Pacific Graphics (2015)
N. J. Mitra, J. Stam, and K. Xu (Editors)

Short Paper

Robust and GPU-friendly Isotropic Meshing Based on Narrow-banded Euclidean Distance Transformation

Y.S. Leung[1] , X. Wang[1] , Y. He[1] , Y.J. Liu[2] , C.C.L. Wang[3]

[1]Nanyang Technological University [2]Tsinghua University [3]The Chinese University of Hong Kong

Abstract

In this paper, we propose a simple-yet-effective method for isotropic meshing via Euclidean distance transformation based Centroidal Voronoi Tessellation (CVT). The proposed approach aims at improving the performance as well as robustness of computing CVT on curved domains while simultaneously maintaining the high-quality of the output meshes. In contrast to the conventional extrinsic methods which compute CVTs in the entire volume bounded by the input model, our idea is to restrict the computation in a 3D shell space with user-controlled thickness. Taking the voxels which contain the surface samples as the sites, we compute the exact Euclidean distance transform on the GPU. Our algorithm is fully parallel and memory-efficient, and it can construct the shell space with resolution up to 2048^3 *at interactive speed. Since the shell space is able to bridge holes and gaps up to a certain tolerance, and tolerate non-manifold edges and degenerate triangles, our algorithm works well on models with such defects, whereas the conventional remeshing methods often fail.*

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 not ready for subsequent geometric processing, since they may contain holes, gaps, noise, degenerate triangles and non-manifold edges.

A popular approach to improve the mesh quality is via 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. Then the isotropic mesh is obtained by the dual graph of the computed CVT. A key step in CVT computation is to construct Voronoi diagrams (VD) in each iteration. Although it is fairly simple to construct VD in Euclidean spaces, computing VD on curved domains is expensive due to lack of closed-form formula of geodesic distance. A practical way is to compute the restricted Voronoi diagrams (RVD) [YLL*09], which is the intersection between the given model and a CVT defined in $\mathbb{R}^3$.

In this paper, we propose a new RVD-based computational framework for isotropic meshing, aiming at improving the performance as well as robustness of computing RVD while simultaneously maintaining the high-quality of output meshes. Rather than computing CVTs in the entire volume bounded by the input model, our idea is to restrict the computation in a 3D shell space with user-controlled thickness. Since the shell space is able to bridge holes and gaps up to a certain tolerance, and also tolerate non-manifold edges and degeneracies, our algorithm works well on imperfect meshes with such defects, whereas the conventional *remeshing* methods often fail. See Figure 1.

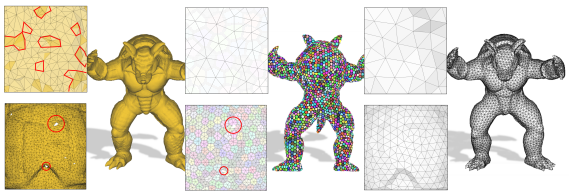


Figure 1: *Isotropic meshing on an imperfect mesh with non-manifold edges, degenerate triangles and holes.*

This paper makes the following contributions:

- We propose an efficient framework for constructing isotropic meshes via voxel representation. It can completely avoid the computationally expensive components, such as implicit function fitting, isosurface extraction, and

DOI: 10.2312/pg.20151273

geodesic distance computation, which are often used in the existing methods.

- Our framework can produce topologically consistent shell space with user control so that it can bridge holes and gaps and tolerate noise to some certain extent. It also works well for models with non-manifold edges and degenerate triangles.
- We present a fast and memory-efficient algorithm for computing narrow-banded distance fields on GPUs. The CVT, RVD and the dual Delaunay triangulations are also computed in parallel on the GPUs.

2. Related Work

There is a large body of literature in Voronoi diagrams, distance computation, Delaunay triangulations and their broad applications. Due to space limit, we review only the most relevant work.

