SAND REPORT

SAND2002-0099
Unlimited Release
Printed February 2002

User Manual and Supporting Information for Library of Codes for Centroidal Voronoi Point Placement and Associated Zeroth, First, and Second Moment Determination

John Burkardt, Max Gunzburger, Janet Peterson, and Rebecca Brannon

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/ordering.htm

SAND2002-0099
Unlimited Release
Printed February 2002

USER MANUAL AND SUPPORTING INFORMATION for LIBRARY OF CODES for CENTROIDAL VORONOI POINT PLACEMENT AND ASSOCIATED ZEROTH, FIRST, AND SECOND MOMENT DETERMINATION

John Burkardt, Max Gunzburger, and Janet Peterson
Iowa State University
Ames, IA 5001-2064

and

Rebecca Brannon
Materials Mechanics
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0893

Abstract

The theory, numerical algorithm, and user documentation are provided for a new "Centroidal Voronoi Tessellation (CVT)" method of filling a region of space (2D or 3D) with particles at any desired particle density. "Clumping" is entirely avoided and the boundary is optimally resolved. This particle placement capability is needed for any so-called "mesh-free" method in which physical fields are discretized via arbitrary-connectivity discrete points. CVT exploits efficient statistical methods to avoid expensive generation of Voronoi diagrams. Nevertheless, if a CVT particle's Voronoi cell *were* to be explicitly computed, then it would have a centroid that coincides with the particle itself and a minimized rotational moment. The CVT code provides each particle's volume and centroid, and also the rotational moment matrix needed to approximate a particle by an ellipsoid (instead of a simple sphere). DIATOM region specification is supported.

USER MANUAL AND SUPPORTING INFORMATION

for

LIBRARY OF CODES

for

CENTROIDAL VORONOI POINT PLACEMENT AND ASSOCIATED ZEROTH, FIRST, AND SECOND MOMENT DETERMINATION

John Burkardt, Max Gunzburger, and Janet Peterson
Iowa State University

Rebecca Brannon
Sandia National Laboratories

December 4, 2001

Contents

Chapter 1

Introduction

This publication describes the use of a library of codes that perform the following task:

> given a region $\Omega \in \mathbb{R}^d$, $d = 2$ or 3, and a positive integer N, construct N regions that tessellate Ω and determine the volumes (the zeroth-order moments), the centroids (the first-order moment vectors), and the second-order moment tensors corresponding to each subregion in the tessellation.

It is natural to choose the N regions to be a Voronoi tessellation of Ω. The question that first arises is: how does one choose the point set that generates the Voronoi tessellation? The particular Voronoi tessellation that our codes uses is a *centroidal Voronoi tessellation* of the given region Ω. In fact, the codes first determine such a tessellation, and subsequently determine the three desired sets of moments for each of the associated Voronoi regions or cells. Centroidal Voronoi tessellations result in very uniformly distributed point sets (or equivalently, subregions with very similar volumes), much more so than other point placement techniques. Centroidal Voronoi point sets also conform well to boundaries and can be characterized, in some sense, as a minimun "energy" configuration.

The next question is how does one efficiently determine the centroidal Voronoi tessellation and the desired moments? We, in fact, never explicitly determine the centroidal Voronoi tessellation, but instead use probabilistic methods to determine its generators. Similarly, we use probabilistic methods to determine the desired moments. As a result, our methodology is quite straightforward and efficient.

In summary, the library of codes performs the given task through the following steps:

- given the region Ω and the positive integer N, the N generators of a centroidal Voronoi tessellation of Ω are determined;
- once the N generators have been determined, the zeroth, first, and second-order moments are determined.

Central to both steps is the random sampling of uniformly distributed points in Ω. In order to do this, the codes require the user to supply a means for determining if a randomly

sampled point in an enclosing box is inside or outside the given region Ω. For example, we have been supplied with an interface to DIATOM[1] which accomplishes this task.

In this publication, we first provide, in Chapter 2, a discussion of the theoretical background underlying our methodology. Then, in Chapter 3, we provide some examples of the use of our codes which illustrate the performance of our methodology and the effects of the few parameters that the user must specify to run our codes. Finally, in Appendix A, we describe the different components of the codes and how to run the codes.

[1] DIATOM is the name of a geometry definitiion syntax used in the Sandia codes CTH [1] and ALEGRA [3].

Chapter 2

Theoretical background

2.1 Centroidal Voronoi tessellations

2.1.1 Voronoi tessellations

Let $|\cdot|$ denote the Euclidean norm in $\mathbb{R}^d$, $d = 2$ or 3. Given an open set $\Omega \subset \mathbb{R}^d$ and a set of points $\{\mathbf{z}_i\}_{i=1}^N$ belonging to $\overline{\Omega}$, let V_i dennote the region in Ω consisting of points that are closer to $\mathbf{z}_i$ than to any other $\mathbf{z}_j$, $j \neq i$, i.e.,

$$V_i = \{\, \mathbf{x} \in \Omega \quad | \quad |\mathbf{x} - \mathbf{z}_i| < |\mathbf{x} - \mathbf{z}_j| \quad \text{for } j = 1, \ldots, N,\, j \neq i \,\} \qquad i = 1, \ldots, N\,. \tag{2.1}$$

The set of regions $\{V_i\}_{i=1}^N$ are non-overlapping and tessellate Ω, i.e.,

$$V_i \cap V_j = \emptyset \quad \text{for } i \neq j \qquad \text{and} \qquad \cup_{i=1}^N \overline{V}_i = \overline{\Omega}\,.$$

The set $\{V_i\}_{i=1}^N$ is referred to as a *Voronoi tessellation* or *Voronoi diagram* of Ω, the members of the set $\{\mathbf{z}_i\}_{i=1}^N$ are referred to as *generating points* or *generators*, and each V_i is referred to as the *Voronoi region* or *Voronoi cell* corresponding to $\mathbf{z}_i$. Examples of Voronoi tessellations are given in Figure 2.1.

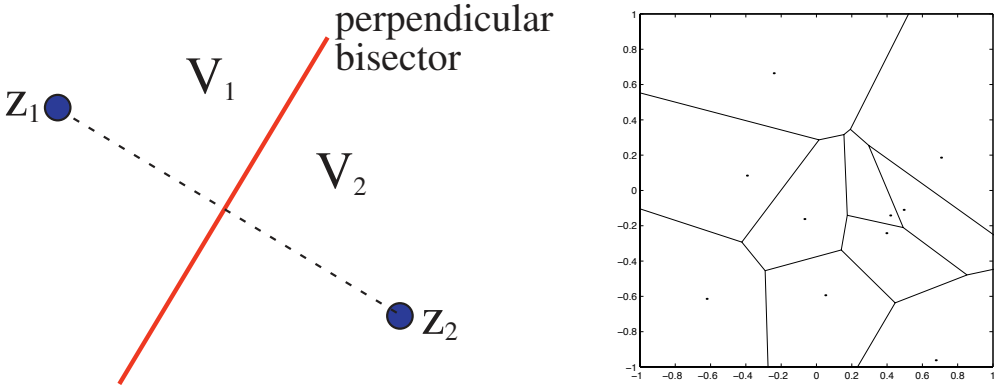

Figure 2.1: Voronoi regions for two points in the plane (left) and the Voronoi tessellation for 10 random points in a square (right).

