An Intrinsic Algorithm for Parallel Poisson Disk Sampling on Arbitrary Surfaces

Xiang Ying, Shi-Qing Xin, Qian Sun, and Ying He

Abstract—Poisson disk sampling has excellent spatial and spectral properties, and plays an important role in a variety of visual computing. Although many promising algorithms have been proposed for multidimensional sampling in euclidean space, very few studies have been reported with regard to the problem of generating Poisson disks on surfaces due to the complicated nature of the surface. This paper presents an intrinsic algorithm for parallel Poisson disk sampling on arbitrary surfaces. In sharp contrast to the conventional parallel approaches, our method neither partitions the given surface into small patches nor uses any spatial data structure to maintain the voids in the sampling domain. Instead, our approach assigns each sample candidate a random and unique priority that is unbiased with regard to the distribution. Hence, multiple threads can process the candidates simultaneously and resolve conflicts by checking the given priority values. Our algorithm guarantees that the generated Poisson disks are uniformly and randomly distributed without bias. It is worth noting that our method is intrinsic and independent of the embedding space. This intrinsic feature allows us to generate Poisson disk patterns on arbitrary surfaces in $\mathbb{R}^n$. To our knowledge, this is the first *intrinsic*, *parallel*, and *accurate* algorithm for surface Poisson disk sampling. Furthermore, by manipulating the spatially varying density function, we can obtain adaptive sampling easily.

Index Terms—Parallel poisson disk sampling, intrinsic algorithm, unbiased sampling, GPU, geodesic distance

1 Introduction

Poisson disk sampling distributes all samples that are uniformly and randomly located. It requires that the samples are as dense as possible but are at a minimum distance apart from one another. Due to its excellent spatial and spectral properties, Poisson disk sampling is widely used in computer graphics and visualization, such as antialiasing, global illumination, nonphotorealistic rendering, remeshing, texture synthesis, vector field visualization, and so on. Dart throwing, proposed by Dippé and Wold [2], is the first accurate approach to generate Poisson disk patterns in $\mathbb{R}^n$. However, this brute-force approach is impractical and inefficient, since a large number of samples are involved in the attempt but only a small percentage of them are eventually inserted into the distribution. Since then, many approaches have been proposed to improve the performance, for example, jittered sampling [3], spatial data structures [4], [5], procedure tiling [6], [7], [8], and hierarchical sampling [9], [10].

While tremendous efforts have been focused on the multidimensional sampling in euclidean space, very few studies have been reported on generating Poisson disk patterns on curved surfaces. A very natural research direction is to extend the key ideas and data structures of the existing 2D sampling algorithms to curved surfaces. However, this extension is usually technically challenging due to the following reasons: first, a surface is a 2D manifold that has arbitrary topology and complicated geometry, and is embedded in $\mathbb{R}^3$ or even higher dimensional space. Second, the generated samples should be randomly and uniformly distributed on surfaces, and exhibit the blue noise pattern without bias. Third, the algorithm can be parallelized and implemented on modern graphics hardware easily. Fourth, the algorithm should be intrinsic to the geometry and insensitive to the mesh resolution and tessellation. Last but not the least, the exact geodesic distance should be used to enforce the minimum distance constraint between any pair of samples.

Recently, Wei [11] pioneered the phase group technique, which subdivides the sample domain into grid cells and then draws samples concurrently from multiple cells that are sufficiently far apart so that their samples cannot conflict one another. The phase group technique is very intuitive, inherently parallel, and highly efficient for Poisson disk sampling in euclidean space of arbitrary dimension. Wei's algorithm can be extended to generate Poisson disks on 3D surface by creating the phase groups in its embedding space [12]. However, due to its dependence of the ambient space, it is difficult to apply the phase graph method [12] to surfaces with complicated geometry/topology or embedded in high-dimensional spaces. Furthermore, as pointed out in [5], the phase group algorithm is approximate in that the generated distribution is not fully random.

In this paper, we present a new method for parallel Poisson disk sampling. Rather than the phase group method that explicitly partitions the sampling domain to generate the samples in parallel, our approach assigns each sample candidate a random and unique priority that is unbiased with regard to the distribution, and organizes the samples by their indices. Hence, multiple threads can process the candidates simultaneously and resolve conflicts by checking

- *The authors are with the School of Computer Engineering, Nanyang Technological University, 50 Nanyang Avenue, BLK N4, Singapore 639798, Singapore. E-mail: {ying0008, sqxin, sunq0004, yhe}@ntu.edu.sg.*

Manuscript received 29 July 2012; revised 15 Jan. 2013; accepted 22 Feb. 2013; published online 8 Mar. 2013.
Recommended for acceptance by P. Cignoni.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-2012-07-0144.
Digital Object Identifier no. 10.1109/TVCG.2013.63.

Authorized licensed use limited to: Nanyang Technological University Library. Downloaded on February 15,2026 at 07:56:29 UTC from IEEE Xplore. Restrictions apply.

the given priority values. Our algorithm is accurate as the Poisson disks are uniformly and randomly distributed without bias. Our algorithm also distinguishes itself from the existing approaches by its intrinsic feature, since all the computations are purely based on the intrinsic metric. This embedding-space-independent feature allows us to generate Poisson disk distribution on arbitrary surfaces in $\mathbb{R}^n$. To our knowledge, this is the first *intrinsic*, *accurate* and *parallel* algorithm for generating Poisson disk samples on arbitrary surfaces. Furthermore, by manipulating the spatially varying radius function, we can obtain adaptive sampling easily. The spectrum analysis shows that the proposed algorithm is accurate and the generated Poisson disk distributions faithfully exhibit blue noise patterns.

2 Related Work

2.1 Poisson Disk Sampling in $\mathbb{R}^n$

Dart throwing [3] is a classical algorithm that generates isotropic blue noise samples sequentially one by one. Dunbar and Humphreys [4] presented a spatial data structure in $\mathbb{R}^2$, called scalloped region, which efficiently represents arbitrary boolean operations on 2D disks. Thus, it can be used to encode the regions of space where the insertion of samples is allowed. However, it is unknown whether their data structure can be extended to three or higher dimension. Gamito and Maddock [5] presented an efficient and accurate Poisson disk sampling algorithm for euclidean space of arbitrary dimension. Performing a subdivision refinement of the allowable space for the insertion of new samples, their algorithm has $O(n \log n)$ space and time complexity, where n is the number of samples. Furthermore, the resultant distribution is maximal in the sense that no further samples can be inserted at the completion of the algorithm. Jones and Karger [13] presented a linear algorithm that optimizes the dart throwing using a spatial data structure. Their algorithm partitions the sampling domain into a uniform grid such that each grid can contain at most one sample and maintains a bucket of regions where a point will be generated. At each step of the algorithm, a region is taken from the bucket, a new point is inserted in that region, and nearby regions are updated and possibly added to the bucket. The bucket is only empty when no more points can be added. Ebeida et al. [14] presented an algorithm for distributing Poisson disks in nonconvex domain with guaranteed maximal property. Their algorithm first generates a near-maximal covering of the domain and then generates the polygonal approximations to the voids. Finally, it places the unbiased samples over these voids by calculating the connected components of the remaining uncovered voids. Although the theoretical time complexity is $O(n \log n)$, they showed that nearly $O(n)$ performance for both time and memory is achieved in practice. Using the implicit "flat quadtree," Ebeida et al. [15] presented a practical algorithm for generating maximal Poisson disk sampling in $\mathbb{R}^d$ with low memory cost.

There are also many approximate algorithms which improve the speed by compromising the sampling quality. The hierarchical approaches [10], [9] allow several Poisson-disk distributions with different radii to be generated from a single run of the algorithm. The tile-based approaches [16], [6], [7], [8] generate infinite nonperiodic tilings on the plane at runtime by carefully designed rules. Dunbar and Humphreys [4] presented a fast approximation algorithm with linear time complexity by sacrificing the uniform distribution property. Recently, Wei [11] pioneered a parallel Poisson disk sampling algorithm by subdividing the sample domain into grid cells and drawing samples concurrently from multiple cells that are sufficiently far apart to avoid conflicts. Wei [17] further extended blue noise sampling to multiple classes where each individual class as well as their unions exhibit blue noise characteristics. Using constrained farthest point optimization, Chen and Gotsman [18] presented a parallel algorithm to generate uniformly distributed point patterns with good blue noise characteristics.

Fattal [19] formulated the blue noise sampling problem using a statistical mechanics interacting particle model. This new formulation generates high-quality point distributions, supports spatially varying spatial point density, and runs in time that is linear in the number of points generated. Balzer et al. [20] presented capacity-constrained point distribution, where each point has a capacity determined by the area of its Voronoi region weighted with a given density function. Demanding each point has the same capacity, the generated distribution possesses high-quality blue noise characteristics and adapts precisely to the underlying density function. Feng et al. [21] modeled anisotropic noise as nonoverlapping ellipses whose size and density match a given anisotropic metric and used a generalized anisotropic Lloyd relaxation to distribute samples evenly. Li et al. [22] generalized the traditional isotropic dart throwing and relaxation for the anisotropic setting, and introduced warping and sphere sampling-based approaches to extend Fourier spectrum analysis for adaptive and/or anisotropic samples. Wei and Wang [23] presented differential domain analysis, which allows quantitatively measurement of the spatial and spectral properties of various nonuniform sample distributions.