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, CVT has been used in a wide range of applications [DGJ02]. CVT can be defined by the critical points of the CVT energy function. A popular way to compute CVT is Lloyd's algorithm [Llo82], which iteratively moves each generator to the corresponding mass center until convergence. Lloyd's method is conceptually simple and easy to implement, however, as a gradient-decent method, it has only linear convergence rate. Liu et al. [LWL*09] proved that the CVT energy function has 2nd order smoothness for most situations encountered in computer graphics, therefore, one can improve the performance of CVT computation by the Newton or quasi-Newton-like optimization methods, which have quadratic or super-linear convergence rate. Rong et al. [RLW*11] developed a GPU-based method for computing the CVT on the plane and observed significant speedup of these GPU-based methods over their CPU counterparts.

To compute CVT on genus-0 surfaces, some researchers [AdVDI03,RJSG11,SGJ13] parameterized the input models to $\mathbb{R}^2$ and computed a 2D CVT whose density function compensates the parameterization distortion. Recently, Wang et al. [WYL*15] proposed an intrinsic method for computing CVT on arbitrary manifold triangle meshes. Rather than computing the mass centers of Voronoi cells that involves area integration, their algorithm computed the Riemannian centers using exponential maps. The parameterization-based and exponential map based CVT algorithms are intrinsic and hereby independent of the embedding space. However, they are computationally expensive and in practical for time-critical applications.

Rather than computing CVT on surfaces directly, Yan et al. [YLL*09] [YWLL13] proposed a novel *indirect* method by computing the restricted Voronoi diagram, that is, the intersection between the input mesh and a 3D CVT. They also adopted the quasi-Newton method for efficient computing the 3D CVT in the volume bounded by the input mesh. Chen et al. [CCW12] proposed an iterative method for generating constrained centroidal Delaunay mesh (CCDM). With local vertex relation, their method does not require geodesic distances and can guarantee the computed CCDM has the same topology as the input mesh. However, their method cannot handle non-manifold edges and degenerate triangles.

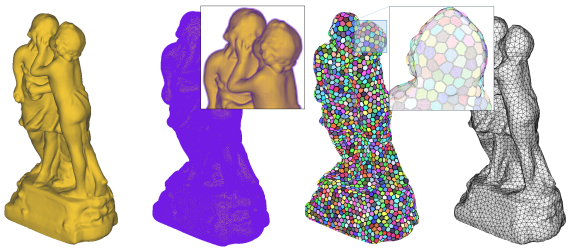


Figure 2: *Overview of our approach on the Sculpture model. (a) Input mesh; (b) Shell space with $d = 3$; (c) CVT with 3K seeds; (d) The output isotropic mesh.*

Li et al. [LZM*14] presented an elegant method for triangulating the conformal uniformization domain via the planar Delaunay refinment. They gave explicit estimates for the Hausdorff distance, the normal deviation, and the differences in curvature measures between the surface and the mesh.

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 $d_p \leq d$ to its closest point on M. The threshold d is specified by the user and 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 the Voronoi diagrams confined in the shell space and moves the seeds toward the corresponding mass centers. The algorithm projects the seeds back to $\bar{P}$ if they are outside the shell space. Upon convergence, it propagates the seed information in the shell space to look for connected Voronoi cells and extracts the dual Delaunay triangulation. Our method is described in detail in the next subsections, and furthermore outlined in Algorithm 1.

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. [CTMT10] to compute Euclidean distance transform (EDT) in the narrow-band manner. Their algorithm partitioned the input domain into small chunks of

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

1: $S \leftarrow k$ random seeds
2: $M \leftarrow$ Voxelization(O, res)
3: $\bar{P} \leftarrow$ ShellSpaceConstruction(M, res, d)
4: **while** convergence not reached **do**
5: $V_k \leftarrow$ SearchClosestSeedInShell($\bar{P}$, S)
6: $C_k \leftarrow$ CenterMass(V_k)
7: $\bar{C}_k \leftarrow$ UpdateSeed($\bar{P}$, C_k)
8: $S \leftarrow \bar{C}$
9: **end while**
10: $V_k \leftarrow$ ShellFlooding($\bar{P}$, S)
11: **return** DualTriangulation(V_k)

equal size, which can be processed in parallel. The results are then merged concurrently. Although their method is exact and efficient, it is not practical for large-scale models due to rapidly growing memory consumption. Our method addresses this memory issue by on-the-fly computation and integrating fast bitmap indexing technology on GPUs. We explain the principle in two dimensions for simplicity; the idea can be easily extended to three dimensions.

