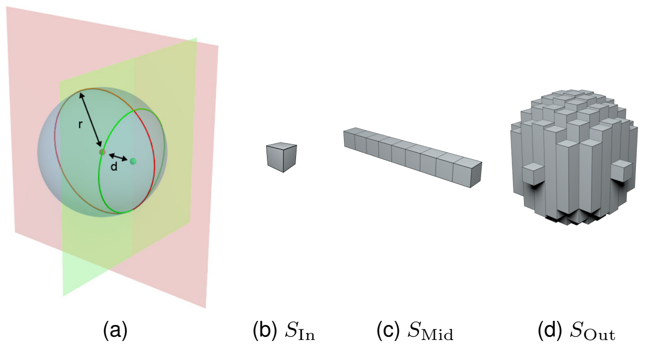

Fig. 4. The dilation operation is performed in two stages (a). A seed is dilated by a radius r along a first axis (in red). The resulting seed (in green) is then dilated along a second axis by a different radius $\sqrt{d^2 - r^2}$, where d is the distance between the two seeds (red point and green point). The equivalent dexel data structure for each pass is shown on right: (b) input dexel, (c) result of the first pass, (d) result of the second pass.

a full volumetric representation: it is possible to represent massive volumes (2048^3 and more) on standard workstations. Additionally, in the context of digital fabrication, 3D printers and CNC milling machines have limited precision: a dexel buffer at the resolution of the printer is sufficient to cover the space of shapes that can be fabricated.

Ray-reps have been used to compute and store the results of Minkoswki sums, offsets, and CSG operations [46]. Hui [47] uses ray-reps to compute a solid sweeping in image-space. In [48], Chen *et al.* use LDNI to offset polygonal meshes by filtering the result of an initial overestimate of the true dilated shape. Wang *et al.* [9] computes the offset mesh as the union of spheres placed on the points sampled by the LDI. Finally, [10] approximates the dilation by a spherical kernel with a zonotope, effectively computing the Minkoswki sum of the original shape with a sequence of segments in different directions. Differently, the method presented in this paper computes the *exact offset* of the discrete input ray-rep (at the resolution of the dexel representation). Despite using a similar data structure, our approach is different from Wang *et al.* [9]: the latter requires a LDI sampled from 3 orthogonal directions, while our method accelerates the offset operation even when a ray-rep from a single direction is available. We provide a comparison and describe the differences in more details in Section 4.

Voronoi Diagrams of Line Segments and Motivations. An example of a Voronoi diagram of line segments is shown in Figure 3. Although the general abstract algorithm for Voronoi diagrams applies (in theory) to line segments [29], the problem is in practice extremely challenging, since the bisector of two segments are piecewise second-order polynomial curves. Some software is readily available to solve the 2D case (e.g., VRONI [50, 51], Boost.Polygon [52], OpenVoronoi [53] and CGAL [54]), but the problem is still open in 3D. Aurenhammer *et al.* [55] present a method for computing the *Voronoi diagram* of *parallel* line segments by reducing the problem to computing the *power diagram* of points in a hyperplane, but the problem of computing the *power diagram of line segments* is still open, even in 2D. To the best of the authors' knowledge, the problem of computing the *power diagram for a set of line segments* was not explored in the literature, not even in 2D. This component is necessary for the second step of our dilation process, and it is one of the contributions of this work. Note that our method does not explicitly store a full Voronoi or power diagram at any time, since the process is *online* (the cells are computed for the current sweepline only and updated as our sweep progresses).

Compared to existing offsetting techniques, there are several advantages to using our discrete approach when the result needs only be computed at a fixed resolution (e.g., the printer resolution in additive manufacturing).

(1) No need for a clean input triangle mesh. As long as the input triangle soup can be properly discretized (e.g., using generalized winding number [56, 57]), the input can have gaps or self-intersecting triangles.

(2) Our method is also directly applicable when the input is a 3D image (e.g., CT scan), where it can be used without any loss of accuracy, providing high performance and low memory footprint.

(3) Another advantage of working directly with the dexel data structure is that CSG operations can be carried out easily in dexel space, providing real-time interactive modeling capabilities [12]. Performing CSG operations on triangle meshes is costly and requires a significant implementation effort to be performed robustly [14]. Compared to [9], our method can achieve higher resolutions when needed, and compared to [10], the result of the dilation by a spherical kernel is computed exactly.

3 Method

The key idea of our algorithm is to decompose the dilation of a point in two 2D dilations along orthogonal axes. Figure 4a illustrates this idea. The result of the 3D dilation of a point—the blue sphere in Figure 4a—can be computed by first performing the dilation along the first axis (in red). Then, the green point can be dilated along a second axis (in green) by a different radius, producing the green circle. By applying this construction directly to a dexel buffer, the input point in Figure 4b is first dilated to produce the line in Figure 4c. Then, each element of this line is dilated again in the orthogonal direction, but with a different dilation radius, producing the final result (Figure 4d). Each stage is a set of 2D dilations: the first stage relying on Voronoi diagrams (uniform offsetting) and the second stage leveraging power diagrams to compute efficient offsets against different dilation radii for each dexel segment. We denote by $S_{\text{In}} \to S_{\text{Mid}} \to S_{\text{Out}}$ the different steps of the pipeline.

We start by formally defining the different concepts and notations in Section 3.1. In Section 3.2, we describe our uniform offsetting method (in 2D), based on efficiently computing the Voronoi diagram of a set of parallel segments. Then, in Section 3.3, we extend the algorithm to compute the power diagram of a set of parallel segments in 2D, which completes the necessary building blocks to extend our algorithm from 2D to 3D.

3.1 Definitions

Dexel Representation. A dexel buffer of a shape $S \subset \Omega$ (with $\Omega \overset{\text{def}}{=} \mathbb{R}^3$) is a set of parallel segments arranged in a 2D grid, where each cell S_{ij} of the grid contains a list of segments sharing the same xy coordinate. Specifically, we discretize S as $S \approx \cup_{ij} S_{ij}$ where $S_{ij} = \{(\boldsymbol{z}_k^{\vdash}, \boldsymbol{z}_k^{\dashv})\}_{k=1}^{n_{ij}}$. Each segment $(\boldsymbol{z}_k^{\vdash}, \boldsymbol{z}_k^{\dashv})$ in the same cell S_{ij} has the same xy coordinate, and represents the intersections of the input shape S with a vertical ray at the same xy coordinate (see Figure 2).

Dilation. Given an input shape S and a radius $r \geqslant 0$ we define the *dilated shape* $\mathcal{D}_r(S)$ as the set of points at a distance less or equal than r from S:

$$\mathcal{D}_r(S) = \{\boldsymbol{p} \in \Omega, \|\boldsymbol{p} - \boldsymbol{x}\| \leqslant r, \boldsymbol{x} \in S\}. \tag{1}$$

Erosion. The erosion of a shape S is equivalent to computing the dilation on the complemented shape $\overline{S}$, and taking the complement of the result. Since the complement operation is trivial under a ray-rep representation, we will focus on describing our algorithm applied to the dilation operation, from which all other morphological operators (erosion, closing, and opening) will follow.

Voronoi Diagram. The *Voronoi diagram* of a set of *seeds* $\mathfrak{S} = \{\mathfrak{s}_i\}_{i=1}^n$ is a partition of the space Ω into different *Voronoi cells* Ω_i:

$$\Omega_i = \{\boldsymbol{p} \in \Omega, \operatorname{dist}(\boldsymbol{p}, \mathfrak{s}_i) \leqslant \operatorname{dist}(\boldsymbol{p}, \mathfrak{s}_j), i \neq j\}. \tag{2}$$

We also define $\operatorname{Vor}(\mathfrak{S})$ to be the interface between each overlapping Voronoi cell:

$$\operatorname{Vor}(\mathfrak{S}) = \cup_{i \neq j} \Omega_i \cap \Omega_j. \tag{3}$$

Notations. In the context of this paper, seeds can be either points or line segments. Throughout the paper, $\mathfrak{s}_i$ shall denote the geometric entity of a seed, whether it is a point or a line segment. $\boldsymbol{s}_i$ shall denote the position of the seed when it is a point, and $(\boldsymbol{s}_i^{\vdash}, \boldsymbol{s}_i^{\dashv})$ shall be used to denote the positions of its endpoints when the seed is a line segment. When the seeds $\{\mathfrak{s}_i\}_i$ are points, $\operatorname{Vor}(\mathfrak{S})$ is a set of straight lines in 2D (planes in 3D), which are equidistant to their closest

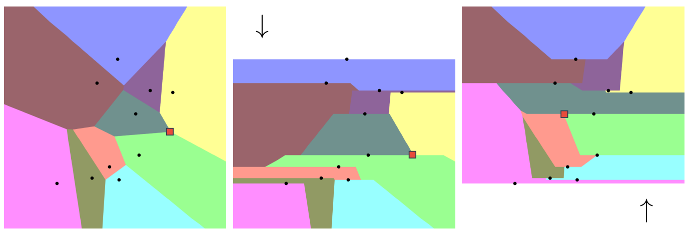

Fig. 5. Voronoi diagram of seed points $\{\mathfrak{s}_i\}_{i=1}^n$. From left to right: Voronoi diagram formed by the full Voronoi cells Ω_i; half-space Voronoi diagram formed by the half-space Voronoi cells $\overrightarrow{\Omega}_i$ and $\overleftarrow{\Omega}_i$ respectively. Note that $\Omega_i \subseteq \overrightarrow{\Omega}_i \cup \overleftarrow{\Omega}_i$. In each diagram, a Voronoi vertex (intersection between 3^+ Voronoi cells) is shown with a red square.

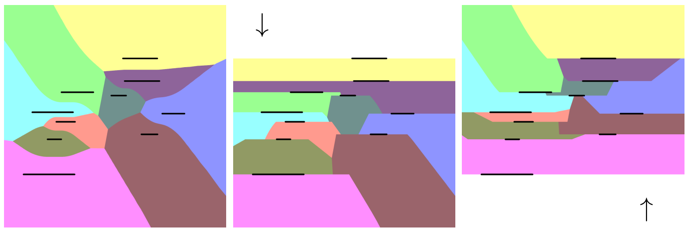

Fig. 6. Voronoi diagram of seed segments $\{\mathfrak{s}_i\}_{i=1}^n$. From left to right: Voronoi diagram formed by the full Voronoi cells Ω_i; half-space Voronoi diagram formed by the half-space Voronoi cells $\overrightarrow{\Omega}_i$ and $\overleftarrow{\Omega}_i$ respectively.

seeds. When the seeds are segments, $\operatorname{Vor}(\mathfrak{S})$ is comprised of parabolic arcs in 2D [31], and parabolic surfaces in 3D.

