Fast generation of 2-D node distributions for mesh-free PDE discretizations

Bengt Fornberg *
Department of Applied Mathematics
University of Colorado
Boulder, CO 80309

Natasha Flyer †
Institute for Mathematics Applied to Geosciences
National Center for Atmospheric Research
Boulder, CO 80305 USA

February 5, 2015

Abstract

Many applications require that nodes be scattered locally quasi-uniformly within 2-D regions or on curved surfaces in 3-D space, while obeying some prescribed spatially varying density function. This study focuses on creating variable density node layouts that are suitable for mesh-free discretizations of PDEs. Another application is 'dithering' to simulate half-tone images. The method is an 'advancing-front type' scheme, inspired by the physical process of dropping of variable sized grains into a bucket.

1 Introduction

Discretizations of PDEs have traditionally relied on structured meshes. Requirements for geometric flexibility, both to conform to irregular geometries and to achieve local refinement in critical areas, have led to an increased use of unstructured meshes, often in the form of triangular/tehrahedral elements. In contrast, RBF-generated finite differences (RBF-FD) is a recent and altogether mesh-free approach. It makes it easy to combine geometric flexibility with high levels of accuracy and computational efficiency [6, 7, 22, 23, 25, 28]. It uses scattered nodes with spatially varying density, without forming any associated 'elements' or connectivites between nodes.

A key to successful implementation of RBF-FD methods is being able to simply and very rapidly generate node layouts, as needed for applications such as real-time weather forecasting, tsunami modeling, and seismic imaging. Current methods in the literature for node distribution can be classified into two categories: 1) iterative methods, where for a given number of nodes the quality

* *Email*: fornberg@colorado.edu
† *Email*: flyer@ucar.edu

of the distribution is dependent on how soon the iteration is stopped, and 2) advancing front methods that start at a boundary and advance forward until the domain is filled.

Examples of iterative-type schemes include minimal energy (ME) distributions [10, 15, 27] (simulating equilibria for freely moving electrons), where the repelling force can be made spatially variable in order to concentrate nodes in areas where increased resolution is needed [5]. It was noted in [11, 24] that the energy minimization can be formulated in such a way that it can be applied to the task of image dithering, in which there is a rich literature. In this context, there may be additional requirements motivated by the characteristics of certain printers. For example, dot sizes may be allowed to be variable, and/or the dot locations may be confined to a Cartesian or hexagonal lattice; see [20] and references therein. Additional iterative strategies include Lloyd's algorithm [1, 16], also known as Voronoi relaxation, and the approach presented in [21], in which Delaunay triangulations are used iteratively to adjust a partly random initial distribution. A Matlab implementation *distmesh2d* of the latter algorithm is available online at http://persson.berkeley.edu/distmesh/, aimed towards finite element diretizations of PDEs.

The algorithms from the literature that are conceptually closest to the present one are known as advancing front methods [4, 14, 26]. These differ however significantly from the present algorithm. In most 2-D cases, triangular mesh generation typically proceeds inwards from boundaries. In some cases, generalizations are available to 3-D. These methods often require background grids, causing effectiveness issues in cases of strong node density variations. Several strategies and implementations are compared in [3]. In cases where no spatial node density variations are needed, the method in [2] is effective in 2-D.

The primary advantages of the present algorithm are computational speed, algorithmic simplicity, and the quality of the generated node distributions. Since it is not iterative, we refer to it as 'node placing'. The user does not need to decide on trade-offs between computer time and distribution quality. Our focus will be on how to implement it simply and effectively, and not on mathematical analysis of either the algorithm itself or of the generated node distributions. For certain aspects (e.g. internal parameter choices) that only minimally influence the quality of the obtained node distributions, our preference has been to make robust ad hoc choices rather than to pursue theoretical analysis of possible variations. Figure 1 illustrates in a very small case the node sets the present procedure will produce very rapidly - in Matlab at a rate of roughly 10,000 nodes per second.

There is a rapidly growing literature on applying RBF-FD to solve many different types of PDEs. This includes a wide range of systems arising in elasticity, geophysics, seismic exploration, electromagnetics, etc., mostly of time dependent character, but also elliptic-type equilibrium equations. The present paper will not repeat this literature but instead focus on associated node generation issues. Although the RBF-FD literature by now is quite extensive, there are as of yet few survey type articles in the field. The first one, with focus on geosciences, appeared recently [7], with an a more general survey article [9] as well as a monograph [8] scheduled for publication in 2015.

The outline of this paper is as follows: Section 2 presents the basic node placing algorithm, with the Matlab code given in the Appendix, as well as a common test case used in image rendering. The paper starts with this test case as it provides a much more challenging example than normally would be found in the PDE discretizations for creating layouts due to its extreme variations in node density. Section 3 discusses the use of a local node repel algorithm for making minor adjustments near non-rectangular boundaries, as required in RBF-FD discretizations. Section 4 describes how to generalize the domain from a rectangle to 2-D surfaces embedded in 3-D space, such as spheres

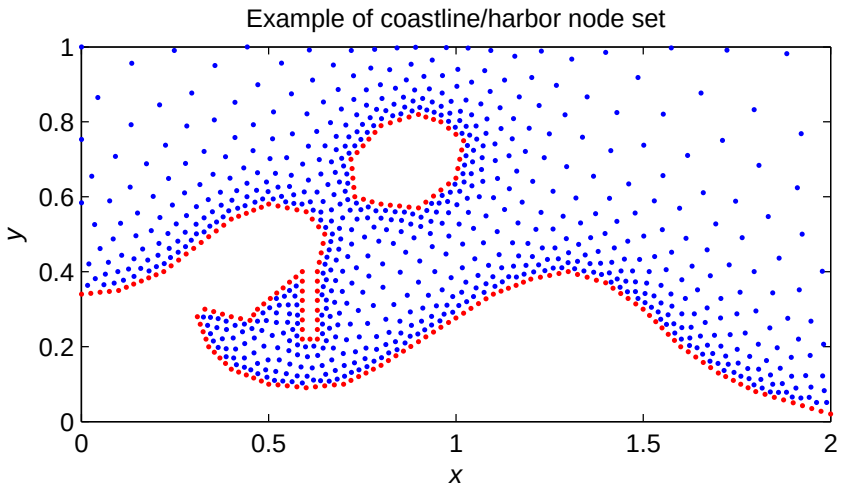

Figure 1: Conceptual illustration of a small ($N = 1438$) RBF-FD node set for coastal hazard modeling including a harbor, breakwater, and nearby island.

and tori. This is followed by some conclusions.

2 The basic algorithm: Creating node sets of variable density in a rectangular domain

2.1 The basic node placing algorithm

We first present the algorithm with the Matlab code (see Appendix) for generating any type of node distribution in a rectangular domain, and then generalize later to irregular boundaries and surfaces. As stated in the introduction, this is an advancing-front type algorithm as shown in Figure 2. It starts by randomly scattering 'potential dot locations' (PDPs) relatively densely along the bottom edge of the rectangle and then proceeds by the following steps, also illustrated in subplots 1-4, respectively, of Figure 3:

repeat until lowest PDP is above the top edge of the domain

1. Identify the lowest PDP (if several are equally low, choose any one), and select it as a new dot location.
2. Mark out a circle centered at the new dot and with a radius matching the desired node separation distance at this location.
3. Keep the new dot, but remove all other PDPs within this circle.
4. Note the directions to the nearest remaining PDP on each side. On the circle periphery, within this sector, place five new PDP equally spaced in angle.

end repeat

In step 2, the radius is what we also describe as the 'grain size'. For image rendering, it will be a mathematical function of the brightness, with large radii in light areas and small in dark

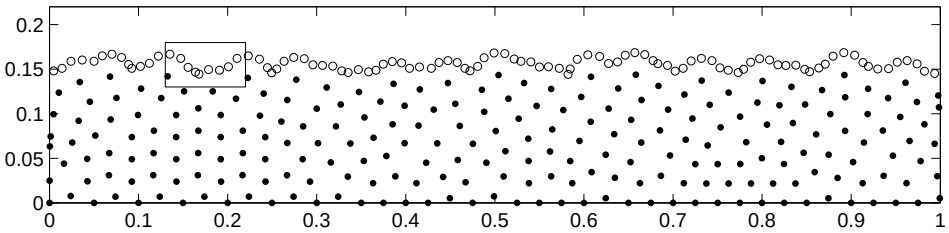

Figure 2: Illustration of the advancing front after several loops of steps 1-4 of the basic algorithm. The dots in the rectangle are used in Figure 3 to demonstrate steps 1-4.