It is well known that Voronoi regions are convex polyhedra and are very useful in a number of applications;[1] see, e.g., [11]. For example, Voronoi polygons and their *dual* tessellation, the *Delaunay triangulation*,[2] the are very useful in numerical computations, e.g., in interpolation, quadrature, partial differential equations, etc. Voronoi and Delaunay tessellations also possess many useful properties. For example, among all possible triangulations of a given set of points in the plane, the Delaunay triangulation is the one with *largest minimum angle*. This, in turn, is desirable, e.g., finite element approximations of partial differential equations. Another example is in "finite difference" discretizations of certain types of partial differential equations for which one must associate variables with *points*, and/or *regions*, and/or *edges*. On arbitrary domains, the best way to do this is to choose regions to be Voronoi polygons and/or Delaunay triangles, and to use the associated edges and vertices as needed.

2.1.2 Centroids

Given a point density function[3] $\mu(\mathbf{x})$ defined on $\overline{\Omega}$, for each Voronoi region V_i, we can define its *centroid*[4] $\mathbf{z}_i^*$ by the well-known formula

$$\mathbf{z}_i^* = \frac{1}{|V_i|}\int_{V_i} \mathbf{x}\mu(\mathbf{x})\,d\mathbf{x} \qquad \text{for } i = 1, \ldots, N\,, \tag{2.2}$$

where $|V_i|$ denotes the volume of V_i and $d\mathbf{x}$ denotes the volume element.[5] The generating points $\{\mathbf{z}_i\}_{i=1}^N$ of a Voronoi tessellation do not, in general, coincide with the centroids $\{\mathbf{z}_i^*\}_{i=1}^N$ of the corresponding Voronoi regions. See, e.g., the left plot in Figure 2.2 for the case of a uniform point density function.

2.1.3 Centroidal Voronoi tessellations

We call the tessellation defined by (2.1) a *centroidal Voronoi tessellation* (CVT) if and only if

$$\mathbf{z}_i = \mathbf{z}_i^* \qquad \text{for } i = 1, \ldots, N\,,$$

[1] In fact, Voronoi sets appear under many names (e.g., Dirichlet regions, Meijering cells, S-mosaics, Thiessen polygons, area of influence polygons, etc.), depending on the application.

[2] Given a Voronoi tessellation of a plane region, the Delaunay triangulation is determined by joining the generators of Voronoi cells that share a common edge as part of their boundaries.

[3] The point density function $\mu(\mathbf{x})$ allows for nonuniform point distributions. Later, we will also use a physical mass density function $\rho(\mathbf{x})$. Our codes apply to the case of uniform point and mass density functions. We treat only uniform point distributions because that is the desired output from our codes. However, for the sake of generality, in the discussion in this chapter, we will allow for both nonuniform point and mass density functions.

[4] A more common name for $\mathbf{z}_i^*$ might be *center of mass*. However, we reserve that name for the physical center of mass of a region determined with respect to a physical mass density; see Section 2.4.

[5] One can also define a centroid in case only a discrete set of points in Ω is given. Given the set of points $W = \{\mathbf{w}_j\}_{j=1}^J, j = 1, \ldots, J$, the centroid $\mathbf{z}^*$ of W is defined by

$$\mathbf{z}^* = \frac{1}{J}\sum_{j=1}^{J} \mu(\mathbf{w}_j)\mathbf{w}_j\,.$$

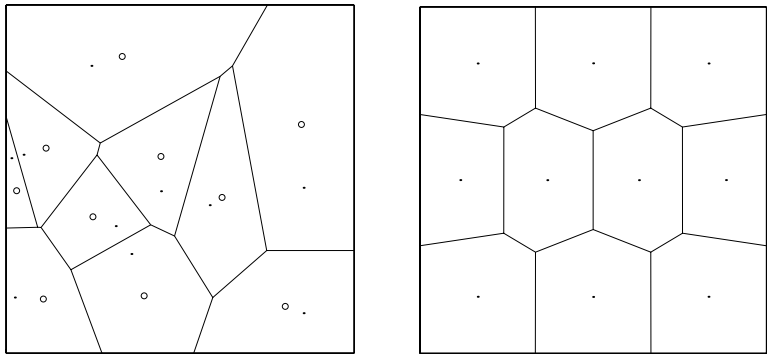

Figure 2.2: On the left, the Voronoi regions, their generators (dots), and their centroids (with respect to a uniform density) (circles) for 10 randomly selected points in a square. Note that the generating points and the centroids do not coincide. On the right, a 10-point centroidal Voronoi tessellation of a square. The dots are the generators of the Voronoi regions and are also the centroids of those regions.

i.e., the points $\{\mathbf{z}_i\}_{i=1}^N$ which serve as the generators associated with the Voronoi regions $\{V_i\}_{i=1}^N$ are also the centroids of those regions. An example of a CVT for a uniform point placement density is shown in the right plot of Figure 2.2.

In general, one *does not have uniqueness,* i.e., for a given positive integer N, there may be more than one N-point CVT of a given region Ω. Examples are provided in Figures 2.3 and 2.4.

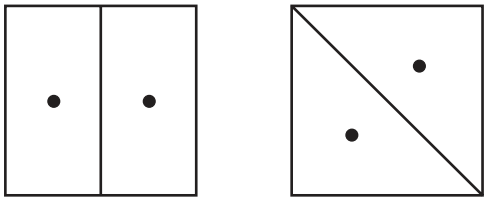

Figure 2.3: Two two-point centroidal Voronoi tessellations of a square.

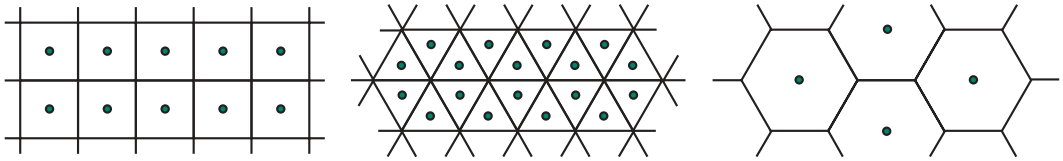

Figure 2.4: The three regular tessellations of the plane are centroidal Voronoi tessellations.

There are many applications for centroidal Voronoi tessellations; see, e.g., [5, 6, 8, 11] for applications in optimal allocation of resources, clustering, numerical integration, territorial behavior of animals, data compression, cell division, grid generation, meshless methods, etc. CVT's also possess many desirable properties. Although here we are mainly concerned with uniform points distributions, we give, in Figure 2.5, two examples of nonuniform CVT's.

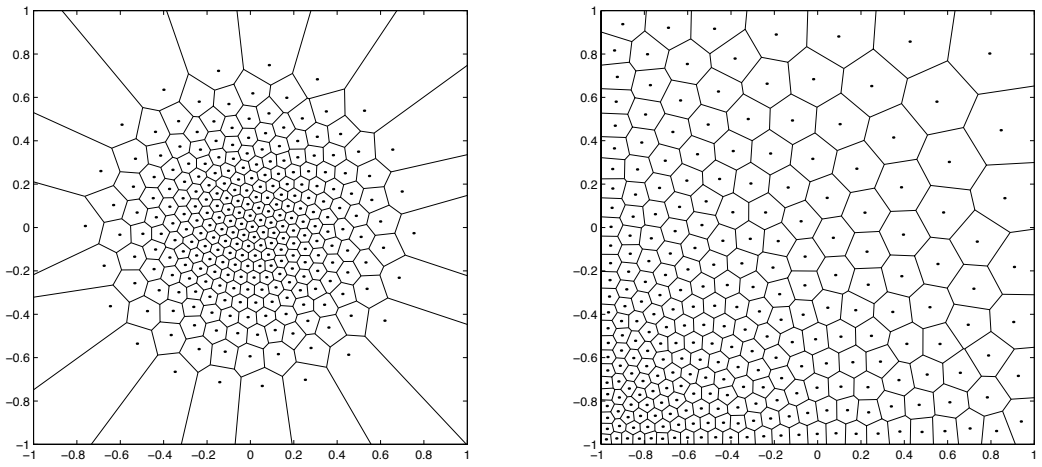

Figure 2.5: Examples of 256 point nonuniform centroidal Voronoi tessellations of a square. On the left, the point density function $\mu(\mathbf{x})$ is largest at the center of the square; on the right, the point density function has a peak at the lower left-hand corner of the square.