In a half-space Voronoi diagram for seed points [15], each seed point $\mathfrak{s}_i$ is associated with a *visibility direction* $\boldsymbol{v}_i$, and a point $\boldsymbol{p}$ is considered in Equation (2) if and only if $(\boldsymbol{p} - \boldsymbol{s}_i) \cdot \boldsymbol{v}_i \geqslant 0$. In this work, we are interested in half-space Voronoi diagrams of seed segments, where each seed is associated to the same visibility direction $\boldsymbol{v}$. Figure 5 (resp. Figure 6) shows the difference between Voronoi diagrams and half-space Voronoi diagrams for point seeds (resp. segment seeds). More precisely, given a set of parallel seed segments $(\boldsymbol{s}_i^{\vdash}, \boldsymbol{s}_i^{\dashv})_{i=1}^n$, and a direction $\boldsymbol{v}$ orthogonal to each segment $\boldsymbol{v} \perp (\boldsymbol{s}_i^{\dashv} - \boldsymbol{s}_i^{\vdash})$, we define the half-space Voronoi cells $\overrightarrow{\Omega}_i$ and $\overleftarrow{\Omega}_i$ as

$$\begin{aligned} \overrightarrow{\Omega}_i = \{&\boldsymbol{p} \in \Omega, \|\boldsymbol{p} - \widetilde{\boldsymbol{p}}_i\| \leqslant \|\boldsymbol{p} - \widetilde{\boldsymbol{p}}_j\|, i \neq j, \\ &(\boldsymbol{p} - \widetilde{\boldsymbol{p}}_i) \cdot \boldsymbol{v} \geqslant 0, (\boldsymbol{p} - \widetilde{\boldsymbol{p}}_j) \cdot \boldsymbol{v} \geqslant 0\} \\ \overleftarrow{\Omega}_i = \{&\boldsymbol{p} \in \Omega, \|\boldsymbol{p} - \widetilde{\boldsymbol{p}}_i\| \leqslant \|\boldsymbol{p} - \widetilde{\boldsymbol{p}}_j\|, i \neq j, \\ &(\boldsymbol{p} - \widetilde{\boldsymbol{p}}_i) \cdot \boldsymbol{v} \leqslant 0, (\boldsymbol{p} - \widetilde{\boldsymbol{p}}_j) \cdot \boldsymbol{v} \leqslant 0\} \end{aligned} \tag{4}$$

where $\widetilde{\boldsymbol{p}}_i$ is the point $\boldsymbol{p}$ projected on the line $(\boldsymbol{s}_i^{\vdash}, \boldsymbol{s}_i^{\dashv})$. Note that the segments $(\boldsymbol{s}_i^{\vdash}, \boldsymbol{s}_i^{\dashv})$ are parallel, following a direction that is orthogonal to the chosen direction $\boldsymbol{v}$. Thus, the dot product $(\boldsymbol{p}-\boldsymbol{q})\cdot\boldsymbol{v}$ in Equation (4) has the same sign for all the points $\boldsymbol{q}$ in the line $(\boldsymbol{s}_i^{\vdash}, \boldsymbol{s}_i^{\dashv})$. Note that we have $\Omega_i \subseteq \overrightarrow{\Omega}_i \cup \overleftarrow{\Omega}_i$ (see Figure 5).

Half-Dilated Shape. We define the *half-dilated shape* $\overrightarrow{\mathcal{D}}(S)$, as the dilation restricted to the half-space Voronoi cells of segments in S:

$$\overrightarrow{\mathcal{D}}_r(S) = \{\boldsymbol{p} \in \overrightarrow{\Omega}_i, \|\boldsymbol{p} - \boldsymbol{x}\| \leqslant r, \boldsymbol{x} \in \mathfrak{s}_i, i \in [\![1, n]\!]\}. \tag{5}$$

Remember that $\mathfrak{s}_i$ is the i-th segment $(\boldsymbol{z}_i^{\vdash}, \boldsymbol{z}_i^{\dashv})$ in the dexel buffer approximating the shape S. The half-space dilated

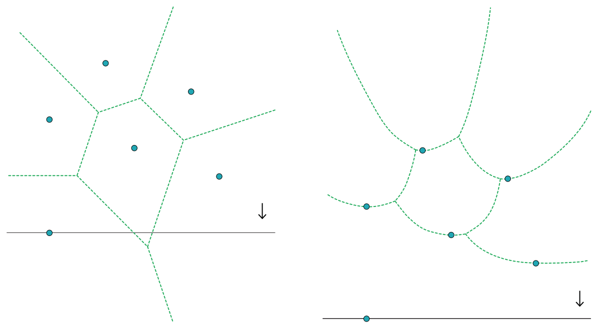

Fig. 7. Fortune's sweepline algorithm [29] requires transforming the point coordinates to compute the correct Voronoi diagram of points in a single sweeping pass, which makes it impossible to compute the result of the dilation in the same sweeping pass (without back-propagating the contribution of each newly inserted seed in the dilated shape).

shape $\overleftarrow{\mathcal{D}}_r(S)$ is defined in a similar manner using $\overleftarrow{\Omega}_i$. For simplicity, we will omit the dilation radius and simply write $\mathcal{D}(S)$, unless there is an ambiguity.

Power Diagram. Finally, the *power diagram* of a set of seeds $\{\mathfrak{s}_i\}_{i=1}^n$ is a weighted variant of the Voronoi diagram. Each seed is given a weight w_i that determines the size of the *power cell* Ω_i^{p} associated to it:

$$\Omega_i^{\mathrm{p}} = \{\boldsymbol{p} \in \Omega, \operatorname{dist}(\boldsymbol{p}, \mathfrak{s}_i)^2 - w_i \leqslant \operatorname{dist}(\boldsymbol{p}, \mathfrak{s}_j)^2 - w_j, i \neq j\}. \tag{6}$$

Intuitively, one could interpret a 2D power diagram as the orthographic projection of the intersection of a set of parabola centered on each seed, where the weights determine the height of each seed embedded in $\mathbb{R}^3$. This definition extends naturally to *half-space power diagrams*.

3.2 Half-Space Voronoi Diagram of Segments and 2D Offsets

Given a 2D input shape S and dilation radius r, we seek to compute the dilated shape $\mathcal{D}(S)$ comprised of the set of points at a distance $\leqslant r$ from S. The input shape is given as a union of disjoint parallel segments evenly spaced on a regular grid (the dexel data structure), and we seek to compute the output shape as another dexel structure (the discretized version of the continuous dilated shape).

The dilated shape $\mathcal{D}(S)$ can be expressed as the union of the dilation of each individual segment of S. To compute this union efficiently, our key insight is to partition the dilated shape based on the Voronoi diagram of the input segments (power diagrams are only needed for the extension to 3D). Within each Voronoi cell Ω_i, if a point $\boldsymbol{p}$ is at a distance $\leqslant r$ from the seed segment $(\boldsymbol{s}_i^{\vdash}, \boldsymbol{s}_i^{\dashv}) \in S$, then $\boldsymbol{p} \in \mathcal{D}(S)$.

The Voronoi diagram of point seeds can be computed in single pass with a sweepline algorithm [29]. The construction requires lifting coordinates in the plane according to the coordinate along the sweep direction, as illustrated in Figure 7. Unfortunately, this approach cannot be used to compute the result of a dilation operation in a single pass without "backtracking". Indeed, whenever a new seed is added to the current sweepline, its dilation will affect rows of the image above the sweepline, which have already processed. Instead,

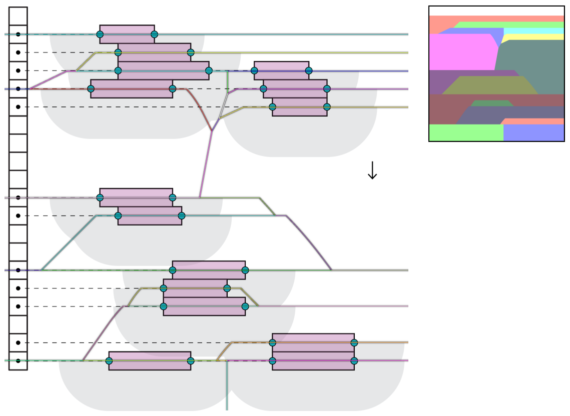

Fig. 8. A single sweep in one direction allows us to compute the half-space Voronoi diagram of parallel line segments (the dexel data structure). This figure illustrates how dexels are stored as nested arrays. The gray area shows the expected result of the half-dilation in one direction, while the solid colored lines show the boundary of the half-space Voronoi diagram of the input line segments (half-space Voronoi cells are shown on the top right).

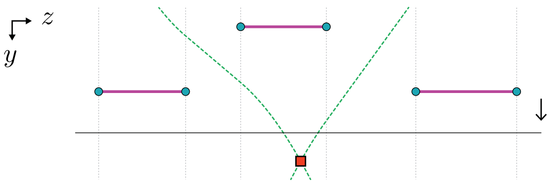

Fig. 9. The bisector of two line segments can be described by a piecewise second-order polynomial curve. Voronoi vertices are located at the intersection between two such bisectors. After this point, the middle segment will always be *further* away from the sweepline than its two neighbors. It will be marked as *inactive* and be removed from $\mathcal{L}$.

we propose a simple construction for seed points and segments, based on half-space Voronoi diagrams—which we extend to power diagrams in Section 3.3. Our key idea is to compute the dilation of a segment $(\boldsymbol{s}_i^{\vdash}, \boldsymbol{s}_i^{\dashv})$ in its Voronoi cell Ω_i as the union of the two half-dilated segments, in $\overrightarrow{\Omega}_i$ and $\overleftarrow{\Omega}_i$. Both $\overrightarrow{\Omega}_i$ and $\overleftarrow{\Omega}_i$ can be computed efficiently in two separate sweeps of opposite direction, without requiring any transformation to the coordinate system as in [29].

