Vis Comput (2014) 30:773–785
DOI 10.1007/s00371-014-0948-z

ORIGINAL ARTICLE

Efficient triangulation of Poisson-disk sampled point sets

Jianwei Guo · Dong-Ming Yan · Guanbo Bao · Weiming Dong · Xiaopeng Zhang · Peter Wonka

Published online: 6 May 2014

Abstract In this paper, we present a simple yet efficient algorithm for triangulating a 2D input domain containing a Poisson-disk sampled point set. The proposed algorithm combines a regular grid and a discrete clustering approach to speedup the triangulation. Moreover, our triangulation algorithm is flexible and performs well on more general point sets such as adaptive, non-maximal Poisson-disk sets. The experimental results demonstrate that our algorithm is robust for a wide range of input domains and achieves significant performance improvement compared to the current state-of-the-art approaches.

Keywords Triangulation · Poisson-disk sampling · Geometric algorithms

1 Introduction

Triangulation is the process of tessellating a given domain with triangles. Due to the different requirements from various applications, many types of triangulation techniques were studied during the last decades.

Among all of the triangulation techniques, Delaunay triangulation [5] has gained most attention since the resulting mesh has many nice geometric properties. There are a lot of mature algorithms or packages for Delaunay (or regular) triangulation, such as CGAL [3], qHull [1] and Triangle [23,24].

The Delaunay triangulation technique has been extensively studied and it is hard to improve. However, if we assume a certain distribution of the input point set, there is still a room to design simpler or faster Delaunay triangulation algorithms. One type of well-spaced point sets that is often used is a Poisson-disk point set, due to its application in rendering, geometry processing, non-photorealistic rendering [26] and modeling [22].

In this paper, we study the triangulation of a Poisson-disk sampled point set. We assume that the grid information used for Poisson-disk sampling is also available. We propose a simple algorithm which is based on a discrete clustering of the grid cells. The input to our algorithm can be a maximal or non-maximal, uniform or adaptive Poisson-disk point set. A GPU implementation of the presented algorithm is also provided which gains significant performance improvement. The main contribution of our work is the speed and robustness of the proposed algorithm. Our implementation works well in practice as demonstrated by various examples and comparison to the state-of-the-art approaches. Figure 1 shows an example of our output.

1.1 Related work

In this section, we briefly review the most related work in Poisson-disk sampling and Delaunay mesh generation. For more details, the readers are referred to [18] for a compre-

J. Guo · D.-M. Yan (✉) · G. Bao · W. Dong · X. Zhang (✉)
LIAMA-NLPR, Institute of Automation, Chinese Academy of Sciences, Beijing, China
e-mail: yandongming@gmail.com

X. Zhang
e-mail: xiaopeng.zhang@ia.ac.cn

J. Guo
e-mail: jianwei.guo@nlpr.ia.ac.cn

G. Bao
e-mail: guanbo.bao@gmail.com

W. Dong
e-mail: weiming.dong@ia.ac.cn

D.-M. Yan · P. Wonka
Visual Computing Center, King Abdullah University of Science and Technology, Thuwal, Kingdom of Saudi Arabia
e-mail: pwonka@gmail.com

Fig. 1 Our algorithm can triangulate a uniform (*top*) and adaptive (*bottom*) Poisson-disk sampled point sets efficiently. From *top to bottom*, and *left to right*: uniform maximal Poisson-disk set, clustering result, and Delaunay triangulation; adaptively sampled Poisson-disk set, clustering result, and regular triangulation

hensive survey of Poisson-disk sampling and the textbooks [5,14] for (Delaunay) mesh generation.

Poisson-disk sampling Cook [9] first proposes an algorithm for Poisson-disk sampling, called dart-throwing. However, it is known that the classic dart-throwing- based approaches are inefficient to achieve the maximal property. Therefore, one area of this research is the acceleration of the generation of Poisson-disk point sets, e.g., using Voronoi Diagrams [17] or scalloped sectors [10].

White et al. [27] propose to use a regular grid to accelerate the Poisson-disk sampling. The cell size of the grid equals to $\frac{r}{\sqrt{2}}$, such that each grid cell can at most receive one disk with radius r, and the conflict checking can be done efficiently in a 5×5 neighborhood. During the sampling process, the partially covered cells are subdivided into smaller fragments in a quadtree manner. Gamito and Maddock [15] extend [27]'s algorithm to higher dimensions. Wei [25] developed a parallel algorithm for Poisson-disk sampling, which also uses the above grid as the background data structure.

More recently, Ebeida et al. [13] propose a two-step unbiased maximal Poisson-disk sampling (MPS) framework that first uses the regular gird for initial sampling and then uses convex polygons to trace and sample the uncovered regions. The follow-up work of Ebeida et al. [12] further extends [27]'s algorithm using a flat array to store the uncovered cells instead of subdividing each cell individually, which is much faster and memory efficient. Yan and Wonka [28] generalize the uniform MPS to various radii, both in Euclidean space and on surface. In summary, most modern Poisson-disk sampling algorithms use a regular grid as acceleration data structure.

Delaunay mesh generation Delaunay mesh generation techniques aim to triangulate a given domain by provably good Delaunay meshes. The research of Delaunay mesh generation became popular since 1980s. A huge amount of work have been proposed [5,14]. Since Delaunay triangulation literature is quite vast, here we only review the most related work to ours.

Recent work of Ebeida et al. [11] reveals that the triangulation of uniform MPS exhibits many nice properties. If a sampled point set satisfies the maximal sampling and minimal distance properties, the corresponding Delaunay triangulation of such a point set satisfies many nice geometric properties, e.g., the edge length bound is $[r, 2r]$ and the angle bound is $[30°, 120°]$ [6]. Ebeida et al. [11] present a new algorithm for computing the conforming Delaunay triangulation [23] for a Poisson-disk sampling point set. However, it is not clear how to use this approach to triangulate non-maximal or non-uniform Poisson-disk point sets.

To accelerate Delaunay mesh generation, several parallel algorithms have been studied [2,7,16,20,21]. The work in [16] computes discrete Voronoi diagrams using graphics hardware. However, it does not contain an algorithm to triangulate the discrete Voronoi diagram in parallel. Rong et al. [21] present a smart method to compute a 2D Delaunay triangulation also using hardware. Its major limitation is that their algorithm is not fully parallel and the stage that uses the CPU for transformation can be very time consuming. Qi et al. [20] extend the work of [21] to make all the phases in parallel. Based on this, they propose the first GPU solution to compute the 2D constrained Delaunay triangulation (CDT) by introducing edge constraints. While related to this work [20], our algorithm is simpler and more efficient in the case of Poisson-disk sampling. The uniform grid we used is adaptively defined by the minimal sampling radius, so that it can handle all the samples at once. As a result, we can avoid the shifting and inserting operations which are the major steps in [20]. Detailed comparison will be given in the experimental section.

1.2 Overview

In this paper, we present a simple algorithm to triangulate a given 2D domain containing a Poisson-disk sampled point set. In the following, we first present the triangulation algorithm on the CPU for uniformly/adaptively sampled Poisson-disk sets in Sect. 2 and Sect. 3, respectively. Then, we present an efficient GPU implementation in Sect. 4. The experimental results are shown in Sect. 5.

2 Triangulation algorithm

The input of our algorithm is a 2D domain D and a set of Poisson-disk sampled points $(\mathbf{x}_i, r)$ (where r is the sampling radius). The 2D domain is represented by a simple polygon, with or without holes. We assume that the distance between any two vertices of the input polygon is larger than the sampling radius. The output is a triangulation that is confined to the input domain. In the following, we first describe the algorithm for maximal Poisson-disk sampling based on a regular grid. Then, we introduce our triangulation algorithm.