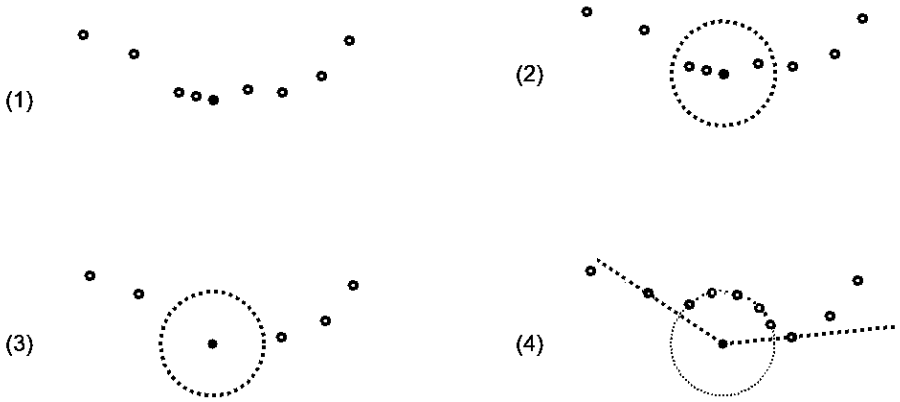

Figure 3: The steps 1-4 in the present algorithm.

ones; cf. Section 2.2.1. Regarding the number of new PDP in step 4, using less than five may degrade the quality of the distribution, and more than eight becomes computationally wasteful. At (non-periodic) side boundaries, new PDPs outside the domain are immediately removed.

2.2 A test case: Dithering of a gray-scale image

Although not directly related to our primary interest in PDE discretizations, we choose as our first demonstration the task of dithering a grey scale image, since

i. The very sharp transitions between very light and very dark regions provide a more challenging test case than what typically arises in the context of PDE discretizations, and

ii. It offers an opportunity to compare the present approach against the state-of-the-art in an area in which numerous node distribution strategies have already been extensively studied.

We consider the 256×256 gray scale image shown in Figure 4 (a). Since it is commonly used for testing image rendering schemes, it is readily found by a web search for 'trui.png'. The rendering shown in Figure 4 (b) is obtained by the Matlab code given in the Appendix. It contains $N = 36,303$ dots and requires 3 seconds of computing on a standard laptop, using only a single core. Since scalar operations dominate, it is expected that conversion to Fortran or C++ will reduce this by more than an order of magnitude, to a fraction of a second. Figure 5 illustrates that the dot distribution

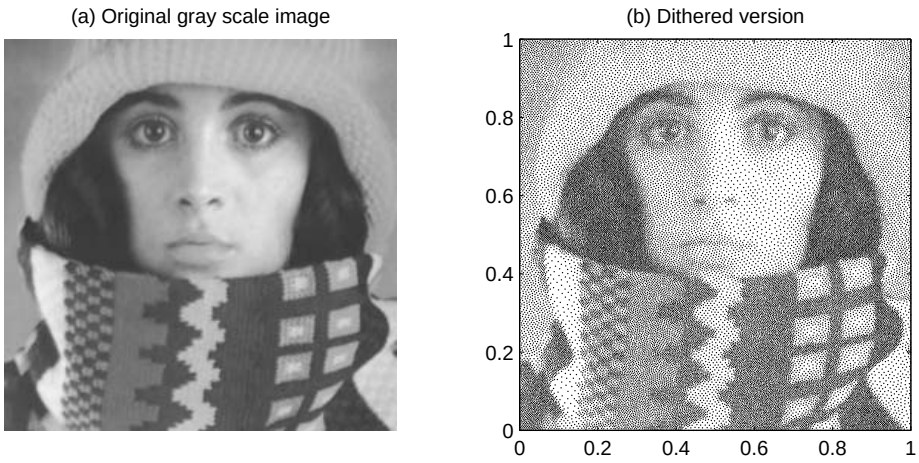

Figure 4: Test image trui.png and its discrete rendering with the present algorithm, using $N = 36,303$ dots/nodes.

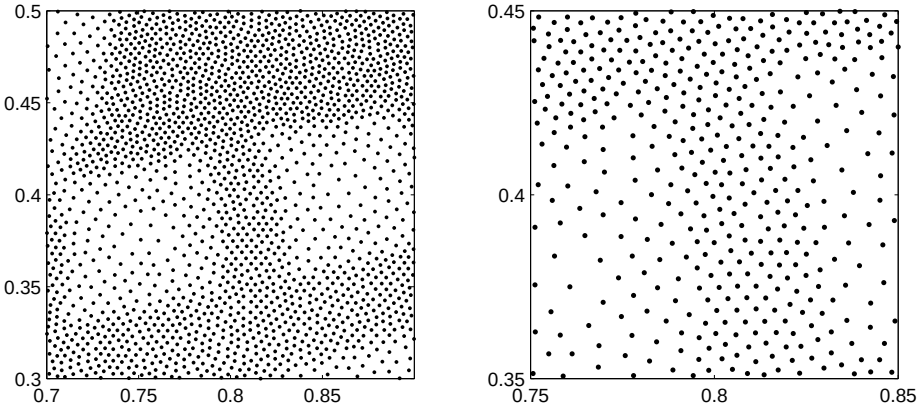

Figure 5: Successive enlargements of Figure 4 (b) around the coordinate location $(0.8, 0.4)$, showing node details also in darker / denser areas of the image.

retains a locally quasi-uniform character also in dark areas and transitions without artifacts where density gradients are large.

The dot distributions shown in Figures 4 and 5 can be compared to similar ones, for the same image, reported in [11, 24]. The numerical approach used there first creates a quadratic functional which incorporates the original image gray-scale information. This is then iteratively minimized until the desired dot distribution is reached. Conjugate gradients (and, for nodes on a sphere/torus, non-equispaced FFTs) can be applied. However, computer times nevertheless become substantial, reported in [11] to be between 15 minutes and one day for $N = 30,150$ renderings of different qualities of this same 'trui' image (with the use of C++ coding).

2.2.1 Conversion between gray levels and grain radii

Each pixel of the original image has a brightness varying from 0 (black) to 255 (white). When scaled to the range $s \in [0, 1]$, the histogram in Figure 6 (a) shows the number of pixels of different brightness levels. When converting each pixel's brightness s to an actual grain radius $r(s)$ (Step 2

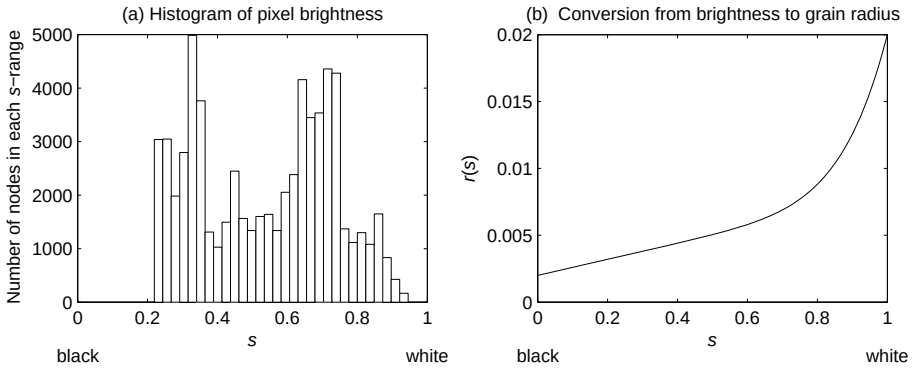

Figure 6: (a) Distribution of brightness for the pixels in the trui image, (b) The conversion function $r(s)$ between pixel brightness and grain radii used for the present trui image rendering.

in the algorithm description in Section 2.1), the main considerations are:

- Solid black ($s = 0$) should correspond to a small but non-zero radius $r(0)$,
- The function $r(s)$ should be monotonically increasing and, in order to represent white well, $r(s)$ should be increasing very rapidly as $s = 1$ is approached,
- The slope of the curve for $r(s)$ controls the contrast of the image.

