Raster2Mesh: Rasterization based CVT meshing

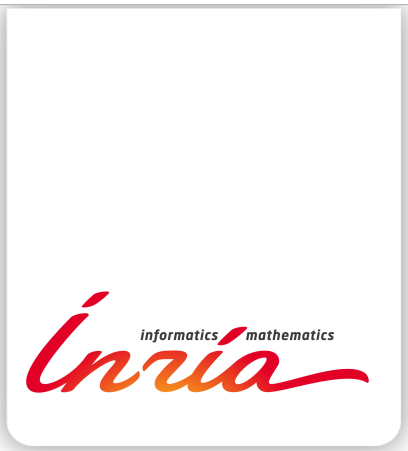
Jonas Martinez, Frédéric Claux, Sylvain Lefebvre

Raster2Mesh: Rasterization based CVT meshing

Jonàs Martínez, Frédéric Claux, Sylvain Lefebvre

RESEARCH REPORT
N° 8684
February 2015
Project-Teams Alice

ISSN 0249-6399 ISRN INRIA/RR--8684--FR+ENG



Raster2Mesh: Rasterization based CVT meshing

Jonàs Martínez, Frédéric Claux, Sylvain Lefebvre

Project-Teams Alice

Research Report n° 8684 — February 2015 — 27 pages

Abstract: In this paper, we propose to extend high quality Centroidal Voronoi Tessellation (CVT) remeshing techniques to the case of surfaces which are *not* defined by triangle meshes, such as implicit surfaces. Our key observation is that rasterization routines are usually available to visualize these alternative representations, most often as OpenGL shaders efficiently producing surface samples (*fragments*) from the surface representation.
Our technique has the ability to mesh any surface for which rasterization routines are available, and runs entirely within the OpenGL rasterization pipeline. There is no intermediate representation: the triangle mesh is computed directly from the surface fragments. Our method produces high quality meshes, as it inherits the properties of CVT meshing. Contrary to existing GPU techniques for CVT computation, it does not require a surface parameterization, and it extracts the mesh topology directly from the surface fragments. Optionally, our algorithm can produce two-manifold, consistently oriented meshes.
We describe our complete implementation and show a variety of applications: direct meshing of implicit surfaces, meshing of operations between solids, mesh repair, and solid sculpting. We analyze performance, correctness and mesh quality.

Key-words: meshing, remeshing, rasterization, Centroidal Voronoi Tessellation

RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Raster2Mesh: génération de maillage de type CVT basée sur la rastérisation

Résumé : Nous proposons d'étendre les méthodes de remaillage basées sur la Tessellation Centroïdale de Voronoï (CVT) à des fins de maillage de surfaces qui ne sont *pas* définies avec des maillages triangulaires, comme par exemple les surfaces implicites. Notre principale observation porte sur la large disponibilité des routines de rastérisation, le plus souvent sous la forme de shaders OpenGL, utilisées pour visualiser ces types de surfaces alternatives. Ces mêmes routines peuvent être efficacement utilisées pour échantillonner des points ou *fragments* sur les surfaces.

Notre méthode peut être utilisée pour mailler n'importe quelle surface pour laquelle des routines de rastérisation sont disponibles et tourne entièrement à l'intérieur du pipeline de rastérisation OpenGL. Aucune représentation intermédiaire n'est nécessaire: nous générons un maillage triangulaire directement à partir des fragments sur la surface. Notre méthode produit des maillages de haute qualité, puisqu'elle hérite des propriétés de la CVT. Contrairement aux méthodes actuelles traitant du calcul de la CVT sur GPU, notre méthode ne requiert pas de paramétrisation de la surface et déduit la topologie du maillage directement à partir des fragments échantillonnés. Notre algorithme peut générer en option un maillage deux-manifold et dont les normales sont orientées de manière homogène.

Nous détaillons notre implémentation et décrivons l'utilisation de notre méthode à diverses fins applicatives : maillage de surfaces implicites, du résultat d'opérations géométriques entre solides, réparation de maillage, sculpture sur solide. Nous analysons les performances et la qualité de notre algorithme.

Mots-clés : maillage, remaillage, rastérisation, Tessellation Centroïdale de Voronoï

1 Introduction

Triangle meshes are an essential, versatile representation of surfaces and solid boundaries. They are supported by the vast majority of modeling, simulation, rendering and manufacturing software. Meshes are also readily renderable by graphics hardware which have dedicated units to process and render large amounts of triangles.

Not all triangle meshes are of equal quality, and a significant amount of research is devoted to the production of meshes with desirable properties, such as being two-manifold, being evenly sampled over the surface they represent and having isotropic triangles. These properties are crucial to allow for a correct definition of enclosed volumes, to ensure a correct sampling of information along surfaces (colors, normals, shading, vector fields), and to obtain well behaved simulations by the Finite Element Method [1].

Remeshing – the ability to obtain a mesh having the aforementioned properties from a mesh having poor quality – has therefore been a long standing problem in the community, for which advanced solutions are now available. However, there are many situations where the surface representation is *not* a mesh. For instance, surfaces can be represented implicitly by the equation of the their locus, by a distance field equation or by an indicator function (0: outside, 1: inside). Remeshing techniques cannot be applied directly in such cases, and an intermediate method is required. For instance, algorithms such as the marching cubes [2] can create a mesh quickly and efficiently from an implicit surface, unfortunately at the expense of quality. Meshing algorithms for solids expressed in ray-representation [3] conveniently used in the area of CSG modeling and 3D printing [4] suffer from similar issues. Of course, the meshes produced can then be remeshed by more advanced techniques, but this cascade of conversions and resampling is wasteful and generally detrimental to the final result quality. Some more general algorithms are available, such as those based on the Delaunay Refinement algorithm [5], but they are difficult to parallelize and generally exhibit low performance, even though they generate meshes of a higher quality.

The key observation we exploit in this paper is that most surface representations are *rasterizable* ; that is, there exists an efficient algorithm to produce surface samples (*fragments*) that are displayed on screen for visualization. Indeed, the vast majority of 3D software applications visualize 3D objects by explicitly providing rasterization routines, most often in the form of GPU pixel shaders. In this paper we propose a highly parallelizable surface meshing algorithm that is *independent* from the surface representation and is applicable as long as rasterization routines are available. Our technique directly produces meshes having a good surface sampling and triangles of high isotropy.

Contributions. Our meshing approach, based on discrete Centroidal Voronoi Tessellation (CVT) [6] techniques, leverages existing, out of the box GPU rasterization routines to perform surface sampling efficiently. Contrary to existing parallel (GPU) approaches, the mesh topology is directly inferred from the rasterization of the surface, and no surface parameterization is required. We calculate vertex connectivity and generate a Restricted Delaunay Triangulation (RDT) using a highly parallel algorithm that deals with most cospherical situations appropriately. The generated isotropic mesh can optionally be made two-manifold and consistently oriented when dealing with surfaces defining solid boundaries. We take a signal processing view on this problem and perform a *local regularization* that filters problematic features at non-manifold locations.

The paper is organized as follows. Section 2 reviews existing approaches for meshing surfaces. Section 3 gives some background on CVT meshing. Section 4 details our algorithm and its implementation. We demonstrate the practical usefulness of our approach in a variety of applications and evaluate the quality of our work in Section 5. We finish with a conclusion in Section 6.

2 Related work

There is a vast literature on surface meshing methods and applications. We focus here on the most common methods, eg. the ones that have received a lot of attention from the community over the past decades. For a detailed review on a wider range of methods, we refer the reader to the book of Boissonnat and Teillaud [7], chapter 5.

Most of the approaches for surface meshing subdivide the space into cells. The methods differ as to what is actually being performed inside each cell, with the goal always being to produce a polygonization for each cell. A well-known method that considers cubical cells is the marching cubes [2]. This family of algorithms is popular as it is well behaved for parallel implementations on the GPU, where processing for each cell can operate independently. The cell size is typically fixed beforehand though, and as a result can only capture details of the input model matching that size. Some algorithms attempt to subdivide the cells adaptively [8], but this recursive nature makes them much more difficult to implement on GPUs. Although it is usually two-manifold, the resulting mesh is generally of poor quality, with many triangles of degenerate shape and size. For this reason, Chen et al. [9] mesh implicit surfaces and then optimize the resulting mesh in a subsequent step, but with only minor regards to the original surface.

Some algorithms work by evaluating point positions at the corners of the cells to infer a polygonization, based on a change of sign. These algorithms are well suited to the meshing of implicit surfaces. Others work by intersecting the input model with the cells. Ray representations are meshed in such a way [2]. Most existing work in this category focuses on the case where three orthogonal ray representations are used, and proceed by labeling which grid cells are intersected by solid ray intervals [10, 11]. Feng and Warren [12] present an approach for a single ray representation that uses a method inspired by dual contouring, an extension of the marching cubes [13]. Although better, the mesh quality suffers from the same issues as the marching cubes method, at an additional computational complexity.

Delaunay refinement algorithms [5] are another class of algorithms. Algorithms in this class start with some initial point sample, and iteratively add surface points to a restricted Delaunay triangulation. They can produce a mesh of relatively good quality but are iterative by nature, and therefore are inefficient to implement on GPUs. Also, they demand at least one seed point per connected component of the original model, a number which may not be known beforehand.

Centroidal Voronoi tessellation on surfaces

Centroidal Voronoi tessellation (CVT) is a large class of algorithms that works by minimizing an energy over a model domain. It operates on models of various types and works in both continuous and discrete contexts. The algorithm works by first computing a Voronoi diagram and then by extracting a mesh from connected regions in the diagram.

The CVT is a special kind of Voronoi diagram, where each seed coincides with the centroid of its Voronoi region [6]. Seeds are typically sampled from the model surface, and an algorithm is then used to move them to their final position. Either the well-known Lloyd method [14] or the method proposed by Liu et al. [15] can be used for this purpose. This latter method is presented as a quasi-Newton method and is called L-BFGS. It converges much faster than Lloyd's method.

CVT can either be used to produce a tetrahedralization of a volume or, in our context, a mesh of a surface or solid boundary. In this latter case, the CVT formulation is said to be *constrained to surfaces*. CVT on surfaces constitutes an effective approach to isotropic meshing. Kunze et al. [16] compute the geodesic Voronoi diagram (GVD) on parameterized surfaces, as the geodesic metric naturally arises as a good option for calculating Voronoi regions, producing cells "along" the surface. Wang et al. [17] intrinsically compute the GVD on triangle meshes without

parameterization. However, the geodesic distance has a very high computation time.

The geodesic distance can be replaced by the Euclidean one to ease the computations. Edelsbrunner and Shah [18] introduced the restricted Voronoi diagram (RVD) on surfaces as the intersection between the Euclidean Voronoi diagram and the surface. Du et al. [19] introduced the constrained CVT (CCVT), where the RVD seeds are constrained to lie on the surface, which tends to improve the quality of the final meshing. Yan et al. [20] presented the localized RVD, which enforces that Voronoi regions form a single connected region and improves remeshing with a low number of vertices. The CVT can also be defined in hyperbolic space [21].

