Parallel Poisson Disk Sampling with Spectrum Analysis on Surfaces

John Bowers* Rui Wang* Li-Yi Wei† David Maletz*

*University of Massachusetts Amherst †Microsoft Research

Abstract

The ability to place surface samples with Poisson disk distribution can benefit a variety of graphics applications. Such a distribution satisfies the blue noise property, i.e. lack of low frequency noise and structural bias in the Fourier power spectrum. While many techniques are available for sampling the plane, challenges remain for sampling arbitrary surfaces. In this paper, we present new methods for Poisson disk sampling with spectrum analysis on arbitrary manifold surfaces. Our first contribution is a parallel dart throwing algorithm that generates high-quality surface samples at interactive rates. It is flexible and can be extended to adaptive sampling given a user-specified radius field. Our second contribution is a new method for analyzing the spectral quality of surface samples. Using the spectral mesh basis derived from the discrete mesh Laplacian operator, we extend standard concepts in power spectrum analysis such as radial means and anisotropy to arbitrary manifold surfaces. This provides a way to directly evaluate the spectral distribution quality of surface samples without requiring mesh parameterization. Finally, we implement our Poisson disk sampling algorithm on the GPU, and demonstrate practical applications involving interactive sampling and texturing on arbitrary surfaces.

Keywords: Poisson disk sampling, manifold surface, parallel computation, mesh Laplacian, spectrum analysis, GPU

1 Introduction

Surface sampling can benefit a variety of graphics applications, such as texturing [Turk 2001; Wei and Levoy 2001; Lagae and Dutré 2005], remeshing [Turk 1992; Alliez et al. 2002; Qu and Meyer 2006], subsurface scattering [Jensen and Buhler 2002], global illumination [Cheslack-Postava et al. 2008; Ritschel et al. 2009], non-photorealistic rendering [Meier 1996], and point-based rendering [Grossman and Dally 1998]. These applications often desire surface samples with a uniform and random distribution. Such a distribution satisfies the blue noise property, i.e. lack of low frequency noise and structural bias in the Fourier power spectrum.

A rich literature of work exists for generating blue noise distributions on the plane. However, it remains difficult to efficiently sample arbitrary non-planar surfaces. Several existing algorithms distribute surface samples using point repulsion [Turk 1992], stratified sampling [Nehab and Shilane 2004], dart throwing [Cline et al. 2009], or pre-computed data sets [Ostromoukhov 2007; Li et al. 2008]. Many of these techniques are either computationally expensive or require surface parameterization, and hence are not suitable for applications involving real-time dynamic or deformable objects with potentially complex shapes.

Figure 1: *A ray traced image showing Poisson disk samples computed using our algorithm on the dragon model. Our algorithm performs at 180,000 sample points per second on complex models.*

With respect to spectrum analysis, the standard Fourier transform has been successfully applied in examining the spectral quality of the sample distribution [Lagae and Dutré 2008]. These methods, however, are only available for planar sampling, and are not yet applicable to sample distributions on arbitrary mesh surfaces.

In this paper we present new methods for Poisson disk sampling with spectrum analysis on surfaces. Our first contribution is a parallel dart throwing algorithm for efficient Poisson disk sampling on arbitrary manifold surfaces. Dart throwing [Cook 1986] is a classical method for generating Poisson disk distributions, in which the samples are randomly located but remain at least a minimum distance r away from each other. Although it is only one way of blue noise sampling, it exhibits excellent distribution quality and is simple and amenable for acceleration. For planar sampling, efficient dart throwing can be achieved using a repertoire of acceleration [Jones 2006; Dunbar and Humphreys 2006; White et al. 2007] and parallelization [Wei 2008] techniques. For surface sampling, Cline et al. [2009] introduces an optimized dart throwing algorithm, but it requires sequential computation. We present the first data-parallel algorithm for Poisson disk sampling on surfaces. Our main idea is to sample the surface into a dense point cloud, and then draw Poisson disk samples from within the set. For this we extend the parallel grid cell sampling approach in [Wei 2008] from Euclidean to geodesic distance metric; since the latter cannot be smaller than the former, we can draw samples directly on surfaces without any parameterization. To account for geodesic distance, we propose a fast approximation that is easy to compute and accurate for close-by sample points.

Our second contribution is a new method for analyzing the spectral distribution quality of surface samples. In planar sampling, the distribution quality is typically measured in terms of the radial means and anisotropy of the Fourier power spectrum [Lagae and Dutré 2008]. We extend these concepts to arbitrary manifold surfaces by employing spectral mesh basis functions, derived as the eigenfunctions of the discrete mesh Laplacian operator [Levy 2006]. These functions define a Fourier-style basis set that exists on the mesh surface, and hence can be used to evaluate the power spectrum of samples distributed over the mesh. We show that regardless of the surface shape and topology, the ideal radial means and anisotropy of the power spectrum are consistent with well-known results obtained in planar sampling. Because the spectral mesh basis obeys surface geodesic distance and does not require any mesh parametrization, it provides a convenient way to study and compare the quality of different surface sampling algorithms.

Finally, we present an implementation of our sampling algorithm

on modern GPUs. Our implementation achieves 180,000 samples computed per second using approximated geodesic distance. This is close to an order of magnitude faster than the state of the art in [Cline et al. 2009]. Since we represent the surface as a large point cloud, our algorithm is flexible with respect to different surface representations, and its speed is insensitive to the mesh complexity. Therefore our algorithm is suitable for applications involving dynamically changing or deformable geometry.

2 Background

Blue noise sampling. A blue noise distribution refers to a sample set that has a uniform and unbiased distribution in the spatial domain as well as absence of low frequency noise and structured bias in the frequency domain. Due to its desirable spatial and spectral properties, blue noise sampling has been widely employed and received significant research attention [Lloyd 1982; Cook 1986; Mitchell 1987; McCool and Fiume 1992; Ostromoukhov et al. 2004; Jones 2006; Dunbar and Humphreys 2006; Kopf et al. 2006; Ostromoukhov 2007; White et al. 2007; Wei 2008; Fu and Zhou 2008; Balzer et al. 2009; Cline et al. 2009].

Surface sampling. Among the numerous techniques for blue noise sampling, we are particularly interested in surface sampling, which can benefit a variety of graphics applications in texturing, rendering, remeshing, and point-based graphics. However, many existing methods are either slow or rely on pre-computed data sets (e.g. point hierarchy [Pastor et al. 2003], parameterization [Alliez et al. 2002], or dual parameterization [Li et al. 2008]) and are thus unsuitable for applications that require fast computation on dynamic/deformable geometry. To overcome these limitations, we aim to provide a surface sampling method that is both parallel (and thus can run fast on parallel processors such as the GPU), and also capable of computing samples on the fly on arbitrary surfaces without parametrization. Our method draws inspirations mainly from the following prior work: [Nehab and Shilane 2004] for the use of voxel grid to store samples, [Wei 2008] for parallel dart throwing, and [Fu and Zhou 2008; Cline et al. 2009] for surface sampling under geodesic distance metric.

Quality analysis. As discussed in [Ulichney 1987] and [Lagae and Dutré 2008], there are two primary methods for measuring the quality of blue noise sample distribution on Euclidean planes: one is spatial uniformity via the relative radius, and the other is Fourier spectrum analysis. Given a set of N samples $\{\mathbf{p}^k\}_{k=0 \text{ to } N-1}$, its Fourier spectrum $F(\mathbf{f})$ ($\mathbf{f}$ being the frequency) can be computed using the standard Fourier transform. Then, three quantities can be derived: the power spectrum/periodogram $|F(\mathbf{f})|^2$, and the corresponding radial mean and anisotropy. A sample set that satisfies the blue noise criteria must exhibit lack of low frequency noise in the radial mean and absence of any structural bias (i.e. a low and flat anisotropy). While the spatial uniformity can be easily extended to Riemannian surfaces, the power spectrum is non-trivial to extend because ordinary Fourier basis does not apply to Riemannian surfaces. Li et al. [2010] extended Fourier power spectrum analysis to sphere surfaces via spherical harmonics. We provide a further extension to arbitrary manifold surfaces using the spectral mesh basis.

Mesh Laplacian and spectral basis. The spectral mesh basis [Karni and Gotsman 2000; Levy 2006], derived from the discrete mesh Laplacian, provides a set of Fourier-style basis functions for manifold surfaces. As such, classical Fourier analysis can be easily extended to functions defined over the surface of a mesh. Due to its flexibility and generality, the spectral mesh basis has found many applications in computer graphics. These applications include mesh compression [Karni and Gotsman 2000], remeshing [Dong et al. 2006], morphing [Alexa 2002], and watermarking [Praun et al. 1999]. Our work is the first to show the application of the spectral basis on analyzing surface sample distributions. By using the spectral basis, we found that the ideal radial means and anisotropy for general surface sampling are consistent with familiar examples in planar sampling.

There are several practical issues for efficiently computing such a basis, including the cost of solving a large matrix eigenvalue problem, and the associated numerical stability problems. [Karni and Gotsman 2000] reduced the cost of constructing the basis by subdividing a large input mesh into smaller partitions, thus deriving basis functions for each partition individually. [Vallet and Lévy 2008] introduced a spectral shift method that is suitable for computing a large number of basis functions at significantly reduced cost. [Dyer et al. 2007] investigated the spectral robustness of different mesh Laplacians with respect to the mesh connectivity and tessellation.