2.2 Poisson Disk Sampling on Surfaces

The majority of the surface Poisson disk sampling algorithms adopt some spatial data structure that either explicitly partitions or subdivides the surface (and its embedding space) or maintains the not-yet-sampled region where the new samples can be drawn. Fu and Zhou [24] extended Dunbar and Humphreys [4]'s scalloped regions to curved surface. Their method is approximate in that samples are not free to be placed anywhere on the surface with equal probability. By using the fast marching method [25] with spherical wavefront to approximate the geodesic distance, Cline et al. [26] proposed the optimized dart throwing algorithm to generate maximal Poisson disk point sets on 3D surface. Their approach efficiently excludes the areas of the domain that are already covered by existing darts and works directly on surfaces of various types, for example, mesh, subdivision, parametric, and implicit surfaces. However, their method requires sequential computation and thus is not suitable for real-time applications involving large-scale objects. Corsini et al. [27] proposed constrained Poisson disk sampling, which allows generating customized set of points with generic geometric constraints. Bowers et al. [12] presented a parallel algorithm for direct sampling on

Authorized licensed use limited to: Nanyang Technological University Library. Downloaded on February 15,2026 at 07:56:29 UTC from IEEE Xplore. Restrictions apply.

arbitrary surfaces. They used grid cell structure in $\mathbb{R}^3$ [11] to efficiently manage the samples and derived a closed-form formula to approximate the geodesic distance by finding the length of a curve smoothly interpolating the normals of the two end points. They also extended the radial means and anisotropy to curved surfaces using manifold harmonic basis, which provides a way to directly evaluate the spectral distribution quality of surface samples without mesh parameterization. Bowers et al.'s approach [12] is fully parallel and highly efficient, however, it is extrinsic and heavily depends on the embedding space.

There are also some numerical approaches to generate Poisson disk patterns. Using capacity-constrained surface triangulation, Xu et al. [28] proposed a relaxation algorithm, which iteratively alternates the optimization between the points' location and their connectivity. Adopting continuous capacity-constrained Voronoi tessellation, Chen et al. [29] proposed a variational framework generating point distributions precisely adapting to given density functions. Both Xu et al. [28] and Chen et al. [29]'s algorithms are able to generate point distributions with high-quality blue noise characteristics on surfaces. However, as these approaches are based on the numerical solver on a global energy function, their performance is much slower than the non-numerical techniques, such as [26], [12]. Also, it is very hard to parallelize these approaches.

Our method is fundamentally different than the existing approaches. First, our method neither partitions the given surface into small patches nor uses any spatial data structure to maintain the voids in the sampling domain. Thus, our method is very easy to implement. Second, our method is intrinsic and independent of the embedding surface. As a result, it works for arbitrary surfaces in $\mathbb{R}^n$. Third, our method is accurate in that the generated samples are distributed uniformly and randomly without bias. The exact geodesic distance also guarantees the samples are free of conflicts. Fourth, our method is parallel in nature and can make full uses of the available threads. The detailed comparison and discussion are provided in Section 6.

3 Algorithmic Overview

3.1 Notations

The notations used throughout this paper include:

- $M = (V, E, F)$: the input triangle mesh, where V, E, and F are the set of vertices, edges and faces;
- A: the area of M;
- $\mathcal{S}$: the set of Poisson disk samples;
- N: the number of Poisson disk samples, i.e., $N = |\mathcal{S}|$;
- $\mathcal{P}$: the set of dense points representing M;
- T: the total number of threads;
- $d(p, q)$: the geodesic distance between points p and q;
- r: the radius of the Poisson disk, so $d(p, q) \geq 2r$ for arbitrary two Poisson disk samples p and q;
- $D(p, r)$: the geodesic disk of radius r centered at point $p \in M$;
- ρ: radius statistics measuring the packing density of Poisson disks. The ideal range for 2D Poisson disk distribution is between 0.65 and 0.85, as suggested in [30].

3.2 Motivation

The dart throwing algorithm generates the Poisson disk distribution by repeatedly drawing a random point on the surface, checking the point's distance with existing samples, and accepting it if no violation is found. Due to its sequential nature and high computational cost, the brute force algorithm is not practical for interactive applications involving large-scale models. Wei [11] pioneered a parallel dart throwing algorithm that starts by partitioning the domain into grid cells such that the diagonal of each cell is $2r$. For each cell, Wei's algorithm makes up to k trials to draw a random point that satisfies the minimum distance requirement with existing samples in the neighboring cells. To achieve parallelism, the cells are organized into subsets such that all cells in a subset are separated by at least a distance of $2r$ from each other, and thus can be processed in parallel without causing conflicts. Although the sampling inside each group is random, the sequence of groups visited for every resolution level follows a predetermined order, which violates the uniform sampling condition.

Our algorithm is based on a different strategy. It is observed that samples have a uniform random position and a uniform random birth time in the sequential dart throwing algorithm. As time progresses, samples are accepted or rejected based on their distance from the samples that have already been born. We parallelize the classic dart throwing algorithm by assigning a *unique random* value to each sample, which represents the priority or order of the sample. The candidate samples can be processed by multiple threads simultaneously. Each thread checks the collision of the processed sample against its neighbors. When conflicts occur, the samples with the higher priority win and are accepted as Poisson disks.

3.3 Data Structure

Our data structure is as follows:

```
enum SampleStatus{
    IDLE,       //not processed
    ACTIVE,     //being processed by a thread
    ACCEPTED,   //accepted as a Poisson disk
    REJECTED,   //rejected
};
struct Sample {
    SampleStatus status;

    //random number between 0 and 1
    float priority;

    //point coordinates
    vector3d pos;

    //the corresponding mesh triangle
    int face_id;
};
```

Each sample has a time-dependent status and a unique random number to represent its priority. During the runtime, each thread takes one active sample. Let p_i be the active sample that is being processed by the ith thread. To ensure

Authorized licensed use limited to: Nanyang Technological University Library. Downloaded on February 15,2026 at 07:56:29 UTC from IEEE Xplore. Restrictions apply.

that no two active samples have the same priority, the priority of p_i is given by

$$priority(p_i) = \frac{rand() * T + i}{RAND_MAX * T},$$

where T is the total number of threads, and the function rand() returns a pseudorandom integer number in the range 0 to RAND_MAX-1.

We follow Osada et al. [31]'s algorithm to generate random points with respect to the surface area, which are unbiased and insensitive to mesh tessellation. For each triangle, we computed its area and stored it in an array along with the cumulative areas of triangles visited so far. Then, we selected a triangle with probability proportional to its area by generating a random number between 0 and the total cumulative area and performing a binary search on the array of cumulative areas. For each selected triangle with vertices (v_1, v_2, v_3), we constructed a random point $(1-\sqrt{\alpha})v_1 + \sqrt{\alpha}(1-\beta)v_2 + \sqrt{\alpha}\beta v_3$, where $\alpha, \beta \in [0,1]$ are two random numbers.

3.4 Geodesic Distance

To enforce the minimal distance constraint between any two Poisson disks, we need to compute the exact geodesic distance. Among the many classical discrete geodesic algorithms, we choose the improved Chen-Han (ICH) algorithm [32] mainly because of its linear space complexity and high performance. The ICH algorithm partitions each mesh edge into a set of intervals, called windows, which are maintained in a priority queue according to the distance from the source. These windows are propagated across the mesh faces: pops a window from the queue and then computes its children windows which can add, modify, or remove existing windows, and updates the queue accordingly. The original ICH algorithm computes the geodesic distance for all mesh vertices (i.e., the "single-source all-destination" problem) by propagating the windows across *all* faces. As Poisson disk sampling needs only to compute geodesic discs, we slightly modify the ICH algorithm to allow the early termination whenever the wavefront exceeds $2r$, i.e., twice the Poisson disk radius. We implement the ICH algorithm on CUDA 4.2. The priority queue is realized as a conventional heap which supports the window propagation from near to far on GPU threads. The mesh is encoded in the half-edge structure and stored in the GPU's global memory in a "read-only" manner. Each GPU thread maintains its own data (i.e., the source point, the wavefront windows and the priority queue) in its own memory pool. Even though two geodesic disks may overlap, the corresponding GPU threads do not have any data conflicts. Therefore, our framework allows us to compute a large number of geodesic disks simultaneously.

3.5 Resolving Conflicts

For each active sample p_i, the corresponding GPU thread computes a local geodesic disk $D(p_i, 2r)$ of radius $2r$ centered at p_i and collects the active and accepted (if any) samples in $D(p_i, 2r)$. See the pseudocode of function DetectCollision in Algorithm 1. We then use the function CheckStatus (see the pseudo code in Algorithm 1) to check whether p_i can be accepted. If there are no active samples

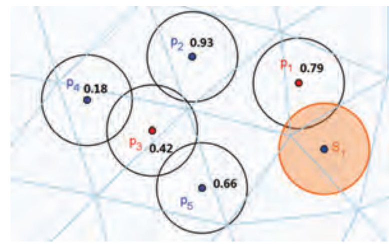


Fig. 1. Illustration of PDTS with five available threads. s_1 is an accepted Poisson disk sample. Each thread handles one active sample p_i, which has a unique and random priority value. PDTS accepts p_2, p_4, and p_5, and rejects p_1 and p_3.

with priority higher than p_i's, p_i is accepted immediately. Otherwise, the function CheckStatus recursively checks whether conflicts occur among the nearby active samples. p_i is rejected if there is any accepted Poisson disk in $D(p_i, 2r)$. Since there are only limited active samples within $D(p_i, 2r)$ and the predefined priority of each sample is unique, such recursive check will terminate in finite (in practice, only a few) steps. See Fig. 1 for an example of our *recursive* strategy to resolve conflicts and Table 1 for the statistical analysis of its performance.