There exist several algorithms to compute the CVT on meshes. Alliez et al. [22] consider a dense set of surface points to compute a discrete RVD. Valette et al. [23] approximate the RVD of triangular meshes by considering that Voronoi regions are composed of clusters of mesh triangles. Yan et al. [24] proposed an exact approach to compute the RVD, by determining the intersection of each triangle mesh with its incident Voronoi regions.

The CVT on surfaces can also be computed by first calculating a surface parametrization. In the work of Alliez et al. [25] and Rong et al. [26], an input mesh is first parameterized onto a 2D space, and the 2D CVT is lifted back to the 3D space. However, finding a parameterization of mesh surfaces is non-trivial and can lead to inaccuracies due to parameterization distortions. Finding a parametrization for arbitrary surfaces is a very difficult problem [27].

To the best of our knowledge, there exist only two approaches to compute the CVT of surfaces on the GPU. Rong et al. [26] first compute a parametrization of the surface onto a 2D geometry image. Due to distortion, each geometry image pixel is assigned a correcting weight. The 2D Voronoi diagram is computed with a GPU jump flooding algorithm. The CCVT is minimized with the Lloyd method (GPU) or with the L-BFGS method (GPU+CPU). Shuai et al. [28] compute an embedding in 2D hyperbolic space. Then, a discrete clustering of triangles in Voronoi regions is performed in the GPU. The CVT energy is minimized with the Lloyd method. Both approaches require a surface parameterization, and do not consider the extraction of the dual CVT triangulation on the GPU.

In contrast, our approach does not require a 2D parametrization of the surface. Instead, we directly compute the restricted CVT from dynamically sampled points over the surface, which we do through the OpenGL rasterization pipeline. Rasterizated fragment positions are accumulated at seed level in the fragment shader (Section 4.2) and feed the Lloyd (GPU) or L-BFGS (CPU) algorithm in a subsequent step. Mesh extraction also takes place on the GPU (Section 4.3), and can optionally be two-manifold. We provide an end-to-end algorithm for arbitrary mesh extraction of solids that works with any representation for which rasterization routines are available.

3 Background

Centroidal Voronoi tessellation

Let $\mathcal{S} = (s_i)_{i=1}^{k}$ be an ordered set of k points in $\mathbb{R}^n$, referred from now as seeds. The Voronoi region Ω_i of s_i is:

$$\Omega_i = \{x \in \mathbb{R}^n : \|x - s_i\| \leq \|x - s_j\|, \forall i \neq j\}$$

Where $\|\cdot\|$ is the Euclidean norm in $\mathbb{R}^n$. The Voronoi regions of all the seeds in S form the Voronoi diagram. For a CVT, each seed s_i coincides with the mass center c_i of its Voronoi region Ω_i, defined as:

$$c_i = \frac{\int_{\Omega_i} \rho(x)\, x d\sigma}{\int_{\Omega_i} \rho(x)\, d\sigma}$$

Where $\rho(x) > 0$ is a user-defined density function and $d\sigma$ is the area differential.

A CVT can equivalently be defined from a variational point of view [6]. The CVT energy function is:

$$F(\mathcal{S}) = \sum_{i=1}^{k} \int_{\Omega_i} \rho(x) \|x - s_i\|^2 d\sigma$$

The CVT corresponds to a critical point of $F(\mathcal{S})$. The gradient of $F(\mathcal{S})$ is:

$$\frac{\partial F}{\partial s_i} = 2(s_i - c_i) \int_{\Omega_i} \rho(x)\, d\sigma$$

Constrained and restricted CVT

Let $\mathcal{O} \subset \mathbb{R}^3$ be a compact surface. The restricted Voronoi regions are $\Omega_i \cap \mathcal{O}$, and form the restricted Voronoi diagram. If we do not constrain $\mathcal{S} \subset \mathcal{O}$ we obtain the so called Restricted CVT (RCVT), which is more suitable for remeshing noisy surfaces [24]. The Lloyd and L-BFGS methods can also be effectively used to compute both the CCVT and RCVT [24].

Restricted Delaunay triangulation

The Restricted Delaunay Triangulation (RDT) is the dual of the RVD. The RDT is a simplicial complex where a vertex is associated to a Voronoi region of RVD, an edge is associated to two Voronoi regions sharing an edge, and a triangle is associated to three Voronoi regions sharing a vertex [18]. Thus, the Delaunay edges of RDT are recovered from pairs of seeds that share a Voronoi edge. The points included in a Voronoi edge are equidistant to both seeds.

4 Discrete RCVT meshing

We present an approach to compute an approximate RCVT of $\mathcal{O}$ by considering a finite, uniformly sampled, and dense set of points $\mathcal{P} \subset \mathcal{O}$. We consider isotropic meshing ($\rho(x) = 1$), and adopt an interleaved discrete optimization and local regularization approach. Figure 1 presents an overview of the method, and a summary of the different shader programs associated to each step.

Our approach is fully integrated in the OpenGL rasterization pipeline, and only considers surface points to construct the surface mesh. We take advantage of atomic memory operations readily available in modern GPUs [29].

First, we distribute the seeds $\mathcal{S}$ over the surface $\mathcal{O}$ (see Section 4.1). Then, we minimize the CVT energy function according to the input surface points $\mathcal{P}$ and the seeds $\mathcal{S}$ (see Section 4.2). Next, we extract a triangular mesh by recovering the Delaunay edges between Voronoi regions (see Section 4.3). In case the mesh it is not two-manifold, we perform a local regularization of $\mathcal{P}$ to ensure convergence (see Section 4.4). The interleaved process finishes when the mesh is two-manifold.

When $\mathcal{O}$ bounds a solid, we use dexel structures [30]. A dexel structure is a compact solid ray representation that allows efficient and robust boolean operations. For a single direction and a uniform grid of rays parallel to that direction, a dexel structure stores the intervals of the rays lying inside a solid; these intervals are called dexels. Dexel structures are required for the

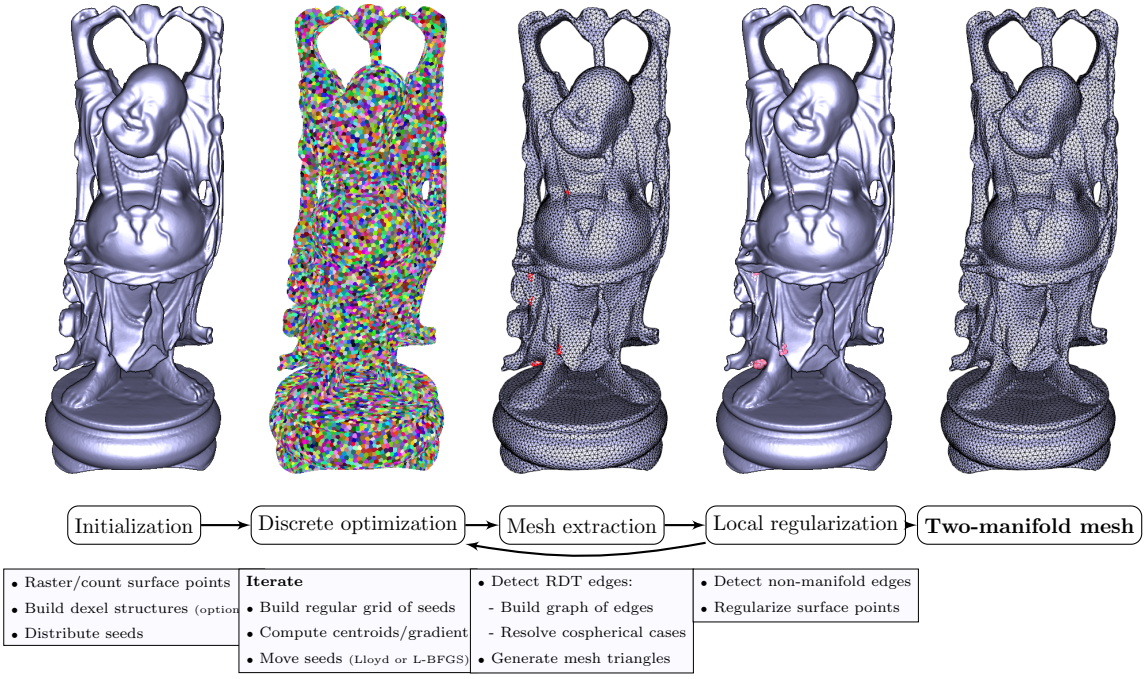


Figure 1: Overview of our approach.

local regularization step. If $\mathcal{O}$ does not bound a solid, we skip the local regularization, and just consider the rasterized surface points.

In each of the following subsections we first introduce the principle behind each step, and discuss its implementation in the graphics pipeline.

4.1 Initialization

Initially, seeds are randomly distributed over the surface. Due to the discrete sampling of $\mathcal{O}$, it is important to maintain a minimum average ratio between the number of seeds and surface points. Given a user-defined sampling density of $\mathcal{P}$, we perform an initial sampling and calculate the number of points $|\mathcal{P}|$, then choose any number of seeds $|\mathcal{S}| \geq \frac{|\mathcal{P}|}{\alpha}$, where α is the desired average number of points per seed. α should be high enough for the CVT computation to be accurate and reliable. The sampling of $\mathcal{P}$ should be an ϵ-covering (see Section 4.3).

Definition 1. *(ϵ-covering) The set of points $\mathcal{P}$ is an ϵ-covering if for any point $x \in \mathcal{P}$ there exists a point $y \in \mathcal{P}$ such that $\|x - y\| < \epsilon$.*

We randomly distribute the seeds over the surface points, with a uniform probability distribution. That is, we approximate a random distribution of seeds according to the surface area of $\mathcal{O}$.

Implementation

We initially rasterize surface points for each three axis-aligned directions. The user-defined sampling resolution $d > 0$ is set uniform for all three directions. Sampling these three directions

is a common technique to obtain a good sample of surface points [10], and forms an ϵ-covering, where $\epsilon = \sqrt{3}d$. If $\mathcal{O}$ bounds a solid, we construct three dexel structures from rasterized points with a fast GPU algorithm [31].

We also count the number of rasterized points, with atomic counters, and constraint the maximum number of sampled seeds $\mathcal{S}$ according to the α ratio. In all our experiments, we set a ratio of $\alpha > 50$ to ensure a good quality of results.