Centroidal Voronoi tessellations are closely related to minimizers of an "energy." Specifically, let

$$\mathcal{E}\left(\{\mathbf{z}_i\}_{i=1}^N, \{V_i\}_{i=1}^N\right) = \sum_{i=1}^N \int_{V_i} |\mathbf{x} - \mathbf{z}_i|^2 \mu(\mathbf{x})\, d\mathbf{x}\,, \tag{2.3}$$

where $\{V_i\}_{i=1}^N$ is a tessellation of Ω and $\{\mathbf{z}_i\}_{i=1}^N$ are points in $\overline{\Omega}$. No *a priori* relation is assumed between the V_i's and the $\mathbf{z}_i$'s. We refer to $\mathcal{E}$ as the *energy*; in the statistics literature, it is called the *variance* or *cost*. It is easy to prove that a necessary condition for $\mathcal{E}$ to be minimized is that $\{\mathbf{z}_i, V_i\}_{i=1}^N$ is a centroidal Voronoi tessellation of Ω; see [5]. However, not all CVT's are minimizers of the energy. For example, the CVT on the right in Figure 2.3 is a saddle point of the energy functional (2.3).

Since a CVT is not uniquely defined, one may ask what CVT does one obtain on a computer? In general, one finds CVT's that minimize the energy functional (2.3). For example, it is clear that there are many $64 = 2^6$-point CVT's of a square, including the Cartesian arrangement into an 8×8 grid depicted in the left-hand plot in Figure 2.6. However, on a computer, one obtains a CVT like the right-hand plot in that figure. In fact, for all common methods for determining CVT's, the Cartesian arrangement is unstable in the sense that if one starts with that arrangement, the methods will evolve the point distribution into one that minimizes the energy. This point is illustrated in Figure 2.6 using a specific method (which will be discussed in Section 2.1.4) for determining CVT's.

2.1.4 Constructing centroidal Voronoi tessellations

As we have seen, the points that generate a Voronoi tessellation are not generally the centroids of the associated Voronoi regions. As a result, one is left with the following

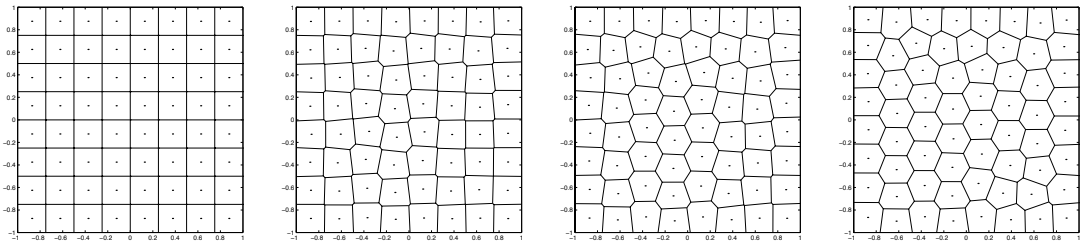

Figure 2.6: A Cartesian CVT (left) with 64 generators deforms into a hexagonal-like CVT (right) through Lloyd's iteration. The pictures are generated at iteration numbers 0, 15, 30, and 120.

construction problem: *given a region* $\Omega \subset \mathbb{R}^d$ *and a positive integer* N*, determine an* N*-point centroidal Voronoi tessellation of* Ω.[6]

There are many known methods for constructing centroidal Voronoi tessellations. In order to understand the method employed in the codes, we first describe the two methods which are perhaps most "basic" and, certainly in the first case, the most used.

First, we have *Lloyd's method* [9] which is the straightforward iteration between constructing Voronoi tessellations and centroids.

Lloyd's method. Start with some initial set of N points $\{\mathbf{z}_i\}_{i=1}^N$ in Ω, e.g., determined using random sampling;

1. construct the Voronoi tessellation $\{V_i\}_{i=1}^N$ of Ω associated with the points $\{\mathbf{z}_i\}_{i=1}^N$;
2. compute the centroids of the Voronoi regions $\{V_i\}_{i=1}^N$ found in Step 1; these centroids are the new set of points $\{\mathbf{z}_i\}_{i=1}^N$;
3. go back to Step 1, or, if happy with convergence, quit.

Lloyd's method and its convergence properties have been analyzed; see [5] and the references cited therein.

A second method is *McQueen's method* [10] which is a random sampling[7] algorithm that doesn't require the explicit construction of Voronoi tessellations or of centroids.

McQueen's method. Start with some initial set of N points $\{\mathbf{z}_i\}_{i=1}^N$ in Ω, e.g., determined using random sampling; set the integer array $J_i = 1$ for $i = 1, \ldots, N$;

1. pick a random point $\mathbf{w} \in \Omega$;
2. find the $\mathbf{z}_i$ closest to $\mathbf{w}$; denote the index of that $\mathbf{z}_i$ by i^*;
3. set $\quad \mathbf{z}_{i^*} \leftarrow \dfrac{J_{i^*}\mathbf{z}_{i^*} + \mathbf{w}}{J_{i^*} + 1} \qquad \text{and} \qquad J_{i^*} \leftarrow J_{i^*} + 1\,;$
4. $\mathbf{z}_{i^*}$ along with the unchanged points $\{\mathbf{z}_i\}_{i=1, i\neq i^*}^N$ are the new set of points;

[6] In this section and in Section 2.2 and 2.3, we will simplify the discussion by considering only uniformly distributed point sets. The discussions of those sections can be easily generalized to treat the nonuniform case.

[7] Throughout our discussion, random sampling implies that points are sampled uniformly.

5. go back to Step 1, or, if happy with convergence, quit.

Note that J_i keeps track of how many times the point $\mathbf{z}_i$ has been previously updated. Remarkably (since neither Voronoi tessellations or centroids appear anywhere in the definition of the algorithm), the McQueen iterates converge to a set of generators of a CVT. The convergence properties of McQueen's methods have been analyzed in [10].

One point is sampled at each McQueen iteration so that the McQueen iterations are cheap, but lots of them are needed. Lloyd's method requires relatively fewer iterations, but each iteration is expensive; a straightforward implementation requires the explicit construction of Voronoi regions and, to determine the centroids, numerical integrations on polyhedra. We have developed (see [7]) a class of probabilistic methods which, similar to McQueen's method, do not require the construction of Voronoi tessellations or centroids but which converge in fewer iterations than does McQueen's method. It is found to be more efficient than both the Lloyd and McQueen methods. The class of methods is characterized by two parameters; see [7] for details. Here, we only consider one particular method within the class that can be viewed as a probabilistic version of Lloyd's method; again, see [7] for details.

Algorithm 1: A probabilistic Lloyd's method. Start with some initial set of N points $\{\mathbf{z}_i\}_{i=1}^N$ in Ω, e.g., determined using random sampling; choose a positive integer N_s;

1. randomly sample $N \cdot N_s$ points in Ω;[8]
2. assign each of the sampled points to a cluster according to which of the points $\{\mathbf{z}_i\}_{i=1}^N$ is closest to it; the points in each cluster are then in the Voronoi region of one of the generators, i.e., in the Voronoi region of $\mathbf{z}_i$ for some i;
3. average the position of the points in each cluster; these cluster averages are the new set of points $\{\mathbf{z}_i\}_{i=1}^N$;
4. go back to Step 1, or, if happy with convergence, quit.