The idea behind our sweeping algorithm is as follows. We advance a sweepline L parallel to the seed segments in the input dexel S. A general view of a 2D dexel buffer and the sweep process is presented in Figure 8, while Figure 9 shows the structure of the Voronoi diagram of three line segments during the sweep process. At each step, for each seed $\mathfrak{s}_i \in S$, we compute the intersection of the current line L with the points in $\boldsymbol{p} \in \overrightarrow{\Omega}_i$ that are at a distance $\leqslant r$. By choosing the visibility direction $\boldsymbol{v}$ to be the same as the sweeping direction, then only the seeds $\{\mathfrak{s}_i\}_i$ previously encountered in the sweep will contribute to this intersection (any upcoming seed will have an empty Voronoi cell $\overrightarrow{\Omega}_i$). Figure 5 shows a Voronoi diagram of points, with two half-space Voronoi diagrams of opposite directions. To make the computation efficient, we do not want to iterate through all the seeds to compute the intersection each time we advance the sweep line (which would make the algorithm $\mathcal{O}(n^2)$ in the number of seeds). Instead, we want to keep only a small list of *active seeds*, that will contribute to the dilated shape $\mathcal{D}(S)$ and intersect the current sweepline L: this will decrease the complexity to $\mathcal{O}(n\log(n)+m)$, where m is the number of line segments generated by the dilation process.

Pseudocode. From an algorithmic point of view, we maintain, during the sweeping algorithm, two data structures: $\mathcal{L}$, the list of *active* seed segments $(\boldsymbol{s}_i^\vdash, \boldsymbol{s}_i^\dashv)$, whose Voronoi cell $\overrightarrow{\Omega}_i$ intersects the current sweepline, (and are thus at distance $\leqslant r$ from the current sweepline), and $\mathcal{Q}$, a priority queue of upcoming *events*. These events indicate when an active seed can safely be removed from the $\mathcal{L}$ as we advance the sweepline, i.e., when it no longer affects the result of the dilation. The events in $\mathcal{Q}$ can be of two types: (1) a Voronoi vertex (the intersection between 3^+ Voronoi cells); and (2) a seed becoming inactive due to a distance $> r$ from the sweepline. Voronoi vertices can be located at the intersection of the bisector curves between 3 consecutive seed segments $a, b, c \in \mathcal{L}$, as illustrated in Figure 9. Beyond this intersection, we know that either a or c will be closer to the sweepline, so we can remove b from $\mathcal{L}$ (its Voronoi cell $\overrightarrow{\Omega}_b$ will no longer intersect the sweepline).

Both $\mathcal{Q}$ and $\mathcal{L}$ can be represented by standard STL data structures. Note that since the seeds stored in $\mathcal{L}$ are *disjoints*, they can be stored efficiently in a `std::set<>` (sorted by the coordinate of their midpoint). A detailed description of our sweep-line algorithm is given in pseudocode in Figures 10 and 11. A full C++ implementation is available on github[1]. In line 13 in Figure 10, the function DILATELINE computes the result of the dilation on the current sweep line, by going through the list of active seeds, and merging the resulting dilated *line* segments. Insertion and removal of seed segments in $\mathcal{L}$ is handled by the functions INSERTSEEDSEGMENT and REMOVESEEDSEGMENT respectively. When inserting a new active *seed* segment in $\mathcal{L}$, to maintain the efficient storage with the `std::set<>`, we need to remove subsegments which are occluded by the newly inserted seed segment (and split partially occluded segments). Indeed, the contribution of such seed segments is superseded by the new seed segment that is inserted. Then, we need to compute the possible Voronoi vertices formed by the newly inserted seed segment and its neighboring seeds in $\mathcal{L}$. The derivation for the coordinates of the Voronoi vertex of 3 parallel seed segments in given in Appendix A.

```
Input: 2D dexel structure $S$ + dilation radius $r$.
Output: Half-dilated dexel shape $S' = \overrightarrow{\mathcal{D}}_r(S)$.
 1: function VORONOISWEEPLINE($S, r$)
 2:     $\mathcal{L} \leftarrow \emptyset$ ▷ Set of active segments on the sweep line
 3:     $\mathcal{Q} \leftarrow \{\}$ ▷ List of removal events marking a seed as inactive
 4:     $S' \leftarrow \emptyset$ ▷ Dilated result
 5:     for i $\leftarrow 0, N-1$ do
 6:         for all $\mathfrak{s}_j \in \mathcal{Q}$ do
 7:             REMOVESEEDSEGMENT($\mathcal{L}, r, \mathfrak{s}_j$);
 8:         end for
 9:         for all $\mathfrak{s}_j^i = (\boldsymbol{z}^\vdash, \boldsymbol{z}^\dashv) \in S_i$ do
10:             INSERTSEEDSEGMENT($\mathcal{L}, \mathcal{Q}, r, \mathfrak{s}_j$);
11:         end for
12:         ▷ Dilate and merge active seeds on the current sweepline
13:         $S' \leftarrow S' \cup$ DILATELINE($\mathcal{L}, i, r$)
14:     end for
15:     return $S'$
16: end function
```

Fig. 10. Sweepline algorithm for computing the half-dilated shape $\overrightarrow{\mathcal{D}}_r(S)$.

```
Input: $\begin{cases} \mathcal{L} \text{ Set of active seeds on the sweep line,} \\ \mathcal{Q} \text{ List of removal events,} \\ r \text{ Dilation radius,} \\ \mathfrak{s} \text{ Seed segment to insert, } \mathfrak{s} = [(y, z^\vdash), (y, z^\dashv)]. \end{cases}$
Output: Updated list of active seeds $\mathcal{L}$ and events $\mathcal{Q}$.
 1: function INSERTSEEDSEGMENT($\mathcal{L}, \mathcal{Q}, r, \mathfrak{s}$)
 2:     REMOVEOCCLUDEDSEGMENTS($\mathcal{L}, \mathfrak{s}$)
 3:     SPLITPARTIALLYOCCLUDED($\mathcal{L}, \mathfrak{s}$)
 4:     $\mathcal{L} \leftarrow \mathcal{L} \cup \mathfrak{s}$ ▷ Overlaps are resolved, insert $\mathfrak{s}$ into $\mathcal{L}$
 5:     $\mathcal{Q} \leftarrow \mathcal{Q} \cup (y + r, \mathfrak{s})$ ▷ After this point, $\mathcal{D}_r(\mathfrak{s})$ will be empty
 6:     while $\exists$ sequence $(\mathfrak{s}_a, \mathfrak{s}_b, \mathfrak{s}) \in \mathcal{L}$ do
 7:         $(y_v, z_v) \leftarrow$ VORONOIVERTEX($\mathfrak{s}_a, \mathfrak{s}_b, \mathfrak{s}$) ▷ See fig. 9
 8:         if $y_v < y$ then
 9:             $\mathcal{L} \leftarrow \mathcal{L} \setminus \mathfrak{s}_b$ ▷ Seed $\mathfrak{s}_b$ becomes inactive
10:         else
11:             $\mathcal{Q} \leftarrow \mathcal{Q} \cup (y_v, \mathfrak{s}_b)$ ▷ Remove $\mathfrak{s}_b$ later on
12:         end if
13:     end while
14:     while $\exists$ sequence $(s, s_a, s_b) \in \mathcal{L}$ do
15:         ▷ Repeat operation on the right side of $s$
16:     end while
17: end function
```

Fig. 11. Insertion of a new seed segment $\mathfrak{s}$ into $\mathcal{L}$.

3.3 Half-Space Power Diagram of Points and 3D Offsets

3D Dilation. As illustrated in Figure 4, the 3D dilation process is decomposed in two stages $S_{\text{In}} \mapsto S_{\text{Mid}} \mapsto S_{\text{Out}}$. We first perform an extrusion along the first axis (in red), followed by a dilation along the second axis (in green). Note that the first operation $S_{\text{In}} \mapsto S_{\text{Mid}}$ is not exactly the same as a dilation along the first axis (red plane): the segments $(\boldsymbol{s}_i^\vdash, \boldsymbol{s}_i^\dashv) \in S_{\text{In}}$ are *extruded*, not dilated (they map to a rectangle, not a disk).

To obtain the final dilated shape in 3D, we need to perform a dilation of the intermediate shape S_{Mid}, where each segment $(\boldsymbol{s}_j^\vdash, \boldsymbol{s}_j^\dashv) \in S_{\text{Mid}}$ is dilated by a different radius r_j along the second axis (green plane in Figure 4), depending on its distance from parent seed (red dot in Figure 4). In the case where the first extrusion of S_{In} produces overlapping segments in S_{Mid}, the overlapping subsegments would need to be dilated by different radii, depending on which segment in S_{In} it originated from. In such a case, where a subsegment $\tilde{\mathfrak{s}}_j \in S_{\text{Mid}}$ has multiple parents $\tilde{\mathfrak{s}}_i \in S_{\text{In}}$, it is enough to dilate $\tilde{\mathfrak{s}}_j$ by the radius of its closest parent in S_{In} (the one for which $r_j^i = \sqrt{d_{ij}^2 - r^2}$ is the largest). In practice, we store S_{Mid} as a set of non-overlapping segments, as computed by the algorithm in Figure 10, with a slight modification to the DILATELINE function (line 13 in Figure 10) to return the *extruded* seeds on the current sweepline instead of the dilated ones.

1. https://github.com/geometryprocessing/voroffset

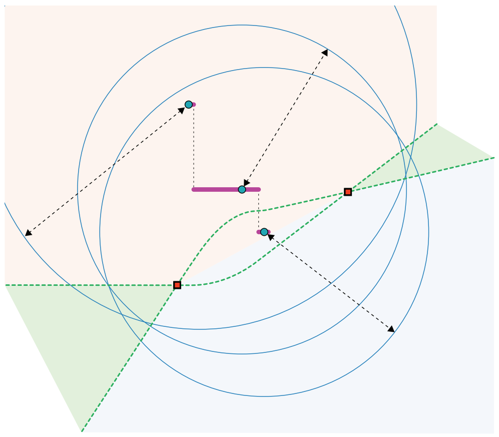

Fig. 12. Special case: the power cell (in green) of a seed segment (central segment in purple) can be a disconnected region of the plane. The blue points represent the center of the power circles (the boundary of the power diagram is the locus of the lines intersecting each pair of circle as the centers move along their respective segment).