The input image is divided into a virtual grid made up of occupied and non-occupied pixels, where the former pixels, denoted as sites $\in S$, are run-length encoded. Every pixel goes through a two-step process : (1) finding the nearest site S_{ij}, among all sites in row j; (2) determining the closest site, among all the nearest sites in the current column i. For the first step we assign one thread to process a row because it's more efficient to do more computation in a single thread than repeatedly accessing global memory with multiple threads. The second step extends the dividing-and-merging approach of PBA, with employing warp[†]-vote and warp-shuffle functions in CUDA to exchange the nearest sites information within a chunk. We make every thread in the same warp doing the same calculation, hence greatly reducing the warp variation that often compromises performance.

Let's take Fig. 3 as an example. When the threads in a warp come to column i, each row will compute their corresponding nearest sites S_{ij}. The nearest site of the current pixel is colored in blue. Note that only some sites (Blue-blank dots) satisfy the the distance constraint. Therefore, the 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 key (boolean) value for every

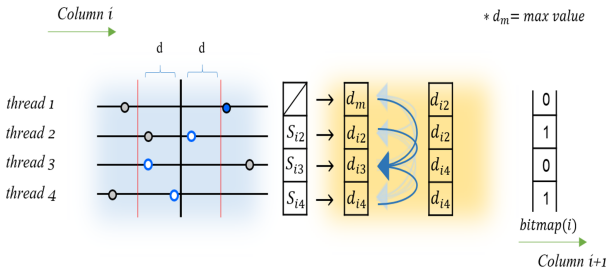


Figure 3: *An illustrative example on distance field computation in a narrow band. See the text for the description and function* ShellSpaceConstruction *for the pseudo code.*

```
// To ease representation, we show the 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
```

pixel. The '0's indicate that the corresponding nearest sites are **not** possible to 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 flag '1' in different chunks and merge them to get the pixels that form the shell region. Since the nearest sites are computed on-the-fly and the temporary result is indexed by bitmap only, our algorithm requires less memory than the PBA method.

3.2. Constructing 3D Voronoi Diagrams in Shell Space

As mentioned above, the shell space represented by $\bar{P}$ is used as constraints to construct Voronoi diagrams and update the positions of seeds. Initially, the seeds $S = \{s_1, .., s_k\}$ are located on the input mesh. We collect points $\in \bar{P}$ that share the same closest seeds to build Voronoi diagrams.

We perform a proximity search to find the closest seed for

[†] A warp is a pool of threads that executes physically in parallel.

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), such that, for a query point q, it just looks up the seeds in the grid cell G_q where q falls into. In case any border of the grid cell is closer to q than the seed found in G_q, query point q looks up the neighbor cell of G_q.

3.3. Computing CVTs

Updating the seeds' position towards uniform distribution is crucial for constructing CVT. Based on the following energy function, optimal positions can be reached by minimizing $E(S)$.

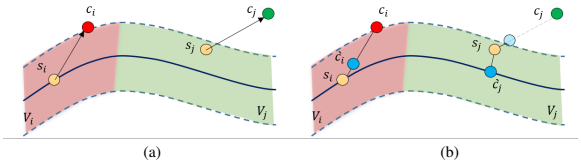
$$E(S) = \sum_{i=1}^{m} \int_{V_i} \rho(p)\|p - s_i\|^2 dp,$$

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.

According to 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 is constructed in the shell space. We consider the following new position to replace mass center for 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 seeds. We observed that if the seeds move in different magnitude, the area of CVTs will largely vary depending on surface curvature. In addition, in order to guarantee the topology consistence, the new center will be projected back to M if it exceeds the shell space, as shown in Fig. 4.



(a) (b)

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

3.4. Computing Dual Triangulations

Upon convergence we have all the generators uniformly distributed. The remain part describes how to extract the dual Delaunay triangulation.