Note that N_s is the average number of points sampled per generator.

In Algorithm 1, the new position of the i-th generator is the average of the points sampled in the Voronoi region corresponding to the current i-th generator. Then, since the points averaged are randomly selected points in the Voronoi region V_i corresponding to the current $\mathbf{z}_i$, one may view the average of those points as a probabilistic approximation to the centroid of V_i. This justifies viewing this method as *a probabilistic version of Lloyd's method*. The more points sampled at each iteration, i.e., the larger is N_s, the better the centroid approximations. Parallel versions of this algorithm having nearly perfect speed-up properties can be easily defined; see [7].[9]

All methods for computing CVT's are heavily influenced by the position of the *initial* guess for the generating points $\{\mathbf{z}_i\}_{i=1}^N$ in the sense that the number of iterations needed to achieve satisfactory convergence is heavily dependent on the initial guess. Generally, the

[8] At least for a uniform point density function, if may be more efficient to sample points on a uniform, regular lattice instead of by random sampling.

[9] McQueen's method does not parallelize well due to the relatively large amount of communications needed; this results from the fact that only one point is sampled and averaged at each iteration.

initial positions of the generators are determined by the random sampling of N-points in Ω. For a uniform distribution of points, quasi-random sampling methods such as those based on Halton sequences have been found to be useful in generating "better"[10] initial point distributions. In the next section, we discuss both random sampling and Halton sequence-based sampling in more detail. We emphasize that Halton sequences can be used only for generating uniformly distributed point sets.

Algorithm 1 requires two auxiliary algorithms. Step 1 requires the sampling of random points in the given region Ω; algorithms for such sampling are discussed in the next section. Step 2 requires the determination of which of the current $\mathbf{z}_i$'s is closest to a sampled point; algorithms for this purpose are discussed in Section 2.3.

2.2 Random and quasi-random sampling

All CVT construction algorithms require an initial set of candidate generating points. These can be determined by random sampling or, in some cases, by using Halton sequences. Algorithm 1 also requires, at each iteration, the random sampling of points. We now discuss both random and Halton sequence-based sampling in the context of general regions in $\mathbb{R}^3$. The discussion can be be applied to regions in $\mathbb{R}^2$ by making obvious changes.

2.2.1 Random sampling in general regions

All computers have a built-in (pseudo-)random number generator that can provide a sequence of (pseudo-)random numbers uniformly distributed on the interval $[0, 1]$. How does one use this capability to generate uniformly distributed points in a general region in $\mathbb{R}^3$? In other words, given a region $\Omega \subset \mathbb{R}^3$, how can one determine a random point in Ω?

If the region Ω is a box in $\mathbb{R}^3$, then it is obvious how one determines a random point in Ω; one merely obtains (from the built-in random number generator) three uniformly distributed random numbers, which suitably translated and stretched, become the coordinates of a random point in the given box. Thus, here the issue is how does one account for the fact that the region is not a box? It is, in fact, easy to treat general regions in $\mathbb{R}^3$. To find a random point inside a given region Ω, we first enclose the given region Ω in a box $[a_1, b_1] \times [a_2, b_2] \times [a_3, b_3]$. We then sample a random point inside the enclosing box. If the point is inside the given region Ω, we accept it; if it is outside Ω we reject it. We keep on sampling in the enclosing box until we find an acceptable point, i.e., a point that is also inside Ω. Note that the only random sampling necessary to implement the algorithm is to sample a random point in the enclosing box. The algorithm just described is summarized as follows.

Algorithm 2: Random sampling in a general region. Given a region $\Omega \subset \mathbb{R}^3$ and a box containing $\overline{\Omega}$, i.e., three ordered pairs of real numbers $\{a_k, b_k\}_{k=1}^3$ such that $\Omega \subset [a_1, b_1] \times [a_2, b_2] \times [a_3, b_3]$;

1. sample three random real numbers x_1, x_2, and x_3 in the interval $[0, 1]$ according to a uniform distribution;

[10] Better in the sense that fewer iterations of the CVT construction algorithm are required.

2. set $x_k \leftarrow (b_k - a_k)x_k + a_k$ for $k = 1, 2, 3$;
3. if $\mathbf{x} = (x_1, x_2, x_3)$ is outside $\overline{\Omega}$, go to step 1; otherwise, exit.

The random point $\mathbf{x}$ determined by the algorithm is in $\overline{\Omega}$ and is uniformly distributed.

The output of the random sampling algorithm (Algorithm 2) can be used in two ways for the determination of a CVT. In the first place, N points sampled according to the algorithm can be used as initial data for any of the CVT construction methods. Second, each iteration of both McQueen's method and Algorithm 1 requires sampled points which are then averaged; the necessary sampled points may be determined by Algorithm 2.

2.2.2 Halton sequence point selection

For *uniform* point distributions, one can also use *Halton sequences* to generate initial point sets for CVT construction algorithms. Halton sequences are based on the following observations. Given a prime number q, any $n \in \mathbb{N}$ can be represented as $n = \sum_i n_i q^i$. Then, one can define a mapping H_q from $\mathbb{N}$ to $[0, 1]$ by $H_q(n) = \sum_i n_i/q^{j+1}$. The Halton sequence of N-points in $[0, 1]$ is then defined as $\{H_q(n)\}_{n=1}^N$. Halton sequences are uniformly distributed pseudo-Monte Carlo sequences.

Given three prime numbers q, r, and s, one may then define the (q, r, s)-Halton sequence of N-points in the unit cube in three dimensions by $\{(H_q(n), H_r(n), H_s(n))\}_{n=1}^N$. By the obvious translations and stretchings, one may then define a set of N Halton points in an arbitrary box in three dimensions. By rejecting sampled points in an enclosing box that lie outside of a given region, one can then generate a Halton point set in an arbitrary region in $\mathbb{R}^3$.

2.2.3 CVT's as an improvement to both random sampling and Halton points

We now know how to generate three types of point sets: randomly selected, Halton-sequence based, and CVT-based. We have discussed the use of the first two of these as initial conditions in construction algorithms for the last. If one wants to obtain a uniformly distributed point set, why not just choose one of the first two and avoid the iteration necessary to determine a CVT?

All three methods (random sampling, Halton, and CVT) will produce, as the number of points $N \to \infty$, uniformly distributed point sets. But, what happens for finite values of N? The typical situation is depicted in Figure 2.7. Note that for 128 points, the CVT point distribution is much more "uniform" than are either of the random sampling or Halton point distributions. This is why it is worthwhile going ahead and determining the CVT point distribution instead of just settling for the random sampling or Halton points. We will discuss this issue again in Section 3.1.2.

From Figure 2.7, we see that the Halton point distribution is more uniform than the randomly sampled points. This is typical and this is why it is usually advantageous to use Halton point sets as initial conditions for CVT construction.