4.2 Discrete optimization

We introduce some RCVT related definitions (see Section 3) in our discrete setting, with $\rho(x) = 1$. The discrete Voronoi region $\overline{\Omega_i}$ of $s_i \in \mathcal{S}$ is defined as the set of surface points (see Figure 2):

$$\overline{\Omega_i} = \{x \in \mathcal{P} : \|x - s_i\| \leq \|x - s_j\|, \forall i \neq j\}$$

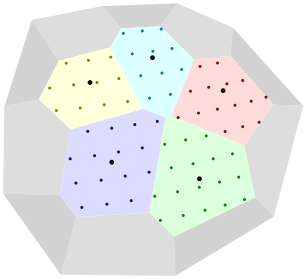


Figure 2: Discrete Voronoi regions of five seeds, composed of surface points with the same color. For illustration purposes, the continuous Voronoi regions are also shaded with the corresponding color.

Analogously, the discrete centroid $\overline{c_i}$ of s_i is:

$$\overline{c_i} = \frac{\sum_{j=1}^{|\overline{\Omega_i}|} x_j}{|\overline{\Omega_i}|} \tag{1}$$

We assume that surface points cover a similar surface area. Thus, we approximate the centroid position.

The centroid computation is similar to the CVT point-based method of Alliez et al. [22], where the input mesh is approximated by a dense set of samples. These samples are surface points together with additional information, like the surface area or the feature edge length. Surface meshing in [22] requires the input mesh, whereas our method only considers the sampled surface points.

With the Lloyd method, the seeds are iteratively moved to their discrete centroids. In order to use the L-BFGS method, we approximate the value of the energy function and its gradient. The discrete CVT energy function is:

$$\overline{F}(\mathcal{S}) = \sum_{i=1}^{k} \sum_{j=1}^{|\overline{\Omega_i}|} \|x_j - s_i\|^2$$

Finally, the discrete gradient of the seed s_i is:

$$2\left(s_i \sum_{j=1}^{|\overline{\Omega_i}|} x_j - \left|\overline{\Omega_i}\right|\right) \tag{2}$$

Importantly, we never explicitly construct the RVD. We just optimize the position of seeds to form an RCVT. The optimization process ends when the discrete gradient norm or the energy function reaches a lower threshold, or alternatively when a maximum number of iterations is exceeded.

Implementation

The CVT optimization using the Lloyd method runs entirely on the rasterization pipeline. The L-BFGS method runs partially on the CPU. For both methods we iteratively: 1) compute the discrete centroid and gradient (only for L-BFGS) of each seed, 2) perform a minimization (move the seeds) of the CVT energy function. With the L-BFGS method we use a publicly available CPU solver [32] to perform step 2 and feed back the new seed positions to the GPU.

In order to compute the discrete centroid and gradient we accumulate (atomic addition) for each seed s_i, both terms $\sum_{j=1}^{|\overline{\Omega_i}|} x_j$ and $\left|\overline{\Omega_i}\right|$ of Equations 1 and 2. To do so, we just need to compute for each surface point x its nearest seed.

To accelerate the closest seed computation we construct a regular three dimensional grid storing the seeds (see Figure 3). In a first pass, we count the number of seeds lying in each grid cell. The obtained set of counters is then converted to prefix sums. These sums indicate where to start storing the seeds of each grid cell. An implementation of this technique is described in [33]. The nearest seed is found by performing progressive lookups on rings of cells around x until the closest one is guaranteed to be found (see Figure 3). For better performance we set the grid cell size to be the average distance between samples, assuming a uniform distribution.

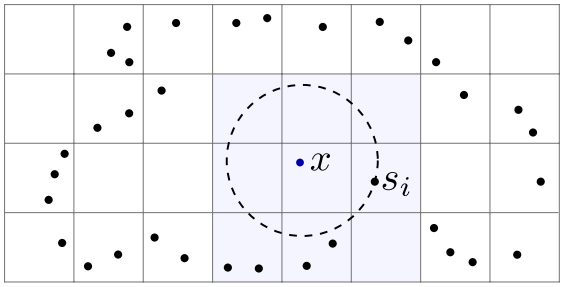


Figure 3: A two-dimensional example of a regular grid storing seeds. From a surface point x, we search around shaded cells until the closest seed s_i is found.

4.3 Mesh extraction

The mesh extraction solely considers the surface points $\mathcal{P}$. First, we recover the RDT edges by looking at the points of $\mathcal{P}$ that approximately lie on the boundary between discrete Voronoi regions. In addition, we introduce a technique to recover RDT edges between seeds in cospherical position. Finally, we extract the mesh triangles from the RDT edges.

In order to characterize the surface points between discrete Voronoi regions, the ϵ-covering value (see Section 4.1) can be used to decide whether two or more seeds are at approximate equal distance.

For $x \in \mathcal{P}$, let $x_{near} = (s_i)_{i=1}^{k}$ be the set of k-nearest seeds of x, ordered by increasing distance to x. That is, $s_1 \in x_{near}$ is the nearest seed to x. Let $x_{edges} \subseteq x_{near}$ be the ordered subset of seeds which are closest to s_1, under the ϵ tolerance (see Figure 4):

$$x_{edges} = \{s_i \in x_{near} : \|x - s_1\| + \epsilon \leq \|x - s_i\|\}$$

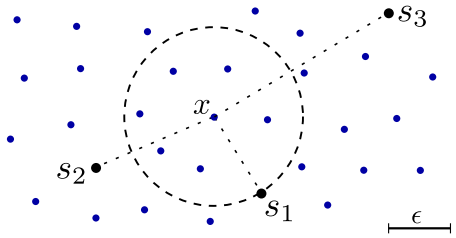


Figure 4: A two-dimensional illustration of seed set $x_{near} = \{s_1, s_2, s_3\}$, and subset $x_{edges} = \{s_1, s_2\}$, with ϵ tolerance.

The two closest seeds $s_1, s_2 \in x_{edges}$ correspond to a Delaunay edge. The remaining different pairs of seeds may correspond to a Delaunay edge.

We employ the following technique in order to recover RDT edges. Consider a graph $\mathcal{G}$, where a vertex corresponds to a seed, and an edge between two seeds $s_a, s_b \in \mathcal{S}$ is marked as Delaunay or candidate.

A *Delaunay edge* is one that has been reported by at least one point of $\mathcal{P}$ as a Delaunay edge. That is, $\exists x \in \mathcal{P}, \exists s_a, s_b \in x_{edges}$ such that s_a and s_b are the two closest. A *candidate edge* is one that has been reported, but never as a Delaunay edge (see Figure 5a). That is, $\forall x \in \mathcal{P}, \forall s_a, s_b \in x_{edges}$, s_a and s_b are not the two closest.

Definition 2. *(ϵ-cospherical) A set of seeds $\mathcal{S}$ is a ϵ-cospherical if* $\forall s_1, s_2, s_3 \in \mathcal{S}$, $|\|s_1 - s_2\| - \|s_1 - s_3\|| \leq \epsilon$.

Observe that candidate edges are likely reported by points of $\mathcal{P}$ lying between the Voronoi region boundaries of ϵ-cospherical seeds. We select an arbitrary subset of candidate edges to be in the RDT, while ensuring that the RDT remains a simplicial complex. For this purpose, we propose an approach that identifies features of the graph $\mathcal{G}$ associated to ϵ-cospherical seeds. We stood upon the observation that minimal cycles of Delaunay edges are likely to be associated with ϵ-cospherical seeds. A cycle is minimal if it does not contain other cycles.

Let x_{edges} contain more than three seeds, which are ϵ-cospherical. If x is not at exactly equal distance to each seed in x_{edges}, then the edge defined by $s_1, s_2 \in x_{edges}$ could correspond to a Delaunay edge that bounds a minimal cycle of ϵ-cospherical seeds. In case x is at exactly equal distance to all seeds in x_{edges}, we do not consider it to compute $\mathcal{G}$.

Let s_{cycle} be a minimal cycle of seeds of $\mathcal{G}$ connected by Delaunay edges. Consider the subset of seeds $s_{vert} \subset s_{cycle}$, where each seed is connected with all the others seeds of s_{cycle}, either with a candidate or a Delaunay edge. We consider that all the edges of $\mathcal{G}$ with origin in an arbitrary seed of s_{vert} belong to the RDT (see Figure 5b).

Once the RDT edges are recovered, we generate the mesh triangles. This is simply done by enumerating all the minimal RDT edge cycles of length three.

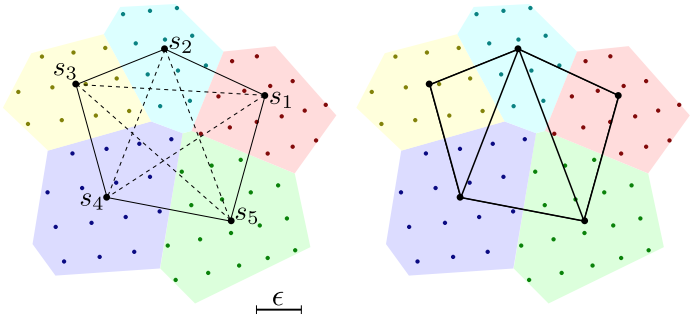


Figure 5: Extracting RDT edges from five seeds in ϵ -cospherical position. (a) Recovered Delaunay (solid lines) and candidate edges (dashed lines). Delaunay edges form a minimal cycle around the five seeds. In this case we have $s_{cycle} = s_{vert} = \{s_1, s_2, s_3, s_4, s_5\}$. (b) Two candidate edges emanating from s_2 are arbitrarily selected to belong to the RDT.

Implementation

First, for every surface point x, we compute the set x_{edges}. To compute the k-nearest seeds from x, we construct the same regular grid described in Section 4.2. We detect at most three nearest seeds ($|x_{near}| = 3$).

The graph $\mathcal{G}$ is stored as an adjacency list. For every seed s, we maintain two unique sets of seeds $del(s)$ and $cand(s)$ that have been reported by some surface point as Delaunay or candidate edge, respectively. Each seed s is identified by a unique integer $id(s) \in \mathbb{N}^+$. See Algorithm 1 for the pseudocode of seed insertion in a unique set.

Algorithm 1 Insertion of seed s in a unique ordered set.

- The function *imageAtomicMax(image, P, data)* atomically computes the maximum of a value with an existing value in memory, stores that value and returns the original value (see [29])
- The unique set starts on *address* of 1D image *uniqueSets*.

```
function INSERTINSET(uniqueSets, address, id)
    c ← 0
    i ← id(s)
    while c < maxSizeSet do
        v ← imageAtomicMax(uniqueSets, address+c, i)
        if v = 0 ∨ v = i then return
        else i ← v
        c ← c + 1
```

To determine if an edge of $\mathcal{G}$ defined by the seeds s_1 and s_2 either is Delaunay or candidate, we iterate over the ordered sets $del(s_1)$ and $cand(s_1)$. If $s_2 \in del(s_1)$, then s_2 is a Delaunay edge. Otherwise, if $s_2 \notin del(s_1)$ and $s_2 \in cand(s_1)$, it is a candidate edge.

