Computational Visual Media
DOI 10.1007/s41095-015-0022-4 Vol. 1, No. 3, September 2015, 239–251

Research Article

A unified framework for isotropic meshing based on narrow-band Euclidean distance transformation

Yuen-Shan Leung[1], Xiaoning Wang[1], Ying He[1] (✉), Yong-Jin Liu[2], and Charlie C. L. Wang[3]

Abstract In this paper, we propose a simple-yet-effective method for isotropic meshing relying on Euclidean distance transformation based centroidal Voronoi tessellation (CVT). Our approach improves the performance and robustness of computing CVT on curved domains while simultaneously providing high-quality output meshes. While conventional extrinsic methods compute CVTs in the entire volume bounded by the input model, we restrict the computation to a 3D shell of user-controlled thickness. Taking voxels which contain surface samples as sites, we compute the exact Euclidean distance transform on the GPU. Our algorithm is parallel and memory-efficient, and can construct the shell space for resolutions up to 2048^3 at interactive speed. The 3D centroidal Voronoi tessellation and restricted Voronoi diagrams are also computed efficiently on the GPU. Since the shell space can bridge holes and gaps smaller than a certain tolerance, and tolerate non-manifold edges and degenerate triangles, our algorithm can handle models with such defects, which typically cause conventional remeshing methods to fail. Our method can process implicit surfaces, polyhedral surfaces, and point clouds in a unified framework. Computational results show that our GPU-based isotropic meshing algorithm produces results comparable to state-of-the-art techniques, but is significantly faster than conventional CPU-based implementations.

Keywords centroidal Voronoi tessellation (CVT); Euclidean distance transformation; GPU; isotropic meshing; polygonal meshes; point clouds; implicit surfaces

1 Introduction

Triangle meshes have found widespread acceptance in computer graphics as a simple, convenient, and versatile representation of surfaces. However, raw meshes obtained from 3D scanners are often unsuitable for subsequent geometric processing, since they may contain holes, gaps, noise, degenerate triangles, and non-manifold edges.

A popular approach to improve mesh quality is to use centroidal Voronoi tessellation (CVT), which can generate a highly regular distribution of sites with respect to a given density function. A typical CVT-based remeshing method iteratively updates the generator of each Voronoi cell until it coincides with its center of mass. An isotropic mesh is then obtained as the dual graph of the computed CVT. A key step in each iteration in CVT computation is to construct a Voronoi diagram (VD). Although this is fairly simple to do in Euclidean space, doing so in curved domains is expensive due to the lack of a closed-form formula for geodesic distance. To tackle this challenge, some researchers parameterize the input models in $\mathbb{R}^2$ and computed a 2D CVT whose density function compensates for the parameterization distortion. These parameterization based methods can compute intrinsic CVTs on simple meshes of good quality, but they do not work for point clouds, imperfect

1 School of Computer Engineering, Nanyang Technological University, Singapore. E-mail: Y.-S. Leung, ysleung@ntu.edu.sg; X. Wang, wang0684@e.ntu.edu.sg; Y. He, yhe@ntu.edu.sg (✉).

2 Department of Computer Science and Technology, Tsinghua University, China. E-mail: liuyongjin@tsinghua.edu.cn.

3 Department of Mechanical and Automation Engineering, The Chinese University of Hong Kong, Hong Kong, China. E-mail: cwang@mae.cuhk.edu.hk.

Manuscript received: 2015-08-17; accepted: 2015-08-29

meshes, or meshes with complicated topology, where parameterization is non-trivial. A practical alternative is to compute a restricted Voronoi diagram (RVD) [1], which is the intersection of the given model and a CVT defined in $\mathbb{R}^3$. RVD methods are easy to implement and can easily handle models with complicated geometry and topology. Although efficient clipping algorithms (e.g., Ref. [2]) have been proposed, their performance is still too low for time-critical applications. Moreover, current RVD methods assume the input is a manifold mesh, and they do not work for other surface representations, such as scanned point samples.

In this paper, we propose a new RVD-based computational framework for isotropic meshing, with the goal of improving performance as well as robustness of computing RVD while simultaneously maintaining the high quality of the output meshes. Rather than computing CVTs in the entire volume bounded by the input model, our idea is to restrict the computation to a 3D shell of user-controlled thickness. Taking voxels which contain surface samples as sites, we compute the exact Euclidean distance transform on the GPU. Our algorithm is parallel and memory-efficient, and can construct a shell space with resolution 2048^3 at interactive speed. The 3D centroidal Voronoi tessellation and restricted Voronoi diagrams are also computed efficiently on the GPU. Computational results show that our GPU-enabled isotropic meshing algorithm produces results comparable to state-of-the-art techniques, but runs significantly faster than conventional CPU-based implementations. It is also worth noting that our method can process various representations, such as implicit surfaces, polyhedral surfaces, and point clouds, in a unified framework. Moreover, since the shell space can bridge holes and gaps smaller than a certain tolerance, and also tolerate non-manifold edges and degeneracies, our algorithm works well on imperfect meshes with such defects, for which conventional *remeshing* methods often fail. See Fig. 1.

This paper makes the following contributions:

- We give a *unified* framework for constructing isotropic meshes from point clouds, polyhedral surfaces, and implicit surfaces via voxel representation. It avoids computationally expensive components often used in existing methods, such as data fitting, isosurface extraction, and geodesic distance computation.
- Our framework produces a topologically consistent shell under user control, so it can bridge holes and gaps, and tolerate noise, to some degree. It also works well for models with non-manifold edges and degenerate triangles.
- We give a fast, memory-efficient algorithm for computing narrow-band distance fields on a GPU. Our implementation on an nVidia Quadro K5000 with 4 GB RAM can compute the *exact* Euclidean distance transform at interactive speed. The CVT, RVD, and the dual Delaunay triangulations are also computed in parallel on the GPU.

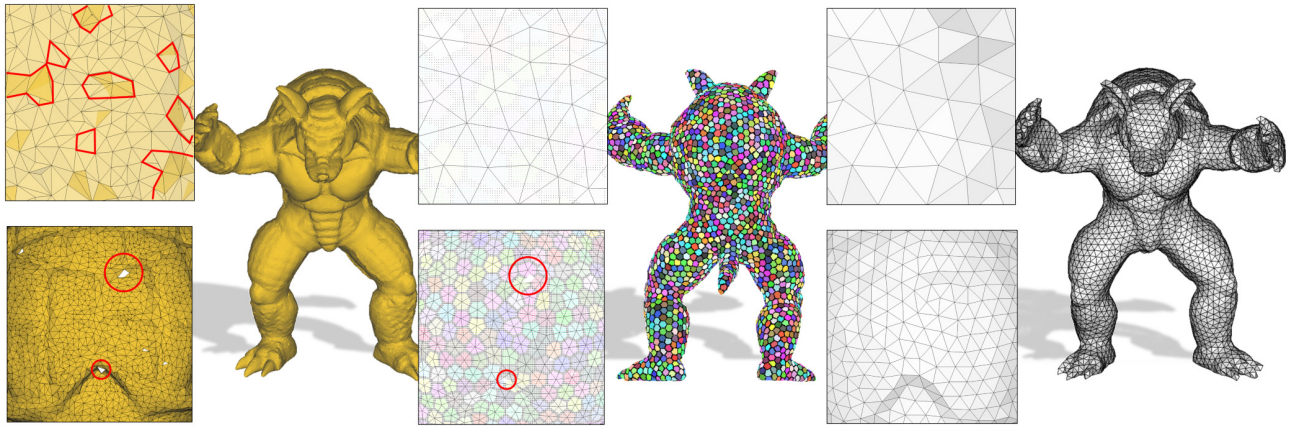


Fig. 1 Isotropic meshing on an imperfect mesh with non-manifold edges, degenerate triangles and holes.