2.1 Poisson-disk sampling

Any existing algorithm for Poisson-disk sampling can be used to generate the input point set. Ebeida et al. [12] present a simple algorithm for unbiased maximal Poisson-disk sampling. An implicit quadtree data structure (based on a uniform grid) is used for this purpose. In our framework, we use this algorithm for the purpose of Poisson-disk sampling. Note that in our algorithm, the Poisson-disk point set is not required to be maximal.

A uniform grid G is first built to accelerate the sampling. The size of each cell is equal to $\frac{r}{\sqrt{2}}$, which ensures that each grid cell can contain at most one sampling point. Each grid cell is equipped with a flag 'occupied' to indicate whether the cell contains a sample point. The flag is initialized as 'false'. The grid cells are classified into three types: interior cells, boundary cells, and exterior cells. Only boundary and interior cells will be used for sampling.

Preprocessing To preserve the domain boundaries, we first perform a preprocessing step to sample the domain boundary, and then sample the interior of D.

We first detect sharp corners of the input boundary and insert them into the sampling set directly. If the angle between two edges is smaller than a threshold (we set it to 160°), we mark the corresponding vertex as a sharp corner and the vertex is inserted in the disk set. Then, we perform a 1D MPS on the boundary. Furthermore, to prevent generating samples that are too close to the boundary, which would result in triangles with unbounded small or large angles, we apply edge length optimization to ensure that the boundary disks are deeply intersected [4]. To do that, we iteratively detect and remove the sample vertices of the long edges, and resample the boundary until all edges are shorter than a value (e.g., $\frac{\sqrt{3}}{2}(r_1 + r_2)$, where r_1, r_2 are the sampling radius of the vertices of the edge). This method can handle most of the input boundaries except in a very few cases. Think two sharp feature vertices on the boundary which have a distance $2r - \varepsilon$, the edge between them is long but we cannot insert any other sample between these two vertices so that it may lead to inner samples very close to the boundary or inner samples in boundary cells. In such case, we use the technique of protecting with interior-disks introduced by [11] to overcome this problem.

All the boundary samples remain fixed during the later sampling process. The result of boundary sampling is shown in the left of Fig. 2. Finally, we perform the (maximal) Poisson-disk sampling in the interior of the input domain (Fig. 2, right).

2.2 Grid cells clustering

Once the sampling is finished, the flags of the grid cells that contain a sample point are marked as 'true'. For all the unoccupied cells, we perform a simple clustering to build a discrete Voronoi diagram.

Boundary clustering After the stage of boundary sampling, the input boundary has been subdivided into many short edges. We cluster the boundary cells using a clustering algorithm, as shown in the left of Fig. 3, so that for any two consecutive points, there will be an edge between them. We mark these boundary cells as locked to avoid clustering them again in the later steps.

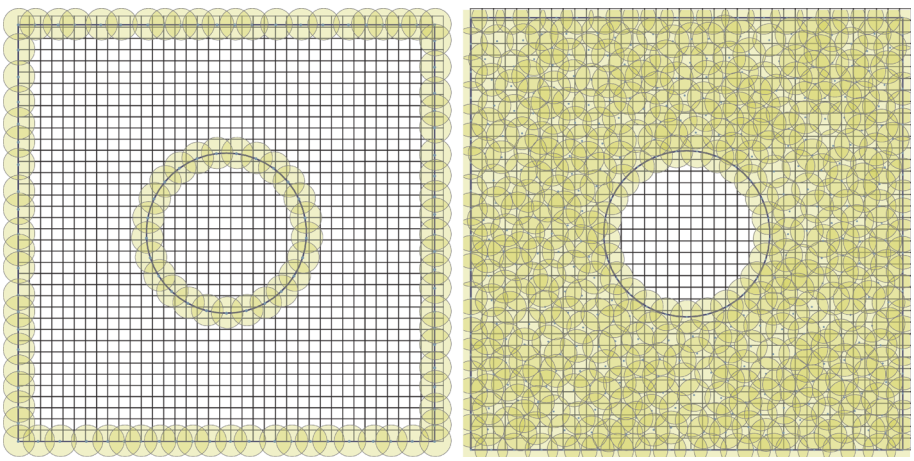


Fig. 2 *Left* Detecting corners and inserting random samples on the boundary. *Right* MPS

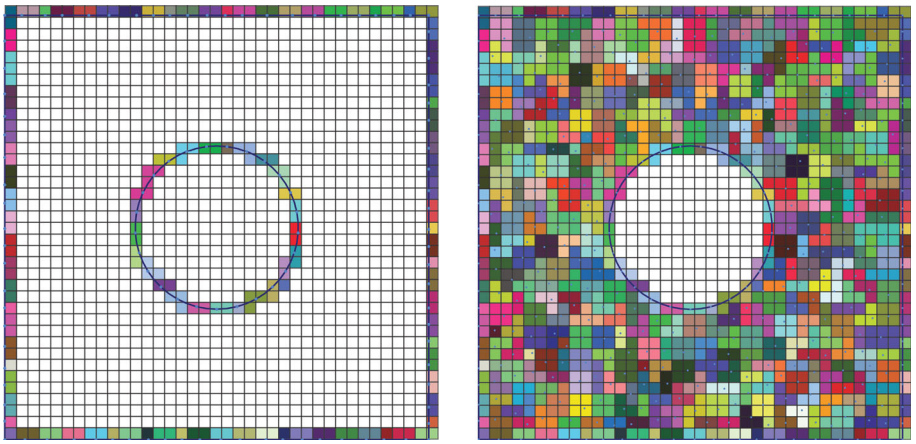


Fig. 3 *Left* Clustering of the boundary cells. *Right* Clustering of the interior grid cells

Interior clustering Next, we cluster the grid cells into groups which are associated with the sample points inside the domain, as shown in Fig. 3(right). In this paper, we apply the variational approximation method [8] accompanied with a priority queue to cluster the 2D grid cells using the Euclidean distance between sample points and cell centers as key for the priority queue. The least distance has the highest priority. One grid cell indexed by (u, v) is represented as $C_{u,v}$ in our paper, and we define its four adjacent cells as those that are indexed by $(u-1, v)$, $(u, v+1)$, $(u+1, v)$ and $(u, v-1)$. First, we collect the grid cells containing sample points as the initial seeds. There is a one-to-one correspondence relationship between each seed cell and the sample point in it. Then, for each seed cell $C_{u,v}$, we insert its four adjacent cells $C_{u',v'}$ (if they are valid) in the global priority queue, with a priority equal to the Euclidean distance between the sample point in $C_{u,v}$ and the center point of $C_{u',v'}$. When inserting, we also assign the candidate label (u, v) to $C_{u',v'}$ as they are being tested against. The region growing process is then performed by repeatedly popping cells with the least distance until the priority queue becomes empty. For each popped cell, there are two cases we should consider: If the cell has been assigned to a sample point, we do nothing and pop another cell from the queue; otherwise, we assign the cell to the sample point indicated by the current candidate label, and push its adjacent and unlabeled cells into the queue with the same label. Finally, we get the clustering result when the queue becomes empty.

2.3 Triangulation

Based on the observation that the clustered cells can be seen as a kind of coarse approximate Voronoi diagram (VD) [16,20], we propose an efficient method to triangulate the sample points. We classify each cell according to the number of different clusters being incident to its lower left hand corner. As shown in Fig. 4, there are four cases: 1-case, 2-case, 3-case and 4-case. In the shown 2-case for example, for the corner p presented by a small red square in the figure, it has two neighboring clusters which are associated with sample point a and b, respectively. For the 3-case and 4-case, we can generate triangles in a very natural manner.