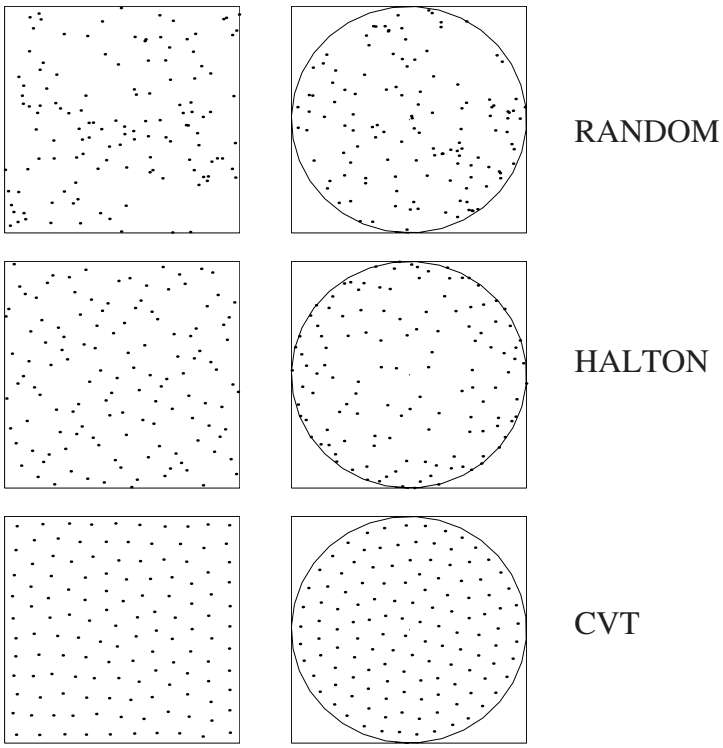

Figure 2.7: Sets of 128 points uniformly distributed in a square (left) and circle (right) determined by the random sampling (top), (2,3) Halton sequence (middle), and centroidal Voronoi tessellation (bottom) point selection methods.

2.3 Closest point determination

From timings of early versions of the code, it became clear that one of the most expensive operations is determining which member of the current set of N generators is nearest to a randomly sampled point; this is a necessary step of Algorithm 1. In a naive implementation[11] of this search, each sampling point incurs a cost of computing the distance to all N generators. Let us see why this is so.

Suppose we have N generators and, at each iteration of Algorithm 1, we sample an average of N_s points per generator so that the total number of points sampled at each iteration is NN_s. Then, the total cost of computing nearest generators over N_{iter} iterations is $N_{iter}(NN_s)N$ since we create NN_s sampled points at each iteration and each sampling point requires N distance computations. Thus, if N_s, the average number of sampling points per generator, is fixed, the amount of work to determine nearest generators increases quadratically with N, the number of generators. No other operation in the code has an N^2 behavior. This means that for even moderate values of N, a naive approach to the nearest generator computation represents the dominant cost.

[11] By this we mean the exhaustive search approach in which the distance from a sampling point to *all* the generators is computed and then compared to find the closest generator.

2.3.1 Using bins for efficient closest point determination

One way to control the cost of closest generator determination is to pay the price of preprocessing the data in some way so that the nearest point can be more quickly determined. The obvious approach is to use bins, that is, a mesh of simple rectilinear subregions, to organize the geometry of the generators [4].

For simplicity of presentation, consider a simple, two-dimensional example. Suppose the physical region of interest is some subset of a square. In this situation, we might subdivide the square by a 10×10 Cartesian grid to produce an array of 100 square bins. After the initial positions of the generators are chosen, we process the generators, constructing a database that allows us to list all the generators in each bin. Then, it is possible to define an algorithm that guarantees that the nearest generator will be located but that "not too many" bins will be unnecessarily searched [2]. If the number of points in each bin is small, say, $\sqrt{N}$, then significant savings can result. The process is fairly easy to describe, although the coding can be elaborate.[12]

Algorithm 3: Determination of the closest point in a set to a given point. Given a set of points $\{\mathbf{z}_i\}_{i=1}^N$ which have been sorted into bins and given a random point $\mathbf{y}$;

1. determine the bin that contains $\mathbf{y}$; if that bin does not contain any of the points in the set $\{\mathbf{z}_i\}_{i=1}^N$ go to Step 2; otherwise, determine the point $\mathbf{z}_{i^*}$ in the bin that is closest to $\mathbf{y}$ and go to Step 3;
2. if the bin containing $\mathbf{y}$ contained no points in the set $\{\mathbf{z}_i\}_{i=1}^N$, search the layer of bins surrounding the bin containing $\mathbf{y}$, or if necessary, "spiraling out" to additional layers of bins until a nonempty bin is found; determine $\mathbf{z}_{i^*}$, the nearest neighbor to $\mathbf{y}$ in that bin;
3. let R denote the distance from $\mathbf{y}$ to $\mathbf{z}_{i^*}$; we continue the spiraling process, expanding the set of searched bins; in each bin, compare R to the distances from $\mathbf{y}$ to the points in the set $\{\mathbf{z}_i\}_{i=1}^N$ contained in the bin, redefining $\mathbf{z}_{i^*}$ and R by a new point and new distance whenever the new point is closer to $\mathbf{y}$ than the candidate point; we stop the search as soon as enough bins have been searched so that they contain the sphere of radius R.

We make some observations about Algorithm 3. First, in Step 1, the bin containing the sampled point $\mathbf{y}$ serves as the center of the search for the closest $\mathbf{z}_i$. After Step 1, if any of the points in the set $\{\mathbf{z}_i\}_{i=1}^N$ are located in the center bin, i.e., in the bin containing $\mathbf{y}$, we have a candidate $\mathbf{z}_{i^*}$ for the nearest point and we can skip the second step. If the center bin contained none of the points in the set $\{\mathbf{z}_i\}_{i=1}^N$ so that we have no candidate closest point, we "spiral out" from the center bin, adding one layer of bins at a time to our list of searched bins. Usually, searching one more layer of bins is sufficient, but in any case, if we add enough layers, we are guaranteed to find a bin containing at least one point $\mathbf{z}_{i^*}$ in the set $\{\mathbf{z}_i\}_{i=1}^N$, and hence, a candidate for the nearest neighbor. Once we know the distance R from some point $\mathbf{z}_{i^*}$ to $\mathbf{y}$, we know the nearest neighbor can be no further away. In Step 3, we are essentially constructing a sphere of radius R centered at the point $\mathbf{y}$ and continue

[12] There have been many other algorithms proposed for efficient closest point determination; in future work, these will be explored.

the spiraling process, expanding the set of searched bins in such a way that they eventually contain the sphere. After Step 3, the distance from $\mathbf{y}$ to every point in the sphere has been considered, and we are guaranteed to have determined the nearest point in the set $\{\mathbf{z}_i\}_{i=1}^N$ to $\mathbf{y}$.

This procedure incurs the costs of additional storage for the bin data, extra preprocessing whenever the generators are modified to resort them into the bins, and the cost of carrying out the bin search procedure. In general, however, for each sampled point, the number of generators that have to be considered is vastly decreased (compared to the exhaustive search approach), and for large enough problems, the decrease in the points that need to be considered can outweigh the overhead of the bin search procedure. If the number of points is of order $\sqrt{N}$, then, using the bind approach, the complexity reduces to $N\sqrt{N}$. This may be further reduced at the cost of more bins and therefore more preprocessing. In our investigations, we have seen that the use of bins on a problem of "interesting" size can speed up the closest point computation by a factor of ten or more.

As mentioned before, as the number of generators increases, the cost of a naive search dominates the whole computation, and so the bin search improvement can be even greater, as long as the number of bins is appropriately increased. The optimal number of bins to use can depend greatly on the geometry of the problem. For example, for a long, thin region, the number of bins in each direction should reflect the aspect ratios of the region since it is best if the bins themselves remain nearly square or cubical.