2 Related work

2.1 Geodesic Voronoi diagrams

Although Voronoi diagrams in Euclidean space have been well studied, Voronoi diagrams in 2-manifold meshes (*geodesic* Voronoi diagrams, GVDs) have received little attention. Liu et al. [3] pointed out the fundamental difference between conventional Voronoi diagrams and GVDs, and they pioneered a practical algorithm for constructing a GVD on a triangle mesh. Given a triangle mesh M with n vertices and m ($\leqslant n$) sites on M, their algorithm has a theoretical time complexity $O(n^2 \log n)$ and runs empirically in linear time when m is close to n. Liu and Tang [4] proved that the combinatorial complexity of a GVD is $O(m(g+n))$, where g is the genus of M. Based on a local Voronoi diagram, Xu et al. [5] developed an exact algorithm for computing a polyline-sourced GVD on a triangle mesh.

2.2 Centroidal Voronoi tessellations

A centroidal Voronoi tessellation is a Voronoi diagram whose generating points are the centers of mass of the corresponding Voronoi cells. Thanks to its many favorable geometric properties, the CVT has been used in a wide range of applications [6]. It can be defined by the critical points of the CVT energy function. A popular way to compute a CVT is Lloyd's algorithm [7], which iteratively moves each generator to the corresponding mass center until convergence. Lloyd's method is conceptually simple and easy to implement, but as a gradient-descent method, it only has linear convergence. Liu et al. [8] proved that the CVT energy function has 2nd order smoothness in most situations encountered in computer graphics, so one can achieve quadratic or super-linear convergence by use of Newton or quasi-Newton optimization methods. Rong et al. [9] developed a GPU-based method for computing a CVT in the plane and observed significant speedup over CPU methods.

To compute the CVT on a genus-0 surface, Alliez et al. [10] conformally parameterized the surface to a disk and evaluated the centroids over a weighted density function expressed in $\mathbb{R}^2$. Rong et al. [11] extended the CVT energy function to the spherical $\mathbb{S}^2$ and hyperbolic $\mathbb{H}^2$ domains, and thus can process surfaces of all topological types. Shuai et al. [12] developed a GPU-enabled algorithm for computing a periodic CVT in $\mathbb{H}^2$ and used it to construct isotropic meshes for high-genus surfaces. Recently, Wang et al. [13] proposed an intrinsic method for computing the CVT on an arbitrary manifold triangle mesh. Rather than computing the mass centers of Voronoi cells which requires area integration, their algorithm computes the Riemannian centers using exponential maps. These Riemannian centers provide a good approximation of the mass centers if there are sufficient seeds.

These parameterization-based and exponential map based CVT algorithms are intrinsic and thereby independent of the embedding space. However, they are computationally expensive and impractical for time-critical applications. Rather than computing CVT on surfaces directly, Yan et al. [1] proposed a novel *indirect* method based on computing a *restricted Voronoi diagram*, that is, the intersection between the input mesh and a 3D CVT. Using a kd-tree to quickly identify and compute the intersection of each triangle face with its incident Voronoi cells, their algorithm computes the RVD in $O(n \log m)$ time, where m is the number of seed points and n is the number of triangles of the input mesh. They also adopted a quasi-Newton method to efficiently compute the 3D CVT in the volume bounded by the input mesh.

Lu et al. [14] computed a CVT using line segments and graphs as generators. CVT can also be defined using anisotropic metrics [15] and L_p distances [16].

Chen et al. [17] proposed an iterative method for generating a constrained centroidal Delaunay mesh (CCDM). With local vertex relation, their method does not require geodesic distances and can guarantee that the computed CCDM has the same topology as the input mesh. However, their method cannot handle non-manifold edges and degenerate triangles. It is also unclear whether their method can be extended to point clouds and implicit functions.

Li et al. [18] presented an elegant method for triangulating the conformal uniformization domain via planar Delaunay refinment. They gave explicit estimates for the Hausdorff distance, normal deviation, and differences in curvature measures between the surface and the mesh.

2.3 Distance computation

A key step when intrinsically computing a CVT on a polyhedral surface is to determine geodesic

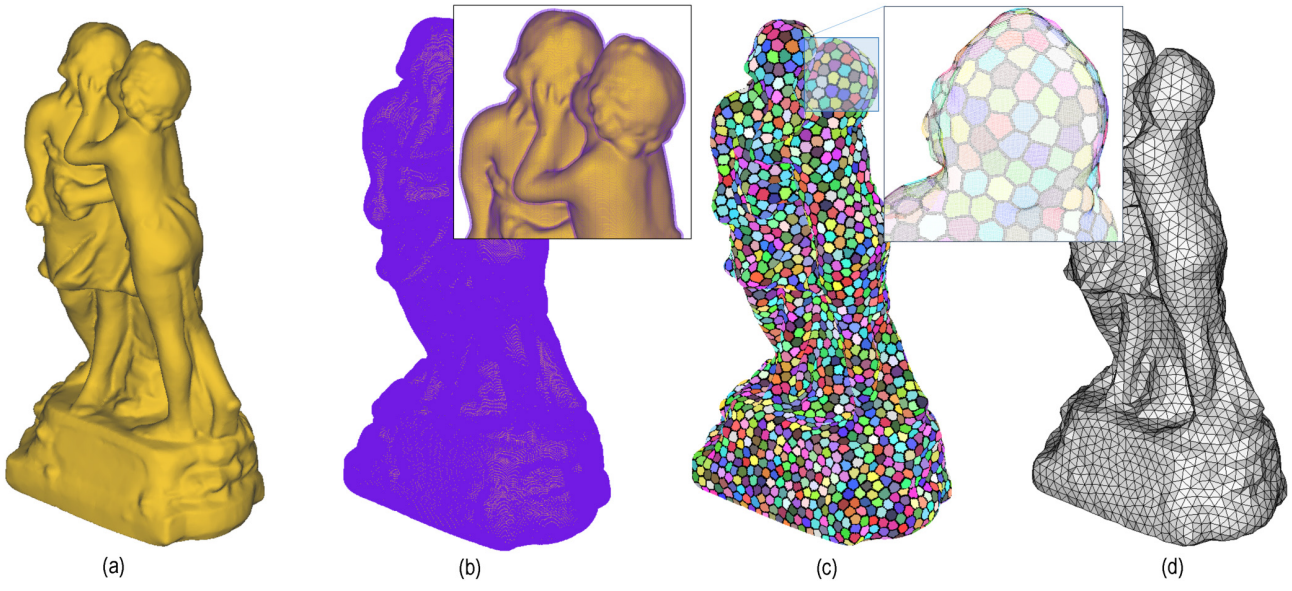


Fig. 2 Overview of our approach using the *Sculpture* model. (a) Input mesh; (b) shell space with $d = 3$; (c) CVT with 3000 seeds; (d) output isotropic mesh.

distances, a fundamental problem in computer graphics and computational geometry. Classical computational geometry approaches include the MMP algorithm [19], the CH algorithm [20], and their many variants (e.g., Refs. [21, 22]). Although these methods can compute the exact solution on arbitrary manifold meshes, they are computationally expensive. Partial differential equation (PDE) approaches, such as the fast marching method [23] and the heat method [24], are efficient and can flexibly handle a wide range of geometric domains, including triangle meshes, point clouds, regular grids and volumes. They can also be easily generalized to an anisotropic setting [25]. However, they only compute a first-order approximation, and thus the results are sensitive to model tessellation and resolution. Motivated by the *local* structure of the discrete geodesic problem, Ying et al. [26] proposed the saddle vertex graph (SVG), a sparse graph which encodes geodesic information on a triangle mesh. Computing a geodesic distance is equivalent to finding a shortest path in this graph. However, the performance of SVG methods depends greatly on the number and distribution of saddle vertices in the input meshes, and it is unclear whether it can be extended to imperfect meshes, implicit surfaces, or point clouds, as it is difficult to determine saddle vertices in such cases.