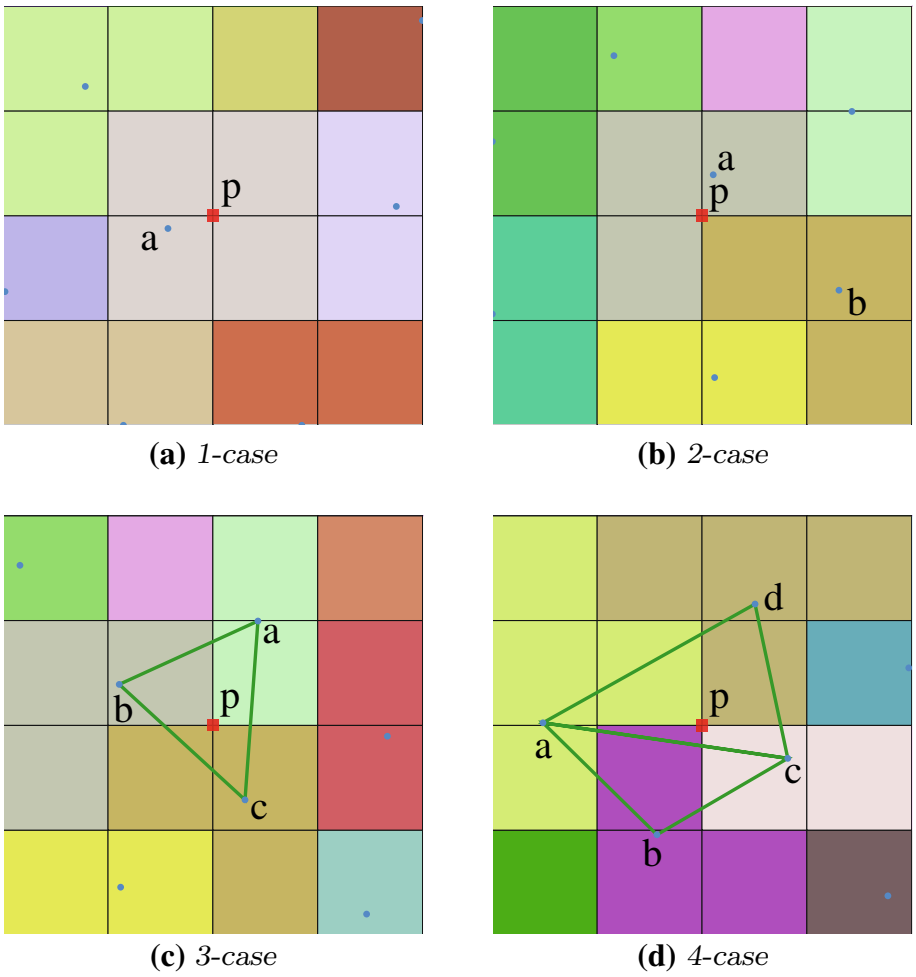
3-case: The cell corner p has three neighboring regions due to sample points a, b, and c. In this case, we generate a triangle $\triangle abc$ directly. See Fig. 4c for an example.

4-case: In this case, we can generate two triangles. For example, see Fig. 4d, we can generate triangles $\triangle bcd$ and $\triangle dab$, or triangles $\triangle abc$ and $\triangle cda$. We perform a Delaunay check to verify which of the triangles are valid. Assume that we first try the triangles $\triangle bcd$ and $\triangle dab$, and if point a is located outside the circumcircle of $\triangle bcd$, then $\triangle bcd$ and $\triangle dab$ are valid. Otherwise, $\triangle abc$ and $\triangle cda$ are valid triangles.

Although we perform a Delaunay check in the 4-case triangulation, there are still some non-Delaunay triangles which are caused by the 3-case triangulation or the concatenation between these two kinds of triangulation [see Fig. 5(left)]. Fortunately, according to our statistics, the Delaunay triangles dominate 90–95 % in the initial triangulation. In the last step, we use the empty circle property of each edge to detect the non-Delaunay edges. For an edge ab of a triangle $\triangle abc$, if the point d of the adjacent triangle $\triangle adb$ is located inside the circumcircle of $\triangle abc$, we call edge ab as a non-Delaunay edge. Then, an edge flipping operation is performed on each non-Delaunay edge. The final result is shown in Fig. 5(right). We provide a formal proof of the correctness of our triangulation algorithm in Appendix A.

3 Triangulation of adaptive sampling

Our algorithm can be easily extended to adaptively sampled Poisson-disk sets. The triangulation then becomes a regular (or weighted Delaunay) triangulation instead of a Delaunay triangulation. Figure 6 shows that the steps of computing a



(a) *1-case* **(b)** *2-case* **(c)** *3-case* **(d)** *4-case*

Fig. 4 Illustration of the four different corner cases and the extracted triangulation for the 3-case and 4-case

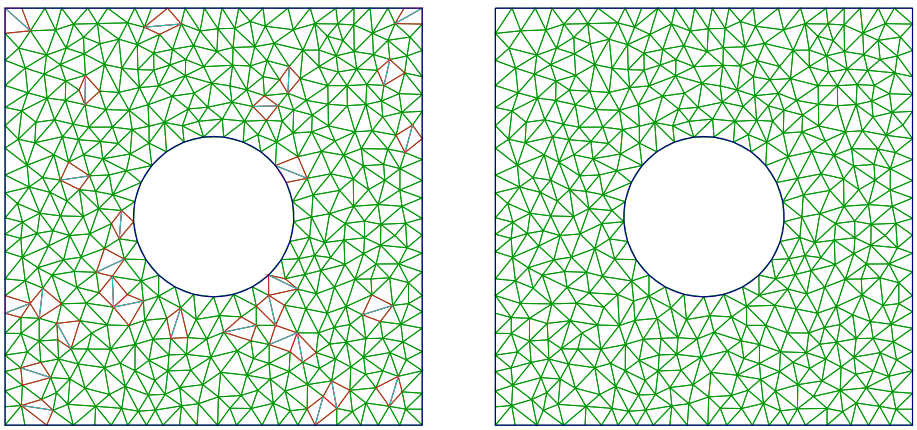


Fig. 5 *Left* A dual mesh is extracted from the clustering result. A large number (93.5 %) of triangles are already Delaunay. The non-Delaunay triangles are marked as *red*. *Right* Edge flips are applied to make the triangulation Delaunay

regular triangulation are the same as those in computing uniform triangulation, except for some differences in the details.

In the sampling step, we follow the definition of [28] for adaptive Poisson-disk sampling. As a result, we generate a set of weighted points $\mathrm{PW} = \{p_i, w_i\}_{i=1}^{n}$, where $w_i = r_i^2$ is the weight of a point, and r_i is the radius of its disk. Each grid cell records the indices of samples that fully cover it. In the clustering step, we just replace the Euclidean distance by the power of two weighted points (Eq. 1). Furthermore, we improve the performance of this step greatly by reducing the number of cells that will be pushed into the global priority queue. For each grid cell $C_{u,v}$, if it is fully covered by only one sample p_i, then we directly assign the cluster of p_i to $C_{u,v}$ and do not insert this cell in the priority queue any more; if $C_{u,v}$