To recover the triangles, we enumerate in parallel the triangles with seeds $s_1 s_2 s_3$, such that $id(s_1) < id(s_2) < id(s_3)$. By doing so, each triangle will be output only once. If all the edges of a triangle are Delaunay, it is directly generated. If not, we detect minimal cycles with length greater than three, that include the triangle, and decide which triangles are generated according to the presented procedure (see Figure 6).

In our implementation, we detect minimal cycles of at most five seeds. This is sufficient to obtain a two-manifold mesh in most situations (see Section 5.2). Nevertheless, if desired, an

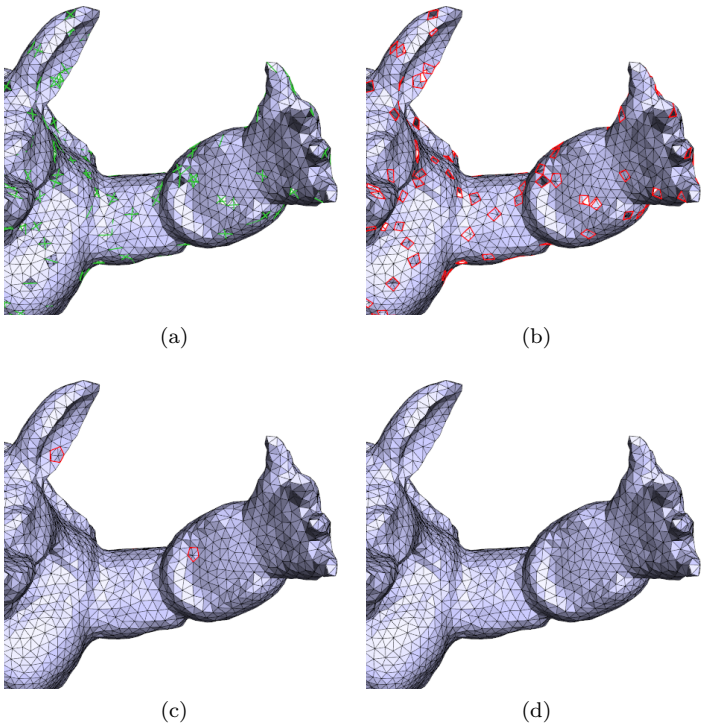


Figure 6: Generating the triangle mesh. (a) Candidate edges shown in green. (b) Candidate edges are not considered. (c) Detection of minimal cycles of length four. (d) Detection of minimal cycles of length four and five.

exhaustive enumeration of minimal cycles could be implemented.

4.4 Local regularization

In this section we present an approach that ensures the convergence of the interleaved process. When $\mathcal{O}$ bounds a solid $\mathcal{O}_S$, we would enforce that the extracted mesh is two-manifold, that is, every RDT edge has exactly two incident triangles.

A well-known problem of RCVT surface meshing is to ensure homeomorphic surface reconstruction from the input. The surface meshing of small features is particularly challenging (see Figure 7). Yan et al. [24] perform a topological ball property test, and insert new seeds around non-manifold features until the topology of the input is fully recovered.

Unfortunately, in our setting we only rely on the sampled surface points and do not have explicit knowledge of the input surface topology. For instance, we loosely assume that if three seeds define a minimal cycle of RDT edges, they also share a common Voronoi vertex (see Section 4.3). Moreover, properly capturing small features of $\mathcal{O}$ will ultimately require an infinite sampling density. Conversely, incrementing the number of seeds will lower the α ratio (see Section 4.1), and therefore deteriorate the quality of the discrete optimization.

To overcome the aforementioned problems, we propose to consider $\mathcal{P}$ as a signal and filter it (regularize) such that a two-manifold mesh can be extracted. Our technique is inspired by the r-regularity property of a surface that we introduce below.

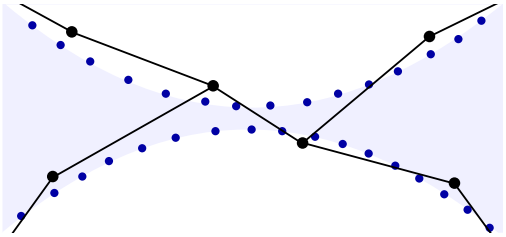


Figure 7: A two-dimensional problematic case of surface meshing. Properly capturing the small feature in the middle requires a higher sampling density and/or more seeds.

Medial axis and r-regularity. The medial axis of $\mathcal{O}$ is the set of points in $\mathbb{R}^3 \setminus \mathcal{O}$ with at least two closest points on $\mathcal{O}$. The medial axis transform (MAT) [34] is composed of the medial balls centered at the medial axis. $\mathcal{O}$ is r-regular if all the medial balls of MAT have radius equal or greater than r. The local feature size lfs is the minimum distance to the medial axis of $\mathcal{O}$.

Observe that non-manifold features of the RDT are expected to arise where the lfs is small (see Figures 8a and 8b). For an r-regular surface, the lfs becomes equal or greater than r everywhere. The points in an r-regular surface that are far away in terms of geodesic distance, are at a distance greater than r. This alleviates the problems related to meshing thin features and small holes. However, both the medial axis and the r-regularization are in general difficult and computationally expensive to compute [35, 34]. Moreover, not all parts of $\mathcal{O}$ may require to be r-regular in order to extract a two-manifold mesh. Thus, global r-regularization is a strong requirement that may introduce unnecessary smoothing of $\mathcal{P}$.

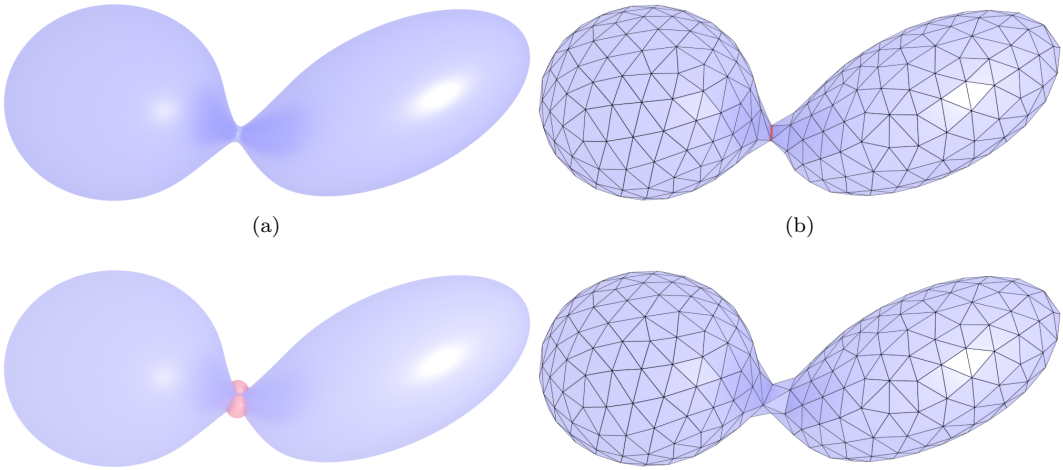


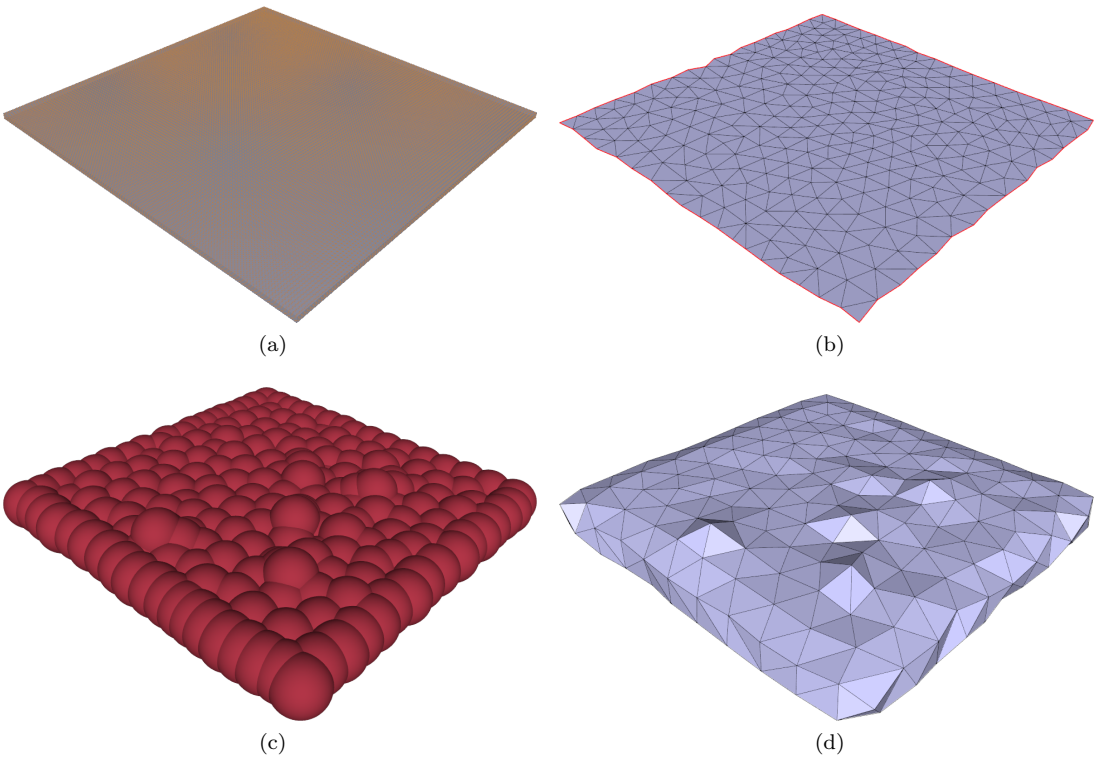
Figure 8: (a) Input solid with a small feature in the middle. (b) Discrete RCVT meshing with four hundred seeds. A non-manifold edge (red color) appears due the thin feature. (c) A solid composed of the union of the input and two balls (red color) sampled along the non-manifold edge. (d) Two-manifold mesh obtained after the local regularization.

We propose a simpler local filtering approach, inspired by the r-regularization. Broadly, the local regularization consists in sampling balls over non-manifold RDT edges, whose union with

$\mathcal{O}_S$ regularizes its surface (see Figures 8c and 8d). During the interleaved process, the union of $\mathcal{O}_S$ and the current set of regularization balls is used as an input in the subsequent discrete optimization step. The regularization balls have an initial radius r_{ini} and a maximum radius r_{max}, that depend on the length l of the sampled non-manifold edge.