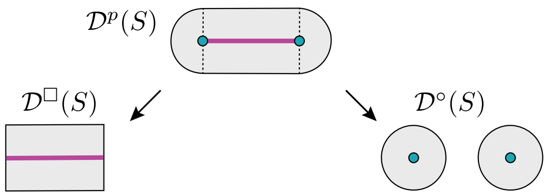

Fig. 13. To simplify the computation of the power diagram in the second stage, the dilation of a segment is separated into a vertical extrusion (left), and the dilation of its two endpoints (right).

2D Power Diagram. We now focus on computing the 2D dilation of a shape S composed of non-overlapping segments $(\boldsymbol{s}_i^{\vdash}, \boldsymbol{s}_i^{\dashv}; r_i)$, where each seed segment is weighted by the dilation radius ($w_i \overset{\text{def}}{=} r_i$ in Equation (6)). In this setting, the computation of the power cells Ω_i^{p} of seed segments becomes more involved, breaking some of the assumptions of the sweepline algorithm. Specifically, the sweepline algorithm (Figure 10) makes the following assumptions about $\mathcal{L}$: the seeds projected on the sweepline are non-overlapping segments, and the Voronoi cells induced by the active seeds are *continuous* regions. Once a seed $(\boldsymbol{s}_i^{\vdash}, \boldsymbol{s}_i^{\dashv})$ is inserted in $\mathcal{L}$, its cell $\overrightarrow{\Omega}_i$ immediately becomes *active*, and once we reach the first removal event in $\mathcal{Q}$, it will become *inactive* and stop contributing to the dilation $\overrightarrow{\mathcal{D}}(S)$. For the power cells of *segments*, the situation is a little bit different, as illustrated on Figure 12: a power cell $\overrightarrow{\Omega^{\mathrm{p}}}_i$ of a line segment can contain *disjoints* regions of the plane. It is not clear how to maintain a disjoint set of seeds in $\mathcal{L}$ if we need to start removing and inserting a seed multiple time, and this makes the number of cases to consider grow significantly.

Instead, we propose to circumvent the problem entirely by making the following observation. A 2D segment dilated by a radius r is actually a capsule, which can be described by two half-disks at the endpoints, and a rectangle in the middle. We decompose the half-dilated shape $\overrightarrow{\mathcal{D}^{p}}(S)$ into the union of two different shapes: $\overrightarrow{\mathcal{D}^{\circ}}(S)$ (the result of the half-dilation of each endpoint) and $\overrightarrow{\mathcal{D}^{\square}}(S)$, where each segment $\mathfrak{s}_i$ is extruded along the dilation axis by its radius r_i. This decomposition is illustrated in Figure 13.

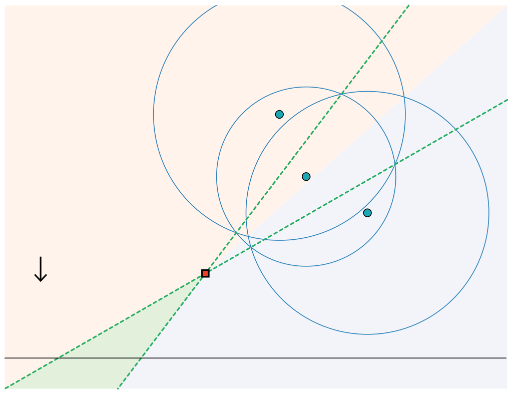

Fig. 14. Sweepline algorithm computing $\overrightarrow{\mathcal{D}^{\circ}}(S)$. When inserting the rightmost point in the set of active seeds $\mathcal{L}$, it may happen that the power vertex with its two neighbors be located further away along the sweep direction. In this case, the middle point should not be removed from $\mathcal{L}$, as its power cell will continue to intersect the sweepline.

Now, the dilated shape $\overrightarrow{\mathcal{D}^{\square}}(S)$ is easy to compute with a forward sweep, since there are no Voronoi diagrams or power diagrams involved: we simply remove occluded subsegments, keeping the ones with the largest radius, and removing a seed $\mathfrak{s}_i$ once its distance to the sweepline is $> r_i$. To compute $\overrightarrow{\mathcal{D}^{\circ}}(S)$, we employ a sweep similar to the one described Section 3.2, but now all the seeds are *points*, which greatly simplifies the calculation of power vertices. Indeed, the power bisectors are now lines, and the power cells are intersections of half-spaces (and thus convex). The derivations for the coordinates of a power vertex is given in Appendix B. The only special case to consider here is illustrated in Figure 14: when inserting a new seed and updating $\mathcal{L}$, the events corresponding to a power vertex do not always correspond to a removal. For example, in the situation illustrated in Figure 14, the power cell of the middle point will continue to intersect the sweepline after it has passed the power vertex, so we should not remove the middle seed from $\mathcal{L}$. Fortunately, this case is easy to detect, and we simply forgo the insertion of the event in $\mathcal{Q}$.

3.4 Complexity Analysis

We analyze the runtime complexity of our algorithm, assuming that the dexel spacing is 1, so that the dilation radius r is given in the same unit as the dexel numbers.

For the first 2D dilation operation, the complexity of combining two dexel data structures is linear in the size of the input, since the segments are already sorted. The cost of the forward dilation operation $\overrightarrow{\mathcal{D}}_r(S)$ requires a more detailed analysis: Let n be the number of input segments, and m be the number of output segments in the dilated shape $\overrightarrow{\mathcal{D}}(S)$. In the worst case, each input segment generates $\mathcal{O}(r)$ distinct output segments, $m = \mathcal{O}(nr)$, and each seed segment is split twice by every newly inserted segment. Since each seed can split at most one element of $\mathcal{L}$ into two separate segments, we have that, at any time, $|\mathcal{L}| = \mathcal{O}(n)$. Moreover, each seed produces at most three events in $\mathcal{Q}$ (a Voronoi vertex with its left/right neighbors, and the moment it becomes inactive because of its distance to the sweepline). It follows that $|\mathcal{Q}| = \mathcal{O}(n)$ as well. Segments in $\mathcal{L}$ are stored in a `std::set<>`, thus insertion and removal (lines 7 and 10 in Figure 10) can be performed in $\mathcal{O}(\log n)$ time. While the line dilation (line 13 in Figure 10) is linear in the size of $\mathcal{L}$, the total number of segments produced by this line cannot exceed m, so the amortized time complexity over the whole sweep is $\mathcal{O}(m)$. This brings the final cost of the whole dilation algorithm to a time complexity that is $\mathcal{O}(n\log(n) + m)$, and it does not depend on the dilation radius r (apart from the output size m). In contrast, the offsetting algorithm presented in [9] has a total complexity that grows proportionally to r^2.

For the second dilation operation, where each input segment is associated a specific dilation radius, the result is similar. Indeed, $\overrightarrow{\mathcal{D}^{\delta}}(S)$ is computed using the same algorithm as before, so the analysis still holds. Combining the dexels in $\overrightarrow{\mathcal{D}^{\delta}}(S)$ with the results from $\overrightarrow{\mathcal{D}^{\square}}(S)$ can be done linearly in the size of the output as we advance the sweepline, so the total complexity of computing $\overrightarrow{\mathcal{D}^{p}}(S)$ is still $\mathcal{O}(n\log(n)+m)$.

For the 3D case, the result of a first extrusion is used as input for the second stage dilation, the total complexity is more difficult to analyze, as it also depends on the structure of the intermediate result. In a conservative estimate, bounding the number of intermediate segments by $\mathcal{O}(nr)$, this bring the final complexity of the 3D dilation to $\mathcal{O}(nr\log(nr) + m)$, where $m = \mathcal{O}(nr^2)$ is the size of the output model. In practice, many segments can be merged in the final output, especially when the dilation radius is large, and m may even be smaller than n (e.g., when details are erased from the surface).

Finally, we note that in each stage of the 3D pipeline, the 2D dilations can be performed completely independently in every slice of the dexel structure, making the process trivial to parallelize. We discuss the experimental performance of a multi-threaded implementation of our method in Section 4.

4 Results

We implemented our algorithm in C++ using Eigen [58] for linear algebra routines, and Intel Threading Building Blocks [59] for parallelization. We ran our experiments on a desktop with a 6-core Intel® Core™ i7-5930K CPU clocked at 3.5 GHz and 64 GB of memory. Our reference implementation is available on github[2] to simplify the adoption of our technique. Note that our results are sensitive for the choice of the dexel direction, since it will lead to different discretizations. In our experiments, we manually select this direction. For 3D printing applications, Livesu *et al.* [5] [Section 3.2] give an overview of algorithms computing an optimized direction to increase the fidelity of the printing process.

Baseline Comparison. We implemented a simple brute-force dilation algorithm (on the dexel grid) to verify the correctness of our implementation, and to demonstrate the benefits of our technique. In the brute-force algorithm, each segment in the input dexel structure generates an explicit list of dilated segments in a disk of radius r around it, and all overlapping segments are merged in the output data structure. Figure 15 compares the two methods using a different number of threads, and with respect to both grid size and dilation radius. In all cases, our algorithm is, as expected, superior not only asymptotically but also for a fixed grid size or dilation radius. Since each slice can be treated independently in our two-stage dilation process, our algorithm is embarrassingly parallel, and scales almost linearly with the number of threads used.

We note that the asymptotic time complexity observed in Figure 15 agrees with our analysis in Section 3.4. Indeed, the dexel grid has a number of dexel $n \propto s^2$ is proportional to the squared grid size s^2, while the (absolute) dilation radius $r_{\text{abs}} \propto s r_{\text{rel}}$ grows linearly with the grid size. Since the complexity analysis in Section 3.4 uses a dilation radius r expressed in dexel units, the observed asymptotic rate of ≈ 3 indicates that our method is indeed $\mathcal{O}(s^3)$.