Since computing geodesic distances is expensive, many applications seek practical ways to give approximate solutions with accuracy control. The most widely used approach is to use a distance field, which encodes the distance from each grid node to the surface of the model (uniformly or adaptively) [27]. An approximate distance field can be efficiently computed by a distance-transformation method, e.g., based on the *chamfer distance transformation* (CDT) [28] or the *vector-city vector distance transform* (VCVDT) [29].

Recent work has focused on the *Euclidean distance transformation* (EDT). Given a d-dimensional grid with $N = n^d$ grid points, where S grid points are colored and the other $N - S$ grid points are not colored, for each grid point, the EDT aims to compute the Euclidean distance to the closest colored grid point. However, this process is typically computationally intensive in 3 or more dimensions. GPU-based algorithms have been developed to provide an efficient solution [30]. Cao et al. [31] presented an efficient algorithm, the *parallel banding algorithm* (PBA), to compute the exact EDT on a GPU. Their algorithm, however, can quickly exhaust graphics memory: the highest resolution that can be achieved by PBA is 512^3 on a cutting-edge graphics card, which does not provide sufficient accuracy to guarantee the correctness of the constructed CVT for many real-world models. The new algorithm proposed in this paper is more sophisticated in

its memory usage and can achieve 3D EDT for resolutions up to 2048^3.

2.4 Surface reconstruction and meshing from points

Constructing a high-quality mesh from unorganized point samples plays an important role in computer graphics. Although numerous techniques are available, all tackle the problem in two *separate* stages, that is, constructing an initial surface first, and then improving the mesh quality using the remeshing techniques. Existing surface reconstruction methods are based on either computational geometry or implicit functions.

Computational geometry approaches aim to generate a piecewise linear surface, topologically equivalent to the sampled surface and also geometrically close. Representative work includes *crust* [32], *cocone* [33], and their many variants. These algorithms provide theoretical guarantees if the input points are densely sampled. However, in practice, this condition may not hold.

In presence of noise and outliers, which is typical for samples obtained by scanning, a common approach is to fit the samples using the zero set of an implicit function. A popular method for *oriented* point samples is *Poisson surface reconstruction* [34]; an implicit surface is first generated and tessellated into a polygonal mesh later. Although tessellation step can be computed on the GPU, fitting an implicit surface uses time-consuming global operators. Sheung and Wang [35] presented a robust mesh reconstruction method for *unoriented* noisy points. An approximate CVT is computed on the point set to down-sample the input point cloud to a smaller number of pointa. Then, a Voronoi diagram based mesh generation method, tight cocone [36], is employed to generate the mesh. As VD-based surface reconstruction is memory intensive, only datasets with a modest number of points can be handled. Moreover, it is hard to parallelize the computation.

Our proposed method is different from existing ones in that it performs surface reconstruction and meshing in an integrated manner. Our method is more efficient and elegant since it can completely bypass implicit fitting as well as isosurface extraction, making it an ideal tool for processing scanned point samples.

3 Algorithm

Let O denote the input 3D mesh. We first construct a voxel representation M of O at a given resolution res. Then we construct a shell space $\bar{P}$ consisting of off-surface points, where each point in $\bar{P}$ has a distance $d_p \leqslant d$ to its closest point on M (see Fig. 2). The threshold d is specified by the user and is model-dependent.

Our isotropic meshing algorithm adopts Lloyd's framework. Starting with k randomly generated seeds, it minimizes the CVT energy by iteratively updating the seed positions. In each iteration, it computes a Voronoi diagram confined to the shell space and moves the seeds toward the corresponding mass centers. The algorithm projects the seeds back to $\bar{P}$ if they move outside the shell space. After convergence, it propagates the seed information in the shell space to determine connected Voronoi cells and extracts the dual Delaunay triangulation. Our method is outlined in Algorithm 1, and described in detail in the following.

Algorithm 1: Isotropic meshing based on EDT

Input: 3D surface O, voxel resolution res, shell space thickness d, convergence threshold ϵ, and number of seeds k.

Output: Isotropic mesh with k vertices.

$S \leftarrow k$ random seeds
$M \leftarrow$ Voxelization(O, res)
$\bar{P} \leftarrow$ ShellSpaceConstruction(M, res, d)
while convergence not reached **do**
  $V_k \leftarrow$ SearchClosestSeedInShell($\bar{P}$, S)
  $C_k \leftarrow$ CenterMass(V_k)
  $\bar{C}_k \leftarrow$ UpdateSeed($\bar{P}$, C_k)
  $S \leftarrow \bar{C}$
end while
$V_k \leftarrow$ ShellFlooding($\bar{P}$, S)
return DualTriangulation(V_k)

3.1 Memory-efficient shell space construction

We introduce a memory-efficient way to construct shell spaces in real-time. We extend the parallel banding algorithm (PBA) of Cao et al. [31] to compute the Euclidean distance transform (EDT) in a narrow-band. Their algorithm partitions the input domain into small chunks of equal size, which can be processed in parallel. The results are then merged

concurrently. Although their method is exact and efficient, it is impractical for large models due to its memory requirements. Our method overcomes this problem by on-the-fly computation and integrating fast bitmap indexing technology on the GPU. We explain the principle in two dimensions for simplicity; the idea easily extends to three dimensions.

The input image is divided into a virtual grid made up of occupied and unoccupied pixels; the former pixels, the *sites* $\in S$, are run-length encoded. Every pixel undergoes a two-step process: (i) find the nearest site S_{ij}, among all sites in row j; (ii) determine the closest site, among all the nearest sites, in the current column i. During the first step we assign a thread to process each row as it is more efficient to do more computation in a single thread than to repeatedly access global memory with multiple threads. The second step extends the divide-and-merge approach of PBA, by employing warp-vote and warp-shuffle functions in NVIDIA's CUDA to exchange information between nearest sites within a chunk. (A warp is a pool of threads that executes in parallel.) Every thread in the same warp does the same calculation, avoiding warp variations that often compromise performance.