Let B_{prev} be the previous set of regularization balls that still do not reach the maximum r_{max} size (initially $B_{prev} = \emptyset$), and $B_{current}$ be the new one. For every non-manifold edge, we check if it intersects any ball $B \in B_{prev}$. If so, we increase the radius of B, by an amount $r_{inc} > 0$, and insert it in $B_{current}$. Otherwise, if a non-manifold edge does not intersect any ball of B_{prev}, we sample new balls with radius r_{ini} on it, and insert them in $B_{current}$. At the end of the local regularization, we compute ${\mathcal{O}_S}' = \mathcal{O}_S \cup B_{current}$, and hand the surface $\mathcal{O}'$ over to the next discrete optimization.

Although the RCVT does not constrain seeds to belong to $\mathcal{O}$, we assume that seeds are at a distance less than r_{max} to $\mathcal{O}$. As a consequence we have that ${\mathcal{O}_S}' \supset \mathcal{O}_S \cup B_{current}$, that is the union of regularization balls is an increasing operator. In the worst case, the increasing union of regularization balls ultimately leads to a super union of regularization balls, with a constrained minimum radius, that can be meshed (see Figure 9c).



(a) (b) (c) (d)

Figure 9: A challenging case for discrete RCVT meshing. (a) Sampled surface points $\mathcal{P}$ over a thin rectangular box. (b) Initial extracted mesh consists of a single connected non-manifold surface. (c) Set of regularization balls iteratively inserted to obtain a two-manifold mesh (d).

Implementation

The local regularization comprises two different steps. First, we detect non-manifold edges of the RDT. Then, we retrieve the current regularization balls and compute the union between them and the three dexel structures. The resulting union is fed back into the discrete optimization step.

We build a hash table E on the GPU, where the key is a RDT edge, and the value is the number of incident triangles. We use the parallel hashing approach of García et al. [36] (based on CUDA) ported to the rasterization pipeline. An edge between two seeds s_1 and s_2 is encoded by the direct concatenation of seed identifiers $\min(id(s_1), id(s_2))$ and $\max(id(s_1), id(s_2))$. For $k \geq 0$, the hash function is defined as:

$$H(s_1, s_2, k) = ((id(s_1) \oplus id(s_2)) + \Theta(k)) \mod |E|$$

Where Θ is a precomputed array of random numbers. To detect non-manifold edges, we process in parallel each entry of E. An entry of E is not empty if the key is greater than zero. Non-manifold edges correspond to all non-empty entries whose associated value is different from two.

The second step involves the detection of the current set of regularization balls $B_{current}$, and the update of the three dexel structures, by computing the union with $B_{current}$. In our current implementation, the detection of non-manifold edges intersecting regularization balls of B_{prev} is performed in the CPU, but it could be easily done on the GPU. Nevertheless, this step currently represents an almost negligible part of computation time (see Section 5.2).

Let e be a non-manifold edge between seeds s_1 and s_2, of length l. We set in our experiments $r_{ini} = l/2$, $r_{max} = l$, and $r_{inc} = l/20$. The number of regularization balls sampled onto e is set aware of the sampling resolution d, and the orthogonal direction with minimal projected edge length. That is, $w \min_i(|s_{1i} - s_{2i}|)/d$, where $w > 0$ is a density weight factor. In our experiments we set $w = 0.5$, which samples the edge densely enough.

Finally, the union of $B_{current}$ and the three current dexel structures is done efficiently by considering the dexel structure of $B_{current}$. Instead of computing the whole union we locally update the current dexel structures only where new regularization balls appear.

5 Results

All the two-manifold extracted meshes by our approach that are shown in this section can be found in the supplementary material.

5.1 Applications

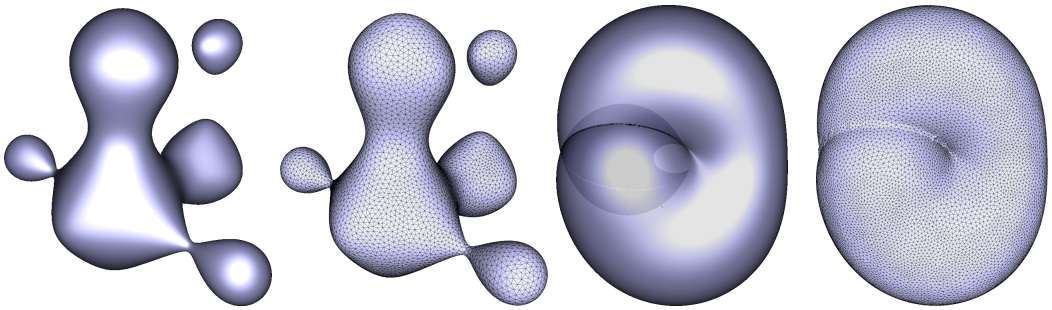
The presented approach can be used in a broad range of applications. In the following, we present some of them.

5.1.1 Meshing of implicit surfaces

The surface points of an implicit surface, and if needed the dexel structure, can be retrieved with ray-tracing rasterization techniques for implicit surface rendering [37] or fractal rendering [38]. In all the examples shown in this paper we either employ regular ray marching, or the sphere tracing technique [39].

Figure 10 shows some examples of meshing implicit surfaces with our method. Compared with state of the art approaches for implicit surfaces, we achieve the highest quality of triangle isotropy (see Section 5.2.1).

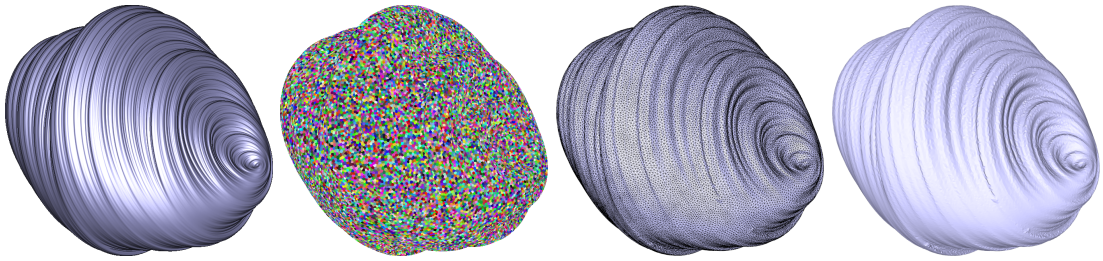
The surface meshing of good meshes for implicit surfaces with singularities is still an open research area [7]. Thanks to the local regularization, we can optionally enforce two-manifold meshing for surfaces with singularities (see Figure 10d).



(a) (b) (c) (d)

Figure 10: Discrete RCVT meshing of implicit surfaces. (a) A non-algebraic surface defined by equation $x^2 + y^2 + z^2 + \sin(4x) - \cos(4y) + \sin(4z) = 0$, and its extracted mesh (b). (c) A sextic surface (see Appendix A in [37]) representing an impression of the Klein bottle, and its extracted mesh (d).

Our approach is also able to obtain a two-manifold mesh from fractal geometry (see Figure 11), which is a challenging case due to their non-differentiable nature. Local regularization plays an important role by effectively filtering the arbitrarily small features that fractals exhibit.



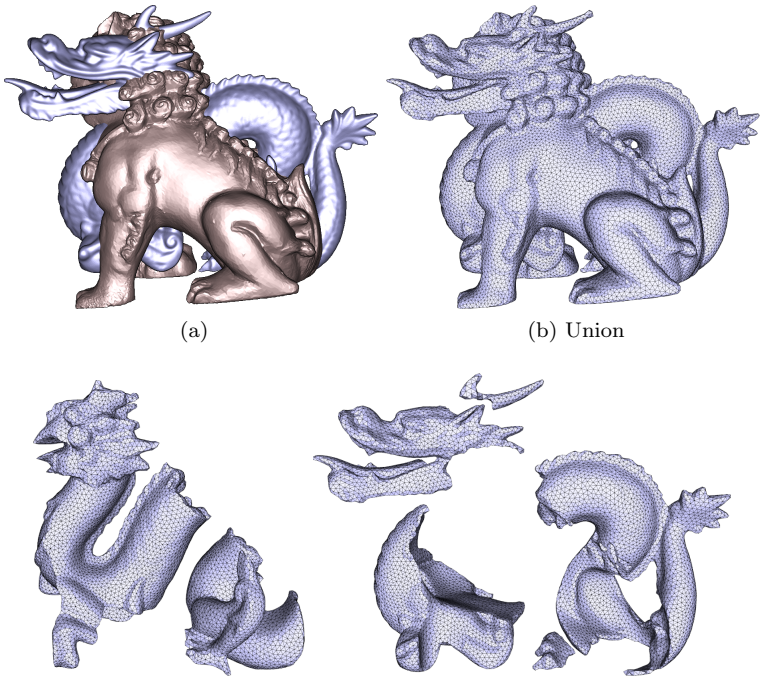
(a) (b) (c) (d)

Figure 11: Discrete RCVT meshing of fractals. (a) A Julia fractal defined by quaternion $(0.1, 0.2, 0.3, 0)$. (b) Discrete Voronoi regions over the fractal surface. (c-d) Extracted two-manifold mesh.

5.1.2 Meshing of boolean operations

Boolean operation between solids [40] are important in solid modeling. For instance, they form the basis for Constructive Solid Geometry (CSG). However, meshing of boolean operations can present several problems, and its interactive modification is often challenging.

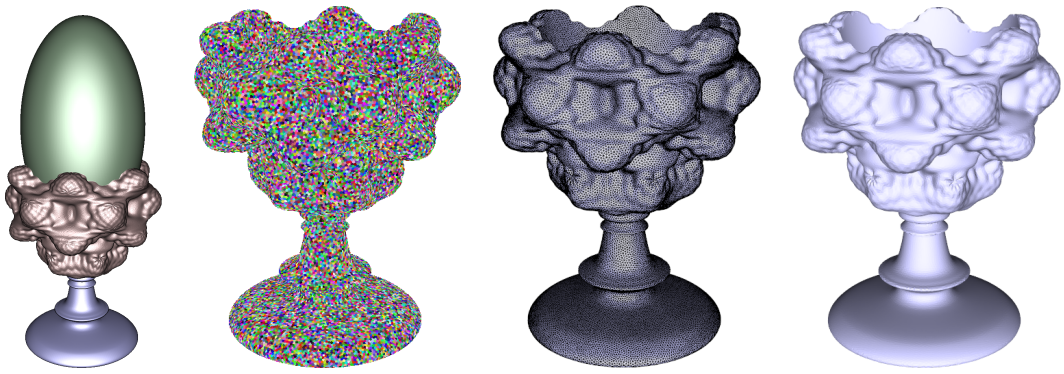
CSG expressions can be computed easily and efficiently with dexel structures [30], by determining which ray intervals lie inside the result (see Figure 12). Our approach allows to mesh indistinctly CSG expressions between any kind of rasterized surfaces (see Figure 13).



(a) (b) Union

(c) Intersection (d) Difference

Figure 12: Discrete RCVT meshing of different CSG expressions between two solids (a).