This simple bin scheme works best when the points are uniformly distributed throughout the rectilinear bounding box. In other cases, the scheme may not work as well. In the DIATOM test case, for example, the region of interest occupies a relatively small portion of the bounding box, so many bins are completely empty. However, a uniform density is used within the region, so this simply means that rather more bins must be used to achieve good results. A more difficult case to handle occurs when the generating points are nonuniformly clustered. A simple, evenly spaced grid of bins will have to be very fine in order to reduce the maximum number of points per bin significantly, and thus to achieve the benefit of reduced computation.

Note that the computation of zeroth, first, and second moments (see Section 2.4) can be optimized by the same process as we just described for optimizing the CVT iteration.

2.4 Determination of zeroth, first, and second moments

So far we have discussed how one determines the generators of an N-point centroidal Voronoi tessellation of Ω. However, recall that the goal of our effort is to determine a set of zeroth moments (volumes), first moments (centroids), and second moments associated with a set of volumes that constitute a "good" tessellation of the region Ω. Having obtained the generators $\{\mathbf{z}_i\}_{i=1}^N$ of an N-point centroidal Voronoi tessellation, it is natural to choose the associated Voronoi polyhedra $\{V_i\}_{i=1}^N$ to be the "good" tessellation of Ω. Thus, we now want to consider how to efficiently compute zeroth, first, and second moments associated with each of the volumes V_i. We assume that we are given a physical *mass density function*[13]

[13]The mass density function $\rho(\mathbf{x})$ should not be confused with the point density function $\mu(\mathbf{x})$. The former is a physical attribute of the media while the latter is a computational artifact that can used in some settings

$\rho(\mathbf{x})$ defined for all $\mathbf{x} \in \overline{\Omega}$.[14]

For $i = 1, \ldots, N$, the *zeroth moment* or *mass* of V_i (a scalar) is defined by

$$M_i = \int_{V_i} \rho(\mathbf{x})\, d\mathbf{x}\,, \tag{2.4}$$

the *first moments* or the *center of mass*[15] V_i (a vector) by

$$\overline{\mathbf{x}}_i = \frac{1}{M_i} \int_{V_i} \mathbf{x}\rho(\mathbf{x})\, d\mathbf{x}\,, \tag{2.5}$$

and the *second moments* (relative to the center of mass) of V_i (a tensor) by

$$\mathbb{M}_i = \frac{1}{M_i} \int_{V_i} (\mathbf{x} - \overline{\mathbf{x}}_i)(\mathbf{x} - \overline{\mathbf{x}}_i)^T \rho(\mathbf{x})\, d\mathbf{x}\,. \tag{2.6}$$

Since we have that

$$\mathbb{M}_i = \mathbb{M}_{0i} - \overline{\mathbf{x}}_i \overline{\mathbf{x}}_i^T\,,$$

where $\mathbb{M}_{0i}$ denotes the second-order moment tensor relative to the origin, i.e.,

$$\mathbb{M}_{0i} = \frac{1}{M_i} \int_{V_i} \mathbf{x}\mathbf{x}^T \rho(\mathbf{x})\, d\mathbf{x}\,,$$

it is clear that if, for each V_i, one determines the three integrals

$$M_i = \int_{V_i} \rho(\mathbf{x})\, d\mathbf{x}\,, \qquad \mathbf{q}_i = \int_{V_i} \mathbf{x}\rho(\mathbf{x})\, d\mathbf{x}\,, \qquad \text{and} \qquad \mathbb{S}_i = \int_{V_i} \mathbf{x}\mathbf{x}^T \rho(\mathbf{x})\, d\mathbf{x}\,, \tag{2.7}$$

then one can easily compute the desired quantities (2.4)–(2.6). In fact, we simply have that

$$\overline{\mathbf{x}}_i = \frac{1}{M_i}\mathbf{q}_i \qquad \text{and} \qquad \mathbb{M}_i = \frac{1}{M_i}\mathbb{S}_i - \overline{\mathbf{x}}_i \overline{\mathbf{x}}_i^T\,.$$

Thus, given the set of points $\{\mathbf{z}_i\}_{i=1}^N$ (the generators of an N-point CVT of Ω), what we now want is an efficient method for determining the set $\{M_i, \mathbf{q}_i, \mathbb{S}_i\}_{i=1}^N$ defined in (2.7).

In principle, knowing the position of the generators $\{\mathbf{z}_i\}_{i=1}^N$ of the CVT, one could generate the corresponding Voronoi tessellation and then use numerical integration rules to evaluate the integrals in (2.7). Since numerical integration formulas for general polyhedra do not exist, one would actually need to subdivide the Voronoi regions into tetahedra and

as a means of obtaining better computational results.

[14] Our codes only treat the case of a uniform physical density; the generalization to the nonuniform case is fairly straightforward. Thus, although we define the moments associated with a give set of regions for the general nonuniform case, our discussion of algorithms and quality measures in this section will be confined to the case of uniform point placement and mass densities.

[15] If the point placement density function $\mu(\mathbf{x})$ that determines the centroidal Voronoi tessellation and the physical density function $\rho(\mathbf{x})$ are multiples of one another, e.g., they are constants, $\mathbf{z}_i = \overline{\mathbf{x}}_i$, the generators of the Voronoi regions coincide with the centers of mass of those regions. In our codes, both these densities are constants, so that, knowing the generators, we already have $\overline{\mathbf{x}}_i$ in hand. However, we choose to recompute it because in more general cases $\mathbf{z}_i \neq \overline{\mathbf{x}}_i$. Furthermore, as we shall see later, the recomputation provides a good check on the overall accuracy of our methodology.

then use numerical integration rules on each of these subregions. *This would be difficult to program and expensive in terms of CPU time.*

Instead, the integrals in (2.7) can be evaluated probabilistically, avoiding not only the use of complicated numerical integration rules on polyhedra, but also avoiding the need to explicitly determine the Voronoi polyhedra. Our discussion so far is summarized in the following algorithm for probabilistically determining, for a uniform mass density, (2.4)–(2.6) from a given set of points[16] $\{\mathbf{z}_i\}_{i=1}^N$ in Ω.

Algorithm 4: Probabilistic determination of zeroth, first, and second moments. Given a region $\Omega \in \mathbb{R}^3$, its volume $|\Omega|$, a set of points $\{\mathbf{z}_i\}_{i=1}^N$ in Ω, and a positive integer N_m;

1. randomly sample $K = N_m N$ points $\{\mathbf{y}_j\}_{j=1}^K$ in Ω;
2. for $i = 1, \ldots, N$, compute the sums

$$K_i = \sum_{\mathbf{y}_j \in V_i} 1\,, \qquad \mathbf{q}_i = \sum_{\mathbf{y}_j \in V_i} \mathbf{y}_j\,, \qquad \text{and} \qquad \mathbb{S}_i = \sum_{\mathbf{y}_j \in V_i} \mathbf{y}_j \mathbf{y}_j^T\,;$$

3. for $i = 1, \ldots, N$, set

$$M_i = \frac{|\Omega|}{K} K_i\,, \qquad \overline{\mathbf{x}}_i = \frac{1}{K_i}\mathbf{q}_i\,, \qquad \text{and} \qquad \mathbb{M}_i = \frac{1}{K_i}\mathbb{S}_i - \overline{\mathbf{x}}_i \overline{\mathbf{x}}_i^T\,.$$