First, we find the direct neighbors of all seeds, where the direct means if there exists two voxels from their Voronoi cells that are connected. The adjacency neighbors can be found by flooding all seeds information to all voxels in the shell $\in \bar{P}$. Each voxel associates a hash table to hold the location of neighbors (26 voxels). Each propagation updates the current seed information to neighbor voxels until all voxels are reached. This approach avoids producing wrong network for seeds that are geometrically close, but topologically far from each other. After that we organize their direct neighbors in clockwise order and finally extract the triangle mesh.

4. Experimental Results

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

4.1. Narrow-banded Distance Fields

Table 1 lists the computational time and the peak memory under varying parameter d and *res* (Fig.5). Clearly, when d increases, the computational time increases insignificantly with the increased amount of nearest sites in the shell. This is due to the low cost of intra-warp communication and the reduction of warp divergence in our algorithm. Also, the memory consumption is remarkably small, considering the scene is in high resolution uniform grid. Traditional algorithms (e.g., [CTMT10]) usually require memory at least 10 times more than ours.

Table 1: *Performance (time in seconds) of our algorithm on different d in resolution 1024^3 and 2048^3.*

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

Table 2 compares our algorithm with PBA on the Sculpture model in resolution 512^3. Since PBA computes a full distance map, their performance is independent of distance d. The result shows that our algorithm consumes significantly less memory and runs much faster than PBA with a reasonably small d.

Table 2: *Comparison of our algorithm with PBA in resolution 512^3 (time in seconds).*

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

4.2. CVT Computation

Similar to [WYL*15], we adopted the following criteria to measure the triangle mesh quality. (1) 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. (2) The smallest

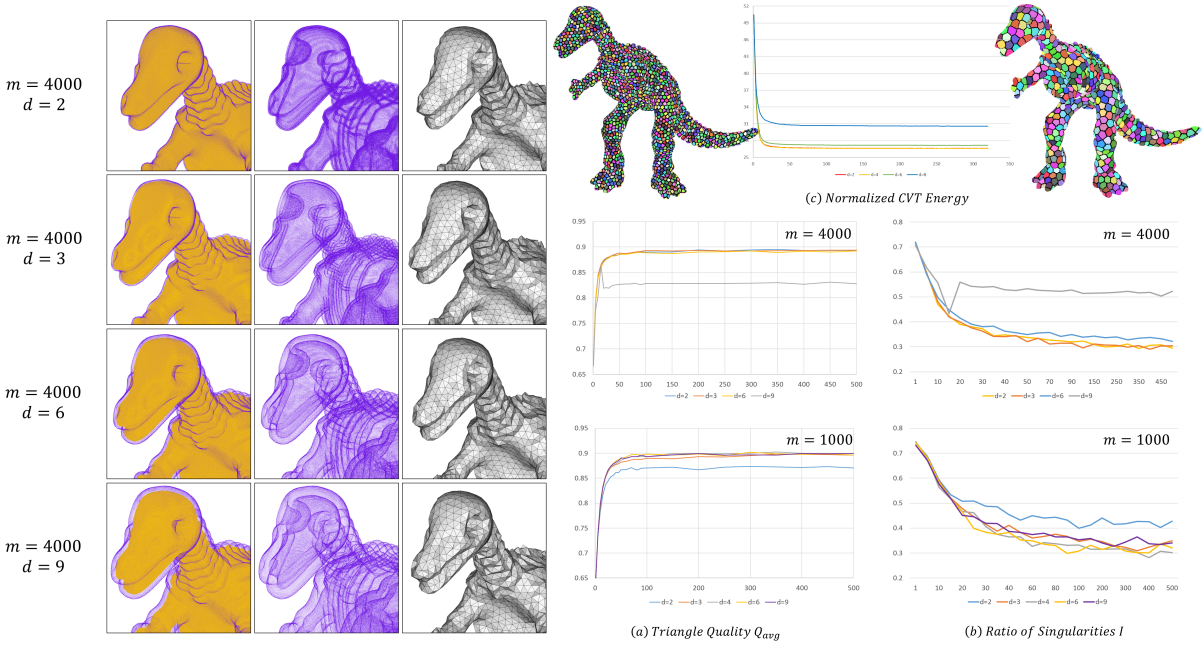


Figure 5: *Evaluating the mesh quality under various shell space parameter d. (a) and (b) show the triangulation quality measure and the singularity ratio. (c) We also observe that our GPU-based Lloyd algorithm converges in usually 100-200 iterations and d has little impact on the convergence rate. The horizontal axis shows the iteration number and the vertical axis is the normalized CVT energy function.*

angle θ_{min} and the average θ_{avg} of minimal angles of all triangles. (3) 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.