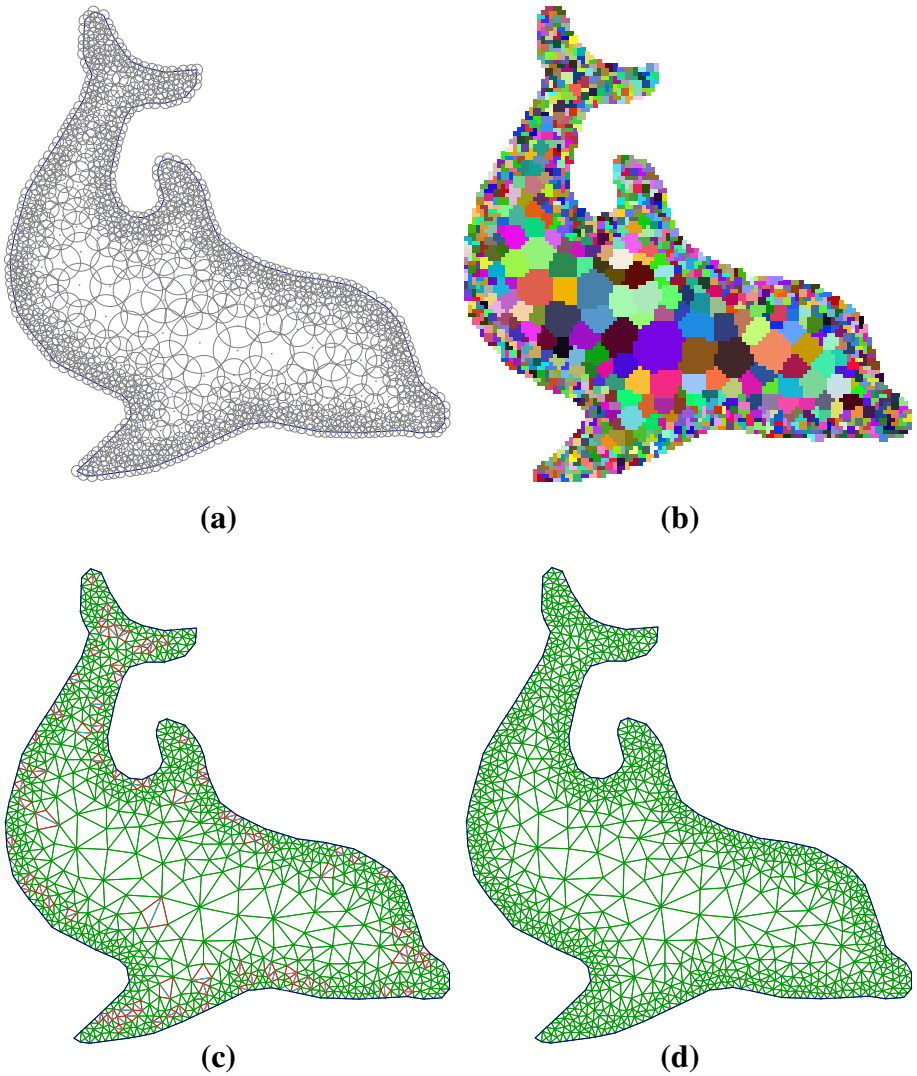


Fig. 6 Triangulation algorithm pipeline for an adaptively sampled Poisson-disk set. **a** Poisson-disk sampling, **b** clustering, **c** initial triangulation, **d** Delaunay triangulation after edge flips

is fully covered by two samples, then we mark it according to the sample with the least power distance; otherwise, this cell can be inserted into the queue.

$$\Pi(p_i, p_j, w_i, w_j) = \|p_i - p_j\|^2 - w_i - w_j \quad (1)$$

Finally, we use the regular property (which just reduces to the Delaunay property when all the weights are null) to check whether a pair of neighboring faces $p_i p_j p_k$ and $p_i p_j p_l$ is locally regular. See the reference manual of [3] for the power test of points (p_i, w_i), (p_j, w_j), (p_k, w_k) and (p_l, w_l). If they are not locally regular, then we perform an edge flip, see Fig. 6.

4 GPU implementation

We designed a GPU version of our algorithm. The random points are generated using the parallel algorithm presented in [12]. For the triangulation and edge flipping operations, the parallel versions can be implemented directly by launching one kernel to deal with one corner or one edge, and the time cost for each kernel is constant. For the clustering step, the method using the priority queue is difficult to be fully parallel. To cluster the cells in parallel, we modify the clustering method slightly.

We maintain a global minimal distance value (initially set to be infinite) for each grid cell and launch one kernel for each sample point. Then, we apply a flood fill algorithm, which is similar to the clustering algorithm in Sect. 2. From each grid cell $C_{u,v}$ which has a sample p in it, we detect the distance between this sample p and each center of its adjacent cell $C_{u',v'}$. If this distance is smaller than the minimal distance of $C_{u',v'}$, we insert $C_{u',v'}$ in a queue and assign label (u, v) to it. The minimal distance of $C_{u',v'}$ is also updated. The flood fill process is then performed by repeatedly popping cells until the queue becomes empty. For each popped cell, we compute the distance between its adjacent cells and the sample p to detect whether we

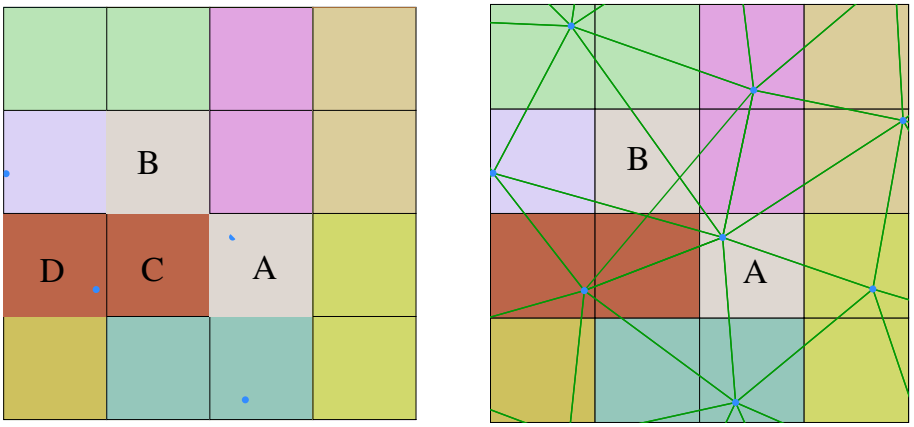
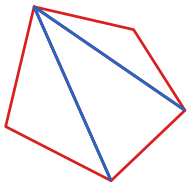


Fig. 7 Local result of a GPU-based triangulation without the post-processing operation. *Left* Cells B and A are in one cluster but they are not connected. *Right* This would generate intersecting triangles in the triangulation step

should insert its adjacent cells in the queue. However, using this method may produce some wrong clusters, even though the possibility is very small (lower than 0.1 % according to our statistics). As shown in Fig. 7, it may happen that cell C is first assigned to the sample point in A, because A's thread might be executed before D's. Then, cell B may be also added to A's cluster. Later, D's thread is executed and it decides that C is actually closer to D, and removes it from A's cluster. Note that B's label is not changed here because it is closer to A than to D. As a result, cell B is isolated from A and this will generate intersecting triangles. To solve this problem, we perform a post-processing operation. For each isolated cell, we recompute the distances between this cell and the sample points indicated by its four adjacent cells. Then, we mark the cluster of this cell according to the closest sample point. If there are two or more such points, we choose the one with the smallest index. The post-processing operation leads to each Voronoi region being connected. As discussed in Appendix A, it guarantees a complete triangular mesh. In addition, the clustering and post-processing operations are all performed in parallel.

Another issue we should consider is that there may be two edges in one triangle that are all non-Delaunay, although they are quite rare. Our Poisson-disk distribution properties prevent more than 5 samples to be concyclic (or nearly so), but 5 nearly concyclic samples can happen. Imprecisions due to the approximations could lead the center triangle of the pentagon to have two non-Delaunay edges (see the blue edges in the embedded figure). In this case, a naive parallel program which uses a single pass of edge flipping could generate intersecting triangles, and it is also insufficient to ensure that all the edges in the final triangulation are Delaunay. In our parallel implementation, we use the atomic operations in CUDA to ensure that we only flip one edge and keep track of the other edge that potentially might require further flipping. Then, we use a second pass (or even more passes) to check these potential non-Delaunay edges and flip them if necessary. Since this case happens rarely, it has little impact on the speed.