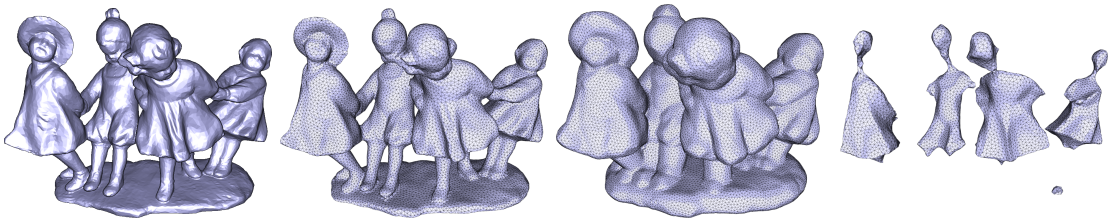
(a) (b) (c) (d)

Figure 13: Discrete RCVT meshing of a CSG expression (a) between a mesh (blue), and implicitly defined ball (green), and a Mandelbrot fractal of degree 7 (orange). We union the mesh and the fractal, and then subtract the ball. The resulting solid resembles a cup. (b) Discrete Voronoi regions. (c-d) Extracted two-manifold mesh.

5.1.3 Meshing of morphological operations

Morphological operations, such as dilation or erosion, are used in several applications. Exact offset surface computation is in general difficult an a number of approximations have been proposed.

There exist different approaches to compute morphological operations with ray-representations of solids. We use the technique presented in [41] designed for dexel structures. It allows us to robustly and efficiently extract a two-manifold offset surface mesh without any explicit handling of topological changes of the offset surface (see Figure 14).



(a) (b) (c) Dilation (d) Erosion

Figure 14: Discrete RCVT meshing of morphological operations of the solid (a). Subfigure (b) shows the meshing of the input solid.

5.1.4 Mesh repair

Mesh repairing techniques [42] try to filter out most of the typical flaws of 3D digitized models, like non-manifold features, holes and gaps, degenerate triangles, etc.

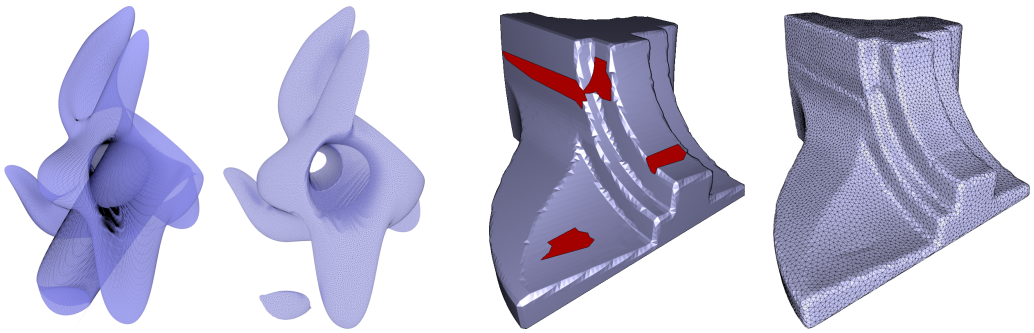
Ray representations of solids have already been used to repair meshes [43], by extracting the outer hull [44] of the mesh. Consider the inside of solid to be not reachable from infinity without crossing the surface. Then, the boundary between the inside and the outside is the outer hull. The orientation sensitive outer hull [44] is obtained by making the reachability sensitive to the surface orientation, ignoring crossing surface parts that are back-facing with respect to the direction.

In our context, it is straightforward to mesh the orientation sensitive outer hull, by also storing the surface sample orientation information (back or front facing) with respect to the rasterization direction (see Figure 15). Then, during the rasterization we just discard the fragments that do not belong to the outer hull.

Moreover, with a technique similar to the ray stabbing method of [43], we are able to remesh and fill the holes and gaps of the input mesh. To do this, we construct dexel structures in different directions, storing the orientation information, and filter out the subset of dexels defined by two endpoints with front and back facing orientation. The filtered set of dexels is likely inside the input mesh. Thus, we mesh the union of all the dexel volumes. With this technique we are able to fill holes and gaps and repair non-manifold features (see Figure 15).

5.1.5 Dynamic meshing

Our approach can be used for meshing of dynamic surfaces. We are inspired by the CVT approach of Lopez and Levy [45] adapted to our discrete setting. It is challenging to achieve isotropic meshing of deforming surfaces at interactive rates [46]. CVT meshing provides a high-quality mesh, but it is computationally expensive, usually taking minutes to process fairly complex



(a) (b) (c) (d)

Figure 15: On the left, computing the orientation sensitive outer hull of a mesh with interpenetrating parts. (a) Input mesh shown in transparent color. (b) Extracted outer hull mesh. On the right, repairing a mesh with holes, shown in red. (c) Input mesh. (d) Repaired mesh, obtained from four different sampling directions.

meshes. Thanks to its high parallelism, our approach enables CVT isotropic meshing of deforming surfaces at almost interactive rates (see Section 5.2.2).

We consider the case where the surface $\mathcal{O}_t$ at time $t \in \mathbb{N}$ is implicitly defined. A time varying implicit function, or a mesh that evolves under an implicit deformation (see Figure 20), are examples of that case. RDT_t is the mesh at time t. Instead of randomly initializing the seeds position for each time step of the process, we use their previous position for a better initialization. We define a minimum and maximum edge length of the dynamic mesh. Usually these bounds will be given by the edge lengths of the initial mesh RDT_0.

For $t > 0$ we compute a temporal mesh RDT_λ, according to $\mathcal{O}_t$ and the seeds of RDT_{t-1}. Then, for every edge of RDT_λ, if the edge length is greater than the maximum one, we insert a new seed in the edge center. If the edge length is lower than the minimum, we erase both edge seed endpoints and insert a new seed in the edge center. In addition, if the discrete Voronoi region of a seed has a low number of surface points, we erase the seed. This is done to maintain a good α ratio (see Section 4.1). All this process is entirely done in the rasterization pipeline.

Finally, we compute RDT_t taking as input $\mathcal{O}_t$ and the seeds of RDT_λ (see Figure 16).

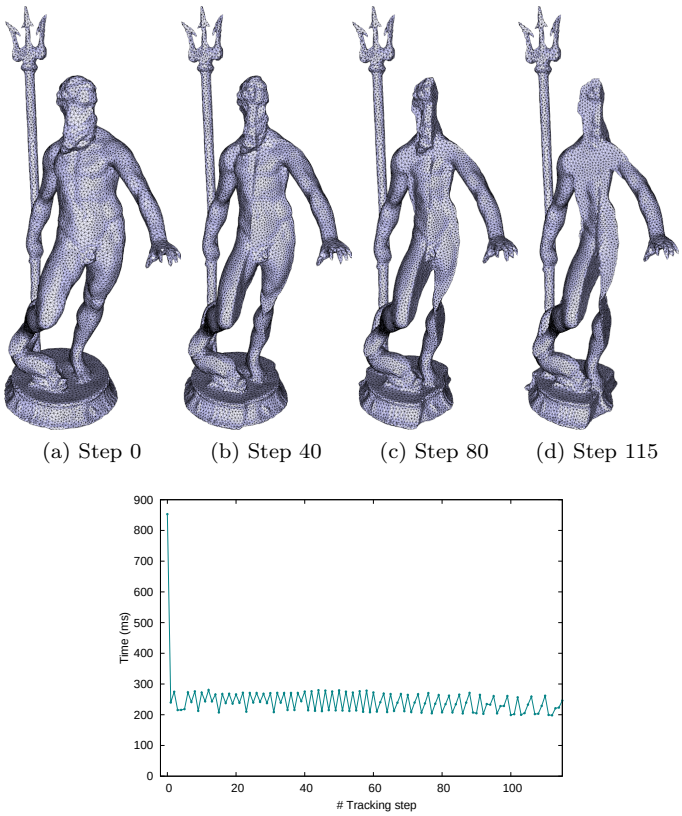
5.1.6 Solid sculpting

The first application of dexel structures was in the context of NC milling rendering [30]. Employing three orthogonal dexel structures is also a common technique for sculpting [47]. Sculpting is directly done using the dexel structure of the input shape. Consider a sculpting tool moving along a path, and a sufficient discrete sampling of path positions. At every path position, the sculpting tool is converted into a dexel structure. Then, the input dexel structure is updated by either performing the union or difference with the tool dexel structure (see Figure 17).

Alternatively, the mesh is also readily available for per-vertex painting and deformation (see Section 5.1.5) through existing sculpting techniques [48].

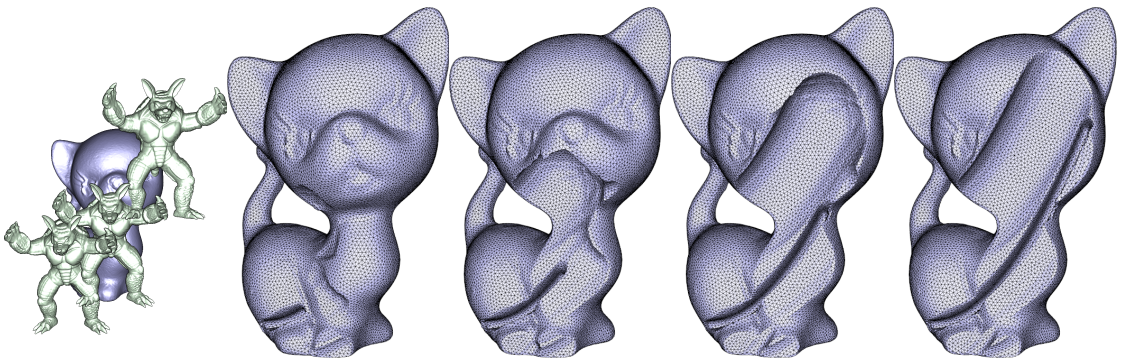
5.2 Evaluation

In this section, we evaluate the meshing quality and performance of the presented approach. All the experiments were carried out on an Intel Core i7 4770k with 16GB of memory, and a GeForce



(a) Step 0 (b) Step 40 (c) Step 80 (d) Step 115

Figure 16: Dynamic meshing of an evolving mesh with self-intersections. The tracking has 115 incremental steps. The initial tracking mesh has 8140 vertices. The dexel structure directly resolves the solid parts from self-intersections. On the right, performance timings of the dynamic meshing.



(a) (b) (c) (d) (e)

Figure 17: Discrete RCVT meshing of a milling sculpting operation. (a) Illustration of the path followed by the sculpting tool (the entire Armadillo model). (b-e) Different steps of the sculpting operation.