Basically any function meeting these criteria will work well. Figure 6 (b) shows the one used in the demonstration code in the Appendix. The total number of dots in an image can be adjusted by multiplying the function $r(s)$ by a scalar.

For PDE applications, the grain radii (node separations) can either be given a priori, or altered dynamically. For the latter case, somewhat similar conversion functions have been described in the finite element literature (in which case node separations typically relate to some local error measure).

3 Generating RBF-FD node sets: Combining the basic algorithm with local node adjustment near boundaries

In the previous section's test case, there was no need to adjust nodes near/along boundaries. However, in generating node layouts for RBF-FD discretizations, the details of how the nodes are distributed along/near the boundaries are critical for accurately imposing a PDE's boundary conditions. The algorithm, as described above, will not automatically conform to the exact boundary; cf. the details shown in Figure 7. In PDE contexts, the domain shape is furthermore often non-rectangular. To address these issues, some remarks on 'node repel' methods are made. Following that, a method of how to combine the node placing algorithm with a local repel adjustment near boundaries is discussed, both in the case of a commonly-used test problem for seismic wave modeling and in the coastline example given in Figure 1.

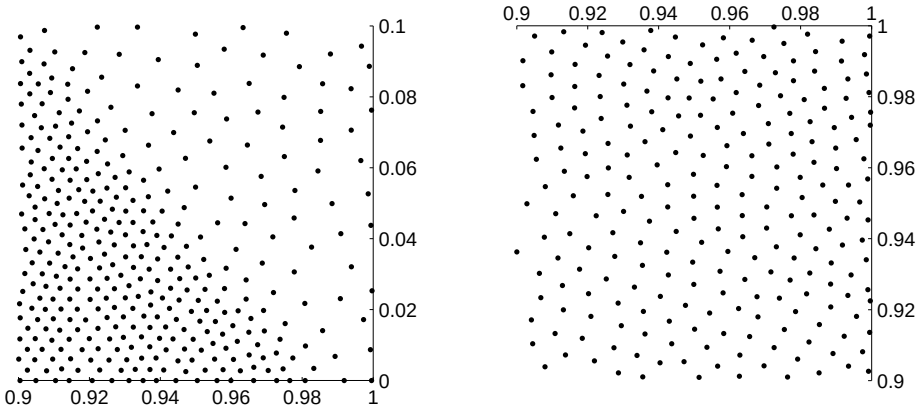

Figure 7: Detail of bottom right and top right corners of the dithered image in Figure 4 (b). Only at the bottom edge do the nodes fall precisely on the boundary.

3.1 Some observations with regard to 'node repel' algorithms

The strategy of adjusting node locations based on a repelling force acting between nearby nodes can be highly efficient for obtaining good distributions *on a local level*. However, it should be noted that this usually becomes very inefficient if the initial node distribution is far from the desired equilibrium. Figure 8 illustrates this. In this simple 1-D test case, it is assumed that each node exerts a force of the strength $\rho(x)/r^2$ on its immediate neighbor on each side, located the distance r away. A node at location x_k, between nodes at locations x_{k-1} and x_{k+1}, will thus reach its local force equilibrium when moved to the adjusted location $\frac{x_{k-1}\sqrt{\rho(x_{k+1})}+x_{k+1}\sqrt{\rho(x_{k-1})}}{\sqrt{\rho(x_{k+1})}+\sqrt{\rho(x_{k-1})}}$. With $\rho(x)$ as shown in Figure 8 (a), we see in part (b) how the movement of initially uniformly placed nodes occurs by a process akin to diffusion. More than 10^4 relaxation steps are needed over all the nodes before the global process has settled. In sharp contrast to 'transporting' nodes much further than their typical separation distances (e.g. as seen in the right part of Figure 8 (b)), merely adjusting node locations to that of their immediate neighbors is a rapid process. It is in this manner that we use node repelling.

The strategy used in *distmesh2d* is to initally distribute nodes on a dense near-hexagonal grid, and then remove nodes by a random process that leads to roughly the desired density variations. This is followed by an iterative scheme to adjust nodes locally using Delaunay trangularizations rather than a local repel process. For the trui test problem, almost independently of the total number of nodes, each of these successive iterations requires about 1/4 of the total time than the present node placing method to complete. However, considerably more than four iterations are typically needed to remove 'artifacts' originating from the initial node layout.

3.2 An RBF-FD node distribution strategy

In the following two subsections, examples are given on how to generate RBF-FD node distributions for domains with boudanries and interfaces. Although the examples differ in the details, the general strategy is the same for both cases. The overall steps are as follows:

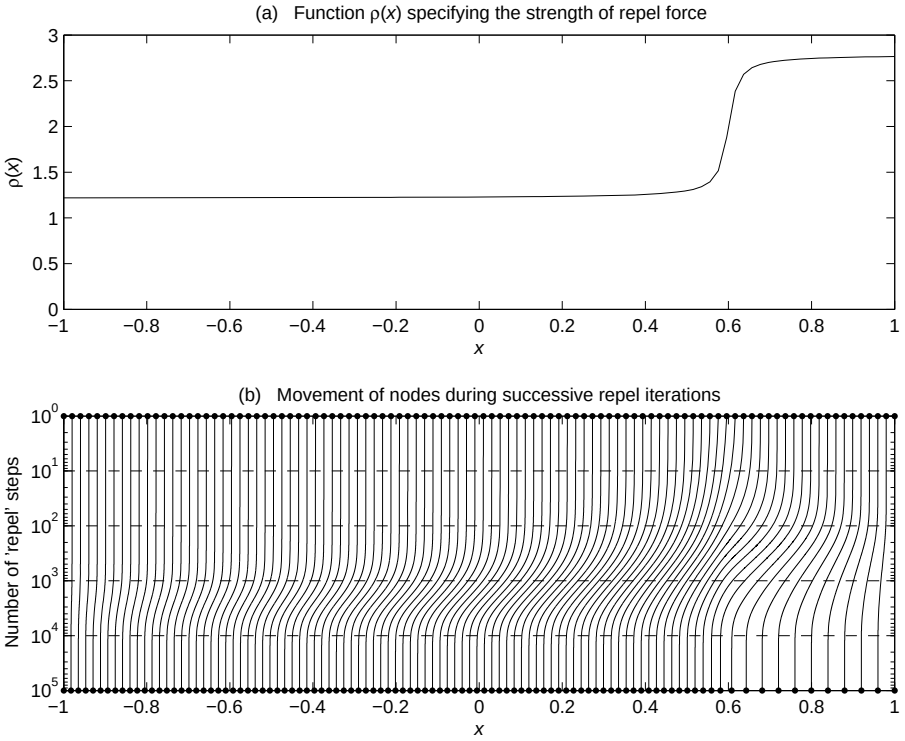

Figure 8: Illustration of repel algorithm in a case where the nodes initially are some distance away from their final locations; (a) 1D repel coefficient $\rho(x) = 2 + \frac{1}{2}\arctan(60(x - 0.6))$; (b) Movement of $N = 100$ nodes during 10^5 repel iterations, each of which adjusts all the nodes as described in Section 3.1.