The input N_m is the average number of points sampled in each Voronoi region for the moment calculation. The sampling in Step 1 can be performed using the same sampling algorithm (Algorithm 2) as that used for determining the generators $\{\mathbf{z}_i\}_{i=1}^N$. In Step 2, for each $i = 1, \ldots, N$, the sums are over all sampled points belonging to the Voronoi region V_i corresponding to the generator $\mathbf{z}_i$. Note that this requires the sorting of the $K = N_m N$ sampled points into the Voronoi regions of the points $\{\mathbf{z}_i\}_{i=1}^N$; this may be accomplished with the help of Algorithm 3. Step 3 requires some explanation. First, K_i is merely the number of sampled points in V_i, i.e., in the Voronoi region corresponding to $\mathbf{z}_i$; $K = N_m N$ is the total number of points sampled over the whole region Ω. Then, the ratio K_i/K is an estimate of the ratio of the volume of V_i to the volume of Ω so that then $|\Omega| K_i/K$ provides an estimate for the volume of V_i. Next, we have the probabilistic approximation to the centroid

$$\overline{\mathbf{x}}_i = \frac{1}{M_i}\int_{V_i} \mathbf{x}\, d\mathbf{x} \approx \frac{1}{M_i}\left(\frac{M_i}{K_i}\sum_{\mathbf{y}_j \in V_i} \mathbf{y}_j\right) = \frac{1}{K_i}\mathbf{q}_i$$

which accounts for the second term in Step 3. The third term in Step 3 can be accounted for in a similar manner.

Note that Algorithm 4 requires the volume $|\Omega|$ of the given region Ω. If this is not known, it can be approximately determined as follows. In Step 1 of Algorithm 4, we must sample K points inside Ω. If we use Algorithm 2 to determine these points, then in determining these

[16] For us, the set of points $\{\mathbf{z}_i\}_{i=1}^N$ will be the generators of a CVT, i.e., the output of a CVT construction algorithm. However, the algorithm given for probabilistically determining (2.4)–(2.6) can be applied to any set of points in Ω.

K points, the points sampled in the enclosing box which were outside of Ω were rejected. If N_b denotes the total number of points sampled in the enclosing box so that K of those points were inside of Ω, then $K|B|/N_b$ is an estimate for the volume of Ω, where $|B|$ is the easily determined volume of the enclosing box.

2.5 Tests for the quality of the solution

Our software determines a set of *uniformly distributed points* in a user specified region; the user also specifies the number of points.[17] Visual examination, of course, can often be used to compare the relative uniformity (or lack thereof) of different sets of points; see, e.g., Figure 2.7. There are other, more quantitative measures of the uniformity of a set of points.

Given any set of points in a region Ω, we can use those points to generate a Voronoi tessellation of Ω.[18] Then, we can associate with each point a corresponding Voronoi region and then determine various quantities associated with the points and the regions, including the zeroth, first, and second-order moments for the regions.[19] These can be used to determine the quality of the set of points.

We will use three particular properties associated with the set of points and the corresponding Voronoi regions as measures of the quality, i.e., the uniformity, of a point set. These are:

- the zeroth-order moments (the volumes) of the Voronoi regions associated with each point;
- the trace[20] of the second-moment matrix associated with each point and its corresponding Voronoi region; and
- the determinant[21] of the deviatoric matrix associated with each point and its corresponding Voronoi region.

For a *perfectly uniform distribution* of N points $\{\mathbf{z}_i\}_{i=1}^N$ in a given region Ω, the corresponding volumes M_i would all be equal, i.e., $M_1 = M_2 = \cdots = M_N$, the corresponding

[17] From these points, the zeroth, first, and second moments are then determined.

[18] Of course, our methodology produces a centroidal Voronoi tessellation of Ω which, as we have seen, is a special Voronoi tessellation.

[19] Indeed, this is desired output of our codes, so that we must determine them in any case.

[20] For each point, the trace of the corresponding second-moment matrix $\mathbb{M}$ is given by

$$T = M_{11} + M_{22} \qquad \text{in two dimensions}$$

$$T = M_{11} + M_{22} + M_{33} \qquad \text{in three dimensions,}$$

where M_{jk} denotes the j,k second moment associated with a point.

[21] For each point, the determinant of the corresponding deviatoric or second-moment matrix $\mathbb{M} - \overline{M}\mathbb{I}$, where $\overline{M} = T/2$ in two dimensions and $\overline{M} = T/3$ in three dimensions, is given by

$$D = \det \begin{pmatrix} M_{11} - \overline{M} & M_{12} \\ M_{12} & M_{22} - \overline{M} \end{pmatrix} \qquad \text{in two dimensions}$$

$$D = \det \begin{pmatrix} M_{11} - \overline{M} & M_{12} & M_{13} \\ M_{12} & M_{22} - \overline{M} & M_{23} \\ M_{13} & M_{23} & M_{33} - \overline{M} \end{pmatrix} \qquad \text{in three dimensions.}$$

traces $\{T_i\}_{i=1}^N$ would also all be equal, i.e., $T_1 = T_2 = \cdots = T_N$, and the corresponding determinants $\{D_i\}_{i=1}^N$ would all vanish, i.e., $D_1 = D_2 = \cdots = D_N = 0$. Thus, we will use the uniformity of these volumes and traces and the smallness of these determinants as measures of the uniformity of a set of points.

In two dimensions, a perfectly uniform set of points would be the centroids of a tessellation of the given region into congruent regular hexagons. Of course, in general, this is impossible since general regions cannot be exactly tessellated by congruent regular hexagons. Similar arguments can be given in three dimensions. Thus, we cannot expect that our criteria for perfect uniformity will ever be satisfied. However, we can certainly use these criteria as quality measures for a set of points.

We can also use the criteria to monitor the convergence of our iterative algorithm (Algorithm 1) for determining a set of generators for a CVT of a given region. As the points converge to the generators of a CVT, the volumes and traces of the points should become more uniform and the determinants should become smaller. In Chapter 3, we will use the criteria in this manner.

The three criteria that we use to test the quality of a set of points or the convergence of an iterative method for determining those points can be generalized to nonuniform point placements and nonuniform physical densities.

For case of uniform point placement and mass densities $\mu(\mathbf{x})$ and $\rho(\mathbf{x})$, respectively, there is another obvious criterion that can be used. In this case, as has already been noted, the generators of the CVT $\{\mathbf{z}_i\}_{i=1}^N$ should coincide with the first moments $\{\overline{\mathbf{x}}_i\}_{i=1}^N$ of the corresponding Voronoi regions; this follows from the defining property of a CVT.[22] Since our codes apply to the constant densities case, one may use this additional criteria to assess the quality of a CVT point set or to monitor the convergence properties of an iterative algorithm for its determination.

[22]It is interesting to observe that in case the point placement and mass densities are uniform, the "energy" (2.3) can be expressed in the form

$$\mathcal{E}\left(\{\mathbf{z}_i\}_{i=1}^N, \{V_i\}_{i=1}^N\right) = \sum_{i=1}^N \text{trace}\left(\int_{V_i} (\mathbf{x}-\mathbf{z}_i)(\mathbf{x}-\mathbf{z}_i)^T \, d\mathbf{x}\right) = \sum_{i=1}^N M_i T_i \,,$$

where M_i and $T_i = \text{trace}(\mathbb{M}_i)$ respectively denote the mass and the trace of the second moment tensor (relative to the center of mass) of the region V_i. Thus, (stable) centroidal Voronoi tessellations can be characterized as minimizers of the sum over the tessellation of the products the mass and second moment traces.

Chapter 3

Numerical experiences

In this chapter, we report on some computational results which illustrate the use of the centroidal Voronoi-based software and which also form the basis for discussions of the performance of that software. We use the quality criteria discussed in Section 2.5 in our discussion.

All computations reported in this chapter were performed on Dec or Compaq Alpha workstations.