Figure 3 shows an example. When the threads in a warp reach column i, each row computes the corresponding nearest sites S_{ij}. The nearest site of the current pixel is shown in blue. Note that only some sites (empty blue dots) satisfy the distance constraint. Therefore, sites S_{i1} of thread 1 can be safely discarded. We set a barrier to ensure that every thread obtains some sites before exchanging information. After synchronization, we sweep each S_{ij} to other threads in the same warp to update the closest site of the current pixel (i, j), based on the distance function d_{ij}. A bitmap stores a Boolean value for every pixel; 0 indicates that the

```
// For simplicity, we give a 2D version here
function ShellSpaceConstruction(M, res, d)
for all thread j = 0 to res in parallel do
  for i = 0 to res do
    S_ij ← GetNearestSite(M, j, d)
    discard S_ij if ‖ d_ij ‖> d
    set barrier // Ensure every thread gets S_ij
    // Compare with other threads in same warp
    // warp size = h, current warp id = k
    C_k ← S_ij
    for x = 0 to h do
      if IsCloser(C_k, C_x) then
        C_k ← C_x
        id ← x
      end if
    end for
    // Mark the closest site of pixel (i, j)
    Bitmap[i][id] = true
  end for
end for
Collect and merge the closest sites if Bitmap[i][j] = true
return P̄ // return pixels located inside the shell
```

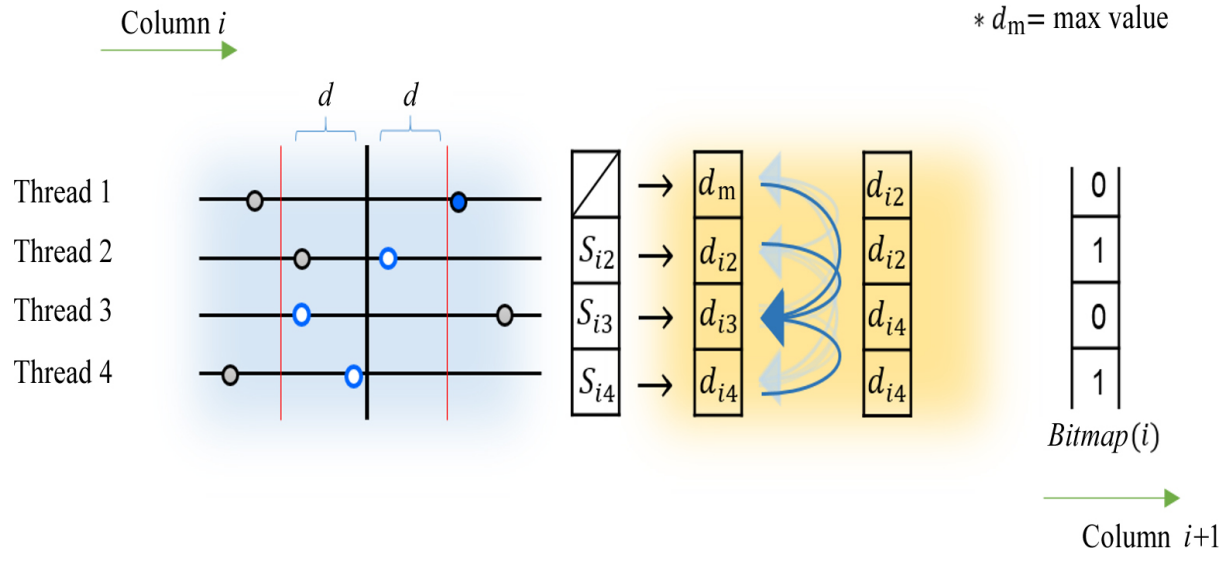


Fig. 3 Illustrative example of distance field computation in a narrow band. See the text for an explanation and function *ShellSpaceConstruction* for the pseudo code.

corresponding nearest sites *cannot* be the closest sites of the current column. Then the threads repeat the same procedure for the next column $i+1$, until they reach the last column. In the final step we collect the closest sites with value 1 in different chunks and merge them to find the pixels that form the shell region. As the nearest sites are computed on-the-fly and the temporary result only needs indexing by a bitmap, our algorithm requires less memory than the PBA method.

3.2 Constructing 3D Voronoi diagrams in shell space

The shell space represented by $\bar{P}$ is used to constrain construction of the Voronoi diagram and update the positions of seeds. Initially, the seeds $S = \{s_1, \ldots, s_k\}$ are located on the input mesh. We collect points $\in \bar{P}$ that share the same closest seeds to build the Voronoi diagram.

We perform a proximity search to find the closest seed for all query points from $\bar{P}$. To speed up the process, the seeds are projected onto a uniform grid G with smaller resolution of res (e.g., 32^3), so that, for a query point q, it just find seeds in the grid cell G_q in which q falls. If any border of the grid cell is closer to q than the seed found in G_q, query point q considers the neighboring cell of G_q.

3.3 Computing the CVT

Updating the seeds' positions towards a uniform distribution is crucial for constructing the CVT. Optimal positions can be reached by minimizing the following energy function:

$$E(S) = \sum_{i=1}^{m} \int_{V_i} \rho(p)\|p - s_i\|^2 \mathrm{d}p$$

where V_i is the Voronoi cell of seed s_i, $p \in \bar{P}$, and ρ is a non-negative user-defined density function.

In Lloyd's algorithm, a seed s_i moves iteratively toward the corresponding mass center c_i of Voronoi diagram V_i until convergence. However, the mass center could be located far from the surface, as it is constructed in the shell space. We determine the following new position to replace the mass center in each iteration:

$$\bar{c}_i = s_i + u\frac{\overrightarrow{s_i c_i}}{\|\overrightarrow{s_i c_i}\|}$$

where $u \in \mathbb{R}^+$ is the magnitude of movement of the seeds. We observe that if the seeds move with different magnitudes, the area of CVT calls will largely vary depending on surface curvature. In order to guarantee topological consistency, the new center is projected back onto M if it moves outside the shell space, as shown in Fig. 4.

3.4 Computing the dual triangulations

Upon convergence, all the generators are uniformly distributed. We next extract the dual Delaunay triangulation.

First, we find the *direct* neighbors of all seeds, where direct means two voxels from their Voronoi cells are connected. Adjacent neighbors can be found by flooding the information from all seeds to all voxels in the shell $\in \bar{P}$. Each voxel stores a hash table to hold the locations of its 26 neighbors. Each propagation step updates the current seed information to neighboring voxels until all voxels have been reached. This approach avoids producing the wrong connectivity for seeds that are geometrically close, but topologically far from each other. We then organize direct neighbors in clockwise

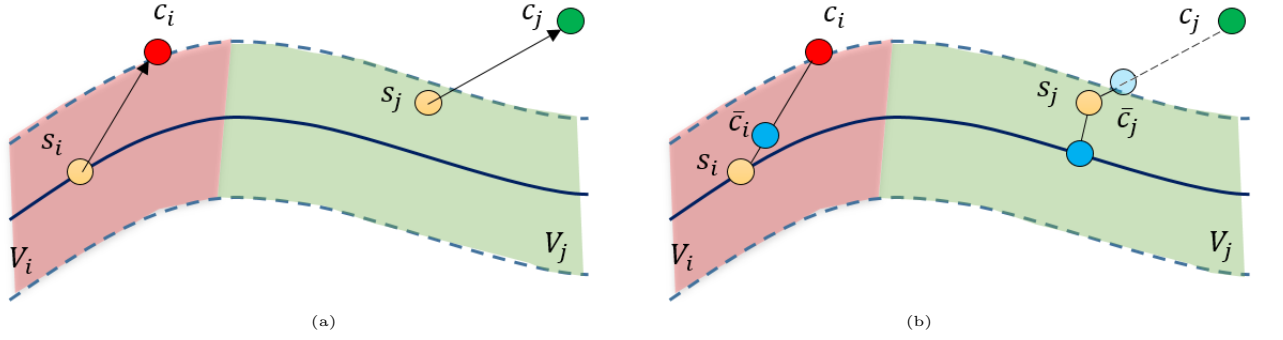


Fig. 4 Illustration of the update process in an iteration for two seeds. (a) Yellow dots: seeds of Voronoi cells V_i (in red) and V_j (in green). Red and green dots: their respective mass centers. (b) The seeds move along vector $\overrightarrow{sc}$ to new centers (blue dots). We project $\bar{c}_j$ (light blue dot) to the surface since it is outside the shell region.

order and finally extract the triangle mesh.

4 Experimental results

All tests were performed on a PC with an Intel Xeon E5 2.5 GHz CPU and an nVidia Quadro K5000 GPU with 4 GB RAM.