Algorithm Parallel Dart Throwing on Surfaces (PDTS)

```
global Mesh M                    //Triangle mesh
global list<Sample> S[|F|]       //S[i] is the list of accepted samples at i-th triangle
global list<Sample> A[T]         //A[i] is the list of active samples colliding with p_i
global list<Sample> D[T]         //D[i] is the list of accepted samples colliding with p_i

function ParallelDartThrowing(r, N)
  //r: Poisson disk radius
  //N: number of Poisson disk samples

  S ← ∅
  while |S| < N do
    parallel for each thread i from 1 to T
      p_i.pos ← random point
      p_i.status ← ACTIVE
      p_i.priority ← (rand()*T+i) / (RAND_MAX*T)
    parallel end for
    parallel for each thread i from 1 to T
      // find the accepted and active samples in geodesic disk D(p_i, 2r)
      {D[i], A[i]} ← DetectCollision(p_i, 2r)
      if D[i] ≠ ∅
        p_i.status ← REJECTED
        A[i] ← ∅
      end if
    parallel end for
    parallel for each thread i from 1 to T
      CheckStatus(p_i)
      if p_i.status == ACCEPTED then
        atomic S[p_i.face_id].samples.insert(p_i)
      end if
    parallel end for
  end while
  return S

function CheckStatus(p_i)
  if atomic p_i.status ≠ ACTIVE then
    return
  end if
  for each q ∈ A[i]
    if q.priority > p_i.priority then
      CheckStatus(q)
      if q.status == ACCEPTED then
        atomic p_i.status ← REJECTED
        return
      end if
    end if
  end for
  atomic p_i.status ← ACCEPTED
  return

function DetectCollision(p_i, 2r)
  //Compute the geodesic disk centered at p_i using the ICH algorithm;
  //Collect the ACTIVE and ACCEPTED samples when wavefronts propagate;
  //Terminate the ICH algorithm when the geodesic disk radius exceeds 2r;
  //Return the collected ACTIVE and ACCEPTED samples.
```

Authorized licensed use limited to: Nanyang Technological University Library. Downloaded on February 15,2026 at 07:56:29 UTC from IEEE Xplore. Restrictions apply.

TABLE 1
Statistics of the Number of Recursive Calls for CheckStatus() on Gargoyle

	# of calls for CheckStatus()									
N/T	1	2	3	4	5	6	7	8	9	≥ 10
5	75.94%	21.45%	2.16%	0.15%	0.13%	0.09%	0.03%	0.02%	0.02%	0.01%
10	84.27%	14.25%	0.81%	0.52%	0.07%	0.04%	0.02%	0.01%	0.01%	0%
20	91.39%	8.30%	0.15%	0.14%	0.02%	0.01%	0%	0%	0%	0%
40	95.32%	4.58%	0.06%	0.04%	0%	0%	0%	0%	0%	0%

Columns show the percentage of the number of function calls.

4 Parallel Poisson Disk Sampling on Surfaces

To facilitate our explanation, we first propose a simple algorithm of parallel dart throwing on surfaces (PDTS) in Section 4.1. PDTS is an accurate algorithm in that it generates randomly and uniformly distributed samples on surfaces at runtime. However, PDTS is not efficient, since it throws a large number of darts, but only accepts a small percentage of them. Then we present an improved algorithm, parallel sampling from dense points (PSDP), in Section 4.2. PSDP first converts the input mesh into a set of dense points $\mathcal{P}$, and then randomly selects the samples from $\mathcal{P}$ and processes them in a similar fashion as PDTS. Once a sample is accepted, all the samples within the geodesic disk of radius $2r$ are rejected. The algorithm terminates when all predefined points have been processed. PSDP is an accurate and efficient algorithm that works for arbitrary surfaces. We present PDTS only for explanation purposes, since the strategy of conflict resolving in PDTS is similar to that of PSDP. Note that all the experimental results reported in Section 5 are actually based on PSDP.

4.1 Parallel Dart Throwing on Surfaces

Starting from an empty set $\mathcal{S}$, our PDTS algorithm iteratively updates the Poisson disk samples. For each iteration, PDTS randomly generates T samples on surfaces, where T is the number of threads. Each sample is marked as *active* and is given a unique random number for the priority. Let p_i denote the ith active sample that is being processed by the ith thread. For each p_i, the corresponding thread computes the geodesic disk $D(p_i, 2r)$ by the CUDA-based ICH algorithm. When the wavefronts propagate, the thread collects the active and accepted samples. The ICH algorithm terminates when the wavefront touches a point with geodesic distance of more than $2r$. If an accepted sample has been found inside this geodesic disk, the current active sample p_i is rejected immediately, since it conflicts with an existing Poisson disk. Otherwise, the thread recursively checks the conflicts among p_i and its nearby active samples. The one with the highest priority wins and is added to the Poisson disk sample set $\mathcal{S}$. It should be noted that all the threads proceed in parallel, and the accepted samples are independent of the order of the threads.

Fig. 1 shows an example of PDTS. Assume the current Poisson disk sample set $\mathcal{S}$ contains one accepted sample s_1 and there are five threads available. PDTS generates five active samples, p_i, $i = 1, \ldots, 5$, at random locations, each of which has a unique and random priority value. All circles in Fig. 1 are of the radius r. The ith thread computes a geodesic disk $D(p_i, 2r)$ centered at p_i with radius $2r$ and collects the active samples inside $D(p_i, 2r)$. Let $\mathcal{A}[i]$ (resp. $\mathcal{D}[i]$) denote the list of active (resp. accepted) samples colliding with p_i. Then, $\mathcal{D}[1] = \{s_1\}$, $\mathcal{D}[2] = \cdots = \mathcal{D}[5] = \mathcal{A}[1] = \emptyset$, $\mathcal{A}[2] = \{p_3\}$, $\mathcal{A}[3] = \{p_2, p_4, p_5\}$, $\mathcal{A}[4] = \{p_3\}$, and $\mathcal{A}[5] = \{p_3\}$.

The sample p_1 is immediately rejected since it collides with an accepted sample s_1, i.e., $\mathcal{D}[1] \neq \emptyset$. Both p_2 and p_5 are accepted, because they do not have conflicted neighbors with higher priority. To determine p_3's status, the third thread invokes CheckStatus(p_3) and checks all elements in $\mathcal{A}[3]$ which have higher priority than p_3. Thus, it will recursively call CheckStatus(p_2) or CheckStatus(p_5), and accepts p_2 or p_5, depending on which element appears earlier in $\mathcal{A}[3]$. For either case, p_3 is rejected. To determine p_4's status, the fourth thread invokes CheckStatus(p_4), which will recursively call CheckStatus(p_3), since p_3 has higher priority than p_4. However, since p_3 is rejected, the fourth thread accepts p_4.

Here, we should mention that the final accepted samples are independent of the order of the threads. To illustrate this, let's consider the third and fourth threads. Depending on their order, there are three cases:

- Case 1: thread 3 is executed before thread 4. Thread 3 sets p_3's status to *REJECTED*. Then thread 4 invokes CheckStatus(p_4), which will call CheckStatus(p_3), since $p_3 \in \mathcal{A}[4]$ has higher priority than p_4. As p_3 has already been rejected, CheckStatus(p_4) set $p_4.status$ to *ACCEPTED*.
- Case 2: thread 4 is executed before thread 3. Thread 4 invokes CheckStatus (p_4), which will call CheckStatus (p_3), since p_3 has higher priority than p_4. The function CheckStatus(p_3) will reject p_3, since its priority is lower than that of p_2 and p_5. Thus, CheckStatus (p_4) sets $p_4.status$ to *ACCEPTED*. Later, when the third thread calls CheckStatus(p_3) again, the function simply returns, since p_3 is not active any more.
- Case 3: thread 3 and thread 4 are executed simultaneously. Thread 3 invokes CheckStatus(p_3) directly and thread 4 invokes CheckStatus(p_4) which in turns calls CheckStatus(p_3). Both threads reject p_3, i.e., $p_3.statu$ is set to *REJECTED* twice.[1] Then p_4 is accepted.

Thus, even though each thread may be executed at different time frames, the accepted samples are totally determined by their priority values and independent of the order of the threads.

4.2 Parallel Sampling from Dense Points

PDTS randomly throws darts on surface in parallel, and abandons the darts that are within $2r$ distance of existing darts. This is not efficient for two reasons: first, PDTS has to compute the geodesic disk for *every* sample; second, a large number of trials does not contribute to the results.

To improve the performance of PDTS, we adopt a different strategy for parallelizing the sampling process in PSDP: given a mesh model M, PSDP converts it to dense point clouds by randomly and uniformly throwing a large number of darts on the surface. To obtain a high-quality

1. The read/write operation of a sample's status is atomic.

Authorized licensed use limited to: Nanyang Technological University Library. Downloaded on February 15,2026 at 07:56:29 UTC from IEEE Xplore. Restrictions apply.

distribution, we require that the presampled points should be at least 10 times the number of Poisson disks, i.e., $|\mathcal{P}| \geq 10N$ (see more discussions in Section 5). Each generated point is initialized as an idle sample. For each iteration in the runtime, PSDP randomly picks T idle samples and updates their status to ACTIVE. Similar to PDTS, each thread takes one active sample, say p_i, computes a geodesic disk $D(p_i, 2r)$ and finds the active and idle samples inside it. The status of p_i is determined by CheckStatus(), which recursively checks the conflicts among the *active* samples inside $D(p_i, 2r)$. If p_i is accepted, then all the idle samples inside $D(p_i, 2r)$ are rejected. PSDP repeats the iterations until all idle points have been processed. PSDP is more efficient than PDTS, since each accepted sample will result in the removal of all idle samples within geodesic distance $2r$. Therefore, no active sample conflicts with the accepted samples in PSDP (see Fig. 2).