3 Parallel Uniform Sampling on Surfaces

Given a surface $\mathcal{M}$, Poisson disk sampling computes a set of samples $\mathcal{S}$ that are randomly distributed on the surface but remain a minimum distance of r away from each other. Mathematically, this means $d(s_i, s_j) \geq r, \forall\, s_i, s_j \in \mathcal{S}$, where d is a given distance metric. Brute force dart throwing generates such a sample set by repeatedly drawing a random point on the surface, checking the point's distance to existing samples, and accepting it if no violation is found. Due to its high computation cost, brute force dart throwing is only feasible for generating a small set of samples.

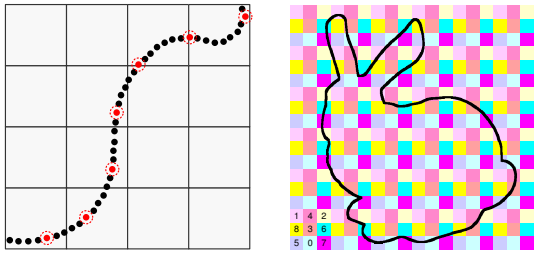
We propose a parallel dart throwing algorithm for efficient Poisson disk sampling on surfaces. In this section we describe our algorithm for uniform surface sampling, where the radius r is uniform everywhere on the surface. We start with Euclidean distance, then extend it to geodesic distance using a fast approximation. For simplicity we only consider triangle mesh surfaces, but other surface types can be easily incorporated. We do not require mesh parametrization.

Our algorithm builds upon [Wei 2008], which performs parallel dart throwing in a continuous n-dimensional Euclidean space. Using 3D as an example: their algorithm starts by partitioning the domain (unit cube) into grid cells (voxel) of size $\frac{r}{\sqrt{3}}$, thus the diagonal of each cell is r. Given the minimum distance requirement, each cell can contain at most one Poisson disk sample. For each cell, the algorithm makes up to k trials to draw a random point that satisfies the minimum distance requirement with existing samples in the neighboring cells. If all k trials are rejected, the cell is left empty. To achieve parallelism, they organize cells into subsets called *phase groups*. All cells in a phase group are separated by at least a distance of r from each other, thus can be processed in parallel without causing conflicts. They process each group sequentially, and use a random ordering of the groups to reduce bias.

3.1 Sampling with Euclidean Distance

Sample space. To perform sampling on surfaces, assuming Euclidean distance, we can use the same grid cell structure as above. However, we must make sure that the random points at each cell are drawn on the mesh surface. One possible solution is to maintain a list of triangles that potentially overlap with each cell, then generate trial points by uniformly sampling the triangles. However, as the triangles are often bigger than the cell size, many generated trial points will be outside the cell and hence be discarded.

Our solution is to compute a large set of initial random points $\mathcal{S}_i$ by uniformly sampling the entire surface, then partition these points to grid cells, and finally draw Poisson disk samples from the points contained in each cell. Thus we have $\mathcal{S} \subset \mathcal{S}_i$. Essentially this means our sample space is a dense set of points that discretely represents the surface. As long as $\mathcal{S}_i$ is sufficiently large and its distribution is uniform with respect to surface area, no additional bias



(a) (b)

Figure 2: *(a) shows that our Poisson disk samples (red) are drawn from the discrete random points assigned to each cell; (b) shows a 2D example of 3×3 phase groups with random ordering.*

will be introduced in the subsequent Poisson disk sampling process.

To generate $\mathcal{S}_i$ for a triangle mesh, we use a standard algorithm that repeatedly selects a triangle with probability proportional to its area, and then uniformly sample the selected triangle using barycentric coordinates computed as $u = 1 - \sqrt{\xi_1}, v = \xi_2\sqrt{\xi_1}$. Here $\xi_1, \xi_2 \in [0, 1]$ are two uniform random numbers. This step can be easily parallelized since each point is independently computed. We store each point's position, triangle id, and barycentric coordinates in a single 1D array. We could also apply stratified sampling, which budgets a certain number of samples per group of triangles, based on the total surface area. This distributes samples more uniformly, which benefits the Poisson disk sampling step. For simplicity we currently do not apply stratified sampling.

Once $\mathcal{S}_i$ is generated, the triangle mesh is no longer needed. Thus our algorithm is flexible w.r.t. different surface types: all we need is a generator that provides uniform random (i.e. white noise) samples on the surface.

Grid partition. Our next step is to partition the points $\mathcal{S}_i$ into grid cells. To do so, we build a 3D grid around the bounding box of $\mathcal{S}_i$, using $\frac{r}{\sqrt{3}}$ as the grid cell size. For each point in $\mathcal{S}_i$, we compute which cell it belongs to, and assign it the cell id. We then use a parallel global sorting to sort the points according to their cell ids, so that those belonging to the same cell are stored together in the resulting array. Note that since the points are initially randomly generated on the surface, after sorting their spatial positions remain in random order within each cell.

A cell is valid (non-empty) if there is at least one point assigned to it; otherwise it is invalid (empty). For each valid cell, we keep track of its starting point in the sorted $\mathcal{S}_i$ array, so that we can access the list of random points assigned to it. Note that because a mesh surface has 2D topology, only a small percentage of the cells are valid. Invalid cells do not contain any actual point and hence are never generated, stored, or processed. For efficient access to the sparse valid cells, we use a hash table described later.

Parallel sampling by phase groups. Once the grid partitioning is completed, we can then perform sampling in each cell. As described in [Wei 2008], the key to enabling parallel sampling is to allow multiple cells, organized as phase groups, to be sampled concurrently. Each phase group contains a set of regularly spaced cells that are sufficiently far apart and thus will not cause conflicts when processed in parallel. Figure 2(b) shows an example of the phase groups. Cells with the same color belong to the same phase group. Again, only valid cells of each phase group will be processed.

To create the phase groups, we simply sort all valid cells by their phase group ID. A random ordering of groups is used to reduce sampling bias. It can be shown in 3D space, the minimum number of phase groups is 3×3×3. Increasing the number of phase groups will help reduce sampling bias, but it also reduces the number of cells per phase group, thus decreasing parallelism.

function $hashtable \leftarrow$ ParallelUniformSurfaceSampling($\mathcal{S}_i$, r, k)
```
// S_i: a dense set of random points sampled on the surface
// k: maximum number of trials per grid cell
// n: sample space dimensionality
μ ← r/√n          // cell size
box ← BoundingBox(S_i)
grid ← ⌈(box.max − box.min)/μ⌉^n
parallel CalculateCellId(S_i, box, μ)
parallel SortPointCloudByCellId(S_i)
parallel hashtable ← InitializeHashtable(S_i)
{p} ← parallel CalculatePhaseGroupID(S_i)
{p} ← parallel SortPhaseGroupID({p})
foreach trial t from 1 to k
  foreach phase group p ∈ {p}
    parallel foreach cell id c ∈ p
      cell ← hashtable.search(c)
      if S_i[cell.first_index + t].cellid ≠ cell.id
        break    // no more random points for the cell
      s ← S_i[cell.first_index + t]
      conflict ← false
      foreach neighboring cell id c_j
        if cell_j ← hashtable.search(c_j) is not null
          if distance(s, cell_j.sample) < r
            conflict←true
            break
      if conflict == false
        cell.sample ← s
    parallel end
  end
end
```

Program 1: *Pseudo-code for parallel uniform surface sampling under Euclidean distance. Extension for geodesic distance is in Section 3.2.*

In contrast to sampling in continuous space, we will draw samples from the discrete set of random points assigned to each cell, as shown in Figure 2(a). Program 1 lists our sampling algorithm, which is similar to [Wei 2008]. For each valid cell in a phase group, we select a trial sample from its list of random points, and check its conflicts with existing samples in the neighboring cells. If no conflict is found, the point is stored as a successful sample. The algorithm then proceeds to the next phase group. As in [Wei 2008], we perform the k loop outside the phase groups. Note, however, we do not perform multi-resolution sampling as suggested in their work. There are two reasons. One is computation efficiency: multi-resolution sampling would require re-assigning the initial random points $\mathcal{S}_i$ to grid cells at each different resolution level, which can cause significant slowdown. The other reason is that in practice, applying the k loops outside the phase groups is already quite effective at eliminating the bias caused by a fixed grid resolution.

Storing cells using hash table. Since valid cells are sparse, we use a hash table to store them and their associated Poisson disk samples. We use the global cell id as a hash key, and a modulo operator as the hash function: *hash_idx = key % hash_table_size*. Our hash table size is 3∼4 times the total number of valid cells. To handle hash collision, we allocate $N_b = 5$ buckets for each hash entry. Cells mapped to the same entry are stored sequentially in the buckets. Figure 3 shows a diagram of the hash table. Each bucket stores the cell id, a pointer to the beginning of the random point list assigned to the cell, and a Poisson disk sample if it has any. Due to sparsity, the total amount of storage for the entire hash table is reasonable (∼50 MB) even for a high resolution grid. While we could use a more sophisticated parallel hashing algorithm such as [Alcantara et al. 2009], our simple hash table is easy to implement and works well in practice.

The hash table is created after the initial random points $\mathcal{S}_i$ are sorted into grid cells. To do so, for every point in parallel, we check its

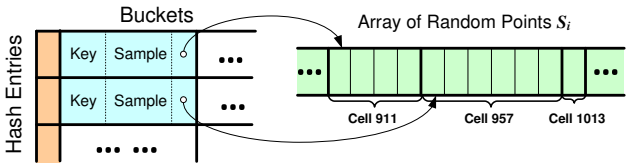


Figure 3: *Our hash table that stores the samples for valid cells.*

cell id against its previous point. If the two ids are different, the current point must be the beginning point of a valid cell, therefore we write its cell id into a bucket at the corresponding hash entry. Each hash entry has an index that keeps track of the next available bucket. We use atomic instructions to update the index, ensuring that concurrent writes to the same hash entry are serialized.