4.1 Narrow-banded distance fields

Table 1 lists the time and peak memory requirements for varying parameter d and res (see Fig. 5). Clearly, as d increases, the time increases insignificantly compared to the increased number of nearest sites in the shell. This is due to the low cost of intra-warp communication and the reduced warp divergence in our algorithm. Also, the memory consumption is remarkably small, considering the scene is represented by a high resolution uniform grid. Previous algorithms (e.g., Ref. [31]) typically require at least 10 times more memory than ours.

Table 2 compares our algorithm with PBA on the Sculpture model at resolution 512^3. Since PBA computes a full distance map, its performance is independent of distance d. Our algorithm consumes significantly less memory and runs much faster than PBA for a reasonably small d.

Table 1 Performance of our algorithm for different d at resolutions 1024^3 and 2048^3

Model	d	Memory (MB)	Time (s)	Memory (GB)	Time (s)
Dinosaur	1	149	1.186	1.18	12.3
	3	174	1.239	1.23	13.1
	6	206	1.313	1.30	13.3
	9	235	1.383	1.36	13.5

Table 2 Comparison of our algorithm to PBA at resolution 512^3

Model	Memory (MB)	d	Time (s)
Sculpture (PBA)	1073	—	0.310
Sculpture (ours)	26.6	1	0.147 (×2.1)
	33.7	3	0.173 (×1.8)
	49.7	6	0.200 (×1.6)
	66.1	15	0.286 (×1.1)
	76.2	20	0.339 (×0.9)

4.2 CVT computation

Following Ref. [13], we adopted the following criteria to measure the triangle mesh quality: (i) triangle quality $Q(t)$ defined by $6P_t/\sqrt{3}H_t$, where P_t and H_t are the inradius and the length of the longest edge of triangle t respectively, (ii) the smallest angle $\theta_{\min}$ and the average θ_{avg} minimal angle for all triangles,

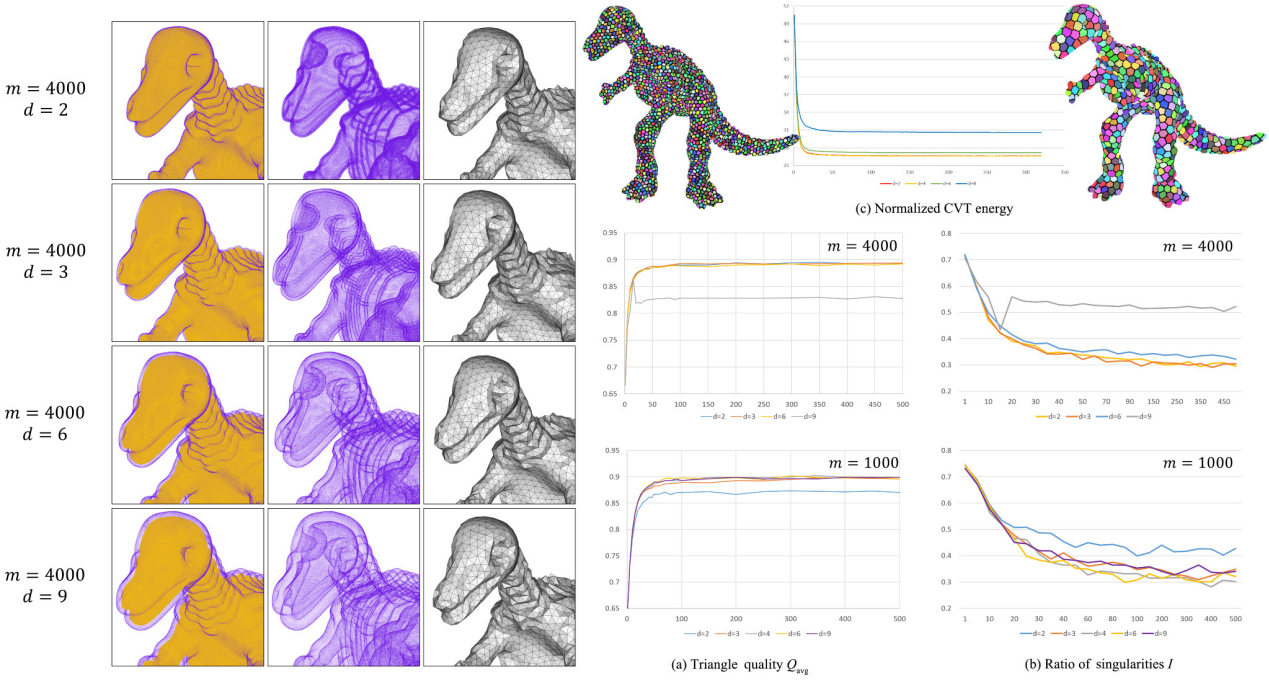


Fig. 5 Mesh quality for various shell space parameter values d. (a) Triangulation quality measure. (b) Singularity ratio. (c) Our GPU-based Lloyd algorithm usually converges in 100–200 iterations; d has little impact on the convergence rate. Horizontal axis: iteration number. Vertical axis: normalized CVT energy function.

(iii) the ratio of singularities, defined by v_s/k, where v_s is the number of non-6-valent vertices and k is the number of vertices.

Flexible inputs. One advantage of our algorithm is that the CVT construction is independent of input topology, thus allowing a wide range of input types such as point clouds, implicit surfaces, and meshes with non-manifold surfaces. Figure 6 shows that our method can be applied to an irregular point cloud. However, the uneven density of points causes failure during triangulation, i.e., holes and a non-manifold mesh. The shell space can help to resolve this issue because it covers the uneven space where points cannot be reached. Figure 7 investigates the influence of shell distance on irregular data sets. It shows that the geometric shape and density of the point cloud directly affect the choice of the shell distance d.

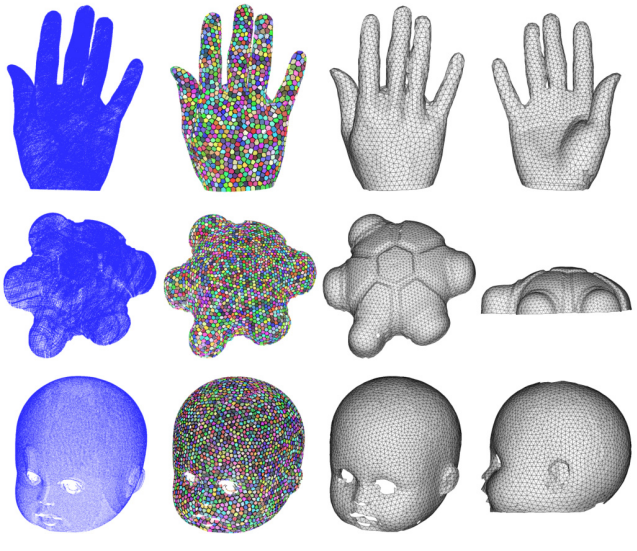


Fig. 6 Experimental results on scanned point samples.

We have also successfully applied our algorithm to implicit surfaces (see Fig. 8) and other implicit representations (see Fig. 9). More examples can be found in Fig. 12.

Figure 1 shows the result from an imperfect mesh with non-manifold edges, degenerate triangles and holes. Thanks to its defect-tolerant nature, our method can handle these issues easily.

Accuracy and efficiency control. We allow the user to balance accuracy and efficiency by choice of the offset d. Figure 5 shows the relationship between the distance d, the number of generators, and the quality of the remeshed surface. As the offset increases to 9, with 4000 generators, the mesh quality of the dinosaur model dramatically drops. This also happens when the offset decreases to 2 with 1000 generators. Figure 5(b) clearly illustrates the difference in quality for different values of d. Along with other examples in Table 3, we can see that the mesh has best quality with offset distance in the range 2–6. Figure 10 compares our method with two parameterization-free isotropic meshing methods, an *intrinsic* CVT method [13] and an *extrinsic* RVD method [1].