Algorithm Parallel Sampling from Dense Points (PSDP)

```
global Mesh M                          //Triangle mesh
global list<Sample> S[|F|]             //S[i] is the list of accepted samples at i-th triangle
global vector<Sample> P                //Dense point clouds of M
global vector<int>PointIndex           //The point indices
global list<Sample> A[T]               //A[i] is the list of active samples colliding with p_i
global list<Sample> I[T]               //I[i] is the list of idle samples colliding with p_i

function ParallelSamplingFromDensePoints(r, L)
  //r: Poisson disk radius
  //L: number of points in |P|

  P ← GenerateDensePoints(L)
  parallel PointIndex ← 1 . . . L
  parallel Randomly shuffle PointIndex
  // partition the PointIndex into T groups, i.e., group_i, i = 1, ··· , T
  for each i from 1 to T
    group_i.range ← ((i-1)/T L, i/T L]
    // the index of the next IDLE sample in group_i
    group_i.curr_index ← (i-1)/T L + 1
  end for
  S ← ∅

  repeat
    parallel for each thread i from 1 to T
        while P[PointIndex[group_i.curr_index]].status ≠ IDLE do
          group_i.curr_index ← group_i.curr_index + 1
          Terminate this thread if group_i.curr_index exceeds its range
        end while
        p_i ← P[PointIndex[group_i.curr_index]]
        p_i.status ← ACTIVE
        p_i.priority ← (rand()*T+i) / (RAND_MAX*T)
    parallel end for
    parallel for each thread i from 1 to T
        //Find the active and idle samples colliding with p_i
        {A[i], I[i]} ← DetectCollision(p_i, 2r)
    parallel end for
    parallel for each thread i from 1 to T
    //Check conflicts among the active samples colliding with p_i
        CheckStatus(p_i)
        if p_i.status == ACCEPTED
          atomic S[p_i.face_id].insert(p_i)
          for each q ∈ I[i]
            atomic q.status ← REJECTED
          end for
        end if
    parallel end for
  until no more IDLE points
  return S

function GenerateDensePoints(L)
  parallel for each i from 1 to L
      p_i.pos ← random point on M
      p_i.status ← IDLE
      P[i] ← p_i
      Insert p_i into the sample list of the corresponding triangle
  parallel end for
  return
```

As shown in Fig. 6, the entry in the array PointIndex is a pointer to the dense point clouds $\mathcal{P}$. To ensure the accepted samples are randomly and uniformly distributed on M, we randomly shuffle the array PointIndex, and then partition PointIndex into T equally sized groups, which allows parallel processing by T threads. It should be noted that that our partition is *not* spatial dependent in that the points in each group are not spatially related, i.e., they are located randomly on M. This distinguishes our algorithm from the existing parallel sampling algorithms [11], [12], which explicitly partition the spatial domain.

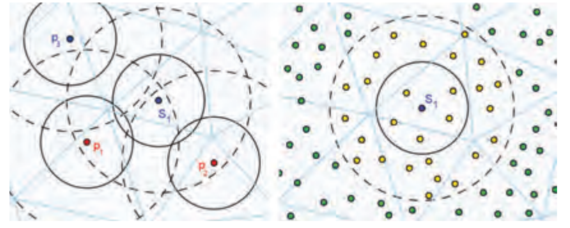


Fig. 2. Comparison between PDTS (left) and PSDP (right). Assume s_1 is an accepted sample. In PDTS, each thread generates one active sample at a random location. Thus, it is likely that the active samples collide with the accepted samples. For example, p_1 and p_2 are rejected due to the collision with s_1, and p_3 is accepted since it does not conflict with any accepted and active samples. Although being parallel, this strategy is not efficient, since two geodesic disks must be computed for p_1 and p_2, respectively. However, p_1 and p_2 do not contribute to the results. By contrast, PSDP randomly picks samples from the predefined dense points $\mathcal{P}$. When s_1 is accepted, PSDP rejects all the idle samples (in yellow) inside $D(s_1, 2r)$. Therefore, in the future iterations, PSDP will pick only the samples (in green) that are outside $D(s_1, 2r)$. In other words, no active samples collide with the accepted samples in PSDP.

4.3 Correctness

We show that our algorithms, PDTS and PSDP, are accurate, i.e., the generated samples are randomly and uniformly distributed in the domain.

Given a sample domain $\mathcal{D}$, Poisson disk sampling is a set $X = \{x_i \in D; i = 1, 2, \ldots, N\}$ of N samples having the properties:

$$\forall x_i \in X, \forall M \subseteq \mathcal{D} : P(x_i \in M) = \int_M dx, \tag{1}$$

$$\forall x_i, x_j \in X : \|x_i - x_j\| \geq 2r, \tag{2}$$

where $P(\cdot)$ is a conditional probability.

The first condition states that a uniformly distributed random sample x_i of X has a probability of falling inside a subset M of $\mathcal{D}$ that is equal to the hypervolume of M. The second condition requires that any two samples are $2r$ apart from each other.

An accurate Poisson disk sampling algorithm must satisfy both conditions. Clearly, our approaches satisfy the second condition, since exact geodesic distances are used. Next, we will show the first condition also holds in PDTS and PSDP.

Let $\mathcal{S} = \{s_1, \ldots, s_k\}$ be a set containing k Poisson disk samples generated by the brute-force dart throwing algorithm. Let p_i, $i = 1, \ldots, T$, be points that are uniformly and randomly distributed on D. If we randomly shuffle the indices $\{1, \ldots, T\}$, and then add the shuffled T points *sequentially* into $\mathcal{S}$ by checking the minimal distance constraint, the resultant set is obviously a Poisson disk set, since the process is the same as a sequential dart throwing. Observe that each trial has a unique order, which is the index of the point.

In PDTS, we assign each sample p_i a *unique* and *random* priority value, which is equivalent to randomly shuffle the

Authorized licensed use limited to: Nanyang Technological University Library. Downloaded on February 15,2026 at 07:56:29 UTC from IEEE Xplore. Restrictions apply.

indices $\{1, \ldots, T\}$. Therefore, PDTS can make T trials *simultaneously*, and the resultant set $\mathcal{S}$ contains uniformly and randomly distributed Poisson disks.

In PSDP, we construct a dense point clouds $\mathcal{P}$, such that the points are randomly and uniformly distributed on $\mathcal{D}$. If we randomly choose T points $p_i \in \mathcal{P}$, $i = 1, \ldots, T$, and make T trials simultaneously, this is equivalent to PDTS algorithm. Note that PSDP randomly shuffles the point indices $\{1, \ldots, |\mathcal{P}|\}$ and then partitions them into T equally sized groups. During the runtime, each thread picks one index from each group. Thus, we only need to show this selection is unbiased.

Let g_i be the number of *available* idle samples in ith group, and $I = \sum_{i=1}^{T} g_i$ be the total number of *available* idle samples in all groups. Without the group partition, the probability for any idle sample being selected is $1/I$. With the group partition, the probability for an arbitrary idle sample q being located in the ith group is g_i/I. Also, the probability for q being selected by the ith thread is $1/g_i$. Therefore, the probability for q being selected is also equal to $1/I$, which implies that the partition based selection is unbiased.

Putting it altogether, PSDP is also accurate which distributes Poisson disks randomly and uniformly on $\mathcal{D}$.

5 Experimental Results and Discussions

5.1 Performance

We implemented our PSDP algorithm on CUDA 4.2, which supports recursion for device function. We tested it on a PC with an Intel Xeon 2.66-GHz CPU and 12-GB memory. The graphics card is an NVIDIA GTX 580 with 512 cores and 1.5-GB memory. Our PSDP program allows the user to specify the Poisson disk radius r and the number of presampled points $L = |\mathcal{P}|$. The program terminates when there is no active samples.

We observed that the sampling speed depends on the number of Poisson disk samples N (see Fig. 5a). Among all the functions in PSDP, DetectCollision() is computationally expensive, since it computes a geodesic disk of radius $2r$ using the ICH algorithm [32] with time complexity $O(n^2 \log n)$, where n is the number of mesh vertices in the geodesic disk. When N is small, the Poisson disk radius r is big, thus, it takes a longer time for DetectCollision() to compute the geodesic disk. Conversely, it takes less time to compute the geodesic disk when N is big and r is small. Thus, the sampling speed increases when N is increasing. However, when N is comparable or exceeds the number of mesh vertices $|V|$, the radius of the geodesic disk is similar to or less than the average edge length. As a result, the time for computing the geodesic disk is insignificant and remains roughly constant. Hence, the sampling speed does not depend on N anymore. We also observed that the sampling speed is closely related to the ratio $|\mathcal{P}|/|\mathcal{S}|$ (see Fig. 5b), since the greater number of predefined points in $\mathcal{P}$, the greater number of idle samples collided; thus, the more time taken for function DetectCollision() to collect the idle samples.

5.2 Quality

We calculated the radius statistics $\rho = r\sqrt{2\sqrt{3}N/A}$, which is a widely used criteria to measure the packing density (thus, the quality) of Poisson disk distribution in $\mathbb{R}^2$ [30]. Such measurement is also adopted in other surface sampling papers [12] [22] [28]. In all cases, our ρ values fall between 0.70 and 0.76, which is within the ideal range of 0.65-0.85 recommended by Lagae and Dutre [30]. As shown in Fig. 5c, the radius statistics ρ of our distributions depends on the number of presampled points $|\mathcal{P}|$. Specifically, the more points in $|\mathcal{P}|$, the larger pool of samples we can choose from, thus, then the higher value of ρ and more Poisson disk samples we can obtain. But the radius statistics ρ grows very slowly when $|\mathcal{P}|/|\mathcal{S}| > 80$. See also Fig. 14 for the plot of ρ in a 2D distribution. As the sampling speed is inversely proportional to the ratio $|\mathcal{P}|/|\mathcal{S}|$, we usually set $|\mathcal{P}|/|\mathcal{S}| = 20$ in our experiments for the tradeoff between quality and performance.

Using our geodesic program, we can easily visualize the Poisson disks on surfaces (see Figs. 9 and 3), which allows us to visually check the sampling quality. We compared our method with the state-of-the-art technique [12]. Our algorithm measures the exact distance between two samples, while Bowers et al. [12] approximate the geodesic distance by a smooth curve interpolating the normals of two endpoints. They derived a closed form formula that produces exact geodesic distances for the plane and sphere, which works well for smooth surfaces, such as Bunny. However, for surfaces with rich details, such as Gargoyle and Lion, their approximation has larger errors, such that some samples violate the minimal distance constraints in the generated distribution (see the highlighted boxes in Fig. 9). Our method, in contrast, is guaranteed to generate conflict-free Poisson disk distributions.