We allow the user to balance accuracy and efficiency in the choice of offset d. Figure 5 describes the relationship between the distance d, the number of generator and the quality of remeshed surface. As the offset increases to 9, with 4k generators, the mesh quality of dinosaur model dramatically drops. This also happens when the offset decreases to 2 with 1k generators. Figure 5(b) illustrates the quality difference between different d clearly. Along with other examples in Table 3, we can show that the mesh is at best quality with offset distance in specified range (2 to 6). Figure 6 compares our method with two parameterization-free isotropic meshing methods, the *intrinsic* CVT method by Wang et al [WYL*15] and the *extrinsic* RVD method by Yan et al. [YLL*09]. Thanks to the GPU-friendly structure and the computational power of modern GPUs, our method runs significantly faster than their CPU-based implementations.

5. Conclusion and Future Work

This paper presents a robust and efficient method for constructing isotropic meshes using Euclidean Distance Transform. Our algorithm constructs a narrow band space 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 it 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 the conventional CPU-based implementations.

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

Acknowledgement. This project was supported by AcRF RG40/12, MOE2013-T2-2-011, NSFC grants (61432003 & 61322206), the TNList Cross-discipline Foundation, and HKSAR Research Grants Council (RGC) General Research Fund (GRF), CUHK/14207414.

References

[AdVDI03] ALLIEZ P., DE VERDIÈRE É. C., DEVILLERS O., ISENBURG M.: Isotropic surface remeshing. In *Proceedings of SMI '03* (2003), pp. 49–58. 2

[CCW12] CHEN Z., CAO J., WANG W.: Isotropic surface remeshing using constrained centroidal Delaunay mesh. *Comput. Graph. Forum 31*, 7-1 (2012), 2077–2085. 2

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)	# of Sites	m	T(s)	n	I	Q_{min}	Q_{avg}	θ_{min}	θ_{avg}	Total Time(s)
Sculpture	3	0.966	1.04×10^6	3K	0.064	100	0.25	0.639	0.907	36.1	52.4	7.2
Heptoroid	2	1.429	2.63×10^6	9K	0.138	120	0.24	0.624	0.902	35.3	51.9	19.5
Helix	2	1.212	4.4×10^5	4K	0.036	100	0.29	0.589	0.889	33.4	50.9	5.14
Pegaso	2	0.948	1.33×10^6	8K	0.063	150	0.26	0.600	0.894	31.5	51.3	10.6
Dinosaur	2	0.880	5.5×10^5	4K	0.025	160	0.28	0.636	0.894	30.2	51.2	4.8
	4	0.913	5.5×10^5	1K	0.046	170	0.31	0.602	0.900	30.6	51.8	9.2
Armadillo	2	0.944	1.26×10^6	4K	0.051	200	0.26	0.613	0.902	30.3	51.9	11.3