When the number of cells mapped to the same hash entry exceeds the number of buckets, we have an overflow situation. Although our hash table is not guaranteed to be overflow-free, in practice we almost never encounter such a situation. Alternatively we could use a standard open addressing scheme such as quadratic probing or double hashing to strictly eliminate overflow, but we found our bucket-based method preserves memory coherence better.

In order to search a cell in the hash table, we find its entry in the table using the cell id, and then linearly scan through the buckets to find a matching id. If no match is found, the cell must be invalid. Otherwise the search returns as soon as a matching id is found. Since each cell can contain at most one Poisson disk sample, there can only be one bucket with the matching id.

3.2 Sampling with Geodesic Distance

For complex shapes with thin features, sampling with the Euclidean distance metric can lead to undesirable distribution. This is because it measures the shortest distance in 3D space as opposed to the surface. In this case, we need to account for geodesic distance, for which we make the following changes to our algorithm.

Geodesic distance approximation. The first change is to replace the distance function d with an approximation that quickly estimates the geodesic distance between two surface points. While techniques are available for computing accurate geodesic distance [Sethian 1995; Surazhsky et al. 2005], they are often expensive to compute and difficult to parallelize on millions of points. Since we represent the surface as a large point cloud, we ideally want a simple approximation that only relies on the points and not on the mesh connectivity or parametrization.

Assume two surface points p_1, p_2 with normals $\vec{n}_1, \vec{n}_2$, respectively. The Euclidean distance between the two points is $d_e = \|p_2 - p_1\|$, and the normalized vector connecting the two points is $\vec{v} = (p_2 - p_1)/d_e$. We compute $c_1 = \vec{n}_1 \cdot \vec{v}$ and $c_2 = \vec{n}_2 \cdot \vec{v}$, which are the cosine angles of the two normals with the direction $\vec{v}$.

We assume that there is a smooth curve on the surface that passes through the two points, and we estimate the geodesic distance d_g as the length of this curve. However, explicitly constructing this curve would be non-trivial, since $\vec{n}_1$, $\vec{n}_2$, and $\vec{v}$ may not be co-planar. Instead, we use a direct integral to estimate d_g. Specifically, we take differential steps along $\vec{v}$. At each step t, we estimate the cosine angle $c(t)$ between the curve normal at this point and direction $\vec{v}$ as: $c(t) = (1-t)\,c_1 + t\,c_2$, which is simply a linear interpolation between c_1 and c_2. Using $c(t)$ we can estimate the differential curve length that is along the tangent direction (see Fig. 4). Finally we

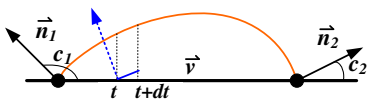


Figure 4: *Geodesic distance approximation. The blue line indicates the differential curve length at integration step t.*

integrate the differential curve length and obtain d_g as:

$$d_g = \int_0^1 \frac{d_e}{\sqrt{1-[(1-t)\,c_1 + t\,c_2]^2}}\,dt = \frac{\arcsin c_1 - \arcsin c_1}{c_1 - c_2} d_e \tag{1}$$

Note that the integral has an analytic solution, and it is a simple scaling of the Euclidean distance d_e. Also note that if $c_1 = c_2$, the formula reduces to $\frac{d_e}{\sqrt{1-c_1^2}}$ using L'Hospital's rule.

When p_1, p_2 are sampled on a plane, we can easily see that our formula gives $d_g = d_e$. Moreover, it can be shown that when p_1 and p_2 are two points sampled on a sphere, our estimated d_g gives the *exact* geodesic distance: $d_g = 2R \arcsin \frac{d_e}{2R}$. Therefore the formula is correct for these two special cases.

The main advantage of this approximation is that it is fast to compute and can easily fit into existing algorithms. On the other hand, since it completely ignores the underlying mesh, it is only accurate when the surface changes smoothly between p_1 and p_2. Note, however, we only need to compute the geodesic distance when $d_e < r$, because if $d_e \geq r$ the two points cannot possibly have a conflict as $d_g \geq d_e$. Usually we are only interested in cases where r is small, thus the two points must be quite close-by spatially, and our surfaces typically do not contain high-frequency changes at this scale level. Therefore in practice we found our approximation to work well in many cases. In Section 5 we discuss the error of this approximation by comparing it to ground truth geodesic distances.

Number of samples per cell. In the Euclidean setting, we can safely assume that each grid cell contains at most one sample. However, this is no longer true for the geodesic setting, as two points in a cell can have $d_g > r$. We handle this case by explicitly allowing the same cell to appear multiple times in the hash table, so that each cell instance can store a different sample. This increases the expected searching time as we must scan through all the available buckets at a hash entry to find all matching cells. But we found this additional overhead to be relatively small. In practice, we rarely observe any cell that contains more than 2-3 samples, especially when generating a large number of samples using a small radius. Therefore we provide an option for the user to enable or disable multiple samples per cell at run-time, to achieve a better tradeoff between quality and performance.

Number of trials. When allowing for multiple samples per cell, we also need to increase the number of trials k in order to increase the likelihood of finding the additional samples. Ideally k should be proportional to the surface area contained in the cell. In practice this can be easily estimated using the number of random points assigned to each cell, because the initial random points $\mathcal{S}_i$ are drawn uniformly from the surface. Again, when generating a large number of samples, we found the effect of varying k per cell to be insignificant in most cases. Therefore our implementation leaves this as an option for the user to adjust at run-time.

Implementation. We implement our parallel surface sampling algorithms on the GPU using NVIDIA's CUDA programming language. The main steps are listed in Program 1. We make use of several parallel computation primitives such as global sorting and list compaction, all of which are available in the CUDPP 1.1 library. For random number generation, we pre-compute a few sets of random numbers on the CPU and upload them as textures on the GPU to be accessed by our CUDA programs.

We typically generate 1∼4 million initial random sample points, which are sufficient for computing a few hundred thousand samples. Our k ranges between 5∼10. We use 3×3×3 phase groups as it results in more valid cells per phase group, thus beneficial for high GPU utilization. For conflict checking, we search the 5×5×5 neighborhood around each cell, in an inside-to-outside fashion sim-

ilar to [Wei 2008]. When using the geodesic distance approximation, these configurations are the same as the Geodesic distance is no less than the Euclidean distance. Note that when enabling multiple samples per cell, the conflict checking must include the current cell itself (in addition to neighbors), as it may contain another sample that could cause a potential radius conflict within the cell.

4 Spectrum Analysis

Our parallel surface sampling algorithm may introduce bias due to several factors: the size of the initial random point set $\mathcal{S}_i$, the number of trials k, and the geodesic distance approximation. Consequently we need a way to evaluate the distribution quality of our samples. While it is possible to parameterize the surface and perform spectrum analysis using standard Fourier transform, the results are often not accurate enough due to the inherent distortion by surface parameterization. Fortunately, there is a correspondence of Fourier basis on manifold surfaces called the *spectral mesh basis* [Karni and Gotsman 2000], which allows us to perform spectrum analysis directly on surfaces without parametrization. Below we provide a brief review of the basis, explain how to use it to evaluate surface sampling bias, and then discuss a few practical issues and show our results.

Review of spectral mesh basis. Consider a manifold surface defined by a triangle mesh $\mathcal{M}$: such a mesh consists of a set of vertices $\mathcal{V}$ and their connectivity. We can define a function f that exists on the mesh by specifying its values at the vertices of the mesh; then the interior values can be linearly interpolated from the vertices using the triangle's barycentric coordinates. If $\mathcal{M}$ has M vertices, f will be represented with a M-dimensional vector.

The second order derivative of a mesh function can be approximated by a discrete mesh Laplacian $\mathcal{L}$, which locally takes a weighted average of the differences between the value of f at a vertex i with its one-ring of neighboring vertices:

$$(\mathcal{L}f)_i = \frac{1}{a_i} \sum_{j \in N(i)} w_{ij}(f_i - f_j) \tag{2}$$

where $N(i)$ is the set of one-ring neighbors of vertex i, w_{ij} are symmetric weights such that $w_{ij} = w_{ji}$, and a_i is a positive value, which in our case is the delta area occupied by each vertex. Thus the sum of a_i over all vertices is the total surface area of the mesh.

Since $\mathcal{L}$ is a linear operator, it can be represented as an $n \times n$ matrix, thus Eq. 2 can be expressed in matrix form as $L = A^{-1}Q$, where A is a diagonal matrix whose diagonal elements are the a_i, Q is a symmetric matrix whose diagonal elements are given by $Q_{ii} = \sum_j w_{ij}$ and whose off-diagonal entries are $-w_{ij}$.

It is well-known that the eigenvectors of L define a set of Fourier-style basis that exists on the mesh, and the associated eigenvalues capture the frequencies of the basis functions. For example, the smallest eigenvalue is always 0, and the associated eigenvector is a constant vector, indicating a zero-frequency (or DC) basis. Larger eigenvalues correspond to higher-frequency basis.

A number of different choices exist to define the weights w_{ij}. We use the cotangent weighting in [Desbrun et al. 2002], which models the differential Laplacian on a smooth Riemannian manifold:

$$w_{ij} = (\cot \alpha_{ij} + \cot \beta_{ij})/2 \tag{3}$$

where α_{ij} and β_{ij} are the two angles opposite to the edge (i, j). Using the cotangent weights, the derived basis set provides expected solutions to a few familiar cases. For example, when $\mathcal{M}$ represents a 2D plane, the result is the real form of 2D Fourier basis; similarly, when $\mathcal{M}$ is a sphere, the result is real spherical harmonics.