5.3 Spectrum Analysis

We applied the spectrum analysis method in [12] to verify our method. As required in [12], we tessellated each model to $|V| = 100{,}000$ vertices and then computed the first $10{,}000$ eigenvectors of the Laplacian matrix. For comparison, we generated 300-500 samples on surfaces by using our method, [12] and the brute-force dart throwing, which is considered as the ground truth. We then evaluated the radial means and anisotropy[2] of the samples power spectrum averaged over 10 runs. Our algorithm is accurate in that it distributes Poisson disks randomly and uniformly on surfaces. As expected, the plots of both the radial mean and anisotropy of our method are highly consistent with the brute-force dart throwing in all examples (see Fig. 4). Bowers et al. [12] embed the surface into $\mathbb{R}^3$ and partitions the domain into grid cells of size $\frac{2r}{\sqrt{3}}$, which are organized into phase groups such that all cells in a phase group are separated by at least a distance of $2r$ from each other. This allows sampling in parallel without causing conflicts. Although the sampling inside each group is random, the sequence of groups visited follows a predetermined order. Therefore, the generated distribution violates the uniform sampling condition because samples within a group cannot be placed until all previous groups have been sampled. We tested our method and the phase group method [12] by using the exact geodesic [32] and the approximate geodesic

2. As in [12], the generated anisotropy plots are centered around the -7.0 dB line instead of the -10 dB line, since the basic functions in manifold harmonics transformation are the real forms of the analytic Fourier basis, which causes a factor 2 of the variance of the power spectrum.

Authorized licensed use limited to: Nanyang Technological University Library. Downloaded on February 15,2026 at 07:56:29 UTC from IEEE Xplore. Restrictions apply.

Fig. 3. Experimental results. For each model we generate three density levels, at approximately 1K, 5K, and 10K sample points. The images are generated at high resolution to allow for zoom-in examination.

used in [12]. As shown in Fig. 5, the spectra of our method are consistent with the blue noise patterns in all examples. In contrast, the anisotropy plots of phase group methods (with and without the exact geodesics), deviate from the ground truth, which implies that such deviation in the anisotropy of the phase group method does *not* come from the distance metric used.

5.4 Efficiency of Our Conflicts Resolving Strategy

Function CheckStatus() plays a key role in our parallel Poisson disk sampling algorithm, since it converts the sequential dart throwing process into a parallel collision detection. Each thread will invoke CheckStatus() for the corresponding active sample, say p_i. If there is an accepted sample inside its geodesic disk $D(p_i, 2r)$, p_i is rejected immediately. Otherwise, CheckStatus() recursively checks whether conflicts have occurred among its active neighbors. The recursive function stops when a winner is identified.

Consider the worst case that the priorities of all samples are in ascending order, i.e., $p_i.priority < p_{i+1}.priority$ and p_i conflicts with p_{i+1}, $i = 1, \ldots, T-1$. Then the ith thread invokes CheckStatus(p_i), which will recursively call CheckStatus($p_{i+1}, \ldots$, until CheckStatus(p_T) to identify the winner p_T. Therefore, the number of calls of CheckStatus() for each thread is of complexity $O(T)$. However, the probability of such a scenario is extremely low in practice. In fact, the number of calls of CheckStatus() depends on the number of samples N and the number of active samples T. The bigger value of N, the smaller radius r, the smaller probability of conflicts among active samples. Let c be the average number of calls of CheckStatus() for each thread. Our statistical results show that $c = 1.09$ when $N/T = 20$, and more than 91 percent function calls returns immediately, since there are no active samples collided (see Table 1). We observed the value of c remains roughly the same for all models, since PSDP converts all models to

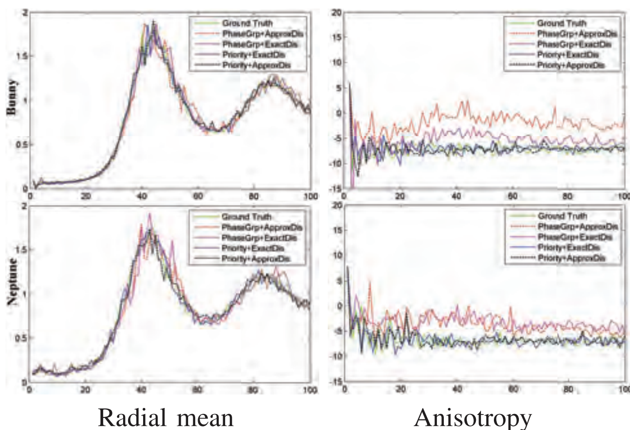


Fig. 4. We applied the manifold harmonics transformation [12] for spectral analysis on surfaces. The horizontal axis is the basis frequency, the vertical axes are the radial mean and the anisotropy. We tested both phase group method and our priority based parallelization with approximate and exact geodesic, respectively. The spectra of our method are consistent with the blue noise patterns in all examples, while the anisotropy plots of the phase group methods (with the exact geodesic [32] and the approximate geodesic [12]) deviate from the ground truth (in green). This implies that such bias of the phase group method does *not* come from the distance metric used.

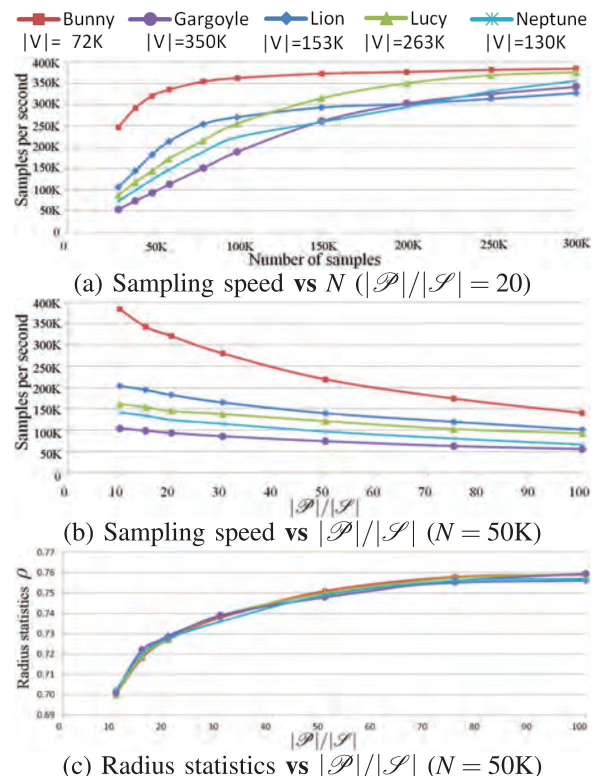


Fig. 5. Performance and quality of our algorithm.

Authorized licensed use limited to: Nanyang Technological University Library. Downloaded on February 15,2026 at 07:56:29 UTC from IEEE Xplore. Restrictions apply.

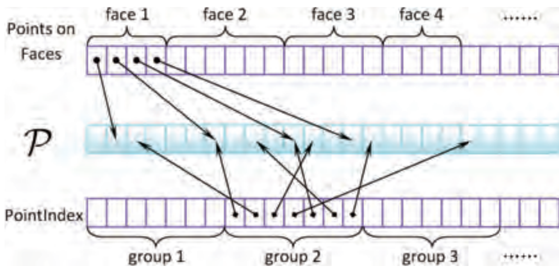


Fig. 6. Data structure of PSDP. The array $\mathcal{P}$ contains the dense points, each point has a random location on M. To speed up the data access by the geodesic algorithm, we insert each sample p_i into the triangle containing it. The array PointIndex is a random permutation of $\{1, \ldots, |\mathcal{P}|\}$, which is partitioned into T equally sized groups.

dense point clouds before sampling, thus the performance is nearly uniform regardless of the mesh complexity.

5.5 Intrinsic Algorithm

Our algorithm is intrinsic in the sense that it depends on the metric rather than the surface embedding, since all distances are measured intrinsically by the exact geodesic algorithm [32]. In contrast, [12] is an extrinsic approach, since it partitions the ambient space $\mathbb{R}^3$ to form the phase groups and approximates the geodesic distance by computing a curve that smoothly interpolates the normals of two end points. Furthermore, our partition method is purely based on the surface area and does not require each partitioned group to be connected, which is fundamentally different from the space or surface segmentation strategies used in other surface sampling algorithms [12], [24]. Therefore, our algorithm works for surfaces in the arbitrary dimension. Fig. 10 shows examples of intrinsic sampling on surfaces in $\mathbb{R}^4$.

Fig. 10 (row 2) also shows two poses of a lion model, which have the same mesh connectivity and are nearly isometric. As our method is intrinsic to the geometry, we can distribute Poisson disks on one pose and the shape correspondence naturally induces the sampling on the other pose, which is guaranteed to be valid, i.e., uniformly distributed and free of conflicts.

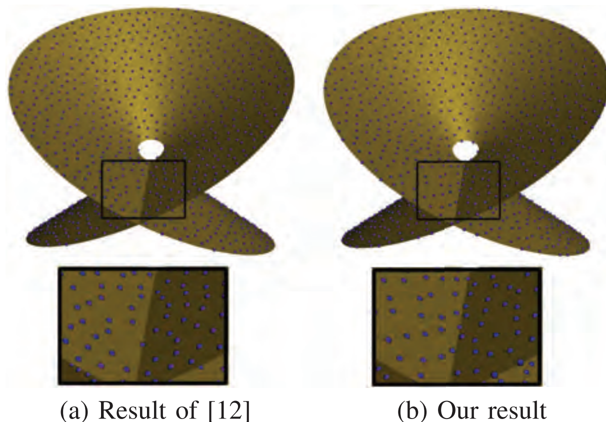


Fig. 7. For the samples near the self-intersection, the extrinsic approach [12] may report "false" conflicts between valid samples since their spatial distance is small. Our method can accurately check the conflicts using the intrinsic geodesic distance, thus, resulting in the valid distribution.