1. Fill the entire domain of a rectangle using the basic node algorithm presented in Section 2.1. The nodes can be quasi-uniform or not.

2. Superimpose equi-spaced boundary and/or interface nodes.

3. Discard nodes that are within a distance of $r/2$ from any boundary and/or interface node.

4. Perform a *few* local repel steps *only* on nodes that are in the *immediate* vicinty of a bounday / interface node (e.g. 20 nearest neighbors).

3.2.1 An RBF-FD node distribution for seismic wave modeling

Figure 9 illustrates the *Marmousi* test case, which is widely used for evaluating seismic wave propagation codes. Pressure waves are initiated at different surface locations (top boundary), and returning pressure and shear waves need to be calculated. These returning signals arise mostly from the numerous material interfaces that separate the different geological strata. For each wave that arrives at an interface, four main outgoing waves are produced - reflected and transmitted pressure and shear waves. The RBF-FD approach to this type of simulations ([17, 18, 19]) uses for each region bound by interfaces a node set that is quasi-uniform both along its boundaries and inside. To visualize individual node locations, the domain is simplified further in Figure 10. Part (a) shows a rectangular region filled with quasi-uniform nodes obtained with the present node placement algorithm (in this example using a grain size of radius $r = 0.03$ throughout). Superimposed on this is the desired node locations along the domain boundary and the interfaces, shown as small blue circles. We next (i) discard all nodes outside the domain boundary and within a distance of $r/2$ of any boundary / interface node, and (ii) run a few node repel steps *only* on nodes in the immediate vicinity of any boundary/interface point. In this process, none of the boundary / interface points are moved. This leads to the node distribution in part (b). All the calculations behind Figure 10 (including the node placement and the repel steps) require only a fraction of a second in Matlab. One contributing factor to the high computational speed is the use of the k-d tree algorithm for finding a given number of nearest neighbors for any subset or even all the nodes in a domain. This algorithm is conveniently available in several versions in Matlab's statistics toolbox (*knnsearch*, *rangesearch*, *createns*).

In the context of seismic exploration, the 'forward' wave propagation problem needs to be solved very large numbers of times in order that 'inverse' solvers will be able to deduce underground structures from reconciling measured and simulated wave responses at the surface. Since the structures are constantly adjusted in this iterative process, it is critical that the necessary node sets be computed very rapidly.

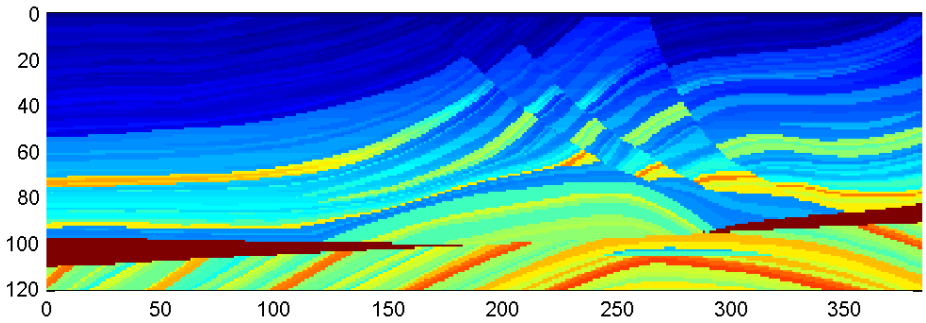

Figure 9: Illustration of the variable medium properties/interfaces in the 2-D Marmousi test case for seismic modeling.

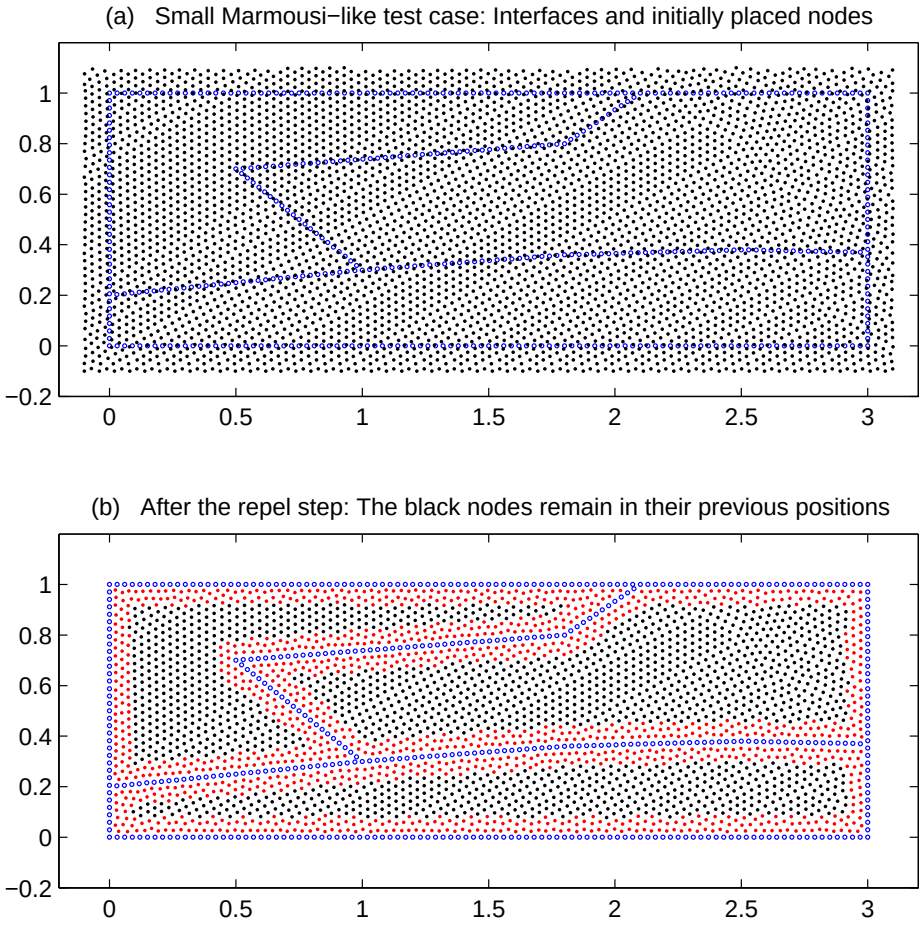

Figure 10: The steps in creating node sets that align with boundaries and interfaces illustrated in a small Marmousi-like test case: (a) Domain boundary and interface nodes (blue circles) are shown superimposed on an independently generated quasi-uniform node set, which extends over a slightly larger domain (black dots), (b) Resulting node distribution after the local repel process. Only the nodes marked red were moved - all the black and blue ones remain in their previous positions.

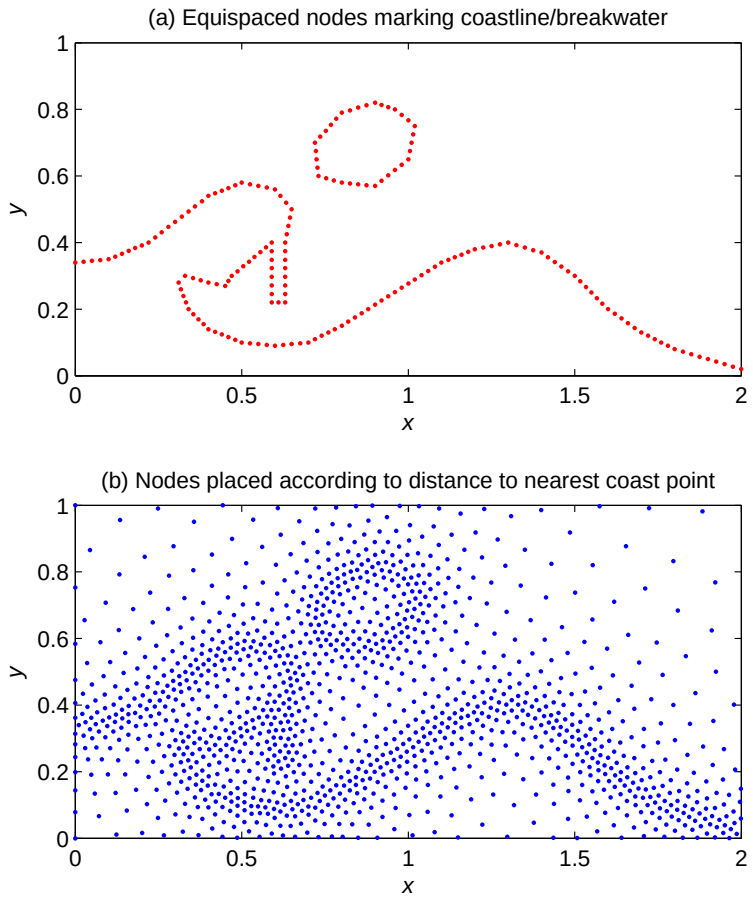

Figure 11: The steps behind the generation of Figure 1.