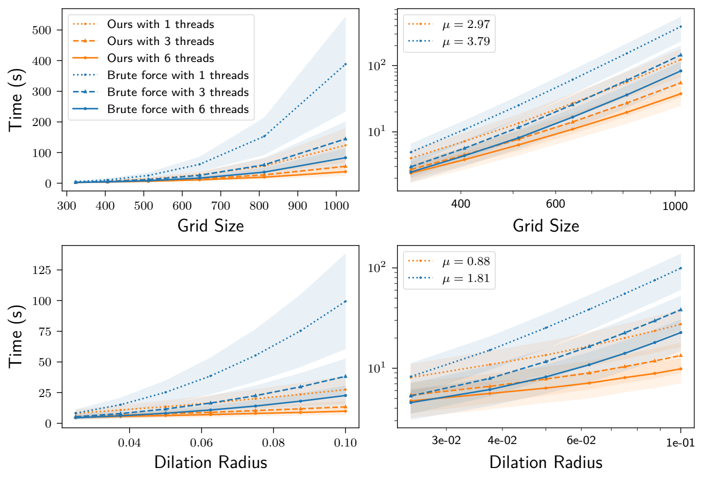

Fig. 15. Total running time averaged on 11 testing models, compared with a direct brute-force implementation. Standard deviation is shown in overlay, and convergence rate μ are reported in the log-log plots. Radii are relative to the grid size. The top row uses a relative radius of 0.05, and the bottom row uses a grid size of 512^2.

Dilation and Erosion. In Figure 16, we compare the performance of the dilation and erosion operator on a small data set of 11 models, provided in the supplemental material. The dilation operator has a higher cost than the erosion, both asymptotically and in absolute running time. This is because the erosion operator reduces the number of dexels, leading to a considerable speedup.

Comparison with Wang *et al.* [9]. The most closely related work on offset from ray-reps representations is Wang *et al.* [9]. It proposes to perform an offset from a LDI sampled from three orthogonal directions. The offset is computed as the union of spheres sampled at the endpoints of each segment from all three directions at once. In contrast, our method relies on a single dexel structure, which has both advantages and drawbacks. It is applicable in a situation where only one view is available, or where one view is enough to describe the model (e.g., modeling for additive manufacturing [12]).

2. https://github.com/geometryprocessing/voroffset

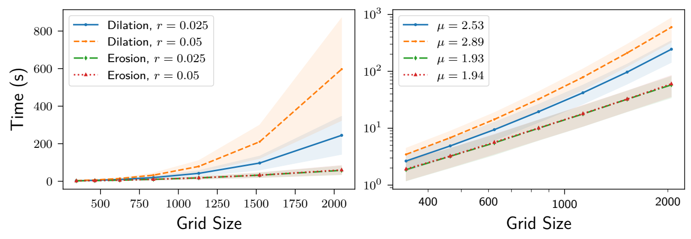

Fig. 16. Total running time for the dilation and erosion operators, for two different radii. Standard deviation is shown in overlay and the convergence rate μ is reported in a log-log plot.

The drawback is that it is less precise on the orthogonal directions (where the 3-views LDI will have more precise samples). However, the difference are minor, as shown in the comparison in Figure 17.

We report a performance comparison in Figure 18, where we compared our 6-core CPU version (running on a Intel® Core™ i7-5930K) with their GPU implementation [9] (running on a GTX 1060). The timings of the two implementations are comparable, suggesting that our CPU implementation is competitive even with their GPU implementation. Their implementation runs out of memory for the larger resolution (2048^3), while our implementation successfully computes the dilation.

Extending our method to the GPU is a challenging and notable venue for future work, that could enable real-time offsetting on large and complex dexel structures.

Comparison with Campen *et al.* [20]. In Figure 17, we compare our dilation algorithm (based on a dexel data structure) and [20] (based on an octree), matching their parameters to get a similar final accuracy. Their running time is higher and depends on the input complexity (45 s for Octa-flower, 519 s for Vase, and 2294 s for Filigree). We computed the Hausdorff distance between our result and theirs (Figure 17). In all cases the error remains in the order of the voxel/dexel size used for the discretization. Our results are visually indistinguishable from theirs, but are computed at a small fraction of the cost.

Topological Cleaning. Our efficient dilation and erosion operators can be combined to obtain efficient opening and closing operators (Figure 19). For example, the closing operation can be used to remove topological noise, i.e., small handles, by first dilating the shape by a fixed offset, and then partially undoing it using erosion. While most regions of the object will recover their original shape, small holes and sharp features will not, providing an effective way to simplify the topology.

Scalability. The compactness of the dexel representation enables us to represent and process immense volumes on normal desktop computers. An example is shown in Figure 20 for the erosion operation. Note that the results on the right have a resolution sufficiently high to hide the dexels: this resolution would be prohibitive with a traditional boundary or voxel representation (Figure 18).

3D Printing. The Boolean difference between a dilation and erosion of a shape produces a shell of controllable thickness. This operation is useful in 3D printing applications, since the interior of an object is usually left void (or filled with support structures) to save material. Another typical use case for creating thick shells out of a surface mesh is the creation of molds [61]. We show a high resolution example of this procedure in Figure 1, and we fabricated the computed shell using PLA plastic on an Ultimaker 3 printer.

Limitations. The main limitation of our method comes from the uneven sampling of the dexel data structure in a single direction (e.g., the flat areas on the sides of the reconstructed models in Figure 17). While this is not a problem if the application only needs a certain resolution to begin with (e.g., 3D printing or CT scans), it is not optimal for the purpose of reconstructing an output mesh, where other approaches such as [20, 9] will lead to a higher fidelity. We could use our method with a dexelized structure in 3 orthogonal directions (similar to LDI [13]), and reconstruct the output surface using [60], but this will lead to a threefold increase of the running times.

5 Future Work and Concluding Remarks

Our current algorithm is restricted to uniform morphological operations. It would be worthwhile to extend it to single direction thickening (for example in the normal direction only) or to directly work on a LDI offset, i.e., representing the shape with 3 dexel representations, one for each axis. A GPU implementation of our technique would likely provide a sufficient speedup to enable real-time processing of ray-reps representation at the resolution typically used by modern 3D printers.

We proposed an algorithm to efficiently compute morphological operations on ray-rep representations, targeting in particular the generation of surface offsets. Beside offering theoretical insights on power diagrams and their application to surface offsets, our algorithm is simple, robust, and efficient: it is an ideal tool in 3D printing applications, since it can directly process voxel or dexel representations to filter out topological noise or extract volumetric shells from boundary representations.

Acknowledgments

This work was supported in part by the NSF CAREER award IIS-1652515, the NSF grant OAC-1835712, a gift from Adobe, and a gift from nTopology.

References

[1] J. Williams and J. Rossignac, “Mason: Morphological simplification,” *Graphical Models*, vol. 67, no. 4, pp. 285–303, Jul. 2005. DOI: 10.1016/j.gmod.2004.10.001.

[2] O. Sigmund, “Morphology-based black and white filters for topology optimization,” *Structural and Multidisciplinary Optimization*, vol. 33, no. 4-5, pp. 401–424, Jan. 2007. DOI: 10.1007/s00158-006-0087-x.

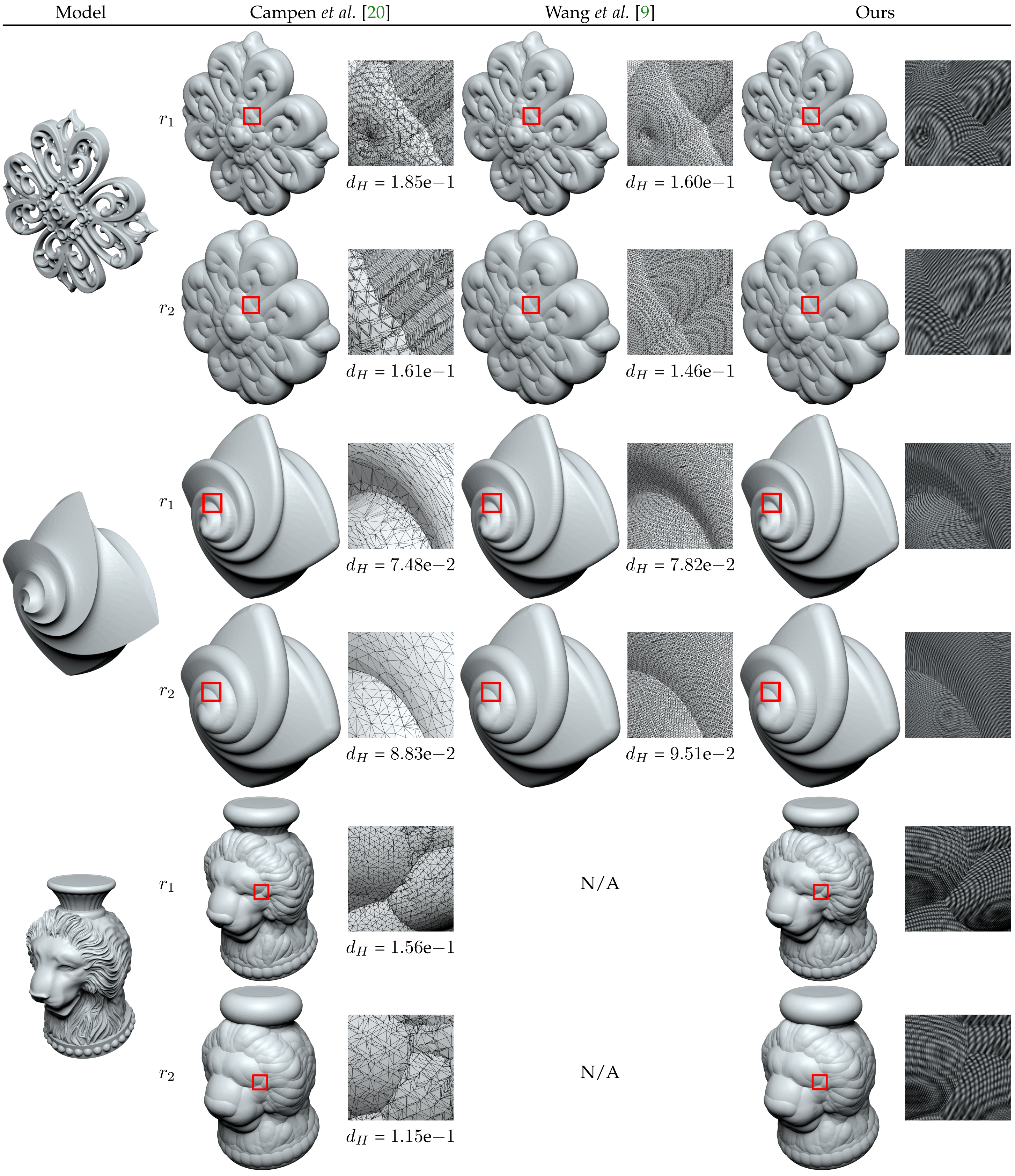

Fig. 17. Quality comparison between [20, 9], and our method. The dilation radii r_1 and r_2 are set to $0.025d$ and $0.05d$ respectively, where d is the bounding box diagonal of each model. The zoom insert corresponds to the red frame on each picture. Both [9] and our method use a discretization of 1024 dexels. The Hausdorff distance d_H between each result and ours is shown under the zoomed pictures, and is expressed as a *percentage of the bounding box diagonal*. Note that the error corresponds roughly to the dexel size used for the discretization. The surface from [20] is reconstructed by marching on cells of an octree, and [9] uses dual-contouring with normal information. For shading purposes, we show in our full-view a surface reconstructed from the dexel samples using [60], while our zoomed view shows the raw dexels. Results for the vase using [9] is unavailable due to a crash in the reference implementation provided by the authors.