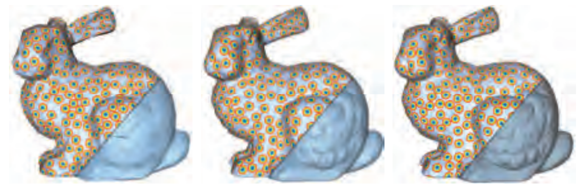


Fig. 8. Our method is insensitive to the mesh resolution and tessellation. From left to right: distributing 700 Poisson disks on Bunny with 1.4K, 14K, and 144K faces, respectively.

5.6 Mesh Size Dependence

It is also worth noting that our algorithm maintains the input mesh only for computing the exact geodesic distance. Since the performance of the discrete geodesic algorithm [32] depends on mesh size, so does our algorithm. For applications where the geodesic accuracy is not a top priority, one could adopt Bowers et al.'s [12] closed-form formula to approximate the geodesic distance. As a result, the input mesh can be discarded once the dense points $\mathcal{P}$ are generated, and then the performance of our algorithm becomes *independent* of the mesh size.

5.7 Robustness

Our method converts the input model into a dense point clouds by Osada et al. [31], which randomly generates points with respect to surface area, i.e., the number of points on a mesh triangle is proportional to its area. Moreover, the exact geodesic distance algorithm [32] is intrinsic to the shape geometry and insensitive to mesh resolution and tessellation, so is our algorithm. See Fig. 8 for sampling on Bunny model with various resolutions.

5.8 Adaptive Sampling

Our algorithm can be easily extended for adaptive sampling, where the Poisson disk radius r is guided by a spatially varying function $g : V \rightarrow [0, 1]$ on the surface, where 0 means dense sampling (small radius) and 1 means sparse sampling (large radius). Let r_{max} be maximal radius of a Poisson disk, specified by the user. Therefore, the

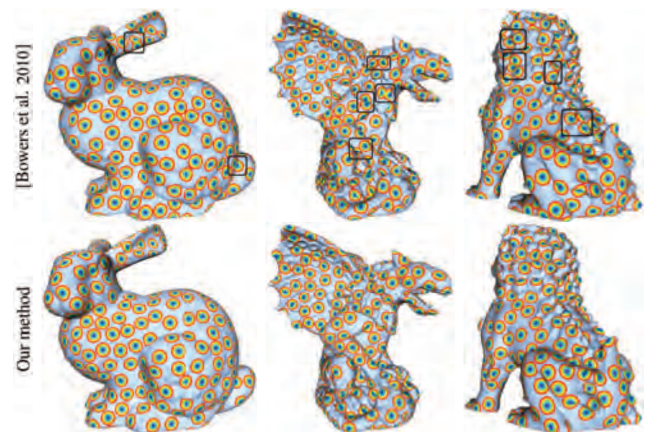


Fig. 9. Sampling quality. We generate samples by using the state-of-the-art technique [12] and our method. To visualize the results, we compute the geodesic disks centered at each sample with radius r. Due to the approximate geodesic distance in [12], some samples violate the minimal distance constraints, as highlighted in the black boxes. Our method measures the exact geodesic distance, thus, can guarantee that the generated distribution is free of conflicts.

Authorized licensed use limited to: Nanyang Technological University Library. Downloaded on February 15,2026 at 07:56:29 UTC from IEEE Xplore. Restrictions apply.

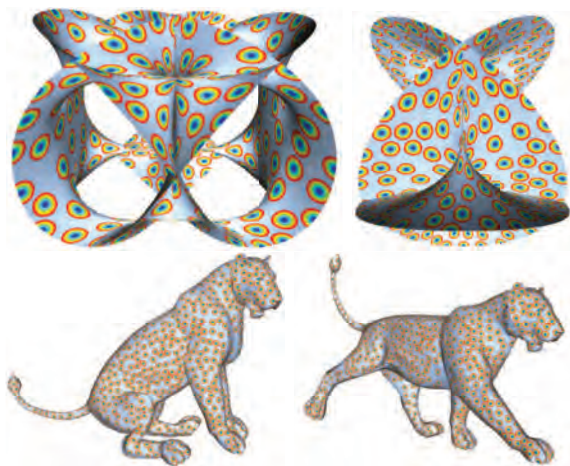


Fig. 10. Our algorithm is independent of the surface embedding. Row 1: two examples of Poisson disk sampling on 4D surfaces. For rendering purpose, the surfaces are projected into $\mathbb{R}^3$; Row 2: we distribute Poisson disks on one pose and the shape correspondence induces the sampling on the other pose. The intrinsic property of our algorithm guarantees the induced Poisson disk distribution is uniformly distributed and free of conflicts.

Poisson disk centered at p has radius $g(p)r_{max}$. Two samples $p, q \in M$ conflict if their geodesic distance $d(p,q) < (g(p) + g(q))r_{max}$. For each active sample p_i, we compute a geodesic disk $D(p_i, (g(p_i)+1)r_{max})$ centered at p_i and with radius $(g(p_i)+1)r_{max}$. Note that this radius is conservative to ensure all the conflicted samples are included. Then for each active/idle sample $q \in D(p_i, (g(p_i)+1)r_{max})$, we detect the collision by checking $d(p_i, q) < (g(p_i) + g(q))r_{max}$. Fig. 11 shows the adaptive sampling guided by user-specified radius function.

5.9 Sampling in $\mathbb{R}^n$

Our algorithm also works for euclidean space of arbitrary dimensions. Given a unit hypercube H in $\mathbb{R}^n$, we maintain a grid structure of hypercells, each has side length $\frac{2r}{\sqrt{n}}$. Then we randomly generate points in each hypercell to make the dense point set $\mathcal{P}$. During the runtime, for each active sample p_i, our algorithm detects the collision of p_i in its neighboring cells. Similar to [12], the predefined points $\mathcal{P}$ are stored in a hash table. The geodesic distance in function DetectCollision() is simply the euclidean distance in $\mathbb{R}^n$. See Fig. 12 for results in $\mathbb{R}^2$ and $\mathbb{R}^3$.

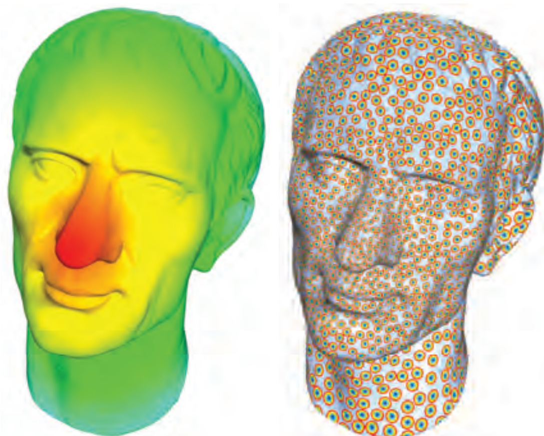


Fig. 11. Adaptive sampling on Caesar model. Left: the spatially varying function specified by the user, where red means dense sampling (small radius) and green means sparse sampling (large radius); Right: the adaptive sampling result.

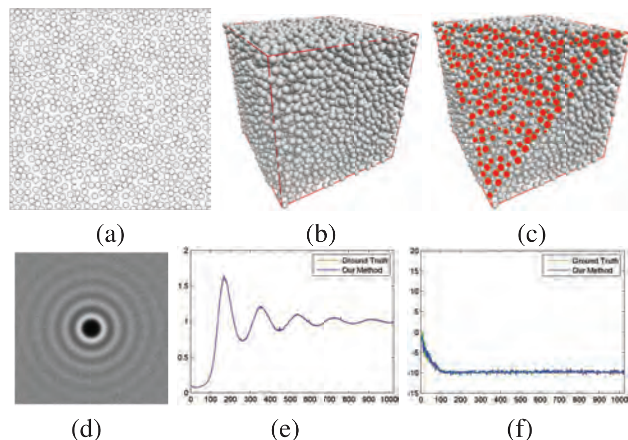


Fig. 12. Our algorithm also works for euclidean space of arbitrary dimensions. (a) 2D case, $r = 0.0147$, # samples $= 787$. (b) 3D case, $r = 0.023$, # samples $= 6064$. (c) Cut view of (b). (d)-(f) The average power spectrum for a 2D case with $r = 0.00268$ and 21,886 samples, the radial power spectrum and the anisotropy spectrum.

6 Comparison and Discussions

The existing algorithms are clearly divided into groups of surface and euclidean space sampling, due to the difference between the segmentation of $\mathbb{R}^n$ and curved surfaces, which also results in different spatial data structures. Table 2 compares our method and the existing Poisson disk sampling algorithms. In the following, we elaborate the key difference between our technique and the existing approaches.

6.1 Comparison to the Phase Group Method [11], [12]

The phase group method, pioneered by Wei [11], is the first technique for parallel Poisson disk sampling. By subdividing the sampling domain into grid cells, darts can be thrown concurrently into multiple cells that are sufficiently far apart to avoid conflicts. The phase group method is highly efficient and can be applied to euclidean space of arbitrary dimension, which makes it very attractive in practice. However, as pointed out in [5], the Poisson disk distribution generated by the phase group method is approximate since the sequence of processing the phase groups follows a predefined order. Although such bias can be reduced by increasing the grid resolution, it will increase the computational cost as well. Our method is proven to be equivalent to the classical dart throwing. As a result, it guarantees to generate Poisson disk distribution without bias. Furthermore, our method does not require the explicit space partitioning. This intrinsic feature makes our method flexible and can be applied to both euclidean space $\mathbb{R}^n$ and arbitrary surfaces easily.

Bowers et al. [12] extended the phase group method to curved surface by subdividing the embedding space. Although their method works fairly well for general models in graphics community, this space depending property may reduce their performance and compromise the quality.

Authorized licensed use limited to: Nanyang Technological University Library. Downloaded on February 15,2026 at 07:56:29 UTC from IEEE Xplore. Restrictions apply.