Thanks to its GPU-friendly structure and the computational power of modern GPUs, our method runs significantly faster than their CPU-based implementations.

Topology consistency. The shell space also guarantees topological consistency between the input and the remeshed surface. Our algorithm constrains the seeds to lie within a shell of width $2d$ during the update process, so the output is topologically consistent with respect to the shell. Figure 11 illustrates the importance of this offset volume. The

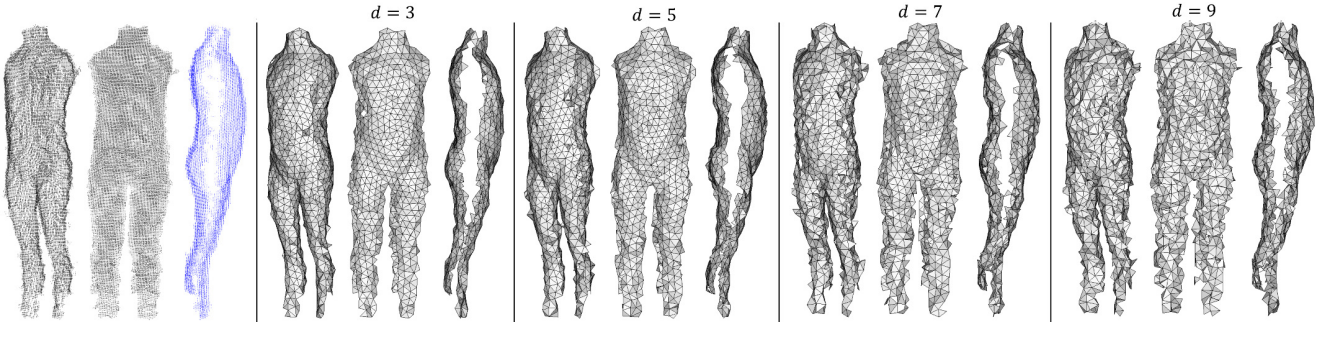


Fig. 7 Isotropic meshing on noisy scanned point samples. We observe that $d \in [3, 5]$ helps to bridge holes and tolerate noise, producing fairly good results. However, if d exceeds that range, there is a large error in the computed CVT, leading to poor results.

Table 3 Model complexity and runtime performance. SS: time (in seconds) for shell construction; m: the number of seeds; T: average time for each Lloyd iteration; n: the number of iterations; I: singularity ratio

Model	d	SS (s)	Number of sites	m	T (s)	n	I	$Q_{\min}$	Q_{avg}	$\theta_{\min}(^\circ)$	$\theta_{\text{avg}}(^\circ)$	Total time (s)
Sculpture	3	0.966	1.04×10^6	3000	0.064	100	0.25	0.639	0.907	36.1	52.4	7.2
Heptoroid	2	1.429	2.63×10^6	9000	0.138	120	0.24	0.624	0.902	35.3	51.9	19.5
Helix	2	1.212	4.4×10^5	4000	0.036	100	0.29	0.589	0.889	33.4	50.9	5.14
Pegaso	2	0.948	1.33×10^6	8000	0.063	150	0.26	0.600	0.894	31.5	51.3	10.6
Dinosaur	2	0.880	5.5×10^5	4000	0.025	160	0.28	0.636	0.894	30.2	51.2	4.8
	4	0.913	5.5×10^5	1000	0.046	170	0.31	0.602	0.900	30.6	51.8	9.2
Armadillo	2	0.944	1.26×10^6	4000	0.051	200	0.26	0.613	0.902	30.3	51.9	11.3
Mask	3	0.897	6.3×10^5	2000	0.042	170	0.28	0.613	0.901	33.4	51.9	8.2
Trung hand (points)	2	0.949	1.17×10^6	2000	0.042	100	0.32	0.198	0.895	12.8	51.5	5.2
Turtle toy (points)	4	0.946	8.2×10^5	2000	0.077	120	0.22	0.373	0.911	23.5	52.6	10.8
Baby head (points)	5	0.26	2.8×10^5	3000	0.069	120	0.29	0.279	0.891	16.9	51.1	9.34

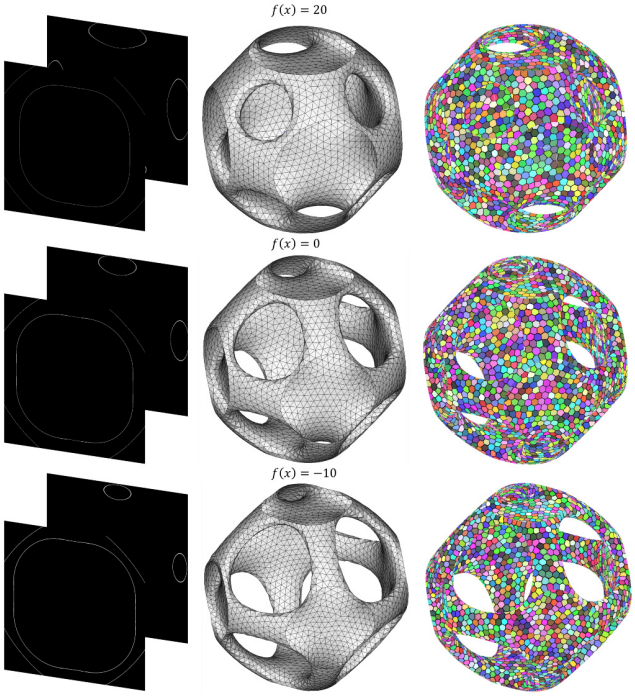


Fig. 8 Generating isotropic meshes from an implicit function. There are 3000 seeds and d=3.

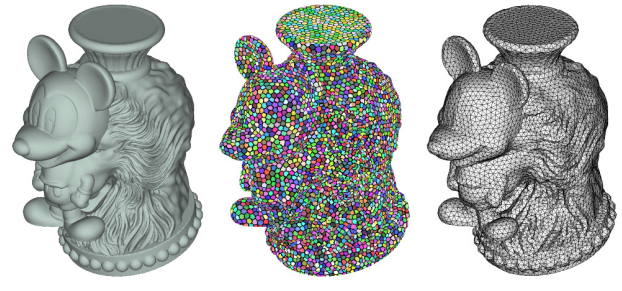


Fig. 9 Generating isotropic meshes with 8000 seeds and d=2 from an implicit representation: layered depth normal image [37].