Since the basis set is represented as discrete functions sampled on the mesh vertices, its robustness may be sensitive to changes in the mesh connectivity and tessellation. As examined by [Dyer et al. 2007], the cotangent weights provide the best spectral robustness among several other choices. Of course to obtain high-frequency basis functions accurately, a fine tessellation of the mesh is still required in order to avoid aliasing.

Frequency spectrum of surface samples. Computing the spectral basis for a mesh $\mathcal{M}$ simply involves constructing the mesh Laplacian matrix L and finding its eigenvectors. We then define each eigenvector as a spectral basis function $\mathbf{B}_k$, and the magnitude of the associated eigenvalue $|\lambda_k|$ as the frequency ω_k. In Figure 5 we show the frequency plots for several models with different shapes and genuses. In all cases we observe that the squared frequencies ω_k^2 largely follow a continuous linear curve, and hence the frequencies ω_k follow an inverse quadratic curve. One special case is the sphere, for which the frequencies exhibit discrete values. Each discrete frequency at level ℓ contains $2\ell + 1$ basis functions, which conforms with the well-known spherical harmonics.

With the basis set, we can easily obtain the frequency spectrum of a function by projecting it onto each basis $\mathbf{B}_k$. In particular, if the function is a discrete set of N surface samples, the projection is:

$$b_k = \int_{\mathcal{M}} \mathbf{B}_k(\mathbf{x}) \frac{1}{N} \sum_{j=1}^{N} \delta(\mathbf{x} - s_j)\, d\mathbf{x} = \frac{1}{N} \sum_{j=1}^{N} \mathbf{B}_k(s_j) \tag{4}$$

which is simply the average of $\mathbf{B}_k$ evaluated at every sample s_j. Note that the basis value at an arbitrary surface sample is interpolated from the triangle vertices. In order to analyze the distribution of the samples, we compute the power spectrum $|b_k|^2$, which is the square magnitude of $|b_k|$; we then multiply it by the total surface area of the mesh to normalize the results.

Radial mean and anisotropy. In planar sampling, the distribution property of random samples is typically measured by the radial mean and anisotropy of the power spectrum [Lagae and Dutré 2008]. It is straightforward to extend these concepts to general surface sampling. To begin with, we partition the entire range of frequencies into a number of equally spaced bands, each with a bandwidth of ω_B. Thus each band contains frequencies in the range of $[(\ell - 1)\omega_B, \ell\,\omega_B]$, where ℓ is the band index. As observed in Figure 5, for all models we tested, the frequency plot roughly follows an inverse quadratic curve. This means that when partitioning the frequencies to equally spaced bands, each band will contain a linearly increasing number of basis functions, which is similar to spherical harmonics. Thus in many cases we can just follow spherical harmonics, assigning $2\ell + 1$ basis functions to each band.

Next, we collect the set of power spectrum values $\{|b_k|^2\}$ whose associated frequencies ω_k fall into each band:

$$\{|b_k|^2\}_\ell = \{|b_k|^2, \forall\, \mathrm{k\,such\,that}\ \omega_k \in [(\ell - 1)\omega_B, \ell\,\omega_B]\} \tag{5}$$

Finally we compute the average and the relative variance of the values in the set $\{|b_k|^2\}_\ell$, and the results correspond directly to the radial mean and anisotropy in standard power spectrum analysis [Lagae and Dutré 2008]. Figure 5 shows the results computed for several models with different sampling algorithms including ours.

Implementation details. We implement the computation of mesh bases in MATLAB. A few practical issues must be considered. First, the mesh needs to be finely tessellated in order to get accurate results for high-frequency bases. We typically subdivide our models to $n = 100,000$ vertices. This means the Laplacian matrix L can be very large. Fortunately since L is extremely sparse, we can make use of the sparse matrix routines in MATLAB to handle such large matrices. Note that the tessellation is only needed in order

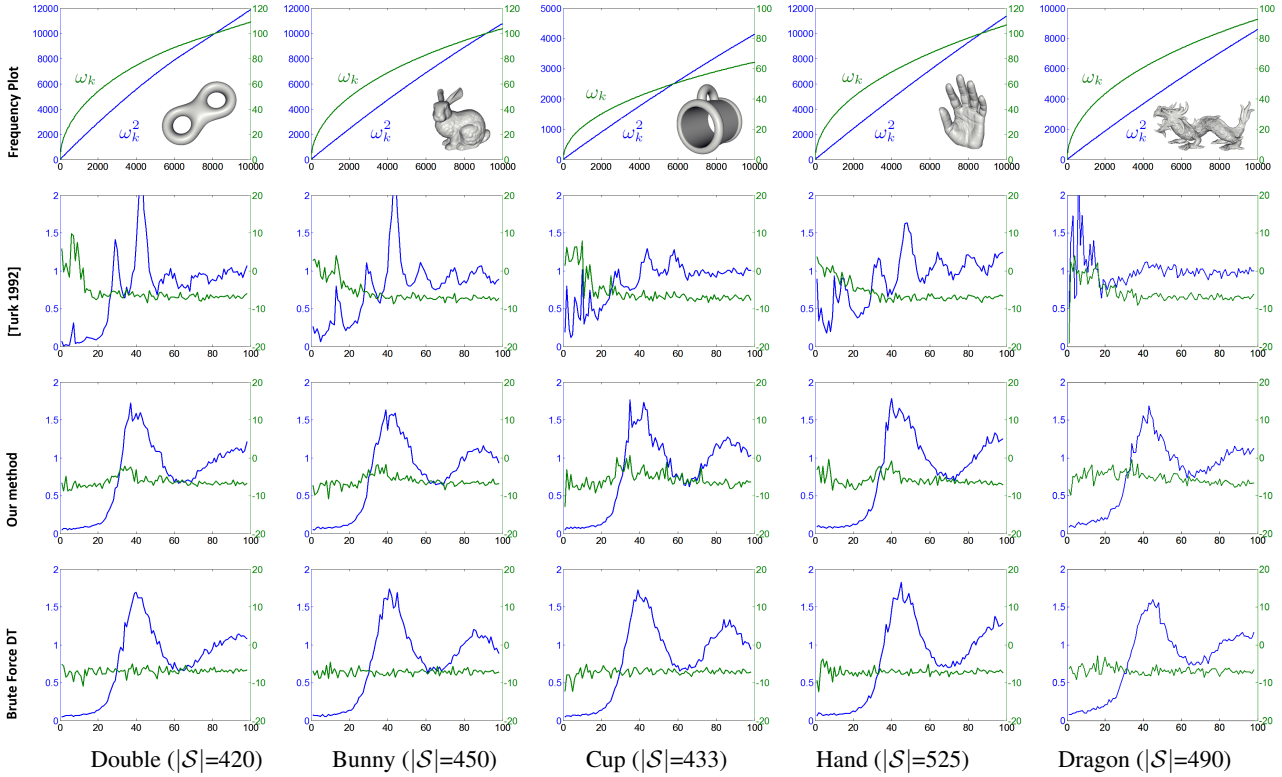


Figure 5: *Spectrum analysis for models with different shapes and genuses. Top row shows the frequency plot of the derived basis set; the next three rows compare the radial means (blue) and anisotropy (green) of samples generated using [Turk 1992], our method, and a reference algorithm (using brute force dart throwing). The x-axis is the basis frequency, the left y-axis is the radial mean, and the right y-axis is the anisotropy. |$\mathcal{S}$| indicates the average number of samples used for each test. We use 10 runs for each test.*

to verify the sample distribution, while the computation of Poisson disk samples does not require any mesh subdivision.

Second, L is generally not symmetric, but is similar to a real symmetric matrix $O = A^{-\frac{1}{2}} Q A^{-\frac{1}{2}}$. Therefore its eigenvalues can be solved by computing an svd of Q, which is more stable. The details can be found in [Dyer et al. 2007]. In addition, we use the spectral shift method described in [Vallet and Lévy 2008] for efficient computation of a large number of singular values. This can be easily implemented via MATLAB's svds routine.

Finally, in order to perform spectrum analysis on a decent number of surface samples, we need a relatively large number of basis functions. Essentially the highest frequency in the basis set determines the number of samples we are able to analyze successfully. Note that increasing the highest frequency leads to a quadratic increase in the number of basis functions, therefore significantly increasing the computation time. In practice, we usually compute 10,000 basis functions, thus the highest relative frequency is roughly 100. We found this sufficient for evaluating $300 \sim 500$ surface samples. The typical computation time for 10,000 basis functions on an 100,000-vertex model is about 2 hours in all our tests.