	Vertices	Q_{min}	Q_{avg}	$d_H(\%)$	Time
Marching cubes	4044	0.002	0.581	2.30	103
	20058	0.001	0.597	0.59	116
	60012	0.001	0.601	0.18	552
Delaunay meshing	4007	0.487	0.842	1.32	410
	20017	0.480	0.856	0.51	1655
	60030	0.473	0.843	0.22	5229
Our method	4000	0.623	0.914	0.86	645
	20000	0.501	0.882	0.52	707
	60000	0.518	0.894	0.21	1467

Table 1: Meshing results of the implicit surface of Figure 18. In red the worst result for each different measure.

Titan Black with 6GB of memory.

5.2.1 Meshing quality

The quality of a triangle is measured by $Q_t = \frac{6}{\sqrt{3}} \frac{A_t}{h_t e_t}$, where A_t is the area, h_t is the half-perimeter, and e_t is the longest edge length of the triangle [49]. Q_{min} is the minimal triangle quality, and Q_{avg} is the average one. This is a common descriptor of isotropic triangle quality [24]. d_H is the two-sided Hausdorff distance between the reference surface and the extracted mesh, divided by the diagonal of the bounding box of the mesh. d_H is approximated by computing a dense sample of points of both surfaces and computing the Hausdorff distance between them.

Figure 18 shows a comparison of meshing methods for implicit surfaces. Our approach achieves always the best triangle quality ratios. Marching cubes generates several triangles with poor quality, while Delaunay meshing exhibits lower quality triangle than our CVT method.

5.2.2 Meshing performance

The time per iteration of the discrete optimization, and the mesh extraction, exhibits an experimental linear complexity with respect to the number of seeds (see Figure 19). The L-BFGS iteration has an overhead compared with the Lloyd method. However, as expected [15], the L-BFGS method minimizes faster the CVT energy than the Lloyd one. The main bottleneck of the computation resides in the computation of the discrete centroids and gradients. The mesh extraction and detection of non-manifold edges also grows linearly in the experiments, and represents an small fraction of the overall computation.

We can see in Table 1 that the performance of our method scales better than the Delaunay meshing. Table 2 shows the meshing performance and quality of different figures in the paper. As expected, the surfaces with small features (e.g. Figures 12d and 10d) require more local regularization iterations. However, after the first iteration, the subsequent ones require less time, because the initialization of seeds becomes better (see Figure 16).

We also computed a mesh from 10000 randomly generated CSG scenes (intersection, difference and union) between a database of 12 meshes. Our method was always able to extract a two-manifold mesh. In average, the discrete optimization represented the 93% of the overall computation time, the local regularization the 6%, and the mesh extraction the remaining 1%.

We also evaluated the average number of RDT edges generated by detecting minimal cycles of length three, four and five, representing respectively 97.85%, 1.77%, and 0.38% of the total

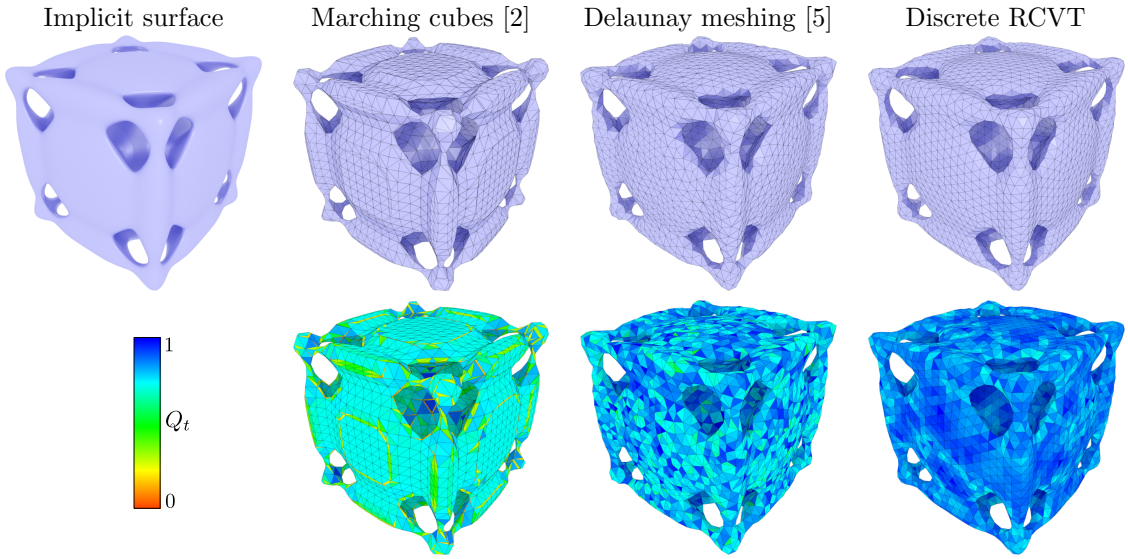
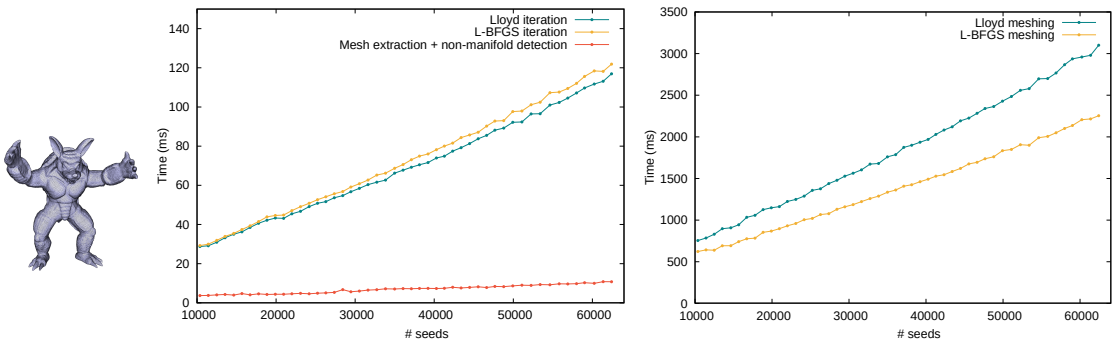


Figure 18: Comparison of different approaches to mesh implicit surfaces. The first row show the extracted meshes, which have a similar number of vertices. The second row shows the triangle quality according to color scale, shown at the left. See Table 1 for the related numerical results. The Chmutov octic surface is defined as $T(x)+T(y)+T(z) = 1.6$, where $T(x) = \left(2x^2\left(3-4x^2\right)\right)^2$. The marching cube mesh extraction uses the single core CPU implementation of Meshlab [50]. The Delaunay meshing method uses the single core CPU implementation of CGAL [51, 52], and a lower bound of 30° for the triangle angles. Our approach uses the L-BFGS method.



(a) (b) (c)

Figure 19: Performance timings of the discrete RCVT meshing with the Armadillo model. (a) Resulting meshing with 30000 seeds. (b) Time per iteration for Lloyd (GPU), and L-BFGS (GPU+CPU) discrete optimization. Time to extract the mesh and detect non-manifold edges (GPU). (c) Total time for meshing with Lloyd and L-BFGS methods. Both methods consider the same stopping threshold of the CVT energy function.

Figure	Vertices	Q_{avg}	Iterations	Time
10b	7890	0.851	1	316
10d	18950	0.860	19	7406
11c	45454	0.850	13	11828
12b	22775	0.856	4	1801
12c	10267	0.853	11	1890
12d	12804	0.862	18	6708
14b	17912	0.858	3	1286
14c	23613	0.866	5	1932
14d	5710	0.856	17	2388

Table 2: Meshing results of different figures along the paper. Iterations denotes the number of local regularization steps that were needed to extract a two-manifold mesh.

number of edges. Thus, detecting cycles with length greater than five has little impact in practice.

Limitations

For thin features of the input surface $\mathcal{O}$ with large surface area the local regularization may require several iterations (see Figure 9). An initial global regularization of $\mathcal{O}$ may solve that problem, but at the cost of losing detail in the parts that could be meshed without performing an r-regularization.

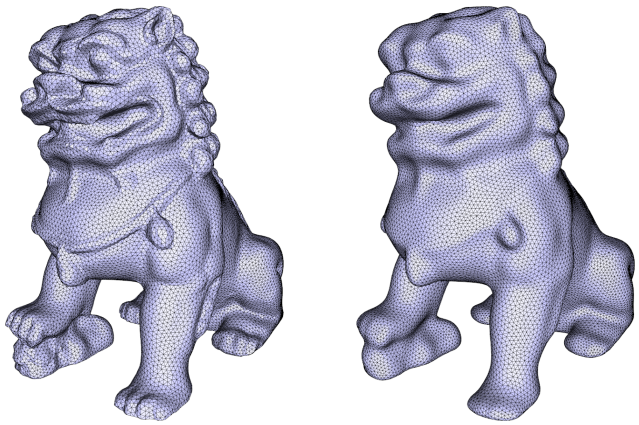
Our approach could be used for interface tracking of surfaces [53]. In that case the surface $\mathcal{O}_t$ is not implicitly defined, and RDT_t (see Section 5.1.5) directly serves as the interface tracking mesh. At each time step the vertices of RDT_t are advected (typically by using a velocity field), seeds are inserted or deleted, and RDT_{t+1} is computed by RCVT meshing with the new set of seeds. Unfortunately, RCVT exhibits quite a bit of diffusion (see Figure 20). Using CCVT instead of RCVT reduces that problem as seeds are constrained to be on the surface. Unfortunately, our tests show that projecting seeds onto the nearest surface point [19] hampers the discrete optimization process.

6 Conclusion

We have presented a method for surface meshing relying on existing rasterization code. We use the rasterization pipeline to sample points and compute an RCVT in a discrete fashion. Mesh extraction is done with a parallel algorithm as well and is capable of handling many cospherical situations.

For solids, obtaining a two-manifold mesh is challenging. Currently, we detect non-manifold edges and employ local regularization by unioning balls of increasing radius into the input solid. With our method, a low number of seeds combined with a low local feature size may lead to quality issues with the meshing. Our algorithm would benefit from a good estimator for the number of seeds in this respect.

To improve our algorithm further and truly adapt the meshing to the local feature size, as discussed in Section 4.4, we could insert new seeds and increase sampling density where appropriate to perform anisotropic meshing. Obtaining the local feature size is computationally expensive though. Sampling points over the surface in a variable-density fashion is also difficult. Not only can the sampling density increase indefinitely as the local feature size converges to zero



(a) (b)

Figure 20: Diffusion of interface tracking. (a) Initial tracking surface. (b) After several RCVT meshing computations, the interface tracking mesh loses detail.

to capture infinitely small features, but it is also challenging to perform variable-density point sampling using the rasterization pipeline, which our method integrates well with.

Our experiments with interface tracking (Figure 20) show off some diffusion issues that could be reduced with CCVT meshing. A surface constraining heuristic for seeds could be investigated for CCVT meshing to successfully take place. This would reduce the diffusion effect as well as improve the mesh quality overall.