	Radius	Resolution	Wang *et al.* [9] Normal	Wang *et al.* [9] Successive	Ours
Filigree	r_1	512	4.94	2.25	1.26
		1024	73.85	12.55	9.96
		2048	-	-	88.97
	r_2	512	10.96	3.64	2.14
		1024	176.65	22.94	19.04
		2048	-	-	280.96
Octa-flower	r_1	512	5.93	1.20	2.39
		1024	90.17	8.90	22.41
		2048	-	-	322.34
	r_2	512	13.68	2.53	5.04
		1024	224.40	19.52	54.99
		2048	-	-	1649.56
Vase	r_1	512	-	-	1.38
		1024	-	-	12.24
		2048	-	-	131.13
	r_2	512	15.71	-	2.57
		1024	-	-	25.61
		2048	-	-	524.90

Fig. 18. Timing (s) comparisons with [9] across different dexel resolutions. Dilation radii r_1 and r_2 are set to $0.025d$ and $0.05d$ respectively, where d is the diagonal of bounding box of the model. The two columns for [9] corresponds to the "GPU Primary" and "GPU SH+P+Succ" in their Table 2 respectively. A "-" indicates that the program terminated with an error (crash or went out of memory). [9] ran on a GeForce GTX 1060, while our comparisons on a 6-core Intel® Core™ i7-5930K CPU. Our multi-thread CPU implementation is competitive with a GPU implementation, while scaling to higher resolutions thanks to the memory efficient dexel data structure.

[3] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M. .-.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino, "Collision detection for deformable objects," *Computer Graphics Forum*, vol. 24, no. 1, pp. 61–81, Mar. 2005. DOI: 10.1111/j.1467-8659.2005.00829.x.

[4] P. Maragos and R. Schafer, "Morphological skeleton representation and coding of binary images," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 34, no. 5, pp. 1228–1244, Oct. 1986. DOI: 10.1109/tassp.1986.1164959.

[5] M. Livesu, S. Ellero, J. Martínez, S. Lefebvre, and M. Attene, "From 3D models to 3D prints: An overview of the processing pipeline," *Computer Graphics Forum*, vol. 36, no. 2, pp. 537–564, May 2017. DOI: 10.1111/cgf.13147.

[6] S. Hornus, S. Lefebvre, J. Dumas, and F. Claux, "Tight printable enclosures and support structures for additive manufacturing," in *Eurographics Workshop on Graphics for Digital Fabrication*, The Eurographics Association, 2016, ISBN: 978-3-03868-003-1. DOI: 10.2312/gdf.20161074.

[7] P. Hachenberger, "Exact minkowksi sums of polyhedra and exact and efficient decomposition of polyhedra in convex pieces," in *Proceedings of the 15th Annual European Conference on Algorithms*, ser. ESA'07, Eilat, Israel: Springer-Verlag, 2007, pp. 669–680.

[8] W. Meng, S. Chen, Z. Shu, S.-Q. Xin, H. Fu, and C. Tu, "Efficiently computing feature-aligned and high-quality polygonal offset surfaces," *Computers & Graphics*, Jul. 2017. DOI: 10.1016/j.cag.2017.07.003.

[9] C. C. L. Wang and D. Manocha, "Gpu-based offset surface computation using point samples," *Computer-Aided Design*, vol. 45, no. 2, pp. 321–330, Feb. 2013. DOI: 10.1016/j.cad.2012.10.015.

[10] J. Martínez, S. Hornus, F. Claux, and S. Lefebvre, "Chained segment offsetting for ray-based solid representations," *Computers & Graphics*, vol. 46, pp. 36–47, Feb. 2015. DOI: 10.1016/j.cag.2014.09.017.

[11] T. Van Hook, "Real-time shaded nc milling display," in *Proceedings of the 13th annual conference on Computer graphics and interactive techniques - SIGGRAPH '86*, Association for Computing Machinery (ACM), 1986. DOI: 10.1145/15922.15887.

[12] S. Lefebvre, "Icesl: A gpu accelerated csg modeler and slicer," in *AEFA'13, 18th European Forum on Additive Manufacturing*, 2013. eprint: http://webloria.loria.fr/~slefebvr/icesl/icesl-whitepaper.pdf.

[13] P. Huang, C. C. L. Wang, and Y. Chen, "Algorithms for layered manufacturing in image space," in *Advances in Computers and Information in Engineering Research, Volume 1*, ASME Press, 2014. DOI: 10.1115/1.860328_ch15.

[14] Q. Zhou, E. Grinspun, D. Zorin, and A. Jacobson, "Mesh arrangements for solid geometry," *ACM Transactions on Graphics*, vol. 35, no. 4, pp. 1–15, Jul. 2016. DOI: 10.1145/2897824.2925901.

[15] C. Fan, J. Luo, J. Liu, and Y. Xu, "Half-plane voronoi diagram," in *2011 Eighth International Symposium on Voronoi Diagrams in Science and Engineering*, Institute of Electrical & Electronics Engineers (IEEE), Jun. 2011. DOI: 10.1109/isvd.2011.25.

[16] A. Meijster, J. B. T. M. Roerdink, and W. H. Hesselink, "A general algorithm for computing distance transforms in linear time," in *Mathematical Morphology and its Applications to Image and Signal Processing*, Kluwer Academic Publishers, 2002, pp. 331–340. DOI: 10.1007/0-306-47025-x_36.

[17] W. Jung, H. Shin, and B. K. Choi, "Self-intersection removal in triangular mesh offsetting," *Computer-Aided Design and Applications*, vol. 1, no. 1-4, pp. 477–484, Jan. 2004. DOI: 10.1080/16864360.2004.10738290.

[18] M. Campen and L. Kobbelt, "Exact and robust (self-)intersections for polygonal meshes," *Computer Graphics Forum*, vol. 29, no. 2, pp. 397–406, May 2010. DOI: 10.1111/j.1467-8659.2009.01609.x.

[19] P. Hachenberger, "3D minkowski sum of polyhedra," in *CGAL User and Reference Manual*, 4.13, CGAL Editorial Board, 2018. [Online]. Available: https://doc.cgal.org/4.13/Manual/packages.html#PkgMinkowskiSum3Summary.

[20] M. Campen and L. Kobbelt, "Polygonal boundary evaluation of minkowski sums and swept volumes," *Computer Graphics Forum*, vol. 29, no. 5, pp. 1613–1622, Sep. 2010. DOI: 10.1111/j.1467-8659.2010.01770.x.

[21] P. Musialski, T. Auzinger, M. Birsak, M. Wimmer, and L. Kobbelt, "Reduced-order shape optimization using

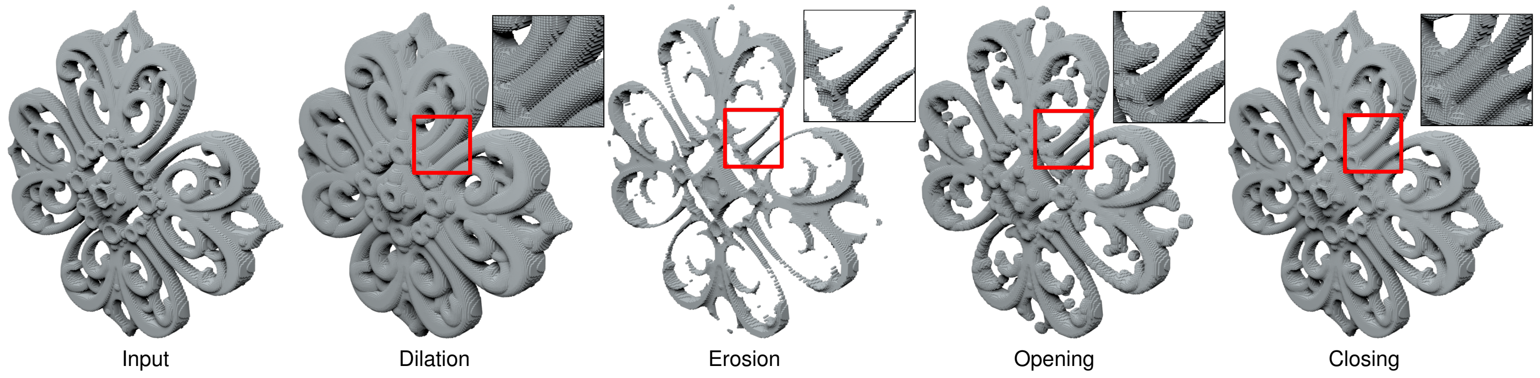

Fig. 19. Topological cleaning of an input shape via morphological operations, using a grid of 256×256 dexels. A hires zoomed-in view of the framed area is shown atop each result.