3.2.2 An RBF-FD node distribution for coastal hazard modeling

Figure 11 (a) shows the boundary coastline / breakwater (with nearby island) marked by a set of roughly equispaced nodes. For any location in the the (x, y)-plane, the distance $d(x, y)$ to the neareast boundary node can be easily calculated. Figure 11 (b) shows the result of using the present node placing algorithm with a variable grain size of radius $r = 0.02 + 0.2\sqrt{d(x, y)}$. The result is that nodes become more densely packed as they approach the boundary coastline, while gently becoming sparser out to sea. In the last step, leading to Figure 1, we (i) superpose the node set in Figure 11 (a) on the node set in part (b), (ii) holding the coastline nodes fixed, run a local repel on the nodes immediately adjacent to these, and (iii) discard nodes outside the domain of interest (in this case the side that represents land).

4 Node distributions on 2-D surfaces embedded in 3-D space

The most straightforward approach for the surface of a sphere, as for many other curved surfaces, is to use conformal mappings to reduce the task to one or several planar node distribution cases. An additional option is noted in Section 4.4.

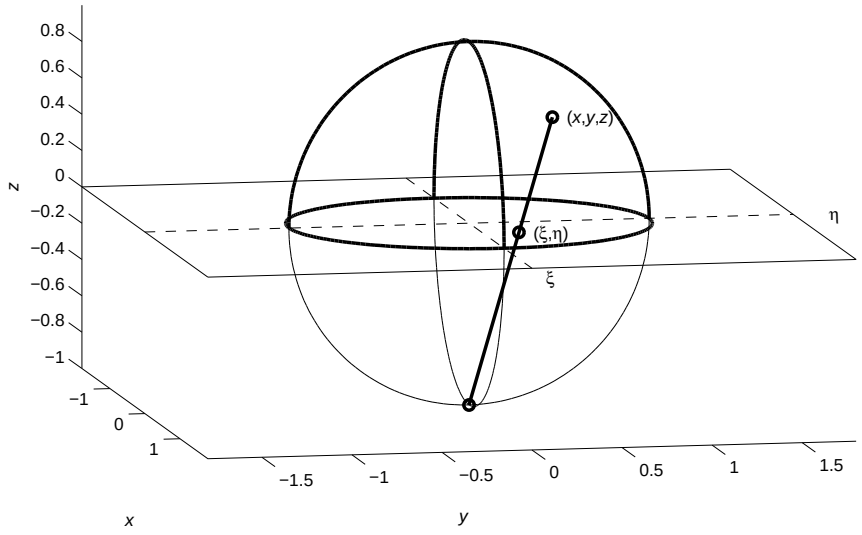

Figure 12: Concept of a stereographic projection - the mapping between the upper half of the unit sphere in x, y, z-coordinates and the unit disk in ξ, η-coordinates, as described by (1).

4.1 Node sets with quasi-uniform density on a sphere

Figure 12 illustrates the standard stereographic (conformal) mapping

$$\left\{ \begin{array}{ll} x = & \frac{2\,\xi}{1+\xi^2+\eta^2} \\ y = & \frac{2\,\eta}{1+\xi^2+\eta^2} \\ z = & \frac{1-\xi^2-\eta^2}{1+\xi^2+\eta^2} \end{array} \right. \quad , \qquad \left\{ \begin{array}{ll} \xi = & \frac{x}{1+z} \\ \eta = & \frac{y}{1+z} \end{array} \right. \quad . \tag{1}$$

between the upper half of a unit sphere in x, y, z-space and the unit disk in a ξ, η-plane (in the figure coinciding with the x, y-plane). A similar mapping for the lower half of the unit sphere is obtained by changing $z \to -z$ in (1). However, each of the two mapping cases can be used also on the opposite hemisphere if one makes the changes

$$\{\xi, \eta\} \leftrightarrow \left\{ \frac{\xi}{\xi^2+\eta^2} \, , \, \frac{\eta}{\xi^2+\eta^2} \right\} \, . \tag{2}$$

A small circle with radius r on the surface of the sphere will, in the ξ, η-plane, correspond to a circle with radius

$$\rho(r) = \frac{r}{1+z} = \frac{r}{2}\,(1+\xi^2+\eta^2) \, . \tag{3}$$

Hence, any node density function on the sphere is trivially translated to its counterpart in the ξ, η-plane. This ρ-function is not explicitly chosen by the user, but follows from the conformal mapping and the desired node separation function $r(x, y, z)$ on the surface. As illustrated in Figure 13 and described in its caption, nodes are placed using the algorithm in Section 2.1 with radius $\rho(r = 0.06)$ in a square in the ξ, η-plane. Then a band of boundary nodes, marked as blue circles in Figure 13 (a), are chosen (see caption). For the opposite hemisphere, in Figure 13 (b), the same interior (black) nodes are used as in Figure 13 (a); however, the location of the boundary nodes are forced by the transformation (2). This then requires a local repel process to be performed on the interior nodes near the boundary (in red). Combining the two hemispheres gives the top row of subplots in

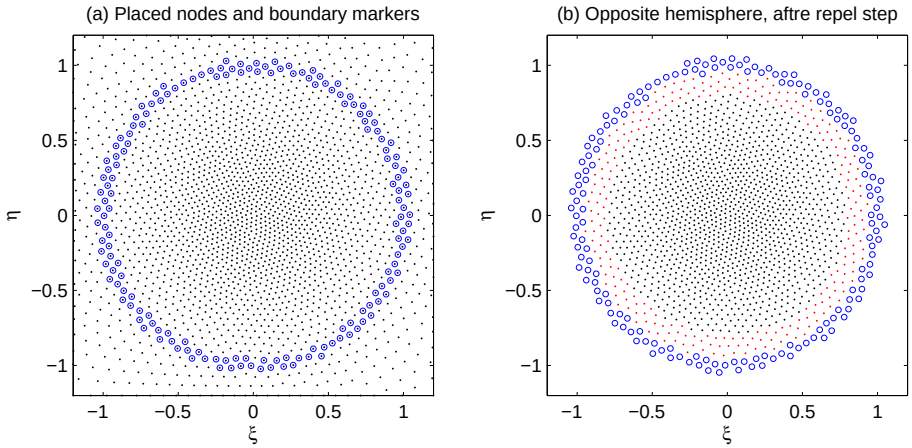

Figure 13: (a) Nodes placed over $\xi, \eta \in [-1.2, 1.2] \times [-1.2, 1.2]$ with radii given by (3) using $r = 0.06$. Nodes located within a thin band around the periphery of the unit circle are marked by small blue circles, and will be referred to as 'boundary' nodes. The band, here given by $0.95 < \sqrt{\xi^2 + \eta^2} < 1.05$, needs to be wide enough that it does not have any gaps along the circle periphery that are longer than typical node separations, (b) The blue circles show the same boundary nodes after the transformation given in (2). Nodes outside this new boundary have been discarded, and the repel process has been carried out on the interior nodes shown in red.