A good initialization of seeds is important in order to speed-up the CVT computation. Our approach could benefit from a more careful initialization [54].

Our experiments show that the discrete optimization step is the main performance bottleneck of our approach (see Section 5.2.2). The discrete optimization can be accelerated by employing a more sophisticate data structure (e.g. a GPU kd-tree) to store the seeds and accelerate the nearest seed query. In addition, the L-BFGS optimization could be done entirely on the GPU by the recent algorithm of Fei et al. [55].

References

[1] J. R. Shewchuk, "What is a good linear element - interpolation, conditioning, and quality measures," in *In 11th International Meshing Roundtable*, 2002, pp. 115–126.

[2] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, 1987, pp. 163–169.

[3] J. Menon, R. J. Marisa, and J. Zagajac, "More powerful solid modeling through ray representations," *IEEE Computer Graphics and Applications*, vol. 14, no. 3, pp. 22–35, 1994.

[4] S. Lefebvre, "IceSL: A GPU accelerated CSG modeler and slicer," in *AEFA'13, 18th European Forum on Additive Manufacturing*, 2013.

[5] J.-D. Boissonnat and S. Oudot, "Provably good sampling and meshing of surfaces," *Graphical Models*, vol. 67, no. 5, pp. 405 – 451, 2005.

[6] Q. Du, V. Faber, and M. Gunzburger, "Centroidal Voronoi tessellations: applications and algorithms," *SIAM Review*, vol. 41, no. 4, pp. 637–676, 1999.

[7] J.-D. Boissonnat and M. Teillaud, *Effective computational geometry for curves and surfaces.* Springer, 2006.

[8] J. M. Snyder, "Interval analysis for computer graphics," in *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, 1992, pp. 121–130.

[9] J. Chen, X. Jin, and Z. Deng, "GPU-based polygonization and optimization for implicit surfaces," *The Visual Computer*, vol. 31, no. 2, pp. 119–130, 2015.

[10] M. O. Benouamer and D. Michelucci, "Bridging the gap between CSG and Brep via a triple ray representation," in *Fourth ACM/Siggraph Symposium on Solid Modeling and Applications*, 1997, pp. 68–79.

[11] C. C. 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, 2010.

[12] P. Feng and J. Warren, "A dual method for constructing multi-material solids from ray-reps," *Lecture Notes in Computer Science*, vol. 7431, pp. 92–103, 2012.

[13] T. Ju, F. Losasso, S. Schaefer, and J. Warren, "Dual contouring of hermite data," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 339–346, 2002.

[14] S. P. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, pp. 129–137, 1982.

[15] 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. 101:1–101:17, 2009.

[16] R. Kunze, F. Wolter, and T. Rausch, "Geodesic Voronoi diagrams on parametric surfaces," in *Proceedings of the Conference on Computer Graphics International*, 1997, pp. 230–237.

[17] X. Wang, X. Ying, Y.-J. Liu, S.-Q. Xin, W. Wang, X. Gu, W. Mueller-Wittig, and Y. He, "Intrinsic computation of centroidal Voronoi tessellation (CVT) on meshes," *Computer-Aided Design*, vol. 58, pp. 51 – 61, 2015.

[18] H. Edelsbrunner and N. R. Shah, "Triangulating topological spaces," *International Journal of Computational Geometry and Applications*, vol. 07, no. 04, pp. 365–378, 1997.

[19] Q. Du, M. Gunzburger, and L. Ju, "Constrained centroidal Voronoi tessellations for surfaces," *SIAM Journal on Scientific Computing*, vol. 24, no. 5, pp. 1488–1506, 2003.

[20] D.-M. Yan, G. Bao, X. Zhang, and P. Wonka, "Low-resolution remeshing using the localized restricted Voronoi diagram," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 10, pp. 1418–1427, 2014.

[21] G. Rong, M. Jin, and X. Guo, "Hyperbolic centroidal Voronoi tessellation," in *Proceedings of the 14th ACM Symposium on Solid and Physical Modeling*, 2010, pp. 117–126.

[22] P. Alliez, D. Cohen-Steiner, M. Yvinec, and M. Desbrun, "Variational tetrahedral meshing," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 617–625, 2005.

[23] S. Valette, J.-M. Chassery, and R. Prost, "Generic remeshing of 3D triangular meshes with metric-dependent discrete Voronoi diagrams," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 2, pp. 369–381, 2008.

[24] D.-M. Yan, B. Lévy, Y. Liu, F. Sun, and W. Wang, "Isotropic remeshing with fast and exact computation of restricted Voronoi diagram," *Computer Graphics Forum*, vol. 28, no. 5, pp. 1445–1454, 2009.

[25] P. Alliez, E. Verdière, O. Devillers, and M. Isenburg, "Centroidal Voronoi diagrams for isotropic surface remeshing," *Graphical Models*, vol. 67, no. 3, pp. 204 – 231, 2005.

[26] G. Rong, Y. Liu, W. Wang, X. Yin, X. Gu, and X. Guo, "GPU-assisted computation of centroidal Voronoi tessellation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 3, pp. 345–356, 2011.

[27] X. Gu and S.-T. Yau, "Global conformal surface parameterization," in *Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, ser. SGP '03, 2003, pp. 127–137.

[28] L. Shuai, X. Guo, and M. Jin, "GPU-based computation of discrete periodic centroidal Voronoi tessellation in hyperbolic space," *Computer-Aided Design*, vol. 45, no. 2, pp. 463 – 472, 2013.

[29] K. A. Mark Segal, "The OpenGL graphics system: a specification (version 4.2)," 2011.

[30] T. Van Hook, "Real-time shaded NC milling display," in *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, 1986, pp. 15–20.

[31] S. Lefebvre, S. Hornus, and A. Lasram, "HA-buffer: Coherent hashing for single-pass A-buffer," INRIA, Tech. Rep. 8282, 2013.

[32] Y. Liu, "HLBFGS," http://research.microsoft.com/en-us/um/people/yangliu/software/HLBFGS/.

[33] M. Maule, J. Comba, R. Torchelsen, and R. Bastos, "Memory-efficient order-independent transparency with dynamic fragment buffer," in *25th SIBGRAPI Conference on Graphics, Patterns and Images*, 2012, pp. 134–141.

[34] D. Attali, J. D. Boissonnat, and H. Edelsbrunner, "Stability and computation of medial axes - a state-of-the-art report," in *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration.* Springer Berlin / Heidelberg, 2009, pp. 109–125.

[35] J. Williams and J. Rossignac, "Mason: morphological simplification," *Graphical Models*, vol. 67, no. 4, pp. 285 – 303, 2005.

[36] I. García, S. Lefebvre, S. Hornus, and A. Lasram, "Coherent parallel hashing," *ACM Transactions on Graphics*, vol. 30, no. 6, pp. 161:1–161:8, 2011.

[37] J. Singh and P. Narayanan, "Real-time ray tracing of implicit surfaces on the GPU," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 2, pp. 261–272, 2010.

[38] J. C. Hart, D. J. Sandin, and L. H. Kauffman, "Ray tracing deterministic 3-D fractals," in *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*, 1989, pp. 289–296.

[39] J. C. Hart, "Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces," *The Visual Computer*, vol. 12, no. 10, pp. 527–545, 1996.

[40] A. A. G. Requicha and H. Voelcker, "Boolean operations in solid modeling: Boundary evaluation and merging algorithms," *Proceedings of the IEEE*, vol. 73, no. 1, pp. 30–44, 1985.

[41] J. Martinez, S. Hornus, F. Claux, and S. Lefebvre, "Chained segment offsetting for ray-based solid representations," *Computers & Graphics*, vol. 46, pp. 36 – 47, 2015.

[42] M. Attene, M. Campen, and L. Kobbelt, "Polygon mesh repairing: An application perspective," *ACM Computing Surveys*, vol. 45, no. 2, pp. 15:1–15:33, 2013.

[43] F. Nooruddin and G. Turk, "Simplification and repair of polygonal models using volumetric techniques," *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, no. 2, pp. 191–205, 2003.

[44] M. Campen and L. Kobbelt, "Exact and robust (self-)intersections for polygonal meshes," *Computer Graphics Forum*, vol. 29, no. 2, pp. 397–406, 2010.

[45] D. Lopez and B. Lévy, "Dynamic mesh optimization for free surfaces in fluid simulation," in *IMR-21th international meshing roundtable-2012*, 2012, pp. 1–5.

[46] M. Dunyach, D. Vanderhaeghe, L. Barthe, and M. Botsch, "Adaptive remeshing for real-time mesh deformation," in *Eurographics Short Papers*, 2013, pp. 29–32.

[47] H. Muller, T. Surmann, M. Stautner, F. Albersmann, and K. Weinert, "Online sculpting and visualization of multi-dexel volumes," in *Proceedings of the Eighth ACM Symposium on Solid Modeling and Applications*, 2003, pp. 258–261.

[48] H. Schäfer, B. Keinert, M. Nießner, and M. Stamminger, "Local painting and deformation of meshes on the GPU," *Computer Graphics Forum*, 2014.

[49] P. J. Frey and H. Borouchaki, "Surface mesh evaluation," in *6th Int. Meshing roundtable*, 1997.

[50] "Meshlab," http://meshlab.sourceforge.net/.

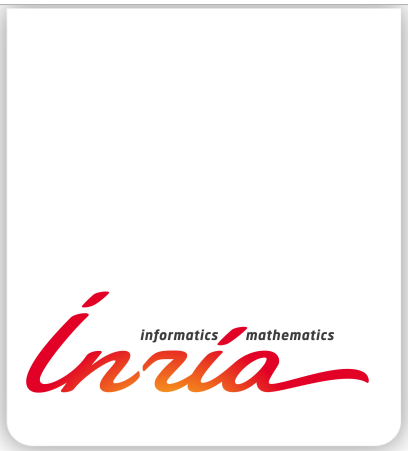
[51] "CGAL, Computational Geometry Algorithms Library," http://www.cgal.org.

[52] L. Rineau and M. Yvinec, "A generic software design for Delaunay refinement meshing," *Computational Geometry*, vol. 38, no. 1–2, pp. 100 – 110, 2007.

[53] X. Jiao, "Face offsetting: A unified approach for explicit moving interfaces," *Journal of Computational Physics*, vol. 220, no. 2, pp. 612 – 625, 2007.

[54] J. Quinn, F. Sun, F. C. Langbein, Y.-K. Lai, W. Wang, and R. R. Martin, "Improved initialisation for centroidal Voronoi tessellation and optimal Delaunay triangulation," *Computer-Aided Design*, vol. 44, no. 11, pp. 1062 – 1071, 2012.

[55] Y. Fei, G. Rong, B. Wang, and W. Wang, "Parallel L-BFGS-B algorithm on GPU," *Computers & Graphics*, vol. 40, pp. 1 – 9, 2014.



RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399