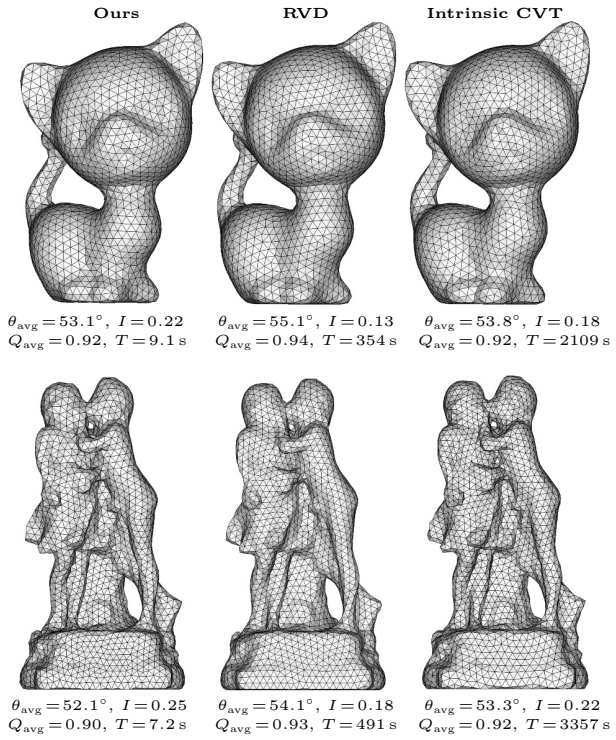


Fig. 10 Comparison with the RVD method [1] and the intrinsic CVT method [13].

bottle model consists of an inner tube which is very close to the outer surface, close enough that only a shell of width 2 ($d = 1$) can separate these two parts when the resolution is 1024^3. However, width 2 is so thin that the seeds can barely move inside, leading to poor remeshing quality ($Q_{\text{avg}} = 0.86$, $Q_{\min} = 0.10$). A possible solution would be to increase the grid size. Since our approach is scalable, this could be improved by using a graphic card with larger memory.

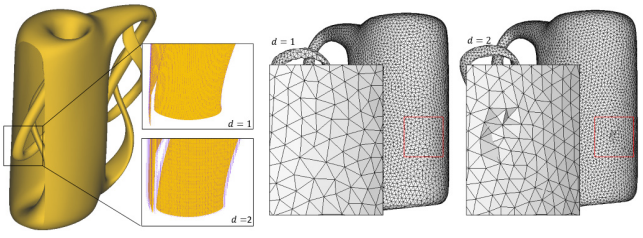


Fig. 11 Remeshing the genus-2 Knotty Bottle model with 9000 seeds. As its inner tube is very close to the outer surface, $d = 1$ is needed (at resolution 1024^3) to produce a shell space with the same topology as the mesh, whereas $d = 2$ does not give such a guarantee.

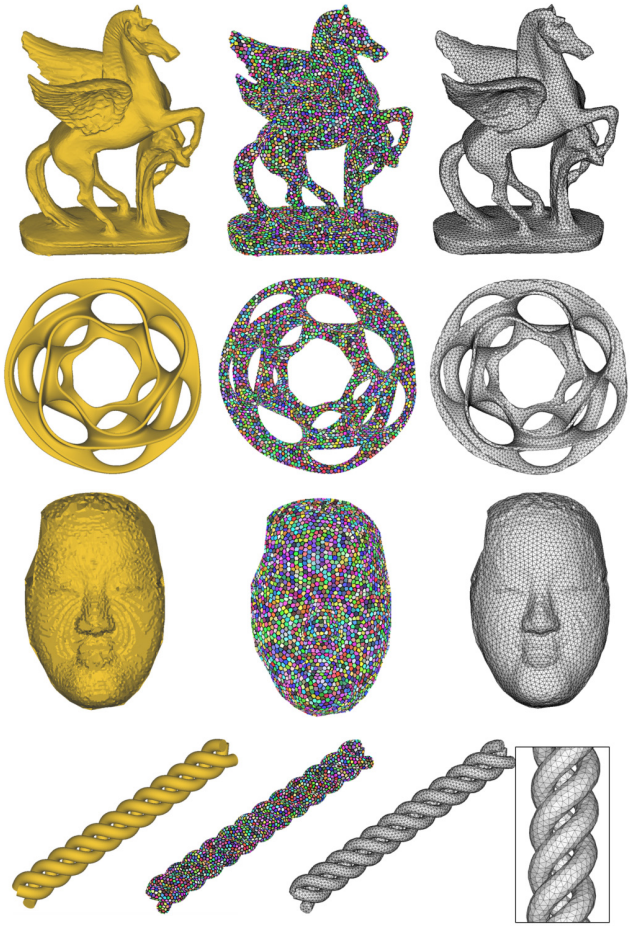


Fig. 12 Experimental results. Images are rendered at high-resolution, allowing zooming in for examination.

5 Conclusions

This paper has presented a robust, efficient method for constructing isotropic meshes using the Euclidean distance transform. Our algorithm constructs a narrow band enclosing the input surface, in which 3D centroidal Voronoi tessellations and restricted Voronoi diagrams are computed. Our algorithm is fully parallel and memory-efficient, and can construct the shell space with resolution up to 2048^3 at interactive speed. Moreover, our method can process implicit surfaces, polyhedral surfaces and point clouds in a unified framework. Computational results show that our GPU-friendly isotropic meshing algorithm produces results comparable to state-of-the-art techniques, but runs significantly faster than conventional CPU-based implementations.

Our current implementation adopts a constant resolution to construct the shell space. This, however, is not optimal, since it is pessimistic for the regions with fairly flat geometry, and it may be inadequate for the highly-curved regions. In the future, we aim to develop a geometry-aware algorithm for constructing in parallel the shell space with adaptive resolution.

Acknowledgements

The NTU authors are partially supported by AcRF RG40/12 and MOE2013-T2-2-011. Y.-J. Liu is partially supported by National Natural Science Foundation of China (Nos. 61432003 and 61322206) and the TNList Cross-discipline Foundation. C. C. L. Wang is partially supported by HKSAR Research Grants Council (RGC) General Research Fund (GRF), CUHK/14207414.

References

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

[2] Yan, D.; Wang, W.; Lévy, B.; Liu, Y. Efficient computation of clipped Voronoi diagram for mesh generation. *Computer-Aided Design* Vol. 45, No. 4, 843–852, 2013.

[3] Liu, Y. J.; Chen, Z.; Tang, K. Construction of iso-contours, bisectors, and Voronoi diagrams on triangulated surfaces. *IEEE Transactions on Pattern*

Analysis and Machine Intelligence Vol. 33, No. 8, 1502–1517, 2011.

[4] Liu, Y.-J.; Tang, K. The complexity of geodesic Voronoi diagrams on triangulated 2-manifold surfaces. *Information Processing Letters* Vol. 113, No. 4, 132–136, 2013.

[5] Xu, C.; Liu, Y.-J.; Sun, Q.; Li, J.; He, Y. Polyline-sourced geodesic Voronoi diagrams on triangle meshes. *Computer Graphics Forum* Vol. 33, No. 7, 161–170, 2014.

[6] Du, Q.; Gunzburger, M. D.; Ju, L. Constrained centroidal Voronoi tessellations for surfaces. *SIAM Journal on Scientific Computing* Vol. 24, No. 5, 1488–1506, 2002.

[7] Lloyd, S. Least squares quantization in PCM. *IEEE Transactions on Information Theory* Vol. 28, No. 2, 129–137, 1982.

[8] Liu, Y.; Wang, W.; Lévy, B.; Sun, F.; Yan, D.-M.; Lu, L.; Yang, C. On centroidal Voronoi tessellation—Energy smoothness and fast computation. *ACM Transactions on Graphics* Vol. 28, No. 4, Article No. 101, 2009.

[9] Rong, G.; Liu, Y.; Wang, W.; Yin, X.; Gu, X. D.; Guo, X. GPU-assisted computation of centroidal Voronoi tessellation. *IEEE Transactions on Visualization and Computer Graphics* Vol. 17, No. 3, 345–356, 2011.

[10] Alliez, P.; de Verdière, É. C.; Devillers, O.; Isenburg, M. Isotropic surface remeshing. In: Proceedings of the Shape Modeling International, 49, 2003.

[11] Rong, G.; Jin, M.; Shuai, L.; Guo, X. Centroidal Voronoi tessellation in universal covering space of manifold surfaces. *Computer Aided Geometric Design* Vol. 28, No. 8, 475–496, 2011.