5 Experimental results

In this section, we first show that our algorithm can efficiently triangulate the input domain with a uniformly/adaptively sampled maximal/non-maximal Poisson-disk set. Then, the performance of our method is compared with the current state-of-the-art approaches. We do not consider the time for the Poisson-disk sampling in the results and we assume that the point set including the grid information and occupancy lists are provided as input to the triangulation stage. All the experimental results in this paper are conducted on a PC with an Intel i7-3770, 3.40 GHz CPU, 16 GB memory and an Nvidia GeForce GTX 770 graphics card.

5.1 Triangulation results

We have tested our algorithm on various data sets as shown in the additional materials. Most of the models are from the repository of Lu et al. [19]. We verified the correctness of our output by comparing with the results of CGAL [3]. Selected meshing results are shown in Fig. 8. We also show the triangulation results of two image stippling point sets, see Fig. 9. Another application is the generation of adaptively triangulated terrains. Given a height map, we can generate a corresponding 3D mesh (Fig. 10), using the smoothed gradient magnitude as density function.

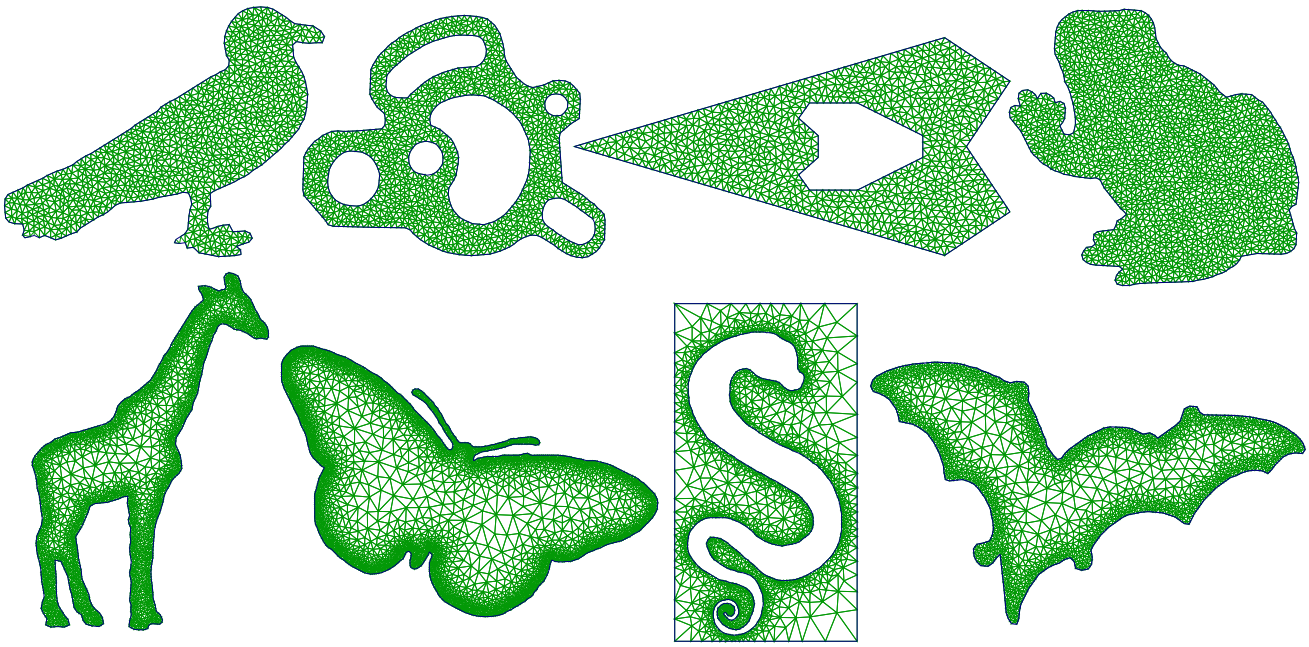


Fig. 8 A gallery of meshing results of uniformly (*top row*) and adaptively (*bottom row*) sampled Poisson-disk point sets

5.2 Performance

In this section, we evaluate the performance of our presented algorithm. Using the serial code, we triangulated 1.17M points/s on the uniform Poisson-disk sets and 0.72M points/s on the adaptive Poisson-disk sets. Furthermore, to show that the differences (such as convex or non-convex, with or without holes, etc.) of input domains have little effect on the running time, we tested our algorithm on the domains of a unit square, Dolphin (Fig. 6) and Snake (Fig. 8) containing adaptively sampled point sets. It took 0.17, 0.20 and 0.19 s, respectively, to triangulate 130K points.

GPU performance We ran our GPU code using a 64-bit Windows 7 operating system. Visual Studio 2010 and CUDA 5.0 Toolkit are used to compile the program. Our algorithm can proceed in a highly parallel manner: in the clustering stage, each thread deals with one sample point; in the triangulation stage, each thread deals with one bottom left corner of grid cell; and in the edge flipping stage, each thread deals with one edge of the triangulation. Using an Nvidia GeForce GTX 770, our fully parallel algorithm triangulated 5.8M points using 0.49 s (about 11.8M points/s) on the uniform Poisson-disk sets, which is a little over 10× speedup over our serial code. In the regular triangulation, since the underlying grid is much finer and has more cells for the same number of points, our parallel method took 0.41 s to triangulate 2.5M points (about 6M points/s). Due to the GPU memory constraints and code overhead for testing and debugging, our parallel algorithm can triangulate about 6 million uniformly sampled points and 2.5 million adaptively sampled points. It is possible to optimize our implementation to reduce the memory usage.

5.3 Comparison

The proposed algorithm can be applied to either a maximal or a near- maximal Poisson-disk set, while Ebeida et al. [11] can only handle maximal point sets. We first compare our approach with previous work [3,11,20,24] with uniform sampling radii. Then, we compare the performance of computing regular triangulation to that of CGAL version 4.1.

Comparison on uniform triangulation To the best of our knowledge, Ebeida et al. [11] were the first to present a conforming Delaunay triangulation algorithm based on uniform Poisson-disk sampling. Triangle [23,24] and CGAL are the most popular and fastest algorithms for Delaunay triangulation. To compare with these algorithms, we generate samples in the unit square with periodic boundaries. We implemented our method, CGAL and Ebeida et al.'s method in C++ and compiled them under 64-bit Linux. The C code for Triangle is also complied under Linux. Figure 11a shows the runtime comparison of the different algorithms. As illustrated in this figure, Ebeida et al.'s method is better than Triangle, but they are very close. Our method achieves 2–3 times speedup over Ebeida et al.'s method. On average, we triangulate 2.8M ($r = 0.0005$) points in 2.4 s (about 1.17M points/s), while Ebeida's takes 6.1 s (about 0.46M points/s). Compared with CGAL (in which we provided all the samples at once, and exact predicates were not used), our serial implementation is also nearly 2 times faster than it.