Figure 14. An alternative, but more laborious and redundant approach, would have been to do a separate node placing for each hemisphere. In many applications that require quasi-uniform nodes on the sphere, details in the distributions matter very little (such as the distinctions between MD and ME nodes; this has in particular been noted for numerical quadrature on the sphere [10]).

There are many ways to measure / illustrate 'node quality', one of which is to create histograms of the distances from each node to its nearest neighbor. Figure 15 shows this for the three node sets displayed in Figure 14. Although the peak locations of the three distributions are somewhat different, the 'spreads' are comparable, indicating that they are all about equally well suited in the present RBF-FD context.

4.2 Node sets with highly variable density on a sphere

We choose here the same dithering test case as is shown in Figure 9 of [11]. The data is available as a 180×360 matrix in the file topo.mat in Matlab's Mapping toolbox, cf. Figure 16. Nodes are placed, as illustrated in Figure 13, but now the grain size is also scaled with the data via the translation function $r(s)$ as in the trui.png image. The execution time in Matlab, on a standard notebook, is again just over 10,000 nodes/second, i.e. the $N = 105,419$ node distribution shown in Figure 17 required just less than 10 seconds.

The projection used in Figure 17 may give the optical illusion that nodes near the sphere's outer edge are 'lined up' more distinctly than as displayed in the center of the image, and also that the edge regions are darker. However, this is not the case. The same effects are seen in all the subplots in Figure 14. The central area of all these figures convey most clearly the node patterns.

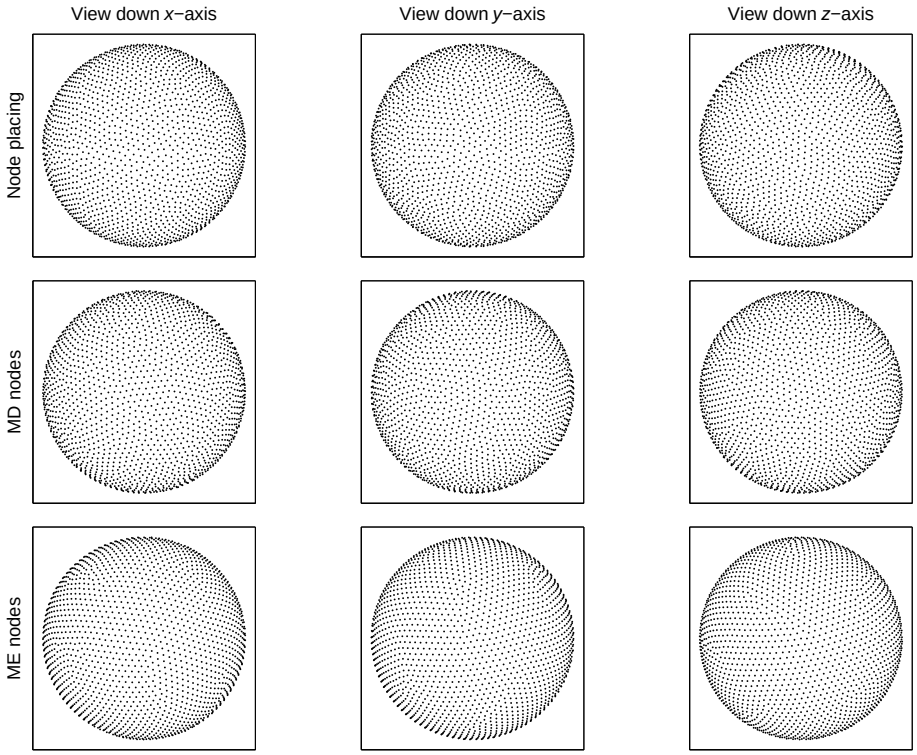

Figure 14: Top row of subplots: The resulting quasi-uniform node distribution obtained when combining the interior nodes from Figures 13 a,b with the boundary nodes, and then transferring these to the sphere by means of the mappings in (1); $N = 3,343$ nodes displayed from three different angles. Bottom two rows of subplots: The corresponding $N = 3,364$ MD and ME node distributions, tabulated in [27].

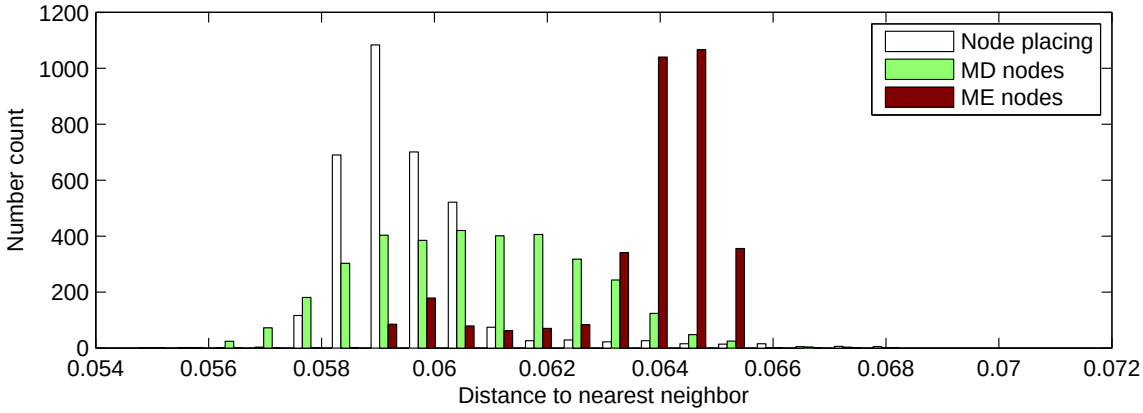

Figure 15: Histograms showing the distance from each node to its nearest neighbor in the three distributions in Figure 14.

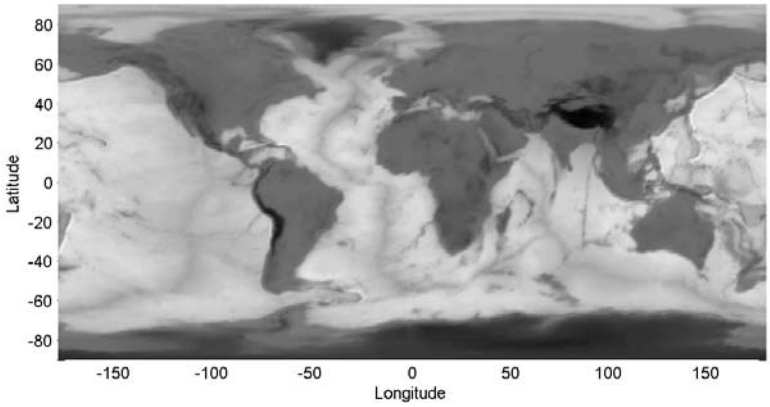

Figure 16: Gray scale rendering of the file topo.mat in Matlab's Mapping toolbox.

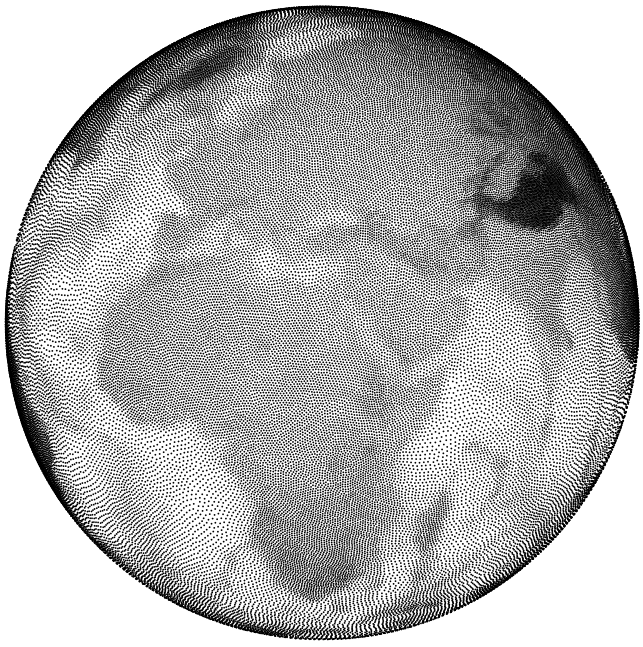

Figure 17: Dithered version of the data from Figure 16, $N = 105,419$ dots.