TABLE 2
Comparison to Other Surface Sampling Algorithms

Method	Sampling Domain	Intrinsic	Parallel	Accurate	Maximal	Numerical solver	Spatial partition	Performance
Dart throwing	$\mathbb{R}^n$	-	No	Yes	No	No	No	Slow
Dunbar and Humphreys [4]	$\mathbb{R}^2$	-	No	Yes	Yes	No	No	Fair
Gamito and Maddock [5]	$\mathbb{R}^n$	-	No	Yes	Yes	No	Yes	Fair
Wei [11]	$\mathbb{R}^n$	-	Yes	No	No	No	Yes	Very Good
Chen and Gotsman [18]	$\mathbb{R}^2$	-	Yes	No	Yes	No	Yes	Good
Ebeida et al. [14]	$\mathbb{R}^2$	-	No	Yes	Yes	No	Yes	Good
Ebeida et al. [15]	$\mathbb{R}^n$	-	Yes	Yes	Yes	No	Yes	Good
Cline et al. [26]	Surfaces in $\mathbb{R}^n$	Yes	No	Yes	Yes	No	Yes	Fair
Fu and Zhou [24]	Surfaces in $\mathbb{R}^n$	Yes	No	No	Yes	No	No	Slow
Bowers et al. [12]	3D surfaces	No	Yes	No	No	No	Yes	Very Good
Chen et al. [29]	Surfaces in $\mathbb{R}^n$	Yes	No	No	No	Yes	No	Fair
Xu et al. [28]	Surfaces in $\mathbb{R}^n$	Yes	No	No	No	Yes	No	Slow
Corsini et al. [27]	3D Surfaces	No	No	Yes	No	No	Yes	Fair
Our method	Surfaces in $\mathbb{R}^n$	Yes	Yes	Yes	No	No	No	Very Good

For example, if the input model has very complicated geometry, the uniform space partition does not lead to a uniform partition of the points inside each cell, i.e., some cells may have significantly larger points than others. Such unbalance would reduce the performance. Our method does not require the space partition. Instead, it simply organizes the sample points by their indices.

Fig. 7 shows another scenario where the input surface is self-intersected. The phase group method may organize spatially-close-but-topologically-far points (imagine the points near the intersection and are on different sheets) into the same cell. Due to the "false" conflicts, Bowers et al.'s method results in fewer samples near the intersected region than our method.

We observed that computing the exact geodesic distance is the bottleneck of our surface sampling framework, while Bowers et al. [12] adopted a simple closed-form formula to approximate geodesics. To make a fair comparison of the performance, we tested our priority based parallelization with the approximate geodesic distance [12]. As shown in Fig. 13b, the performance of our priority-based parallelization is about 85-95 percent of the phase group method, which demonstrates that our method is fairly efficient.

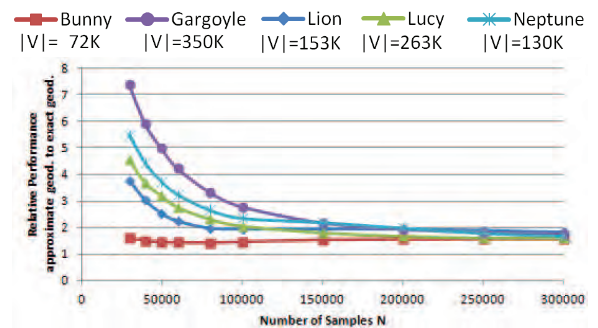
We also compared the performance of our algorithm to Wei's algorithm [11]. Our implementation on NVIDIA GTX 580 is able to generate 13M 2D samples/second and 1.5M 3D samples/second, while Wei's algorithm produces 15M 2D and 1.7M 3D samples/second, respectively.

6.2 Comparison to the Optimization-Based Techniques [28], [29]

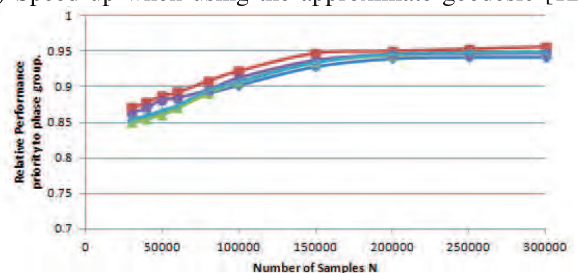
From quite a different perspective, Chen et al. [29] and Xu et al. [28] formulated the Poisson disk sampling on surface into an optimization problem. Both approaches are intrinsic and independent of the ambient space. Although the spectrum analysis shows the patterns generated by Chen et al. [29] and Xu et al. [28] exhibit blue noise characteristics, these approaches are only approximate, since the samples not distributed in a fully random and uniform manner. Furthermore, both approaches are computational expensive due to solving the centroidal Voronoi tessellation [29] and computing capacity-constrained triangulation. Furthermore, it is unclear whether [29] and [28] can be implemented in parallel. Our method is simple and robust, and significantly faster than [29], [28].

6.3 Comparison to Hierarchical Dart Throwing [27]

Corsini et al. elegantly extended the hierarchical dart throwing [33] to 3D surface by subdividing the 3D space to reduce the sampling domain during the sample generation. Both our method and theirs adopt the presampling strategy, i.e., pre-generate a suitable set of samples on the mesh and then remove samples from the pool. The difference is fourfold: first, their approach requires an explicit partition



(a) Speed up when using the approximate geodesic [12].



(b) Relative performance.

Fig. 13. (a) To make a fair comparison of the performance to [12], we adopt the approximate geodesic distance into our framework, and observe a significant boost the performance when N is small. However, such speedup is not significant when the sample number is very high, since each geodesic disk becomes very small. (b) The relative performance of our priority-based parallelization to the phase group method [12]. Our method is about 5-15 percent slower than their method due to the fact that some computing samples may be rejected in our approach.

Authorized licensed use limited to: Nanyang Technological University Library. Downloaded on February 15,2026 at 07:56:29 UTC from IEEE Xplore. Restrictions apply.

of the 3D space, so it is extrinsic, while ours is intrinsic and can be applied to surfaces in $\mathbb{R}^n$. Second, their approach does not support parallel computing, while our method does. Third, their method is very flexible to generate constrained Poisson disk distribution, but our method cannot. Last, once pregenerated the samples on the mesh, their method then discards the mesh completely. They follow Bowers et al's closed form formula to approximate the geodesic distance between two samples. As a result, the time complexity of their approach is independent of the mesh size. Our method still maintains the mesh to compute the exact geodesic distance, thus, the runtime performance of our method depends on the mesh complexity, especially when the number of Poisson disks $\mathcal{S}$ is small. However, such dependence becomes very loose when $\mathcal{S}$ is large, due to the small radius of the Poisson disk, i.e., the geodesic algorithm quickly stops if the covered area is beyond the Poisson disk radius r. See also the performance curve plot in Fig. 5a.

6.4 Comparison to Maximal Poisson Disk Sampling [15]

Ebeida et al. [15] presented a simple yet effective method to generate maximal Poisson disk sampling in $\mathbb{R}^n$ in a parallel manner. Their idea is to maintain an implicit flat quadtree data structure to keep track of uncovered regions of the domain. The grid cells are further refined (subdivided) during the iteration. A cell is discarded if it is fully covered by a sample. The algorithm terminates when no more active cells remain. Ebeida et al.'s algorithm is theoretically sound and also practical in dimension d up to 5 (serial) and 3 (parallel).

Our method is different than [15] in two aspects. First, although Ebeida et al. [15] work well for euclidean space, it would be very difficult to extend their idea to curved surfaces, since there is no natural way to maintain such *grid* cells as well as *quadtree* tessellation on arbitrary surfaces. One may argue to adopt the strategy used in [12] which converts the surface sampling problem into a special case of *space* sampling. However, as mentioned above, such conversion is extrinsic and space dependent, which may fail for models with complicated geometry/topology and is not suitable for surfaces in high-dimensional space either.

Second, the main advantage of their algorithm with respect to our is the guaranteed maximal property of the generated Poisson disk distribution. We applied our algorithm to $\mathbb{R}^2$ and computed the radius statistics ρ to measure how far our distribution is from the maximal distribution. To obtain the ground truth, we applied their program to a unit square with $r = 0.005$ and run it for 10 times. We found the values of ρ fall in the range $[0.778, 0.780]$ with average 0.7791. Then we applied our program with various number of presampled points $|\mathcal{P}|$. For each configuration, we also run our program 10 times and obtained the average ρ. As shown in Fig. 14, the value of our ρ (the blue curve) tends to approach the ground truth (the red line) when increasing $|\mathcal{P}|$. This result is not a surprise, since the more number of predefined samples, the larger pool our algorithm can choose from, thus, more closer the generated distribution to the maximal distribution.

6.5 Limitations

First, although it can make full use of all the available GPU threads to obtain maximal performance, our algorithm has to synchronize all threads three times for each iteration.

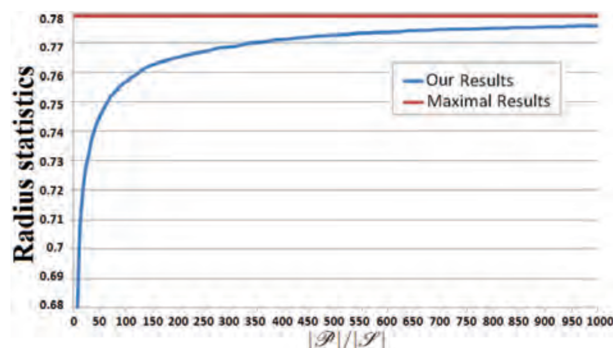


Fig. 14. The distribution generated by our method is not maximal. For the Poisson disk sampling on a unit square with $r = 0.005$, the radius statics ρ of the maximal sampling algorithm [15] is very stable, all values fall in the range [0.778, 0.780] with an average 0.7791. Our ρ depends on the dense-points-to-Poisson-disks ratio $|\mathcal{P}|/|\mathcal{S}|$.