Next, we compared with the current state-of-the-art GPU Delaunay triangulations [11,20], as shown in Fig. 11b. To get a fair comparison, the running time of ours and Ebeida et al.'s method contains the time for building the uniform

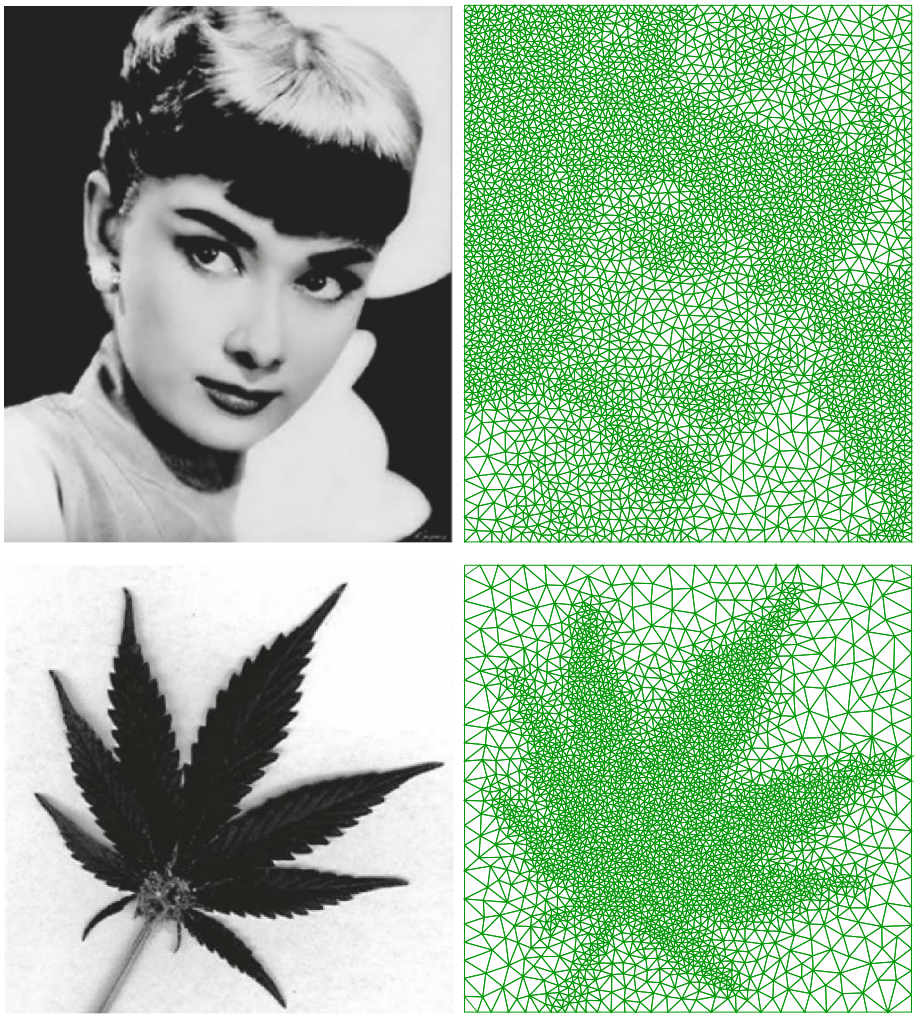


Fig. 9 Triangulation of two image stippling results

grid. We re-implemented the GPU version of Ebeida et al. [11], and downloaded the software GPUDT [20] for comparison. From this figure, we can see that Ebeida et al.'s GPU algorithm is about 7 times slower than our method. That is because their GPU algorithm is not fully parallel. In their algorithm, each thread deals with a 4×4 grid of center cells sequentially, and this degrades the performance. For the GPUDT, the generated Poisson-disk points are used as their input and the number of constraints is set to 0. We also chose double precision and enabled computational capability 3.0 using the switch -sm_30. Compared with GPUDT, we combine the usage of a uniform grid and the good distribution of Poisson-disk sampling points to speedup the triangulation. For different number of points, we achieve about 3.5–4 times speedup over GPUDT. In addition to this, given an adaptively sampled point set with varying radii, the GPUDT is only applicable to compute Delaunay triangulation, while our approach can generate regular triangulation.

Furthermore, in Ebeida et al.'s algorithm, they construct the CDT-star based on the locality (a 7×7 template of cells with corners removed) determined by the background grid and the bounded edge lengths. As a result, their algorithm is only applicable to uniform maximal Poisson-disk sampling. Instead, our proposed algorithm clusters the grid cells independent of localization and extracts the triangulation directly from the clustered cells. Therefore, it also works when the Poisson-disk sampling is not maximal (Fig. 12).

Comparison to regular triangulation Finally, we compared our regular triangulation algorithm to CGAL. While there are some similarities between regular triangulation and Delaunay triangulation, it is not easily possible to extend Triangle to handle regular triangulations.

To compare the performance of our algorithm with that of CGAL, we tested them on the Poisson-disk sets with varying radii. Note that there are two implementations in CGAL: providing all the samples at once (we call it as CGAL_RT1) and inserting samples one by one (CGAL_RT2). The former can be speeded up using spatial sorting and hierarchy data structure. We tested and compared with both of these cases, where we chose the

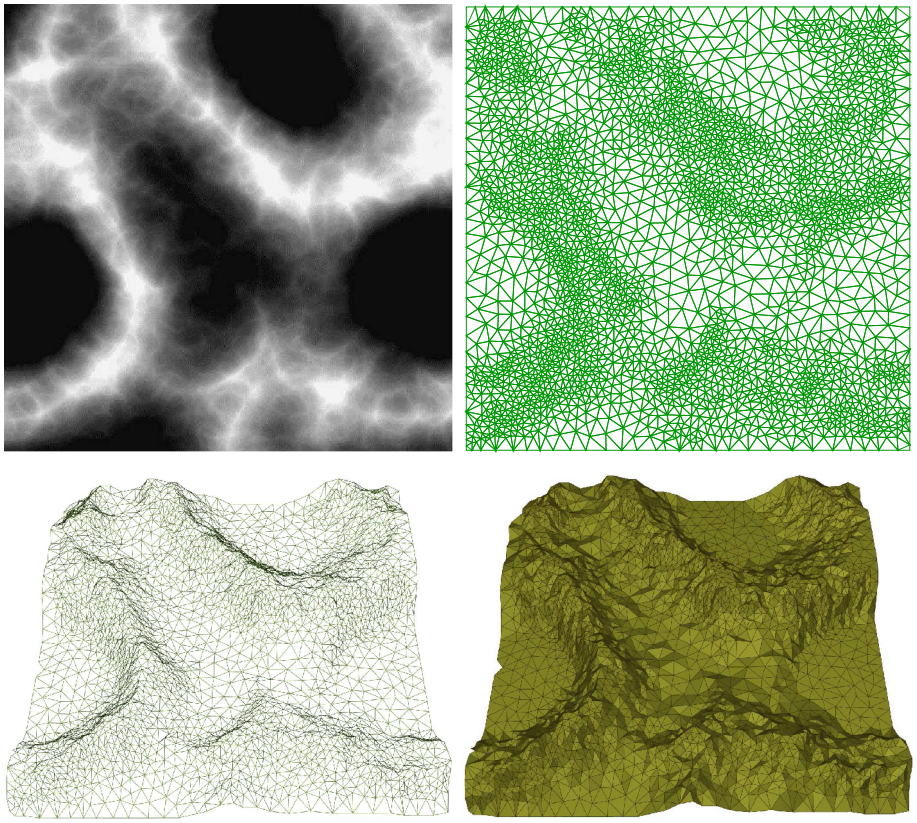


Fig. 10 Illustration of a 3D terrain mesh generation. From *left to right* and *top to bottom*: input height map, 2D triangulation result, 3D mesh and the rendered result