4.3 Nodes on a torus

The case of a torus (also considered in [11]) is simpler than that of a sphere in that its surface can be conformally mapped to a single rectangle (i.e. does not require the composite of two planar cases). Figure 18 illustrates the conformal mapping between a torus with aspect ratio $\sqrt{2}$ and a periodic square. It is given by

$$\left\{ \begin{array}{rcl} x &=& f(\xi)\ \cos\ \pi\eta \\ y &=& f(\xi)\ \sin\ \pi\eta \\ z &=& f(\xi)\ \sin\ \pi\xi \end{array} \right. \quad , \qquad \left\{ \begin{array}{rl} \xi = & \text{atan2}(z, \sqrt{2}\sqrt{x^2+y^2}-1) \\ \eta = & \text{atan2}(y,x) \end{array} \right. \tag{4}$$

where $f(\xi) = 1/(\sqrt{2} - \cos\ \pi\xi)$ and $\text{atan2}(y', x')$ denotes the polar coordinate angle for a general Cartesian location (x', y'). The counterpart to (3) becomes in this case

$$\rho = \frac{r}{\pi\ \sqrt{x^2+y^2}} = \frac{r}{\pi}\ (\sqrt{2} - \cos\ \pi\xi)\ .$$

Tori with other aspect ratios can similarly be conformally mapped to rectangles.

4.4 Nodes on general surfaces

If the upper half of a sphere is generalized to a simply connected part of an arbitrarily curved surface, conformal mappings can still be found to planar 2-D regions, making the presented algorithm directly applicable. For a general surface, the mappings are not available in closed form, but Table 1 in [12] lists no less than five different computational options. The surfaces can be extraordinarily convoluted, with the provided examples including a hand and a foot, the head of Michelangelo's David statue, and the heavily folded surface of the brain. Two of the methods are reviewed in more detail in [13] (based on holomorphic differentials and on surface Ricci flows, respectively).

As an alternative to relying on conformal mappings, the node placing procedure for 2-D rectangles presented in Section 2.1 can be directly generalized to curved surfaces. For example, in the case of a sphere, nodes can directly be placed at the lowest PDP available on its surface. Then, the only location at which a final repel adjustment becomes called for will be in the immediate vicinity of the North Pole itself, involving only the last 10 or so placed nodes. However, the algebra becomes slightly more involved than for the conformal mapping approach described in Section 4.1.

5 Conclusions

Numerous applications require node distributions of greatly variable density while still locally maintaining a quasi-uniform structure. The proposed scheme is very simple to implement (cf. the Matlab code in the Appendix). It provides a computationally fast alternative to algorithms based on energy minimization and Voronoi tessellations, with the additional benefit of both algorithmic and mathematical simplicity. The high computational speed is especially significant for time dependent problems that require the distribution to be updated frequently.

For irregularly shaped 2-D regions, node distributions can be specified as desired along the boundaries. This provides an approach for creating variable density node sets not only for planar regions, but also for curved surfaces. In some applications, such as creating node sets for seismic modeling, the presence of many separate regions allows straightforward parallelization on distributed memory computers.

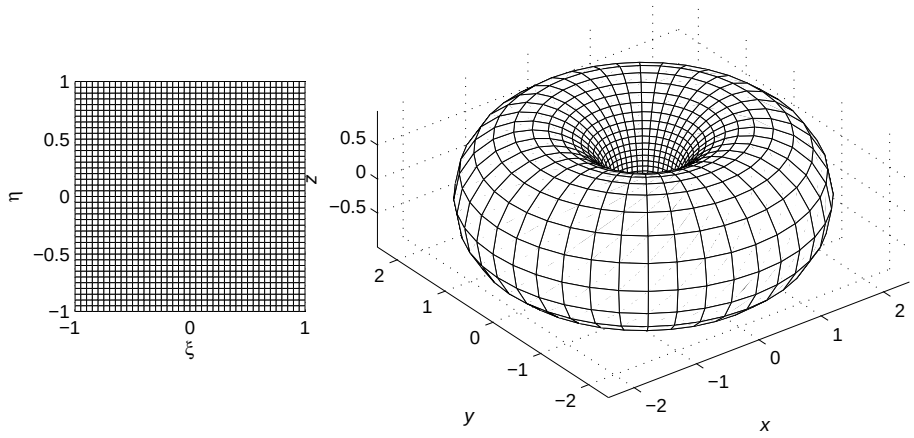

Figure 18: The *conformal* mapping between the periodic square $\xi, \eta \in [-1, 1]$ and a torus, as described by equation (4).

6 Appendix: Matlab code for rendering the trui image

The function *node_placing* is generic for placing nodes in a rectangle of any size. The brief 'main script' and the function *radius_trui* provide an example of how the function *node_placing* can be called. The values for the variables ninit and dotmax are only used to size internal arrays. Their values can be vastly increased without any adverse effects apart from proportional increases in memory allocations. If the file trui.png is not conveniently available, any other .png image file can be substituted with only trivial changes in the main script.

```
function xy = node_placing (box, ninit, dotmax, radius)

% --- Input parameters ---
%   box     Size of box to be filled by nodes:  [xmin,xmax,ymin,ymax]
%   ninit   Upper limit on number of PDP entries
%   dotmax  Upper bound for number of dots to place
%   radius  The function radius(xy) provides grain radius to be used
%           at location (x,y).

% --- Output parameter ---
%   xy      Array xy(:,2) with the generated node locations

dotnr = 0;                          % Counter for the placed dots
rng(0);                             % Initialize random number generator
pdp     = [linspace(box(1),box(2),ninit)',box(3)+1e-4*rand(ninit,1)];
pdp_end = ninit;                    % Number of PDPs in use
xy = zeros(dotmax,2);               % Array to store produced dot locations
[ym,i] = min(pdp(1:pdp_end,2));     % Locate PDP with lowest y-coordinate

while ym <= box(4) && dotnr < dotmax;% Loop over all dots that are to be placed
    dotnr = dotnr + 1;              % Keep count for next dot to be placed
    xy(dotnr,:) = pdp(i,:);         % Place the dot
    r = radius(xy(dotnr,:));        % Get grain radius to be used

% --- Calculate the distance from the placed dot to all present PDPs
    dist2 = (pdp(1:pdp_end,1)-pdp(i,1)).^2+(pdp(1:pdp_end,2)-pdp(i,2)).^2;

% --- Find nearest old PDP to the left, outside the new circle
    ileft = find(dist2(1:i) > r^2,1, 'last' );
    if isempty(ileft );
        ileft    = 0;               % Special case if no such point
        ang_left = pi;
    else
        ang_left = atan2(pdp(ileft,2)-pdp(i,2),pdp(ileft,1)-pdp(i,1));
    end

% --- Find nearest old PDP to the right, outside the new circle
    iright = find(dist2(i:pdp_end) > r^2,1, 'first');
    if isempty(iright);
        iright    = 0;              % Special case if no such point
        ang_right = 0;
    else
        ang_right = atan2(pdp(i+iright-1,2)-pdp(i,2),pdp(i+iright-1,1)-pdp(i,1));
    end

% --- Introduce five new markers along the circle sector, equispaced in angle
    ang     = ang_left-[0.1;0.3;0.5;0.7;0.9]*(ang_left-ang_right);
    pdp_new = [pdp(i,1)+r*cos(ang),pdp(i,2)+r*sin(ang)];
    ind     = pdp_new(:,1) < box(1) | pdp_new(:,1) > box(2);
    pdp_new(ind,:) = [];            % Remove any new PDPs outside the domain
    nw = length(pdp_new(:,1));      % Number of new markers to be inserted

% --- Remove obsolete and insert new PDPs in the array pdp
    if iright == 0                  % Place rightmost block of old PDPs
        pdp_end = ileft+nw;         % to the right of the block of the new PDPs
    else
        ind     = i+iright-1:pdp_end;
        pdp_end = ileft+nw+pdp_end-i-iright+2;
        pdp(ileft+nw+1:pdp_end,:)  = pdp(ind,:);
    end
    pdp(ileft+1:ileft+nw,:)  = pdp_new; % Insert the new PDPs into pdp

% --- Identify next dot location, then iterate until all dots are placed
[ym,i] = min(pdp(1:pdp_end,2));
end
xy = xy(1:dotnr,:);                 % Remove unused entries in array xy
```

```
% --- Main script for generating the dithered trui image
close all; clear all;
ninit  = 5e+4;          % Upper bound on potential dot positions (PDPs) to use
dotmax = 5e+5;          % Upper bound on number of dots to place

% --- Carry out the node placing
xy = node_placing([0 1 0 1],ninit,dotmax,@radius_trui);
% --- Display the resulting dithered image
plot(xy(:,1),xy(:,2),'k.','MarkerSize',2.1); axis square
```

```
function r = radius_trui(xy)            % Return grain radius at location (x,y)
persistent F

if isempty(F)                           % Read in the trui image once
    A = double(imread('trui.png','PNG')); A = flipud(A(:,:,1));
    rf = @(s) 0.002+0.006*s+0.012*s.^8; % Conversion of brightness to grain radius
    F = rf(A/255);
end

ixy = round(255*xy);                    % Given a location (x,y), evaluate
r = F(ixy(:,2)+256*ixy(:,1)+1);         % the corresponding grain radius
```