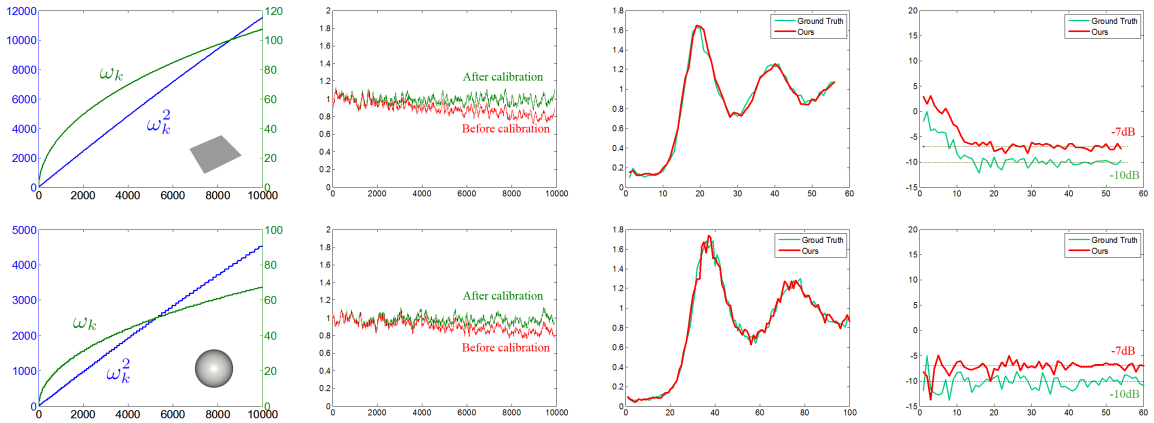
White noise calibration. Due to numerical inaccuracies in the basis computation, we have observed a slight linear decay in the calculated spectral power, as shown in Figure 6(b). To correct for this error, we use white noise spectrum to calibrate the decay. Specifically, we use white noise (uniform random) samples generated on the surface to compute the spectrum, average the results over many runs, and then plot the results. In theory this should give a straight line that is roughly constant. In practice we get a slanted line that declines at a roughly constant rate. We calculate the slope of the line, and scale the power spectrum by the inverse of the decay to bring the white noise spectrum back to straight. For most models we tested, the decay is about -0.2∼-0.25 over 10,000 basis functions. We apply the correction on the power spectrum $|b_k|^2$ computed in all our tests.

Verification of our method. We verify our spectrum analysis method by using the plane and sphere as two special cases, for which the analytic basis functions are known. Figure 6(c) and (d) show the results comparing our results with the ground truth. In both cases we generate Poisson disk samples using brute force dart throwing, and average the results over 10 runs. The radial mean plots exhibit familiar blue noise patterns in both examples. However, one important difference is that **our anisotropy plot is centered around the -7.0dB line** instead of the -10dB line in the ground truth. This means there is a difference by a factor of 2, and we found the difference is consistent throughout our tests.

The reason for this **double anisotropy** is because our derived mesh basis functions are the real forms of the analytic Fourier basis. This causes the anisotropy, which measures the variance of the power spectrum, to differ by a factor of 2, while the radial means remain the same. In the supplemental document, we show a proof of the double anisotropy for the case of 1D white noise samples.

Through experiments, we found that our method works well for general manifold surfaces, regardless of their shape and topology. Specifically, the ideal radial means and anisotropy using our analysis are consistent with the blue noise patterns found in familiar examples. These results are shown in Figure 5 on various models. Therefore we conclude that our method provides an effective way for evaluating the distribution quality of surface samples.

Discussion. While the number of basis functions we compute (currently 10,000) may seem a limiting factor of our technique, note that the main purpose of our spectrum analysis tool is to evaluate the distribution property of a surface sampling algorithm. We found



(a) Frequency plot (b) White noise calibration (c) Radial means (d) Anisotropy

Figure 6: *Verification of our spectrum analysis method using plane and sphere. The ground truth are generated by analytic Fourier basis and spherical harmonics respectively. We generate ∼270 samples for the plane and ∼320 samples for the sphere using brute force dart throwing, and average the results over 10 runs. Note that our radial means are consistent with the ground truth, but the anisotropies differ by a factor of 2 (-7dB vs. -10dB).*

that 300 to 500 samples are usually sufficient to characterize an algorithm, as shown in Fig 5. When it is necessary to evaluate the distribution property of more samples, we can split the mesh into smaller partitions, each containing 300 to 500 samples. We can then analyze each partition separately.

5 Results and Applications

We test our parallel sampling algorithms on a 2.66 GHz PC with an NVIDIA 280 GTX graphics card. Our programs make use of CUDPP 1.1 and are compiled using CUDA 2.3. We typically use 1∼4 million initial random sample points $\mathcal{S}_i$; the phase group size is 3×3×3, and the number of trials $k = 5 \sim 10$. Figure 9 shows examples of Poisson disk samples we generated for several models using different radii, resulting in different sampling densities.

Performance. Figure 7 plots the performance of our algorithm tested under both Euclidean and approximated Geodesic distance metrics for three models and two choices of k. The timing includes all steps listed in Program 1. The breakdown is roughly 30% for two global sorts, and 70% for the Poisson sampling loop. The remaining steps take an insignificant amount of time. The plots show that the performance remains roughly the same for all models. This shouldn't be surprising as our algorithm converts all models to a dense point cloud before sampling, thus the performance is nearly uniform regardless of the mesh complexity.

We observed that increasing the number of trials k from 5 to 10 generally results in about 10∼15% of drop in performance. On the other hand, using the approximated geodesic distance vs Euclidean only results in very small performance differences. This is because 1) our geodesic distance approximation is very fast to compute; and 2) it results in more sample points being placed on the surface, therefore increasing the relative performance with respect to the number of samples per second. Overall we are able to achieve about 180,000 samples per second, which is 5∼10 times faster than the state of the art such as [Cline et al. 2009].

Efficiency. When the Poisson disk radius is large, there will be only a small number of valid cells, which leads to poor utilization of the GPU. Ideally we need a large number of valid cells in order to achieve high efficiency on the GPU. Thus our algorithm is more efficient at generating a large number of sample points. In Figure 8 we show statistics on the number of valid cells and the number of Poisson disk samples computed using three different radii. For these examples, the number of valid cells is sufficiently large to keep the GPU fully utilized. Note that the number of computed

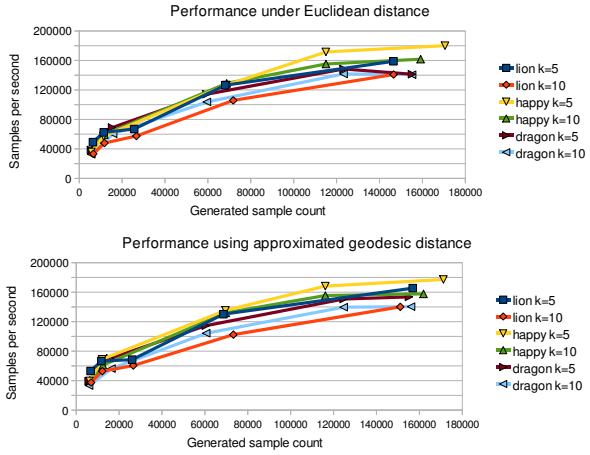


Figure 7: *Performance of our algorithm on three different models.*

samples is typically around $1/5 \sim 1/6$ of the number of valid cells, which is consistent in all our examples.

Quality. We measure the quality of our Poisson disk sampling using two criteria. First, to verify the density of sample points, we calculate the radius statistics ρ [Lagae and Dutré 2008] defined as:

$$\rho = \frac{r/2}{(2\sqrt{3} \cdot N/A_{\mathcal{M}})^{-1/2}} \tag{6}$$

where r is the Poisson disk radius (note that the typical convention of the Poisson radius is half of our r), N is the total number of Poisson disk samples computed, and $A_{\mathcal{M}}$ is the surface area of the mesh. The results are included in Figure 8. In all cases, our ρ values fall between 0.71∼0.75, which is within the ideal range of 0.65∼0.85 recommended by [Lagae and Dutré 2008].

Second, we apply our proposed spectrum analysis to examine the radial means and anisotropy. To do so, we tessellate the test models to about 100,000 vertices, and compute 10,000 spectral mesh basis functions as described in Section 4, then evaluate the radial means and anisotropy of the samples' power spectrum averaged over 10 runs. The number of available mesh basis functions allow us to evaluate 300∼500 sample points and observe the expected patterns. The results are shown in Figure 5. For comparison, we provide results for two additional sampling algorithms: the point repulsion algorithm by [Turk 1992], and a brute force dart throwing algorithm (using our geodesic distance approximation) as reference. Note that

			r=0.0089			r=0.0111			r=0.0139		
model	#tri	area	#v.c.	#s.	r.s.	#v.c.	#s.	r.s.	#v.c.	#s.	r.s.
Bunny	5k	12.37	508k	93k	0.715	354k	61k	0.726	239k	40k	0.736
Dragon	239k	3.99	193k	32k	0.739	128k	21k	0.741	83k	13k	0.743
Lion	306k	3.17	156k	26k	0.741	103k	16k	0.742	67k	11k	0.742
Hand	50k	7.48	338k	58k	0.727	228k	38k	0.736	151k	25k	0.740

Figure 8: *From left to right, the table lists the number of triangles and surface area for each test model, the number of valid cells (#v.c.) and Poisson disk samples (#s) computed using our algorithm with three difference radii (r), and the corresponding radius statistics (r.s.).*