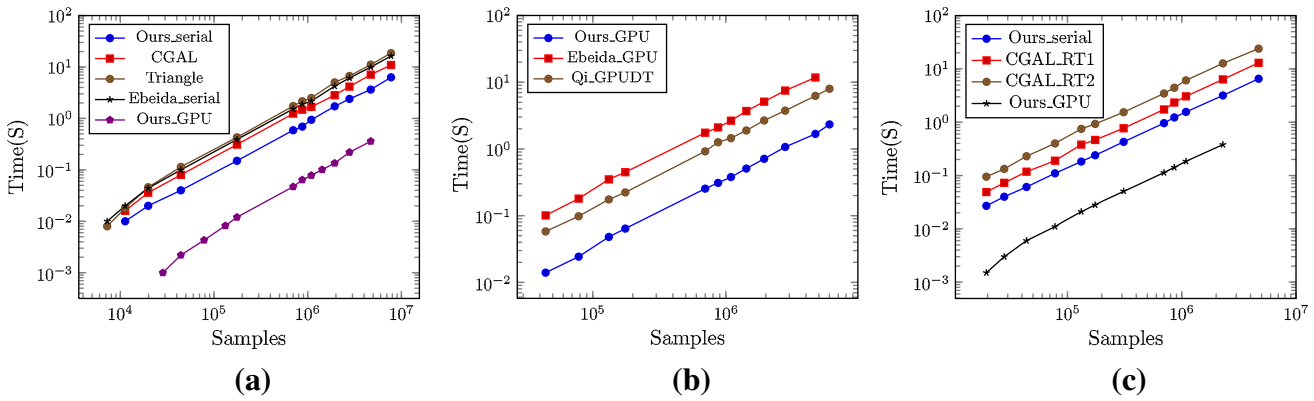


Fig. 11 **a** Comparison of our serial triangulation algorithm with CGAL [3], Triangle [23] and Ebeida et al. [11]'s method. We generate the uniform sampling sets in a unit square. Here, our GPU algorithm is used for ground truth. **b** Comparison of our GPU algorithm with previous GPU approaches [11,20], where the time for building the uniform grid is considered in ours and Ebeida et al. [11]'s method. We also test these algorithms using a unit square as input. **c** Comparison of our algorithm with CGAL for triangulating Poisson-disk point sets with varying radii. The point sets are sampled in the Dolphin domain shown in Fig. 6

"Exact_predicates_inexact_constructions_kernel" and double precision. As shown in Fig. 11c, we have indeed the fastest algorithm. The performances of our serial algorithm are 2 times and 4.5 times better than those of CGAL_RT1 and CGAL_RT2, respectively. Finally, it is also worth pointing out that, our GPU implementation remains much faster, which can achieve significant speedup over CGAL_RT1, of up to an order of magnitude.

6 Conclusions and future work

In this paper, we have presented a simple algorithm for triangulating a 2D input domain containing a Poisson-disk sampled point set. The proposed algorithm utilizes the grid information used for Poisson-disk sampling which greatly improves the efficiency of the triangulation compared with previous approaches. One current limitation of our algorithm

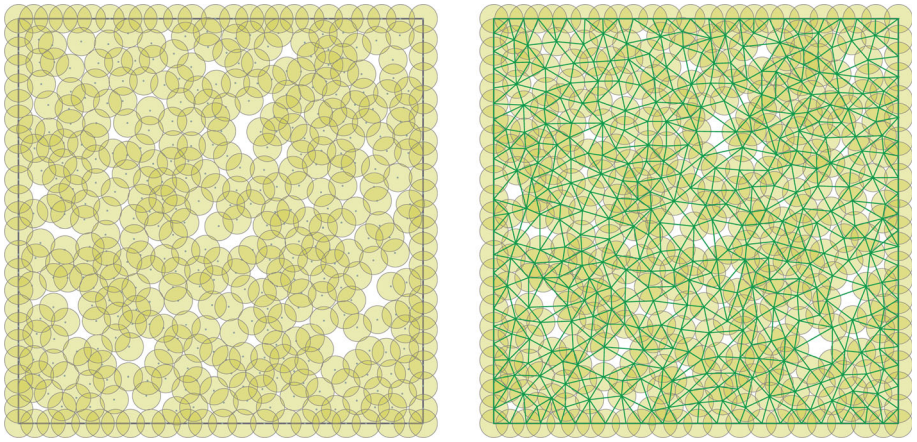


Fig. 12 Triangulation of a non-maximal Poisson-disk set

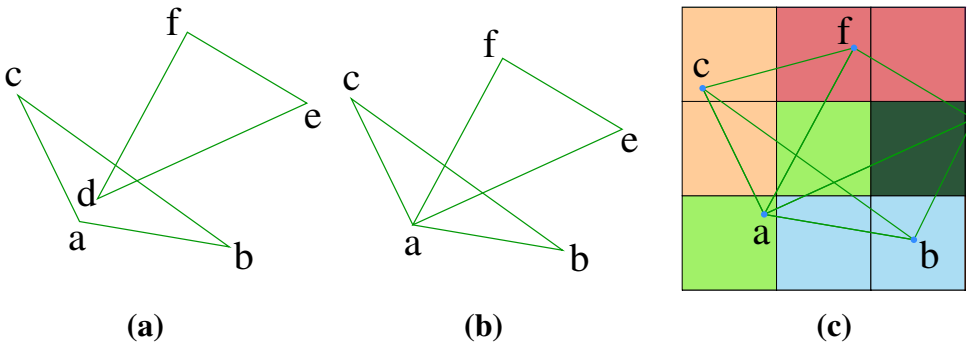


Fig. 13 **a**, **b** The two configurations of the intersection of triangles. **c** The local Voronoi regions corresponding to the intersecting case in **b**

is that we cannot deal with the very thin features of the boundary if two input edges are too close. In such a case, the disks of samples on one input edge may intersect another input edge, and this will generate triangles across the boundary. We would like to address this issue in our future work. In addition, we plan to generalize our approach to surface and volumetric mesh generation.

Acknowledgments This research was partially funded by National Natural Science Foundation of China (Nos. 61372168, 61172104, 61331018, and 61271431), the KAUST Visual Computing Center, and the National Science Foundation.

Appendix A: Proof of correctness

In this appendix, we prove that our triangulation algorithm correctly computes a complete triangular mesh. Here, the collection of the cells that belong to one sample p is called the Voronoi region of p.

Property 1 *Each Voronoi region generated by grid cell clustering is connected and any two regions are non-overlapping.*

Proof In the clustering step, we adopt a propagation algorithm equipped with a priority queue on the grid to approximate the Voronoi regions. Each time we choose a grid cell that has the highest priority and assign the label of its nearest sample to it. Note that each grid cell is connected to a neighbor cell having the same label, which is in turn inductively connected to its nearest sample. In addition, each grid cell will be assigned only once and never changes its label. As a result, the propagation algorithm ensures that each Voronoi region is singly connected and any two regions are not overlapping.

Property 2 *No two triangles in the triangulation intersect each other.*

Proof Suppose that a triangle $\triangle abc$ crosses another triangle $\triangle def$. There are two configurations of the intersection: the two triangles do not share common vertices (Fig. 13a) and share common vertices (Fig. 13b, we take the 1 common vertex as example, the proof for the other cases is straightforward). The former configuration is impossible; otherwise, the Voronoi region of d must connect to the Voronoi regions of a, b and c, and this will generate triangles $\triangle abd$ and $\triangle adc$ instead of $\triangle abc$. The latter configuration happens only if some Voronoi regions are not connected, such as the green cells shown in Fig. 13c. This contradicts the fact that all the Voronoi regions are connected (Property 1). Thus, no two triangles in the triangulation intersect each other.

Property 3 *The union of all triangles fully covers the input domain. In other words, there are no holes in the triangulation.*

Fig. 14 Input domains we used to test our algorithm

Proof Firstly, our boundary sampling and clustering steps guarantee that for any two consecutive points on the boundary there will be an edge between them. This ensures the output triangulation is tightly confined to the input domain.

Next, we demonstrate that any holes in the interior of the domain will be filled by triangles. Before that, we note all the grid cells have been assigned a label (Property 1). Now suppose there exists a hole, which is a polygon consisting of three or more vertices. We consider the Voronoi regions of any three consecutive vertices a, b, c in counterclockwise order. There are only two cases. First, if the three Voronoi regions connect to each other, then our algorithm will generate a triangle $\triangle abc$ and the hole shrinks to a smaller one. Second, if the Voronoi regions of a and c are not adjacent, but they both connect to that of b, then there must be another vertex d whose Voronoi region connects to all of the three Voronoi regions. In such a case, it will generate two triangles $\triangle abd$ and $\triangle bcd$ and the hole also shrinks to a smaller one. As such, the hole becomes smaller and smaller by repeatedly processing like this, and finally it will disappear.