Our experimental results show that each synchronization usually takes 0.04-0.05 milliseconds. Note that this overhead is independent of the number of threads T. The number of iterations for PSDP to terminate is in the range of $[|\mathcal{S}|/T, |\mathcal{P}|/T]$. Therefore, the total number of synchronization is between $3|\mathcal{S}|/T$ and $3|\mathcal{P}|/T$. The number of synchronization required for [12] is gl, where g is the number of phase groups (e.g., $3 \times 3 \times 3$), and l is the number of trials in each phase group (e.g., 10). Thus, the overhead in [12] is independent of the sampling complexity and GPU configuration, and much less than ours. Second, our method converts the input model to dense point clouds on surface M. To obtain high-quality sampling, we usually require the ratio $|\mathcal{P}|/|\mathcal{S}| \geq 20$, which results in high memory costs. Thus, our method is limited by the available physical memory of the graphics card. Third, as mentioned before, our method does not generate *maximal* distribution at the completion of the algorithm and it is difficult to control the exact number of samples generated.

7 Conclusion

This paper presents an accurate and parallel algorithm for Poisson disk sampling. Our contributions are threefold. First, we propose a new technique for parallelizing the dart throwing. Rather than the conventional approaches that spatially partition the sampling domain to generate the samples in parallel, our approach assigns each sample candidate a random and unique priority that is unbiased with regard to the distribution, and organizes the samples by their indices. Hence, multiple threads can process the candidates simultaneously and resolve conflicts by checking the given priority values. Second, our algorithm is accurate as it distributes the Poisson disks without bias. Third, our framework is general since it works for both euclidean space of any dimension and arbitrary surfaces embedded in $\mathbb{R}^n$, and all computations are based on the intrinsic metric and independent of the embedding space. By manipulating the spatially varying density function, we can obtain adaptive sampling easily. To our knowledge, this is the first *intrinsic*, *accurate* and *parallel* algorithm for generating Poisson disks on arbitrary surfaces.

Authorized licensed use limited to: Nanyang Technological University Library. Downloaded on February 15,2026 at 07:56:29 UTC from IEEE Xplore. Restrictions apply.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their constructive comments and Prof. Rui Wang for sharing his codes of parallel Poisson disk sampling and spectrum analysis. This work was supported by an MOE Tier1 grant. All correspondence should be addressed to Ying He.

REFERENCES

[1] Y. Xiang, S.-Q. Xin, Q. Sun, and Y. He, "Parallel and Accurate Poisson Disk Sampling on Arbitrary Surfaces," *Proc. ACM SIGGRPAH ASIA 2011 Technical Sketches*, article no. 18, 2011.
[2] M.A.Z. Dippé and E.H. Wold, "Antialiasing through Stochastic Sampling," *Proc. ACM SIGGRAPH '85*, pp. 69-78, 1985.
[3] R.L. Cook, "Stochastic Sampling in Computer Graphics," *ACM Trans. Graphics*, vol. 5, pp. 51-72, 1986.
[4] D. Dunbar and G. Humphreys, "A Spatial Data Structure for Fast Poisson-Disk Sample Generation," *ACM Trans. Graphics*, vol. 25, pp. 503-508, 2006.
[5] M.N. Gamito and S.C. Maddock, "Accurate Multidimensional Poisson-Disk Sampling," *ACM Trans. Graphics*, vol. 29, pp. 8:1-8:19, 2009.
[6] A. Lagae and P. Dutré, "A Procedural Object Distribution Function," *ACM Trans. Graphics*, vol. 24, pp. 1442-1461, 2005.
[7] J. Kopf, D. Cohen-Or, O. Deussen, and D. Lischinski, "Recursive Wang Tiles for Real-Time Blue Noise," *Proc. ACM SIGGRAPH '06*, 2006.
[8] V. Ostromoukhov, "Sampling with Polyominoes," *ACM Trans. Graphics*, vol. 26, 2007.
[9] D.P. Mitchell, "Spectrally Optimal Sampling for Distribution Ray Tracing," *Proc. ACM SIGGRAPH '91*, pp. 157-164, 1991.
[10] M. McCool and E. Fiume, "Hierarchical Poisson Disk Sampling Distributions," *Proc. Conf. Graphics Interface*, pp. 94-105, 1992.
[11] L.-Y. Wei, "Parallel Poisson Disk Sampling," *ACM Trans. Graphics*, vol. 27, 2008.
[12] J. Bowers, R. Wang, L.-Y. Wei, and D. Maletz, "Parallel Poisson Disk Sampling with Spectrum Analysis on Surfaces," *ACM Trans. Graphics*, vol. 29, 2010.
[13] T.R. Jones and D.R. Karger, "Linear-Time Poisson-Disk Patterns," arXiv:1107.3013v1, 2011.
[14] M.S. Ebeida, A.A. Davidson, A. Patney, P.M. Knupp, S.A. Mitchell, and J.D. Owens, "Efficient Maximal Poisson-Disk Sampling," *ACM Trans. Graphics*, vol. 30, pp. 49:1-49:12, 2011.
[15] M.S. Ebeida, S.A. Mitchell, A. Patney, A.A. Davidson, and J.D. Owens, "A Simple Algorithm for Maximal Poisson-Disk Sampling in High Dimensions," *Computer Graphics Forum*, vol. 31, no. 2, pp. 785-794, 2012.
[16] M.F. Cohen, J. Shade, S. Hiller, and O. Deussen, "Wang Tiles for Image and Texture Generation," *Proc. ACM SIGGRAPH '03*, 2003.
[17] L.-Y. Wei, "Multi-Class Blue Noise Sampling," *ACM Trans. Graphics*, vol. 29, 2010.
[18] R. Chen and C. Gotsman, "Parallel Blue-Noise Sampling by Constrained Farthest Point Optimization," *Computer Graphics Forum*, vol. 31, no. 5, pp. 1775-1785, 2012.
[19] R. Fattal, "Blue-Noise Point Sampling Using Kernel Density Model," *ACM Trans. Graphics*, vol. 30, no. 4, 2011.
[20] M. Balzer, T. Schlömer, and O. Deussen, "Capacity-Constrained Point Distributions: A Variant of Lloyd's Method," *ACM Trans. Graphics*, vol. 28, no. 3, pp. 86:1-86:8, 2009.
[21] L. Feng, I. Hotz, B. Hamann, and K. Joy, "Anisotropic Noise Samples," *IEEE Trans. Visualization and Computer Graphics*, vol. 14, no. 2, pp. 342-354, Jan. 2008.
[22] H. Li, L.-Y. Wei, P.V. Sander, and C.-W. Fu, "Anisotropic Blue Noise Sampling," *ACM Trans. Graphics*, vol. 29, 2010.
[23] L.-Y. Wei and R. Wang, "Differential Domain Analysis for Non-Uniform Sampling," *ACM Trans. Graphics*, vol. 30, 2011.
[24] Y. Fu and B. Zhou, "Direct Sampling on Surfaces for High Quality Remeshing," *Proc. ACM Symp. Solid and Physical Modeling (SPM '08)*, pp. 115-124. 2008.
[25] J.A. Sethian, "A Fast Marching Level Set Method for Monotonically Advancing Fronts," *Proc. Nat'l Academy of Sciences of USA*, vol. 93, pp. 1591-1595, 1996.
[26] D. Cline, S. Jeschke, A. Razdan, K. White, and P. Wonka, "Dart Throwing on Surfaces," *Computer Graphics Forum*, vol. 28, no. 4, pp. 1217-1226, 2009.
[27] M. Corsini, P. Cignoni, and R. Scopigno, "Efficient and Flexible Sampling with Blue Noise Properties of Triangular Meshes," *IEEE Trans. Visualization and Computer Graphics*, vol. 18, no. 6, pp. 914-924, Jan. 2012.
[28] Y. Xu, R. Hu, C. Gotsman, and L. Liu, "Blue Noise Sampling of Surfaces," *Computers and Graphics*, vol. 36, no. 4, pp. 232-240, 2012.
[29] Z. Chen, Z. Yuan, Y.-K. Choi, L. Liu, and W. Wang, "Varational Blue Noise Sampling," *IEEE Trans. Visualization and Computer Graphics*, vol. 18, no. 10, pp. 1784-1796, Oct. 2012.
[30] A. Lagae and P. Dutre, "A Comparison of Methods for Generating Poisson Disk Distributions," *Computer Graphics Forum*, vol. 27, pp. 114-129, 2008.
[31] R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin, "Shape Distributions," *ACM Trans. Graphics*, vol. 21, pp. 807-832, 2002.
[32] S.-Q. Xin and G.-J. Wang, "Improving Chen and Han's Algorithm on the Discrete Geodesic Problem," *ACM Trans. Graphics*, vol. 28, no. 4, pp. 1-8, 2009.
[33] K.B. White, D. Cline, and P.K. Egbert, "Poisson Disk Point Sets by Hierarchical Dart Throwing," *Proc. IEEE Symp. Interactive Ray Tracing (IRT '07)*, pp. 129-132, 2007.

Xiang Ying received the BS degree in software engineering from Tianjin University, China. He is currently working toward the PhD degree with the School of Computer Engineering, Nanyang Technological University, Singapore. His research interests include general purpose GPU computing, computational geometry, and computer graphics.

Shi-Qing Xin received the MS and PhD degrees in applied mathematics from Zhejiang University, China, in 2009. Since 2009, he has been working at Nanyang Technological University as a research fellow. His research interests include computational geometry, computer graphics, and topology analysis.

Qian Sun received the BS degree in pharmacy from Tianjin University, China. She is currently working toward the PhD degree with the School of Computer Engineering, Nanyang Technological University, Singapore. Her current research interests include human-computer interaction and computer graphics.

Ying He received the PhD degree in computer science from Stony Brook University, New York. He is currently an associate professor at the School of Computer Engineering, Nanyang Technological University. His research interests include the general areas of visual computing. He is particularly interested in the problems which require geometric analysis and computation. For more information, visit http://www.ntu.edu.sg/home/yhe.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.

Authorized licensed use limited to: Nanyang Technological University Library. Downloaded on February 15,2026 at 07:56:29 UTC from IEEE Xplore. Restrictions apply.