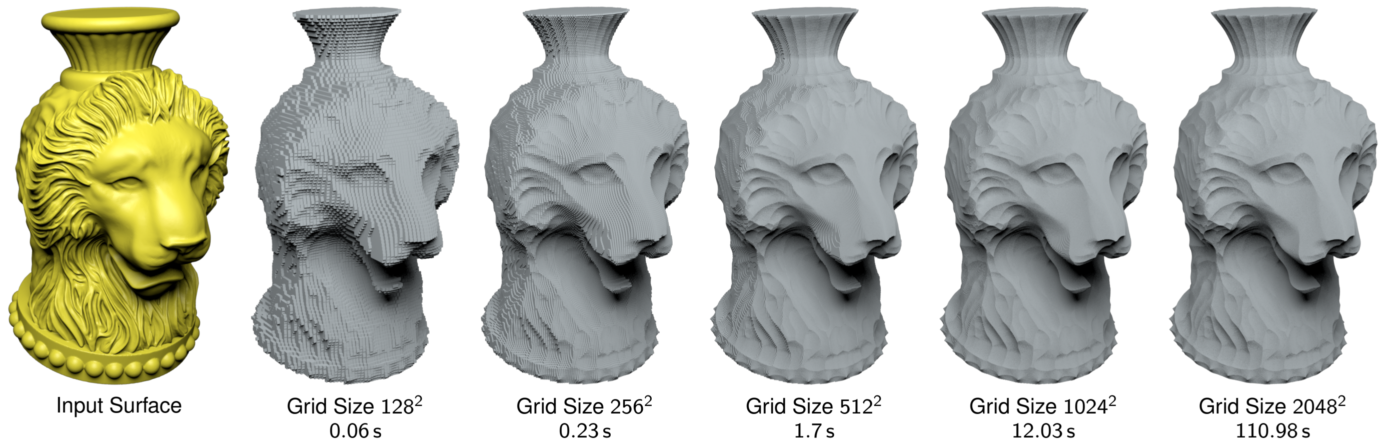

Fig. 20. Result of an erosion operation using different grid resolutions.

offset surfaces," *ACM Transactions on Graphics*, vol. 34, no. 4, 102:1–102:9, Jul. 2015. DOI: 10.1145/2766955.

[22] G. Varadhan and D. Manocha, "Accurate minkowski sum approximation of polyhedral models," *Graphical Models*, vol. 68, no. 4, pp. 343–355, Jul. 2006. DOI: 10.1016/j.gmod.2005.11.003.

[23] M. Peternell and T. Steiner, "Minkowski sum boundary surfaces of 3D-objects," *Graphical Models*, vol. 69, no. 3-4, pp. 180–190, May 2007. DOI: 10.1016/j.gmod.2007.01.001.

[24] D. Pavić and L. Kobbelt, "High-resolution volumetric computation of offset surfaces with feature preservation," *Computer Graphics Forum*, vol. 27, no. 2, pp. 165–174, Apr. 2008. DOI: 10.1111/j.1467-8659.2008.01113.x.

[25] S. Liu and C. C. L. Wang, "Fast intersection-free offset surface generation from freeform models with triangular meshes," *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 2, pp. 347–360, Apr. 2011. DOI: 10.1109/tase.2010.2066563.

[26] S. Calderon and T. Boubekeur, "Point morphology," *ACM Transactions on Graphics*, vol. 33, no. 4, pp. 1–13, Jul. 2014. DOI: 10.1145/2601097.2601130.

[27] Y. Liu, W. Wang, B. Lévy, F. Sun, D.-M. Yan, L. Lu, and C. Yang, "On centroidal voronoi tessellation—energy smoothness and fast computation," *ACM Transactions on Graphics*, vol. 28, no. 4, pp. 1–17, Aug. 2009. DOI: 10.1145/1559755.1559758.

[28] S.-Q. Xin, B. Lévy, Z. Chen, L. Chu, Y. Yu, C. Tu, and W. Wang, "Centroidal power diagrams with capacity constraints," *ACM Transactions on Graphics*, vol. 35, no. 6, pp. 1–12, Nov. 2016. DOI: 10.1145/2980179.2982428.

[29] S. Fortune, "A sweepline algorithm for voronoi diagrams," *Algorithmica*, vol. 2, no. 1-4, pp. 153–174, Nov. 1987. DOI: 10.1007/bf01840357.

[30] F. Aurenhammer, "Power diagrams: Properties, algorithms and applications," *SIAM Journal on Computing*, vol. 16, no. 1, pp. 78–96, Feb. 1987. DOI: 10.1137/0216006.

[31] L. Lu, B. Lévy, and W. Wang, "Centroidal voronoi tessellation of line segments and graphs," *Computer Graphics Forum*, vol. 31, no. 2pt4, pp. 775–784, May 2012. DOI: 10.1111/j.1467-8659.2012.03058.x.

[32] C. R. Maurer, R. Qi, and V. Raghavan, "A linear time algorithm for computing exact euclidean distance transforms of binary images in arbitrary dimensions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 2, pp. 265–270, Feb. 2003. DOI: 10.1109/tpami.2003.1177156.

[33] R. Fabbri, L. D. F. Costa, J. C. Torelli, and O. M. Bruno, "2D euclidean distance transform algorithms: A com-

parative survey," *ACM Computing Surveys*, vol. 40, no. 1, pp. 1–44, Feb. 2008. DOI: 10.1145/1322432.1322434.

[34] A. Tagliasacchi, T. Delame, M. Spagnuolo, N. Amenta, and A. Telea, "3d skeletons: A state-of-the-art report," *Computer Graphics Forum*, vol. 35, no. 2, pp. 573–597, May 2016. DOI: 10.1111/cgf.12865.

[35] L. Lam, S. .-.-W. Lee, and C. Y. Suen, "Thinning methodologies-a comprehensive survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 9, pp. 869–885, 1992. DOI: 10.1109/34.161346.

[36] L. Liu, E. W. Chambers, D. Letscher, and T. Ju, "A simple and robust thinning algorithm on cell complexes," *Computer Graphics Forum*, vol. 29, no. 7, pp. 2253–2260, Sep. 2010. DOI: 10.1111/j.1467-8659.2010.01814.x.

[37] N. Amenta, S. Choi, and R. K. Kolluri, "The power crust, unions of balls, and the medial axis transform," *Computational Geometry*, vol. 19, no. 2-3, pp. 127–153, Jul. 2001. DOI: 10.1016/s0925-7721(01)00017-7.

[38] D. Attali, J.-D. Boissonnat, and H. Edelsbrunner, "Stability and computation of medial axes - a state-of-the-art report," in *Mathematics and Visualization*, Springer Science + Business Media, 2009, pp. 109–125. DOI: 10.1007/b106657_6.

[39] Y. Yan, K. Sykes, E. Chambers, D. Letscher, and T. Ju, "Erosion thickness on medial axes of 3D shapes," *ACM Transactions on Graphics*, vol. 35, no. 4, pp. 1–12, Jul. 2016. DOI: 10.1145/2897824.2925938. [Online]. Available: https://doi.org/10.1145/2897824.2925938.

[40] Y. Yan, D. Letscher, and T. Ju, "Voxel cores," *ACM Transactions on Graphics*, vol. 37, no. 4, pp. 1–13, Jul. 2018. DOI: 10.1145/3197517.3201396. [Online]. Available: https://doi.org/10.1145/3197517.3201396.

[41] J. Shade, S. Gortler, L.-w. He, and R. Szeliski, "Layered depth images," in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques - SIGGRAPH '98*, Association for Computing Machinery (ACM), 1998. DOI: 10.1145/280814.280882.

[42] L. Carpenter, "The a-buffer, an antialiased hidden surface method," in *Proceedings of the 11th annual conference on Computer graphics and interactive techniques - SIGGRAPH '84*, Association for Computing Machinery (ACM), 1984. DOI: 10.1145/800031.808585.

[43] M. Maule, J. L. Comba, R. P. Torchelsen, and R. Bastos, "A survey of raster-based transparency techniques," *Computers & Graphics*, vol. 35, no. 6, pp. 1023–1034, Dec. 2011. DOI: 10.1016/j.cag.2011.07.006.

[44] T. Saito and T. Takahashi, "Nc machining with g-buffer method," *ACM SIGGRAPH Computer Graphics*, vol. 25, no. 4, pp. 207–216, Jul. 1991. DOI: 10.1145/127719.122741.

[45] C. C. L. Wang, Y.-S. Leung, and Y. Chen, "Solid modeling of polyhedral objects by layered depth-normal images on the gpu," *Computer-Aided Design*, vol. 42, no. 6, pp. 535–544, Jun. 2010. DOI: 10.1016/j.cad.2010.02.001.

[46] E. E. Hartquist, J. P. Menon, K. Suresh, H. B. Voelcker, and J. Zagajac, "A computing strategy for applications involving offsets, sweeps, and minkowski operations," *Computer-Aided Design*, vol. 31, no. 3, pp. 175–183, Mar. 1999. DOI: 10.1016/s0010-4485(99)00014-7.

[47] K. C. Hui, "Solid sweeping in image space—application in nc simulation," *The Visual Computer*, vol. 10, no. 6, pp. 306–316, Jun. 1994. DOI: 10.1007/bf01900825.

[48] Y. Chen and C. C. L. Wang, "Uniform offsetting of polygonal model based on layered depth-normal images," *Computer-Aided Design*, vol. 43, no. 1, pp. 31–46, Jan. 2011. DOI: 10.1016/j.cad.2010.09.002.

[49] M. van Kreveld and W. van Toll, *Lecture notes in geometric algorithms*, slides 7b, Apr. 2017. [Online]. Available: http://www.cs.uu.nl/docs/vakken/ga/.

[50] M. Held, "VRONI: An engineering approach to the reliable and efficient computation of voronoi diagrams of points and line segments," *Computational Geometry*, vol. 18, no. 2, pp. 95–123, Mar. 2001. DOI: 10.1016/s0925-7721(01)00003-7.

[51] M. Held and S. Huber, "Topology-oriented incremental computation of voronoi diagrams of circular arcs and straight-line segments," *Computer-Aided Design*, vol. 41, no. 5, pp. 327–338, May 2009. DOI: 10.1016/j.cad.2008.08.004.

[52] Boost, *The Boost.Polygon Library*, https://www.boost.org/doc/libs/1_69_0/libs/polygon/doc/index.htm, 2010.

[53] A. Wallin, *OpenVoronoi*, https://github.com/aewallin/openvoronoi, 2018.