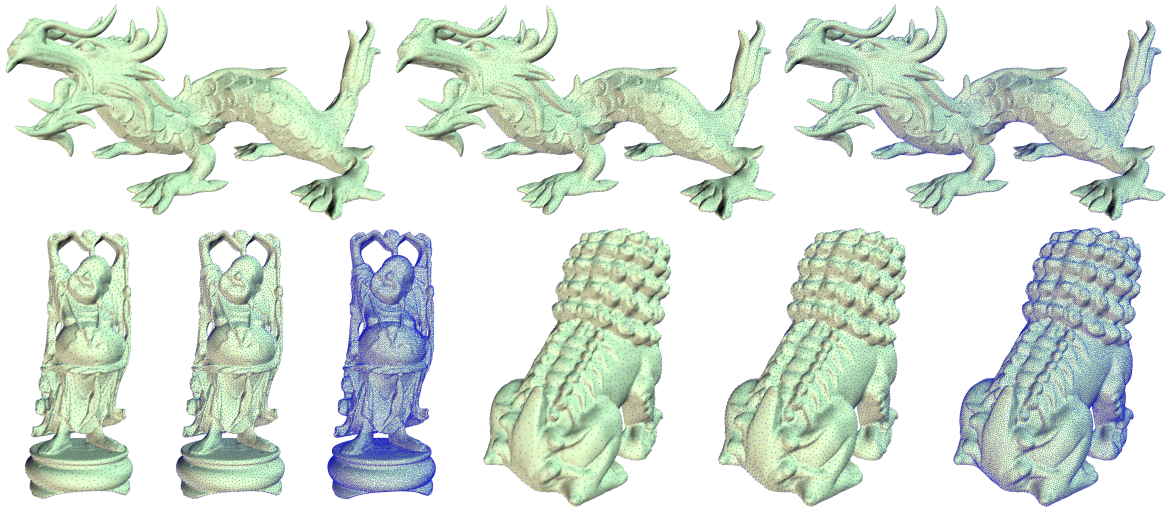
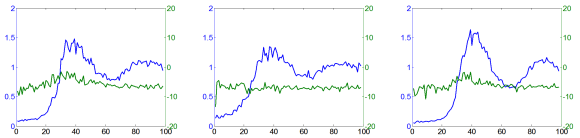


Figure 9: *Poisson disk sampling using our algorithm for three models: Dragon, Buddha, and Lion. For each model we generate three density levels, at approximately 5K, 10K and 50K sample points. The images are generated at high resolution to allow for zoom-in examination.*



$k = 1, |\mathcal{S}_i| = 1M$; $k = 10, |\mathcal{S}_i| = 1000$ $k = 10, |\mathcal{S}_i| = 1M$

Figure 10: *Here we show the comparisons of the radial means and anisotropy when adjusting the parameters k and $|\mathcal{S}_i|$. The tests were performed on the Bunny model (refer to Figure 5).*

our results are qualitatively similar to the reference. We do observe some bias around the principle frequencies, manifested by the imperfect shapes of radial means and the slightly higher anisotropy around those regions. However, overall the results look consistent with expected blue noise distribution patterns. The point repulsion algorithm, on the other hand, produces relatively large bias, especially in the radial means.

A number of parameters influence the quality, such as the number of trials k and the size of initial sample points $\mathcal{S}_i$. In Figure 10 we show the radial means and anisotropy for sampling on the Bunny model when reducing k to 1, or reducing the size of the initial sample points to $|\mathcal{S}_i| = 1000$. Note that in both cases the radial means become flatter compared to when $k = 1$ and $|\mathcal{S}_i| = 1,000,000$.

Geodesic distance approximation. To examine the accuracy of our geodesic distance approximation, in Figure 11 we compute the percentage of correct distance checking results under both Euclidean and our approximated Geodesic distance metrics. We generate a large number of uniform random points on the surface. Then for every pair of points that have an Euclidean distance smaller than the Poisson radius r, we obtain their true geodesic distance using [Surazhsky et al. 2005]. This provides the ground truth answer as to whether the two points should reject or accept each other. At the same time we use our approximated geodesic distance to compute the prediction, and report the rate of correct predictions. Note that the Euclidean distance will always predict rejection since we only consider points within a Euclidean distance of r. This results in errors when the true geodesic distance is in fact $\geq r$. In Figure 11 we can see that in most cases our approximation provides more accurate answers, especially for smooth surfaces such as the double torus. We perform the analysis this way because it's unrealistic and unnecessary for us to have an accurate geodesic distance approximation over large distances.

Interactive surface sampling. Our parallel surface sampling algorithm is fast and hence suitable for interactive settings where the user changes the sampling density in real-time. Figure 12 shows an example where we apply our algorithm on a deformable model, and the samples are updated interactively in response to the surface changes. Notice that even though we do not explicitly enforce temporal coherence (as in [Vanderhaeghe et al. 2007; Yu et al. 2009]), our results nevertheless appear relatively coherent in general. If stricter coherence is required we could possibly incorporate similar mechanisms as in [Yu et al. 2009] into our framework.

Surface texturing application. One direct application of our method is surface texturing [Lagae and Dutré 2005]. This includes a variety of different types of textures, such as 2D textures, 3D geometry, spatially varying BRDF or BTF textures. These can all be achieved in real-time using a method similar to texture-bombing [Glanville 2004]. The basic idea is to assume that a texture exemplar is placed at the center of every Poisson disk sample; then for every surface point to be shaded, we find its closest samples, project the point at each sample's local frame to obtain the texture coords, then interpolate the results to achieve smooth blending. Our implementation gains an additional benefit in that our hash table is already a spatial data structure that can be used directly to search for the closest samples for a query point. Figure 14 shows several examples of surface texturing using our samples. By changing the Poisson disk radius, the user can adjust the distribution of textures in real-time. This does not require any surface parametrization.

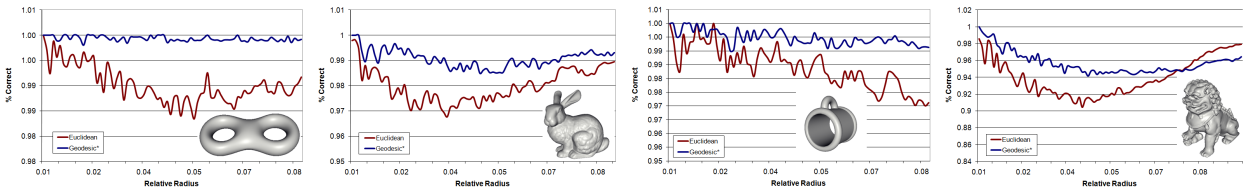


Figure 11: *Comparison of Euclidean vs. our approximated geodesic distance. We generate the ground truth geodesic distance using [Surazhsky 2005]; then for a number of different Poisson disk radius r, we select a large number of uniform sample points on the surface, and report the rate of correct acceptance or rejection for any two points that have an Euclidean distance smaller than r.*

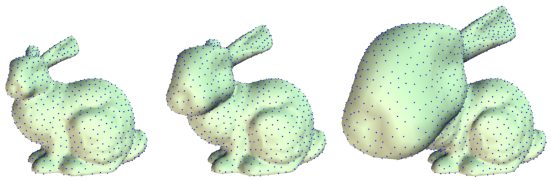


Figure 12: *Real-time sampling on a deformable Bunny. Note how the samples maintain a uniform distribution as the surface deforms.*

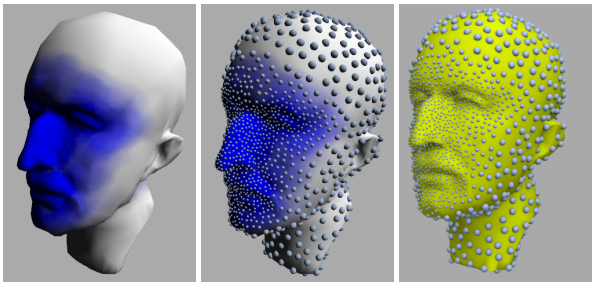


Figure 13: *Adaptive sampling guided by user-painted radius function. While not real-time, our adaptive algorithm can still compute more than 10,000 points in 1∼2 seconds. The right image shows an offline rendering.*

Parallel adaptive surface sampling. We extend our parallel uniform sampling algorithm to adaptive sampling, where the Poisson disk radius r is defined by a spatially varying function $r(\cdot)$ on the surface. Assuming that $r(\cdot)$ has an upper bound r_{max}, (i.e. the maximum radius defined by the function), our algorithm works similarly to before: first, we partition the initial random points $\mathcal{S}_i$ into grid cells using $\frac{r_{max}}{\sqrt{3}}$ as the cell size; then, we draw samples in parallel for each valid cell belonging to the same phase group. When checking the radius conflicts, we use $\max(r(s), r(s'))$ as the distance threshold, where s is an existing sample and s' is a trial sample. By using r_{max} to compute the cell size, no two cells being processed concurrently can place conflicting samples. On the other hand, each cell must be allowed to contain multiple samples. Note that this is already allowed in our hash table (see Section 3.2). Here we only need to suitably enlarge the number of trials k as well as the bucket size in order to accommodate additional samples.

In general we can view the grid cells as a simple space partitioning scheme that allows multiple regions of the surface to be sampled in parallel. One downside is that it is not adaptive, so its efficiency can decrease significantly if the radius function $r(\cdot)$ changes dramatically over the surface. In this case, the number of samples placed in each cell will be highly non-uniform, causing increased searching time in the hash table and divergence in parallel computation. A more efficient solution would be to use an hierarchical approach similar to [Wei 2008]. This remains our future work.

Figure 13 shows an example of our adaptive surface sampling algorithm: the user directly paints onto the mesh to indicate a desired radius function $r(\cdot)$. Our algorithm then recomputes adaptive surface samples according to $r(\cdot)$. While not yet real-time, our algorithm capable of computing 10,000 sample points in 1∼2 seconds.

6 Limitations and Future Work

In summary, we have presented a parallel Poisson disk sampling algorithm suitable for fast sampling on arbitrary surfaces. Our algorithm is flexible and produces high-quality surface samples at interactive rates. For analysis, we introduce a new method, based on the spectral mesh basis, to evaluate the spectral distribution quality of our sampling algorithm using radial means and anisotropy. Our method perform power spectrum analysis directly on manifold surfaces without requiring parametrization.