Acknowledgments:

The presented research was supported by the NSF grants DMS-0914647, DMS-0934317, OCI-0904599 and by Shell International Exploration and Production, Inc.

References

[1] M. Balzer, T. Schlömer, and O. Deussen. Capacity-constrained point distributions: A variant of Lloyd's method. *ACM Trans. Graphics*, 28(3):86:1–86:8, 2009.

[2] R. Bridson. Fast Poisson sampling in arbitrary dimensions. *ACM SIGGRAPH, Sketches Program*, page (DOI) 10.1145/1278780.1278807, 2007.

[3] G. M. Ortigoza Capetillo. Triangular grid generators for the eigenvalue calculation with edge elements. *Revista Mexicana de Fisica*, 55:154–160, 2009.

[4] Y. Choi, K-Y. Kwon, S-W. Chae, and D-M. Kim. A modified advancing layers mesh generation for thin three-dimensional objects with variable thickness. *Comp. & Graph.*, 33:195–203, 2009.

[5] N. Flyer and E. Lehto. Rotational transport on a sphere: Local node refinement with radial basis functions. *J. Comput. Phys.*, 229:1954–1969, 2010.

[6] N. Flyer, E. Lehto, S. Blaise, G. B. Wright, and A. St-Cyr. A guide to RBF-generated finite differences for nonlinear transport: Shallow water simulations on a sphere. *J. Comput. Phys*, 231:4078–4095, 2012.

[7] N. Flyer, G. B. Wright, and B. Fornberg. Radial basis function-generated finite differences: A mesh-free method for computational geosciences. In W. Freeden, M.Z. Nashed, and T. Sonar, editors, *Handbook of Geomathematics*, Berlin, 2014. Springer Verlag.

[8] B. Fornberg and N. Flyer. *A Primer on Radial Basis Functions with Applications to the Geosciences*. SIAM, Philadelphia, 2015.

[9] B. Fornberg and N. Flyer. Solving PDEs with radial basis functions. *Acta Numerica*, 2015.

[10] B. Fornberg and J. M. Martel. On spherical harmonics based numerical quadrature over the surface of a sphere. *Adv. Comput. Math.*, pages (DOI) 10.1007/s10444–014–9346–3, 2014.

[11] M. Gräf, D. Potts, and G. Steidl. Quadrature errors, discrepancies, and their relations to halftoning on the torus and the sphere. *SIAM J. Sci. Comput.*, 34:A2760–A2791, 2012.

[12] X. Gu, Y. Wand, T. F. Chan, P. M. Thompson, and S.-T. Yau. Genus zero surface conformal mapping and its application to brain surface mapping. *IEEE Trans. Medial Imgaing*, 23:949–958, 2004.

[13] X. D. Gu, W. Zeng, F. Luo, and S.-T. Yau. Numerical computation of surface conformal mappings. *Comp. Meth. and Function Th.*, 2:747–787, 2011.

[14] Z. Guan, J. Shan, Y. Zheng, and Y. Gu. An extended advancing front technique for closed surfaces mesh generation. *Int. J. Num. Meth. Eng.*, 74:642–667, 2008.

[15] D. P. Hardin and E. B. Saff. Discretizing manifolds via minimum energy points. *Notices Amer. Math. Soc.*, 51:1186–1194, 2004.

[16] S. P. Lloyd. Least square quantization in PCM. *IEEE Trans. Information Th.*, 28(2):129–137, 1982.

[17] B. Martin, N. Flyer, B. Fornberg, and A. St-Cyr. Development of meshless computational algorithms for seismic exploration. *SEG Technical Program Expanded Abstracts*, pages 3543–3547, 2013.

[18] B. Martin, B. Fornberg, and A. St-Cyr. Seismic modeling with radial basis function generated finite differences (RBF-FD). *Geophysics, to appear*, 2015.

[19] B. Martin, B. Fornberg, A. St-Cyr, and N. Flyer. Seismic modeling with radial basis function-generated finite differences (RBF-FD). *SEG Technical Program Expanded Abstracts*, pages 3546–3550, 2014.

[20] W.-M. Pang, Y. Qu, T.-T. Wong, D. Cohen-Or, and P.-A. Heng. Structure-aware halftoning. *ACM Tans. Graph.*, 27:89:1–89:8, 2008.

[21] P.-O. Persson and G. Strang. A simple mesh generator in Matlab. *SIAM Rev.*, 46:329–345, 2004.

[22] Y. Y. Shan, C. Shu, and Z. L. Lu. Application of local MQ-DQ method to solve 3D incompressible viscous flows with curved boundary. *Comp. Mod. Eng. & Sci.*, 25:99–113, 2008.

[23] C. Shu, H. Ding, and K. S. Yeo. Local radial basis function-based differential quadrature method and its application to solve two-dimensional incompressible Navier-Stokes equations. *Comput. Meth. Appl. Mech. Engrg.*, 192:941–954, 2003.

[24] T. Teuber, G. Steidl, P. Gwosdek, C. Schmaltz, and J. Weikert. Dithering by differences of convex functions. *SIAM J. Imag. Sci.*, 4:79–108, 2011.

[25] A. I. Tolstykh and D. A. Shirobokov. On using radial basis functions in a "finite difference mode" with applications to elasticity problems. *Comput. Mech.*, 33:68–79, 2003.

[26] J. R. Tristano, S. J. Owen, and S. A. Canann. Advancing front surface mesh generation in parametric space using a Riemann surface definition. In *Proceedings of the 7th International Meshing Roundtable*, pages 429–445, 1998.

[27] R. S. Womersley and I. H. Sloan. Interpolation and cubature on the sphere. Website, 2003/2007. http://web.maths.unsw.edu.au/~rsw/Sphere/.

[28] G. B. Wright and B. Fornberg. Scattered node compact finite difference-type formulas generated from radial basis functions. *J. Comput. Phys.*, 212:99–123, 2006.