[12] Shuai, L.; Guo, X.; Jin, M. GPU-based computation of discrete periodic centroidal Voronoi tessellation in hyperbolic space. *Computer-Aided Design* Vol. 45, No. 2, 463–472, 2013.

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

[14] Lu, L.; Lévy, B.; Wang, W. Centroidal Voronoi tessellation of line segments and graphs. *Computer Graphics Forum* Vol. 31, No. 2, 775–784, 2012.

[15] Lévy, B.; Bonneel, N. Variational anisotropic surface meshing with Voronoi parallel linear enumeration. In: Proceedings of the 21st International Meshing Roundtable, 349–366, 2013.

[16] Lévy, B.; Liu, Y. L_p centroidal Voronoi tessellation and its applications. *ACM Transactions on Graphics* Vol. 29, No. 4, Article No. 119, 2010.

[17] Chen, Z.; Cao, J.; Wang, W. Isotropic surface remeshing using constrained centroidal Delaunay mesh. *Computer Graphics Forum* Vol. 31, No. 7, 2077–2085, 2012.

[18] Li, H.; Zeng, W.; Morvan, J. M.; Chen, L.; Gu, X. Surface meshing with curvature convergence. *IEEE Transactions on Visualization and Computer Graphics* Vol. 20, No. 6, 919–934, 2014.

[19] Mitchell, J. S. B.; Mount, D. M.; Papadimitriou, C. H. The discrete geodesic problem. *SIAM Journal on Computing* Vol. 16, No. 4, 647–668, 1987.

[20] Chen, J.; Han, Y. Shortest paths on a polyhedron. In: Proceedings of the 6th Annual Symposium on Computational Geometry, 360–369, 1990.

[21] Xu, C.; Wang, T.; Liu, Y.; Liu, L.; He, Y. Fast wavefront propagation (FWP) for computing exact discrete geodesics on meshes. *IEEE Transactions on Visualization and Computer Graphics* Vol. 21, No. 7, 822–834, 2015.

[22] Ying, X.; Xin, S.-Q.; He, Y. Parallel chen-han (PCH) algorithm for discrete geodesics. *ACM Transactions on Graphics* Vol. 33, No. 1, Article No. 9, 2014.

[23] Kimmel, R.; Sethian, J. A. Computing geodesic paths on manifolds. *Proceedings of the National Academy of Sciences of the United States of America* Vol. 95, 8431–8435, 1998.

[24] Crane, K.; Weischedel, C.; Wardetzky, M. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Transactions on Graphics* Vol. 32, No. 5, Article No. 152, 2013.

[25] Campen, M.; Heistermann, M.; Kobbelt, L. Practical anisotropic geodesy. *Computer Graphics Forum* Vol. 32, No. 5, 63–71, 2013.

[26] Ying, X.; Wang, X.; He, Y. Saddle vertex graph (SVG): A novel solution to the discrete geodesic problem. *ACM Transactions on Graphics* Vol. 32, No. 6, Article No. 170, 2013.

[27] Jones, M. W.; Baerentzen, J. A.; Sramek, M. 3D distance fields: A survey of techniques and applications. *IEEE Transactions on Visualization and Computer Graphics* Vol. 12, No. 4, 581–599, 2006.

[28] Marchand-Maillet, S.; Sharaiha, Y. M. Euclidean ordering via chamfer distance calculations. *Computer Vision and Image Understanding* Vol. 73, No. 3, 404–413, 1999.

[29] Satherley, R.; Jones, M. W. Vector-city vector distance transform. *Computer Vision and Image Understanding* Vol. 82, No. 3, 238–254, 2001.

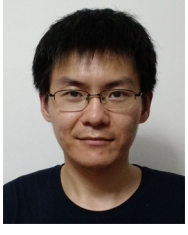
[30] Hoff III, K. E.; Keyser, J.; Lin, M.; Manocha, D.; Culver, T. Fast computation of generalized Voronoi diagrams using graphics hardware. In: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, 277–286, 1999.

[31] Cao, T.-T.; Tang, K.; Mohamed, A.; Tan, T.-S. Parallel banding algorithm to compute exact distance transform with the GPU. In: Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, 83–90, 2010.

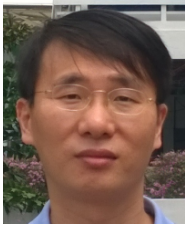
[32] Amenta, N.; Bern, M.; Kamvysselis, M. A new Voronoi-based surface reconstruction algorithm. In: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, 415–421, 1998.

[33] Amenta, N.; Choi, S.; Dey, T. K.; Leekha, N. A simple algorithm for homeomorphic surface reconstruction. In: Proceedings of the 16th Annual Symposium on Computational Geometry, 213–222, 2000.
[34] Kazhdan, M. M.; Bolitho, M.; Hoppe, H. Poisson surface reconstruction. In: Proceedings of the 4th Eurographics Symposium on Geometry Processing, 61–70, 2006.
[35] Sheung, H.; Wang, C. C. L. Robust mesh reconstruction from unoriented noisy points. In: Proceedings of SIAM/ACM Joint Conference on Geometric and Physical Modeling, 13–24, 2009.
[36] Dey, T. K.; Goswami, S. Tight cocone: A water-tight surface reconstructor. In: Proceedings of the 8th ACM Symposium on Solid Modeling and Applications, 127–134, 2003.
[37] Wang, C. C. L.; Leung, Y.-S.; Chen, Y. Solid modeling of polyhedral objects by layered depth-normal images on the GPU. *Computer-Aided Design* Vol. 42, No. 6, 535–544, 2010.

Yuen-Shan Leung received her B.Eng. degree and her Ph.D. degree from the Chinese University of Hong Kong. She is currently a research fellow at the School of Computer Engineering, Nanyang Technological University, Singapore. Her research interests include geometric & solid modeling, computer-aided design & manufacturing, and computer graphics.

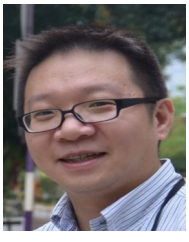


Xiaoning Wang received the B.Eng. degree from Tianjin University, China, in 2011. He is currently a Ph.D. candidate in the School of Computer Engineering, Nanyang Technological Univeristy, Singapore. His research interests are geometry analysis and computer graphics.



Ying He is currently an associate professor at the School of Computer Engineering, Nanyang Technological University, Singapore. He received the B.S. and M.S. degrees in electrical engineering from Tsinghua University, China, and the Ph.D. degree in computer science from Stony Brook University, USA. His research interests fall into the general area of visual computing, and he is particularly interested in the problems which require geometric analysis and computation. For more information, please visit http://www.ntu.edu.sg/home/yhe.

Yong-Jin Liu received his B.Eng. degree from Tianjin University, China, in 1998, and his M.Phil. and Ph.D. degrees from Hong Kong University of Science and Technology (HKUST), Hong Kong, China, in 2000 and 2004, respectively. He is now an associate professor at the Tsinghua National Laboratory for Information Science and Technology, the Department of Computer Science and Technology, Tsinghua University, China. His research interests include computational geometry, multimedia, computer graphics, and computer-aided design. For more information, please visit http://cg.cs.tsinghua.edu.cn/people/∼Yongjin/yongjin.htm.



Charlie C. L. Wang is a Fellow of the American Society of Mechanical Engineers (ASME) with expertise in geometric computing, design, and manufacturing. He received his Ph.D. in mechanical engineering from HKUST in 2002. After that, he joined the Chinese University of Hong Kong in 2003, and is now a full professor of mechanical and automation engineering. His research interests include geometric computing, computer-aided design, advanced manufacturing, and computational physics.