[54] M. Karavelas, "2d segment delaunay graphs," in *CGAL User and Reference Manual*, 4.13, CGAL Editorial Board, 2018. [Online]. Available: https://doc.cgal.org/4.13/Manual/packages.html#PkgSegmentDelaunayGraph2Summary.

[55] F. Aurenhammer, B. Jüttler, and G. Paulini, "Voronoi diagrams for parallel halflines and line segments in space," en, 2017. DOI: 10.4230/lipics.isaac.2017.7.

[56] A. Jacobson, L. Kavan, and O. Sorkine-Hornung, "Robust inside-outside segmentation using generalized winding numbers," *ACM Transactions on Graphics*, vol. 32, no. 4, p. 1, Jul. 2013. DOI: 10.1145/2461912.2461916.

[57] G. Barill, N. G. Dickson, R. Schmidt, D. I. W. Levin, and A. Jacobson, "Fast winding numbers for soups and clouds," *ACM Transactions on Graphics*, vol. 37, no. 4, pp. 1–12, Jul. 2018. DOI: 10.1145/3197517.3201337.

[58] G. Guennebaud, B. Jacob, *et al.*, *Eigen v3*, http://eigen.tuxfamily.org, 2010.

[59] J. Reinders, *Intel Threading Building Blocks - Outfitting C++ for Multi-core Processor Parallelism*. O'Reilly Media, Mar. 2010, pp. I–XXV, 1–303, ISBN: 978-0-596-51480-8.

[60] D. Boltcheva and B. Lévy, "Surface reconstruction by computing restricted voronoi cells in parallel," *Computer-Aided Design*, vol. 90, pp. 123–134, Sep. 2017. DOI: 10.1016/j.cad.2017.05.011.

[61] L. Malomo, N. Pietroni, B. Bickel, and P. Cignoni, "Flexmolds: Automatic design of flexible shells for molding," *ACM Transactions on Graphics*, vol. 35, no. 6, pp. 1–12, Nov. 2016. DOI: 10.1145/2980179.2982397.

Zhen Chen is a first year PhD student at the Department of Computer Science in the University of Texas at Austin. He earned his Bachelor degree in Computational Mathematics at the University of Science and Technology of China in 2018. From June to August 2017, he has been a visiting student at the Courant Institute of Mathematical Sciences (New York University, USA). His research interests are 3D printing, geometry processing and shell deformation.

Daniele Panozzo Daniele Panozzo is an assistant professor of computer science at the Courant Institute of Mathematical Sciences in New York University. Prior to joining NYU he was a postdoctoral researcher at ETH Zurich (2012–2015). He earned his PhD in Computer Science from the University of Genova (2012) and his doctoral thesis received the EUROGRAPHICS Award for Best PhD Thesis (2013). He received the EUROGRAPHICS Young Researcher Award in 2015 and the NSF CAREER Award in 2017. Daniele is leading the development of libigl (https://github.com/libigl/libigl), an award-winning (EUROGRAPHICS Symposium of Geometry Processing Software Award, 2015) open-source geometry processing library that supports academic and industrial research and practice. Daniele is chairing the Graphics Replicability Stamp (http://www.replicabilitystamp.org), which is an initiative to promote reproducibility of research results and to allow scientists and practitioners to immediately benefit from state-of-the-art research results. Daniele's research interests are in digital fabrication, geometry processing, architectural geometry, and discrete differential geometry.

Jérémie Dumas Jérémie Dumas is a research engineer at nTopology in New York. Prior to joining nTopology Inc. he was a postdoctoral fellow at the Courant Institute of Mathematical Sciences in New York University. Jérémie completed his PhD at INRIA Nancy Grand-Est in 2017, under the direction of Sylvain Lefebvre. His doctoral thesis received the EUROGRAPHICS Award for Best PhD Thesis (2018). His work focuses on shape synthesis for digital fabrication, shape optimization, simulation, microstructures, and procedural synthesis.

Appendix A
Voronoi Vertex between Three Parallel Segments in 2D

The bisector of two parallel segment seeds is 2D is a piecewise-quadratic curve, as illustrated in Figure 9. A Voronoi vertex is a point at the intersection of the bisector curves between three segment seeds. Because the segment seeds are non-overlapping, the Voronoi vertex can be either between three points (Section A.1) or between one segment and two points (Section A.2).

A.1 Voronoi Vertex between Three Points

Let $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3$ be three points, with coordinates $\boldsymbol{p}_i = (y_i, z_i)$. A point $\boldsymbol{p}$ lying on the bisector of $(\boldsymbol{p}_1, \boldsymbol{p}_2)$ satisfies

$$\begin{aligned} \|\boldsymbol{p}-\boldsymbol{p}_1\|^2 &= \|\boldsymbol{p}-\boldsymbol{p}_2\|^2 \\ \iff (y-y_1)^2 + (z-z_1)^2 &= (y-y_2)^2 + (z-z_2)^2 \end{aligned} \tag{7}$$

After simplification, we get

$$2(y_2-y_1)y + 2(z_2-z_1)z = (y_2^2+z_2^2) - (y_1^2+z_1^2) \tag{8}$$

Similarly, for a point lying on the bisector of $(\boldsymbol{p}_2, \boldsymbol{p}_3)$,

$$2(y_3-y_2)y + 2(z_3-z_2)z = (y_3^2+z_3^2) - (y_2^2+z_2^2) \tag{9}$$

We can get the coordinate of Voronoi vertex between three points by solving the system formed by Equations (8) and (9):

$$2\begin{pmatrix} y_2-y_1 & z_2-z_1 \\ y_3-y_2 & z_3-z_2 \end{pmatrix}\begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} (y_2^2+z_2^2)-(y_1^2+z_1^2) \\ (y_3^2+z_3^2)-(y_2^2+z_2^2) \end{pmatrix} \tag{10}$$

A.2 Voronoi Vertex between a Segment and Two Points

Let $\boldsymbol{p}_1(y_1, z_1), \mathfrak{s}(y_s, z_s^{\vdash}, z_s^{\dashv}), \boldsymbol{p}_2(y_2, z_2)$ be three seeds. We only need to consider the case where $y_s < y_1 < y_2$, other cases are similar. For the Voronoi vertex $\boldsymbol{p} = (y, z)$ to intersect the bisectors where it is closest to the interior of $\mathfrak{s}$, and not one its endpoints, we need to have $z_s^{\vdash} < z < z_s^{\dashv}$. It follows the distance from $\boldsymbol{p}$ to the segment $\mathfrak{s}$ is:

$$\begin{aligned} \operatorname{dist}(\boldsymbol{p}, \mathfrak{s}) &= \inf_{(a,b)\in\mathfrak{s}} (y-a)^2 + (z-b)^2 \\ &= (y-y_s)^2 \quad \text{when } (a,b) = (y_s, z) \end{aligned} \tag{11}$$

By definition of the Voronoi vertex,

$$\operatorname{dist}(\boldsymbol{p}, \mathfrak{s}) = \operatorname{dist}(\boldsymbol{p}, \boldsymbol{p}_1) \tag{12}$$

$$\operatorname{dist}(\boldsymbol{p}, \boldsymbol{p}_1) = \operatorname{dist}(\boldsymbol{p}, \boldsymbol{p}_2) \tag{13}$$

Developing Equation (12), we get

$$\begin{aligned} (y-y_s)^2 &= (y-y_1)^2 + (z-z_1)^2 \\ \iff 2(y_1-y_s)y - (z-z_1)^2 &= z_1^2 - y_s^2 \end{aligned} \tag{14}$$

Similarly, developing Equation (13) leads to

$$2(y_1-y_2)y + 2(z_1-z_2)z = (y_1^2+z_1^2) - (y_2^2+z_2^2) \tag{15}$$

Now, let

$$\begin{cases} u = 2(y_1 - y_s) \\ w = -y_1^2 + y_s^2 \\ a = 2(y_1 - y_2) \\ b = 2(z_1 - z_2) \\ c = (y_1^2 + z_1^2) - (y_2^2 + z_2^2) \end{cases}$$

We can rewrite the system of equations as

$$\begin{cases} uy - (z - z_1)^2 + w = 0 \\ ay + bz = c \end{cases} \tag{16}$$

$$\implies az^2 + (bu - 2az_1)y + az_1 - cu - aw = 0 \tag{17}$$

Solving Equation (17), if the roots exist, we will have

$$\Delta = (bu - 2a_1z)^2 - 4a(az_1^2 - cu - aw)$$
$$z_{1,2}^* = \frac{2az_1 - bu \pm \sqrt{\Delta}}{2a}$$

Choosing the solution that belongs to $[z_s^{\vdash}, z_s^{\dashv}]$, and substituting into $ay + bz = c$, we will get the y-coordinate of our Voronoi vertex.

Appendix B
Power Vertex between Three Points in 2D

Let $\boldsymbol{p}_1(y_1, z_1; r_1), \boldsymbol{p}_2(y_2, z_2; r_2), \boldsymbol{p}_3(y_3, z_3; r_3)$ be three seeds. A power vertex can be computed from the intersection of two bisector lines.

A point $\boldsymbol{p}(y, z)$ lying on the bisector of $(\boldsymbol{p}_1, \boldsymbol{p}_2)$ satisfies:

$$\|\boldsymbol{p} - \boldsymbol{p}_1\|^2 - r_1^2 = \|\boldsymbol{p} - \boldsymbol{p}_2\|^2 - r_2^2$$
$$\iff (y_1 - y)^2 + (z - z_1)^2 - r_1^2 = (y_2 - y)^2 + (z - z_2)^2 - r_2^2$$
$$\iff 2(y_2 - y_1)y + 2(z_2 - z_1)z = (y_2^2 + z_2^2 - r_2^2) - (y_1^2 + z_1^2 - r_1^2) \tag{18}$$

A similar equation holds for the bisector of $(\boldsymbol{p}_2, \boldsymbol{p}_3)$. This translates into the following system of equations:

$$2\begin{pmatrix} y_2 - y_1 & z_2 - z_1 \\ y_3 - y_2 & z_3 - z_2 \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} (y_2^2 + z_2^2 - r_2^2) - (y_1^2 + z_1^2 - r_1^2) \\ (y_3^2 + z_3^2 - r_3^2) - (y_2^2 + z_2^2 - r_2^2) \end{pmatrix} \tag{19}$$