There are several limitations of our work that should be addressed in future work. First, a number of design decisions we made for the parallel sampling algorithm are centered around high performance on a parallel processor. For example, we favor a small number of phase groups to improve GPU utilization, we use a simple geodesic distance approximation to avoid expensive computation, and we currently do not use hierarchical sampling in order to avoid the overhead in building the associated data structures. Some of these choices can lead to sampling bias that needs to be examined more carefully. Second, although we've shown the extension of our algorithm to adaptive surface sampling, it is not currently running at interactive rates. We would like to exploit hierarchical sampling or other methods that are more suitable for spatially non-uniform sampling patterns. Finally, our spectrum analysis currently does not apply to adaptive sampling, which is still an open research problem. We would like to investigate this problem in the future.

Acknowledgments. The authors would like to thank SIGGRAPH Asia anonymous reviewers for their feedback and comments. Rui Wang is supported in part by NSF grant CCF-0746577. John Bowers is supported by an NSF graduate research fellowship.

References

ALCANTARA, D. A., SHARF, A., ABBASINEJAD, F., SENGUPTA, S., MITZENMACHER, M., OWENS, J. D., AND AMENTA, N. 2009. Real-time parallel hashing on the GPU. *ACM Trans. Graph. 28*, 5, 154:1–9.

ALEXA, M. 2002. Recent advances in mesh morphing. *Comput. Graph. Forum 21*, 2, 173–196.

ALLIEZ, P., MEYER, M., AND DESBRUN, M. 2002. Interactive geometry remeshing. *ACM Trans. Graph. 21*, 3, 347–354.

BALZER, M., SCHLÖMER, T., AND DEUSSEN, O. 2009. Capacity-constrained point distributions: a variant of Lloyd's method. *ACM Trans. Graph. 28*, 3, 86:1–8.

CHESLACK-POSTAVA, E., WANG, R., AKERLUND, O., AND PELLACINI, F. 2008. Fast, realistic lighting and material design using nonlinear cut approximation. *ACM Trans. Graph. 27*, 5, 128:1–10.

CLINE, D., JESCHKE, S., WHITE, K., RAZDAN, A., AND WONKA, P. 2009. Dart throwing on surfaces. *Computer Graphics Forum 28*, 4, 1217–1226.

COOK, R. L. 1986. Stochastic sampling in computer graphics. *ACM Transactions on Graphics 5*, 1, 51–72.

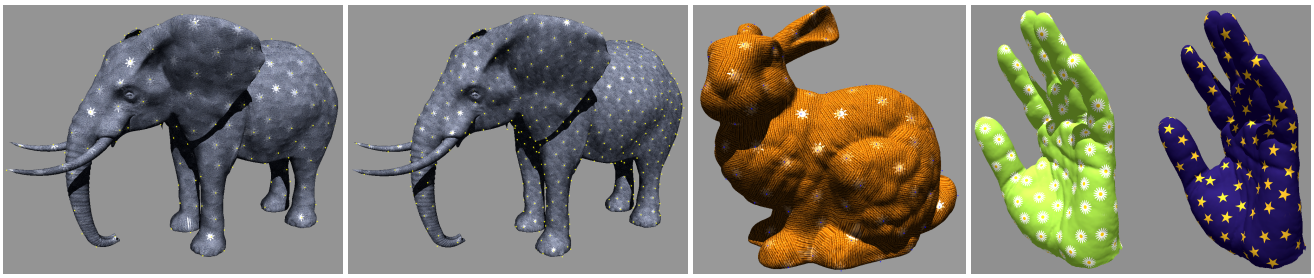


Figure 14: *Here we apply our Poisson disk samples to guide the placement of surface textures using the texturing bombing [Glanville 2004] technique. The user can adjust the sampling density in real-time. No surface parametrization is needed.*

DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. H. 2002. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and Mathematics III*, Springer-Verlag, 35–57.

DONG, S., BREMER, P.-T., GARLAND, M., PASCUCCI, V., AND HART, J. C. 2006. Spectral surface quadrangulation. *ACM Trans. Graph. 25*, 3, 1057–1066.

DUNBAR, D., AND HUMPHREYS, G. 2006. A spatial data structure for fast Poisson-disk sample generation. *ACM Trans. Graph. 25*, 3, 503–508.

DYER, R., ZHANG, H., MOELLER, T., AND CLEMENTS, A. 2007. An investigation of the spectral robustness of mesh Laplacians. Tech. Rep. TR-2007-17, School of Computing Science, Simon Fraser University.

FU, Y., AND ZHOU, B. 2008. Direct sampling on surfaces for high quality remeshing. In *SPM '08*, 115–124.

GLANVILLE, S. 2004. Texture bombing. *GPU Gems*.

GROSSMAN, J. P., AND DALLY, W. J. 1998. Point sample rendering. In *Rendering Techniques*, 181–192.

JENSEN, H. W., AND BUHLER, J. 2002. A rapid hierarchical rendering technique for translucent materials. *ACM Trans. Graph. 21*, 3, 576–581.

JONES, T. R. 2006. Efficient generation of Poisson-disk sampling patterns. *Journal of Graphics Tools 11*, 2, 27–36.

KARNI, Z., AND GOTSMAN, C. 2000. Spectral compression of mesh geometry. In *SIGGRAPH '00*, 279–286.

KOPF, J., COHEN-OR, D., DEUSSEN, O., AND LISCHINSKI, D. 2006. Recursive Wang tiles for real-time blue noise. *ACM Trans. Graph. 25*, 3, 509–518.

LAGAE, A., AND DUTRÉ, P. 2005. A procedural object distribution function. *ACM Trans. Graph. 24*, 4, 1442–1461.

LAGAE, A., AND DUTRÉ, P. 2008. A comparison of methods for generating Poisson disk distributions. *Computer Graphics Forum 27*, 1, 114–129.

LEVY, B. 2006. Laplace-Beltrami eigenfunctions towards an algorithm that "understands" geometry. In *Proc. of SMI 2006*, 13.

LI, H., LO, K.-Y., LEUNG, M.-K., AND FU, C.-W. 2008. Dual Poisson-disk tiling: An efficient method for distributing features on arbitrary surfaces. *IEEE TVCG 14*, 5, 982–998.

LI, H., WEI, L.-Y., SANDER, P., AND FU, C.-W. 2010. Anisotropic blue noise sampling. *ACM Trans. Graph. 29*, 5, to appear.

LLOYD, S. 1982. Least squares quantization in pcm. *IEEE Transactions on Information Theory 28*, 2, 129–137.

MCCOOL, M., AND FIUME, E. 1992. Hierarchical Poisson disk sampling distributions. In *Graphics Interface '92*, 94–105.

MEIER, B. J. 1996. Painterly rendering for animation. In *SIGGRAPH '96*, 477–484.

MITCHELL, D. P. 1987. Generating antialiased images at low sampling densities. In *SIGGRAPH '87*, 65–72.

NEHAB, D., AND SHILANE, P. 2004. Stratified point sampling of 3D models. In *Proc. of PBG*, 49–56.

OSTROMOUKHOV, V., DONOHUE, C., AND JODOIN, P.-M. 2004. Fast hierarchical importance sampling with blue noise properties. *ACM Trans. Graph. 23*, 3, 488–495.

OSTROMOUKHOV, V. 2007. Sampling with polyominoes. *ACM Trans. Graph. 26*, 3, 78:1–6.

PASTOR, O. M., FREUDENBERG, B., AND STROTHOTTE, T. 2003. Real-time animated stippling. *IEEE CG&A 23*, 4, 62–68.

PRAUN, E., HOPPE, H., AND FINKELSTEIN, A. 1999. Robust mesh watermarking. In *SIGGRAPH '99*, 49–56.

QU, L., AND MEYER, G. W. 2006. Perceptually driven interactive geometry remeshing. In *I3D '06*, 199–206.

RITSCHEL, T., ENGELHARDT, T., GROSCH, T., SEIDEL, H.-P., KAUTZ, J., AND DACHSBACHER, C. 2009. Micro-rendering for scalable, parallel final gathering. *ACM Trans. Graph. 28*, 5, 132:1–8.

SETHIAN, J. A. 1995. A fast marching level set method for monotonically advancing fronts. In *Proc. Nat. Acad. Sci*, 1591–1595.

SURAZHSKY, V., SURAZHSKY, T., KIRSANOV, D., GORTLER, S. J., AND HOPPE, H. 2005. Fast exact and approximate geodesics on meshes. *ACM Trans. Graph. 24*, 3, 553–560.

TURK, G. 1992. Re-tiling polygonal surfaces. In *SIGGRAPH '92*, 55–64.

TURK, G. 2001. Texture synthesis on surfaces. In *SIGGRAPH '01*, 347–354.

ULICHNEY, R. 1987. *Digital halftoning*. MIT Press, Cambridge, MA.

VALLET, B., AND LÉVY, B. 2008. Spectral geometry processing with manifold harmonics. *Comput. Graph. Forum 27*, 2, 251–260.

VANDERHAEGHE, D., BARLA, P., THOLLOT, J., AND SILLION, F. X. 2007. Dynamic point distribution for stroke-based rendering. In *EGSR'07*, 139–146.

WEI, L.-Y., AND LEVOY, M. 2001. Texture synthesis over arbitrary manifold surfaces. In *SIGGRAPH '01*, 355–360.