Appendix B: Input domains

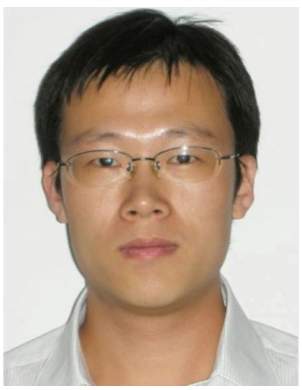
See Fig. 14.

References

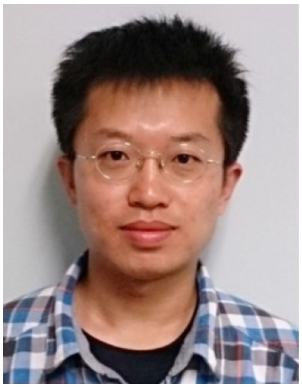
1. Barber, C.B., Dobkin, D., Huhdanpaa, H.: The quickhull algorithm for convex hulls. ACM Trans. Math. Softw. **22**(4), 469–483 (1996)
2. Blelloch, G.E., Miller, G.L., Hardwick, J.C., Talmor, D.: Design and implementation of a practical parallel delaunay algorithm. Algorithmica **24**(3–4), 243–269 (1999)
3. CGAL, Computational Geometry Algorithms Library. http://www.cgal.org
4. Cheng, S.W., Dey, T.K., Levine, J.A.: A practical Delaunay meshing algorithm for a large class of domains. In: Proceedings of the 16th International Meshing Roundtable, pp. 477–494 (2007)
5. Cheng, S.W., Dey, T.K., Shewchuk, J.R.: Delaunay Mesh Generation. CRC Press, Boca Raton (2012)
6. Chew, L.P.: Guaranteed-quality triangular meshes. Department of Computer Science Tech Report 89-983, Cornell University (1989)
7. Chrisochoides, N., Nave, D.: Parallel delaunay mesh generation kernel. Int. J. Numer. Methods Eng. **58**(2), 161–176 (2003)
8. Cohen-Steiner, D., Alliez, P., Desbrun, M.: Variational shape approximation. ACM Trans. Graph. (Proc. SIGGRAPH) **23**(3), 905–914 (2004)
9. Cook, R.L.: Stochastic sampling in computer graphics. ACM Trans. Graph. **5**(1), 69–78 (1986)
10. Dunbar, D., Humphreys, G.: A spatial data structure for fast poisson-disk sample generation. ACM Trans. Graph. (Proc. SIGGRAPH) **25**(3), 503–508 (2006)
11. Ebeida, M.S., Mitchell, S.A., Davidson, A.A., Patney, A., Knupp, P.M., Owens, J.D.: Efficient and good delaunay meshes from random points. Comput. Aided Des. **43**(11), 1506–1515 (2011)
12. Ebeida, M.S., Mitchell, S.A., Patney, A., Davidson, A.A., Owens, J.D.: A simple algorithm for maximal poisson-disk sampling in high dimensions. Comput. Graph. Forum (Proc. EUROGRAPHICS) **31**(2), 785–794 (2012)
13. Ebeida, M.S., Patney, A., Mitchell, S.A., Davidson, A., Knupp, P.M., Owens, J.D.: Efficient maximal poisson-disk sampling. ACM Trans. Graph. (Proc. SIGGRAPH) **30**(4), 49:1–49:12 (2011)
14. Edelsbrunner, H.: Geometry and Topology for Mesh Generation. Cambridge University Press, Cambridge (2001)
15. Gamito, M.N., Maddock, S.C.: Accurate multidimensional poisson-disk sampling. ACM Trans. Graph. **29**(1), 8:1–8:19 (2009)
16. 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, SIGGRAPH '99, pp. 277–286 (1999)

17. Jones, T.R.: Efficient generation of poisson-disk sampling patterns. J. Graph. Tools **11**(2), 27–36 (2006)
18. Lagae, A., Dutré, P.: A comparison of methods for generating poisson disk distributions. Comput. Graph. Forum **27**(1), 114–129 (2008)
19. Lu, Y., Lien, J.-M., Ghosh, M., Amato, N.M.: Alpha-decomposition of polygons. Comput. Graph. (Proc. SMI) **36**(5), 466–476 (2012)
20. Qi, M., Cao, T.T., Tan, T.S.: Computing 2d constrained delaunay triangulation using the gpu. IEEE Trans. Vis. Comput. Graph. **19**(5), 736–748 (2013)
21. Rong, G., Tan, T.S., Cao, T.T., et al.: Computing two-dimensional Delaunay triangulation using graphics hardware. In: Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games, pp. 89–97. ACM (2008)
22. Schechter, H., Bridson, R.: Ghost sph for animating water. ACM Trans. Graph. (Proc. SIGGRAPH) **31**(4), 61:1–61:8 (2012)
23. Shewchuk, J.R.: Triangle: engineering a 2d quality mesh generator and delaunay triangulator. In: Lin, M.C., Manocha, D. (eds.) Applied Computational Geometry: Towards Geometric Engineering. Lecture Notes in Computer Science, vol. 1148, pp. 203–222. Springer, Berlin (1996)
24. Shewchuk, J.R.: Delaunay refinement algorithms for triangular mesh generation. Comput. Geom. Theory Appl. **22**(1), 21–74 (2002)
25. Wei, L.Y.: Parallel poisson disk sampling. ACM Trans. Graph. (Proc. SIGGRAPH) **27**(3), 20:1–20:9 (2008)
26. Wei, L.Y.: Multi-class blue noise sampling. ACM Trans. Graph. (Proc. SIGGRAPH) **29**(4), 79:1–79:8 (2010)
27. White, K.B., Cline, D., Egbert, P.K.: Poisson disk point sets by hierarchical dart throwing. In: Proceedings of the IEEE Symposium on Interactive Ray Tracing, pp. 129–132 (2007)
28. Yan, D.M., Wonka, P.: Gap processing for adaptive maximal poisson-disk sampling. ACM Trans. Graph. **32**(5), 148:1–148:15 (2013)

Jianwei Guo received his bachelor degree from Shandong University in 2011, and he is currently a Ph.D. candidate in National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences. His research interests include 3D shape analysis and geometry processing.



Dong-Ming Yan is a research scientist at King Abdullah University of Science and Technology (KAUST), and he is an Associate Professor at LIAMA-NLPR, CASIA. He received his PhD from Hong Kong University in 2010, and his Master and Bachelor degrees both from Tsinghua University in 2005 and 2002, respectively. His research interests include computer graphics, geometric processing and visualization.



Guanbo Bao is an assistant professor at Institute of Automation, Chinese Academy of Sciences. He received his Ph.D. degree in computer science from Institute of Automation, Chinese Academy of Sciences, in 2012. His research interests include high performance rendering, geometry processing and image editing.

Weiming Dong is an Associate Professor in LIAMA-NLPR, CASIA. He received his BSc and MSc degrees in computer science in 2001 and 2004, both from Tsinghua University, P.R. China. He received his Ph.D. in computer science from the University of Henri Poincaré Nancy 1, France, in 2007. His research interests include image synthesis and image analysis. Weiming Dong is a member of ACM and IEEE.

Xiaopeng Zhang is a Professor in National Laboratory of Pattern Recognition at Institute of Automation, Chinese Academic of Sciences (CAS). He received his Ph.D. degree in Computer Science from Institute of Software, CAS in 1999. He received the National Scientific and Technological Progress Prize (second class) in 2004. His main research interests include computer graphics and image processing.

Peter Wonka is Associate Professor at KAUST (King Abdullah University of Science and Technology) and ASU (Arizona State University). His research interests include topics in computer graphics, visualization, computer vision, remote sensing, image processing, and machine learning.