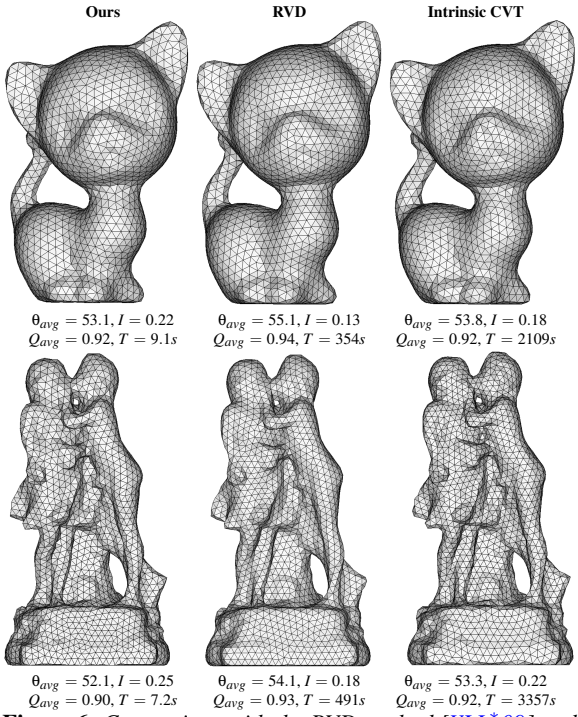


Figure 6: *Comparison with the RVD method [YLL*09] and the intrinsic CVT method [WYL*15].*

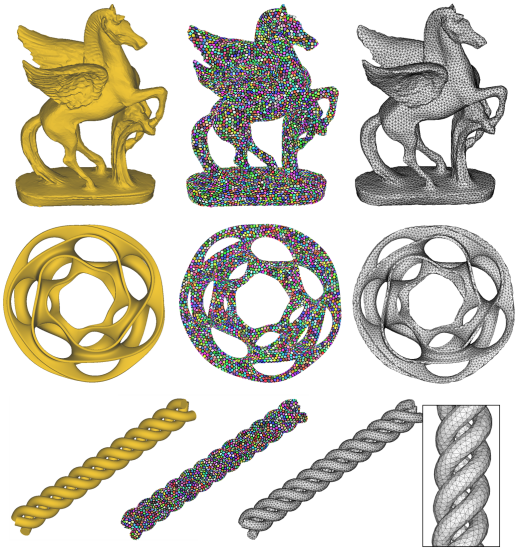


Figure 7: *Experimental results. Images are rendered in high-resolution, allowing zooming in examination.*

[CTMT10] CAO T.-T., TANG K., MOHAMED A., TAN T.-S.: Parallel banding algorithm to compute exact distance transform with the GPU. In *Proceedings of ACM I3D '10* (2010), pp. 83–90. 2, 4

[DGJ02] DU Q., GUNZBURGER M. D., JU L.: Constrained centroidal Voronoi tessellations for surfaces. *SIAM J. Sci. Comput. 24*, 5 (2002), 1488–1506. 2

[Llo82] LLOYD S.: Least squares quantization in PCM. *IEEE Trans. Inform. Theory. 28*, 2 (1982). 2

[LWL*09] LIU Y., WANG W., LÉVY B., SUN F., YAN D.-M., LU: On centroidal Voronoi tessellation: Energy smoothness and fast computation. *ACM Trans. Graph. 28*, 4 (2009). 2

[LZM*14] LI H., ZENG W., MORVAN J., CHEN L., GU X.: Surface meshing with curvature convergence. *IEEE Transactions on Visualization and Computer Graphics 20*, 6 (2014), 919–934. 2

[RJSG11] RONG G., JIN M., SHUAI L., GUO X.: Centroidal Voronoi tessellation in universal covering space of manifold surfaces. *CAGD 28*, 8 (2011), 475–496. 2

[RLW*11] RONG G., LIU Y., WANG W., YIN X., GU X. D., GUO X.: Gpu-assisted computation of centroidal Voronoi tessellation. *IEEE Trans. Vis. Comput. Graph. 17*, 3 (2011), 345–356. 2

[SGJ13] SHUAI L., GUO X., JIN M.: GPU-based computation of discrete periodic centroidal Voronoi tessellation in hyperbolic space. *Computer-Aided Design 45*, 2 (2013), 463–472. 2

[WYL*15] 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 58* (2015), 51–61. 2, 4, 5, 6

[YLL*09] YAN D.-M., LÉVY B., LIU Y., SUN F., WANG W.: Isotropic remeshing with fast and exact computation of restricted Voronoi diagram. *Comput. Graph. Forum 28*, 5 (2009), 1445–1454. 1, 2, 5, 6

[YWLL13] YAN D., WANG W., LÉVY B., LIU Y.: Efficient computation of clipped Voronoi diagram for mesh generation. *Computer-Aided Design 45*, 4 (2013), 843–852. 2