WEI, L.-Y. 2008. Parallel Poisson disk sampling. *ACM Trans. Graph. 27*, 3, 20:1–9.

WHITE, K., CLINE, D., AND EGBERT, P. 2007. Poisson disk point sets by hierarchical dart throwing. In *Symposium on Interactive Ray Tracing*, 129–132.

YU, Q., NEYRET, F., BRUNETON, E., AND HOLZSCHUCH, N. 2009. Scalable real-time animation of rivers. *Comput. Graph. Forum 28*, 2, 239–248.

Supplement: Radial Mean and Anisotropy of White Noise Samples

John Bowers* Rui Wang* Li-Yi Wei† David Maletz*

*University of Massachusetts Amherst †Microsoft Research

In this supplemental document for [Bowers et al. 2010], we derive the analytic radial means and anisotropy of 1D white noise samples. We show that with the real form of Fourier basis, the computed radial mean is the same as using the complex form of Fourier basis; however, the anisotropy computed using the former is twice as much as when using the latter. This is manifested in our experimental results in the paper. The definition of radial means and anisotropy can be found in literature work such as [Lagae and Dutré 2008]. We use 1D white noise as an example, but the proof can be straightforwardly extended the 2D case.

We assume there are N independent and uniformly random samples x_j distributed between [0, 1] (i.e. these are white noise samples). In the following we derive the analytic radial mean and anisotropy of the samples' power spectrum, and compare the results using 1D complex Fourier basis vs. 1D real Fourier basis.

1 Using 1D Complex Fourier Basis

It's well-known that each 1D complex Fourier basis is in the form:

$$f_\omega(x) = e^{-i\,2\pi\,\omega\,x} = \cos 2\pi\omega x + i\,\sin 2\pi\omega x \tag{1}$$

where ω is the frequency. The Fourier transform of a set of impulse samples is the sum of the Fourier basis evaluated at each sample:

$$F(\omega) = \sum_{j=1}^{N} f_\omega(x_j) = \sum_{j=1}^{N} \cos 2\pi\omega x_j + i \sum_{j=1}^{N} \sin 2\pi\omega x_j \tag{2}$$

Power Spectrum. The power spectrum is defined as the squared magnitude of $F(\omega)$. We further divide it by N in order to normalize it. Therefore the power spectrum $P(\omega)$ is:

$$P(\omega) = \frac{1}{N}|F(\omega)|^2 = \frac{1}{N}\left[\left(\sum_{j=1}^{N}\cos 2\pi\omega x_j\right)^2 + \left(\sum_{j=1}^{N}\sin 2\pi\omega x_j\right)^2\right] \tag{3}$$

Radial Mean. The radial mean R_m is defined as the average of $P(\omega)$ within a given frequency range $[\omega_1, \omega_2]$. When the range being considered is sufficiently small, it suffices to examine the expected value of $P(\omega)$ at a specific frequency ω. In order words, $R_m = E[P(\omega)]$, where E denotes the expected value with respect to the joint distribution of the samples. It is calculated as:

$$E[P(\omega)] = \int_0^1 ... \int_0^1 P(\omega)\,\mathrm{d}x_1\,\mathrm{d}x_2 ... \mathrm{d}x_N \tag{4}$$

While we could expand the quadratic terms in Eq. 3 to evaluate the integral, doing so makes it hard to compute the anisotropy later. So instead, we use an inductive approach. Let R_m^N denote the expectation of Eq. 3 with the sum running from $j = 1$ to N, and R_m^{N-1} denote that with the sum running from $j = 2$ to N, and so on. By integrating Eq.4 over $\mathrm{d}x_1$, we find that:

$$R_m^N = \frac{1}{N} + R_m^{N-1} + \Delta_{rem} \tag{5}$$

where Δ_{rem} is a residual term:

$$\Delta_{rem} = \frac{2\,(N-1)\,(\sin\pi\omega)^2}{N\,(\pi\,\omega)^2} \tag{6}$$

For any given finite N, Δ_{rem} will rapidly fall off to zero for as the frequency ω becomes large. Thus for a sufficiently large ω:

$$R_m^N = \frac{1}{N} + R_m^{N-1} \tag{7}$$

Using induction, and the fact that $R_m^1 = \frac{1}{N}$, we get:

$$E[P(\omega)] = R_m^N = \frac{1}{N} + \frac{1}{N} + ... + \frac{1}{N} = 1 \tag{8}$$

In other words, the power spectrum has an expected value of 1.0, and this is independent of the frequency ω. This conforms with the observation that the radial mean plot is centered around the 1.0 line.

Anisotropy. The anisotropy A_r is defined as the variance of $P(\omega)$ relative to the squared radial mean R_m^2. Since we have already calculated R_m, we now only need to compute the expected value of $P^2(\omega)$, which is:

$$P^2(\omega) = \frac{1}{N^2}\left[\left(\sum_{j=1}^{N}\cos 2\pi\omega x_j\right)^2 + \left(\sum_{j=1}^{N}\sin 2\pi\omega x_j\right)^2\right]^2 \tag{9}$$

Similarly to before, we use an inductive approach. Let S_m^N denote the expected value of the above equation with the sum running from $j = 1$ to N, and S_m^{N-1} denote that with the sum running from $j = 2$ to N, and so on. By integrating Eq. 9 over $\mathrm{d}x_1$, we find that:

$$S_m^N = \frac{1}{N^2} + S_m^{N-1} + \frac{4}{N}R_m^{N-1} + \Delta_{rem} \tag{10}$$

where Δ_{rem} is again a residual term that can be ignored for a sufficiently large ω. Using induction:

$$S_m^N = \frac{N}{N^2} + \frac{4}{N}\left(\frac{N-1}{N} + \frac{N-2}{N} + ... + \frac{1}{N}\right) = 2 - \frac{1}{N} \tag{11}$$

For a large enough N, we can see $S_m^N \approx 2$.

Now, the variance of $P(\omega)$ is simply $E[P^2(\omega)] - E^2[P(\omega)]$, or in other words $S_m - R_m^2$, thus

$$\mathrm{Var}(P(\omega)) = S_m - R_m^2 = 2 - 1^2 = 1 \tag{12}$$

Again, this conforms with our observation that the anisotropy plot is centered around the 1.0 line.

2 Using 1D Real Fourier Basis

The 1D real-form Fourier basis consists of only the real portion of the 1D complex Fourier basis:

$$f_\omega(x) = \sqrt{2}\cos 2\pi\omega x \tag{13}$$

where ω is integer frequency: $\omega = 0, 1, 2, ...$, and $\sqrt{2}$ is a normalization factor for the basis set to remain orthonormal. Such a basis

can be derived from the eigenfunctions of the Laplacian operator, as described in our paper. Given the form of the basis, the Fourier transform of a set of samples now becomes:

$$F(\omega) = \sum_{j=1}^{N} \sqrt{2} \cos 2\pi\omega x_j \tag{14}$$

Therefore the **power spectrum** is:

$$P(\omega) = \frac{1}{N}|F(\omega)|^2 = \frac{2}{N}\left(\sum_{j=1}^{N} \cos 2\pi\omega x_j\right)^2 \tag{15}$$

Radial Mean and Anisotropy. We use the same inductive method as before. For radial mean, the induction formula is:

$$R_m^N = \frac{1}{N} + R_m^{N-1} + \Delta_{rem} \tag{16}$$

This is the same with Eq. 5, thus the **radial mean** remains the same:

$$E[P(\omega)] = R_m = 1$$

To compute anisotropy, we first compute expected value of $P^2(\omega)$ which we denote as S_m. In this case, the induction formula is:

$$S_m^N = \frac{3}{2N^2} + S_m^{N-1} + \frac{6}{N} R_m^{N-1} + \Delta_{rem} \tag{17}$$

Therefore:

$$S_m = \frac{3N}{2N^2} + \frac{6}{N}\left(\frac{N-1}{N} + \frac{N-2}{N} + ... + \frac{1}{N}\right) = 3 - \frac{3}{2N} \tag{18}$$

For a sufficiently large N, we have $S_m^N \approx 3$.

Given R_m and S_m, the variance of $P(\omega)$ is:

$$\mathrm{Var}(P(\omega)) = S_m - R_m^2 = 3 - 1^2 = 2 \tag{19}$$

This conforms with our observation that the anisotropy plot in this case is centered around the 2.0 line. Note that this is twice as much as the anisotropy when using the complex 1D Fourier basis.

2.1 Multiple Experimental Runs

In practice, we typically perform multiple experimental runs, each run generating N random samples. We compute the power spectrum of each run, average the results over all runs, and then use the averaged power spectrum to compute radial mean and anisotropy. Typically we use $K = 10$ runs.

Under multiple runs, the radial mean still has an expected value of 1.0, since the expected value of each run is 1.0. On the other hand, the anisotropy is inversely proportional to K. This is because the variance of the average of K independent identically-distributed (i.i.d.) variables is $\frac{1}{K}$ of the variance of each random variable by itself. Therefore, with $K = 10$ runs, the anisotropy using real-form Fourier basis is centered around the 0.2 (or -7dB) line. For comparison: the anisotropy using complex 1D Fourier basis is centered around the 0.1 (or -10dB) line.

References

BOWERS, J., WANG, R., WEI, L.-Y., AND MALETZ, D. 2010. Parallel Poisson disk sampling with spectrum analysis on surfaces. In *SIGGRAPH Asia '10*.

LAGAE, A., AND DUTRÉ, P. 2008. A comparison of methods for generating Poisson disk distributions. *Computer Graphics Forum 27*, 1, 114–129.