3.1 An example in two dimensions

The domain used for the two-dimensional example is the region specified by the DIATOM test case which can also be analytically defined by

$$\begin{aligned} 45 \le x \le 55 \qquad &\text{for} \qquad 40 \le y \le 100 \\ 30 \le (x-50)^2 + y^2 \le 40 \qquad &\text{for} \qquad 0 \le y < 40\,. \end{aligned}$$

3.1.1 Initial point sets

Since the method used is iterative in nature, a set of initial points must be generated. As indicated in Section 2.2, the initial points can be chosen either by a random sampling (Monte Carlo) approach using a random number generator or by using Halton sequences. In Figure 3.1, we plot a set of 256 initial points using a $(2, 3)$ Halton sequence along with the absolute value of the determinant of the deviatoric matrix at each of the points. In Figure 3.2, we plot a set of 256 initial points which are generated by random sampling along with the determinant of the deviatoric matrix at each point. As can be seen by comparing the figures, the Halton points represent a better starting set for this example. It has been our experience that in most situations using Halton points requires fewer iterations to reach the centroidal Voronoi generators and so is the preferred method to use for generating initial point distributions. However, the software allows the user to choose either method for generating an initial set of points.

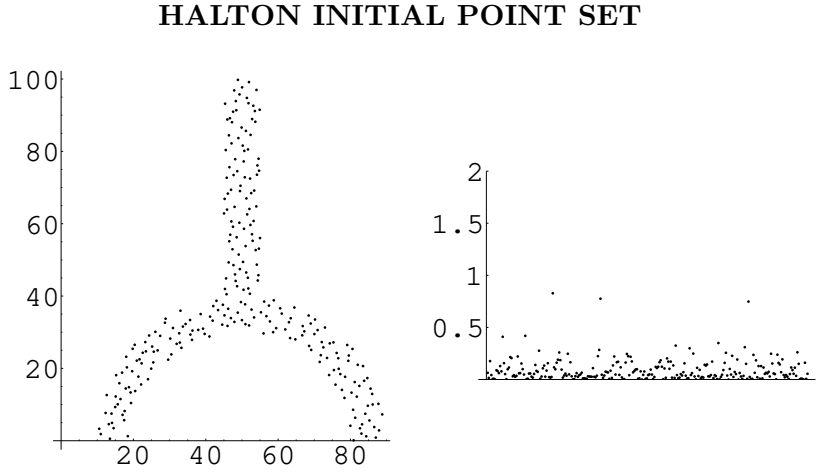

Figure 3.1: Initial set of 256 points using Halton sequence along with the magnitude of the determinant of the deviatoric matrix at each point.

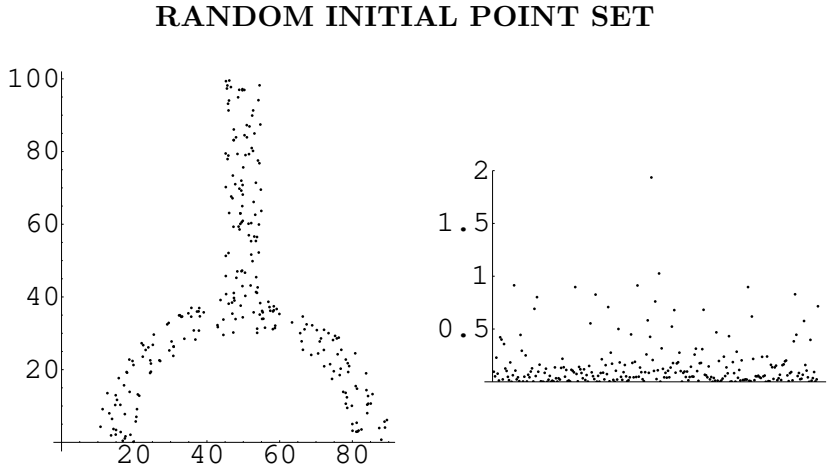

Figure 3.2: *Initial* set of 256 points using random sampling along with the magnitude of the determinant of the deviatoric matrix at each point.

3.1.2 Quality of the point sets

There are several parameters which are input to the code which affect the accuracy and performance of the methods implemented. These include:

- the number of Voronoi generators N, i.e., the number of points;
- the number of sampling points per generator N_s; and
- the maximum number of iterations N_{iter}.

The number of generators or points is determined by the needs of the user.

In the code, the total number of sampling points is set to the number of generators times the user specified number of sampling points per generator. The number of iterations

depends on the properties of the iterative method with respect to the number of generators, sampling points, etc., and the initial distribution of points. Hence it is not always easy to determine the number of iterations *a priori.* We will investigate an appropriate choice for the number of iterations in the following discussion.

We demonstrate the convergence of the iterative method by plotting the position of the points and the magnitude of the determinants of the deviatoric matrices associated with each point at several stages of the iterative process for determining the centroidal Voronoi set of points. In Figures 3.3, 3.4, and 3.5, we respectively plot the distribution of 128, 256, and 512 points after 10, 20, 40, 60, 80, and 100 iterations of our iterative point placement method (Algorithm 1).[1] In Figures 3.6, 3.7, and 3.8, we plot the corresponding determinants of the deviatoric matrix at each point. The data in Figures 3.6, 3.7, and 3.8 are summarized in Figures 3.9 and 3.10 in which respectively the average and standard deviation of the magnitude of the determinant of the deviatoric matrix vs. the number of iterations for 128, 256, and 512 generating points is plotted. Note that it is always beneficial to try to determine a centroidal Voronoi point placement, i.e., that there is a substantial improvement over the initial Halton point set. However, we also see that there no need to perform a large number of iterations; the improvements gained by doing additional iterations are small, i.e., performing further iterations does not change the distribution of points and hence the determinant of the deviatoric matrix is not reduced. As can be seen from the figures, a higher number of points requires fewer iterations. This is partly due to the fact that the initial condition is better for a higher number of points, i.e., in the limit as the number of points approaches infinity, the Halton sequence points (or randomly sampled points for that matter) approach a uniform distribution of points.[2] To illustrate this point, we present the initial conditions using a Halton sequence for 128, 256, and 512 points in Figure 3.11. Note that in the case where 128 points are specified, the determinant of the deviatoric matrix is much larger than the other two cases. This is due to the fact that 128 points are insufficient for a Halton sequence to produce a truly quasi-uniform point distribution.

Even though the quality of the Halton points improves as the number of points increases, they are still not as good as the CVT points. This can be seen from Figure 3.12 where, using the same scale, the magnitude of the determinant of the deviatoric matrix for both the initial Halton and converged CVT sets of points are shown for the case of 512 points.

In Figures 3.13–3.15, the other quality criteria discussed in Section 2.5 are used to examine the quality of the CVT points produced after 100 iterations; sets of 128, 256, and 512 points are compared. Specifically, in Figure 3.13, the volumes (zeroth moments) of the associated Voronoi regions are given for each point; recall that for a perfectly uniform point distribution, these should be the same. In Figure 3.14, the traces of the associated second moment matrices are given for each point; recall that for a perfectly uniform point distribution, these should also be the same. In Figure 3.15, the distance between the CVT generators and the calculated first-moment vectors of the corresponding regions are given for each point; recall that for a perfectly uniform point distribution of CVT points (and only for CVT points), these should vanish. From Figures 3.13–3.15, we again see that for

[1] The initial set of points were determined using a Halton sequence.

[2] It is also due to the fact that as the number of points increases, the same number of sampling points per generator yields more accurate approximations to the centroids of the Voronoi regions.

the same number of sampling points per generator and the same number of iterations, the greater the number of generators the better is the quality of the point sets.

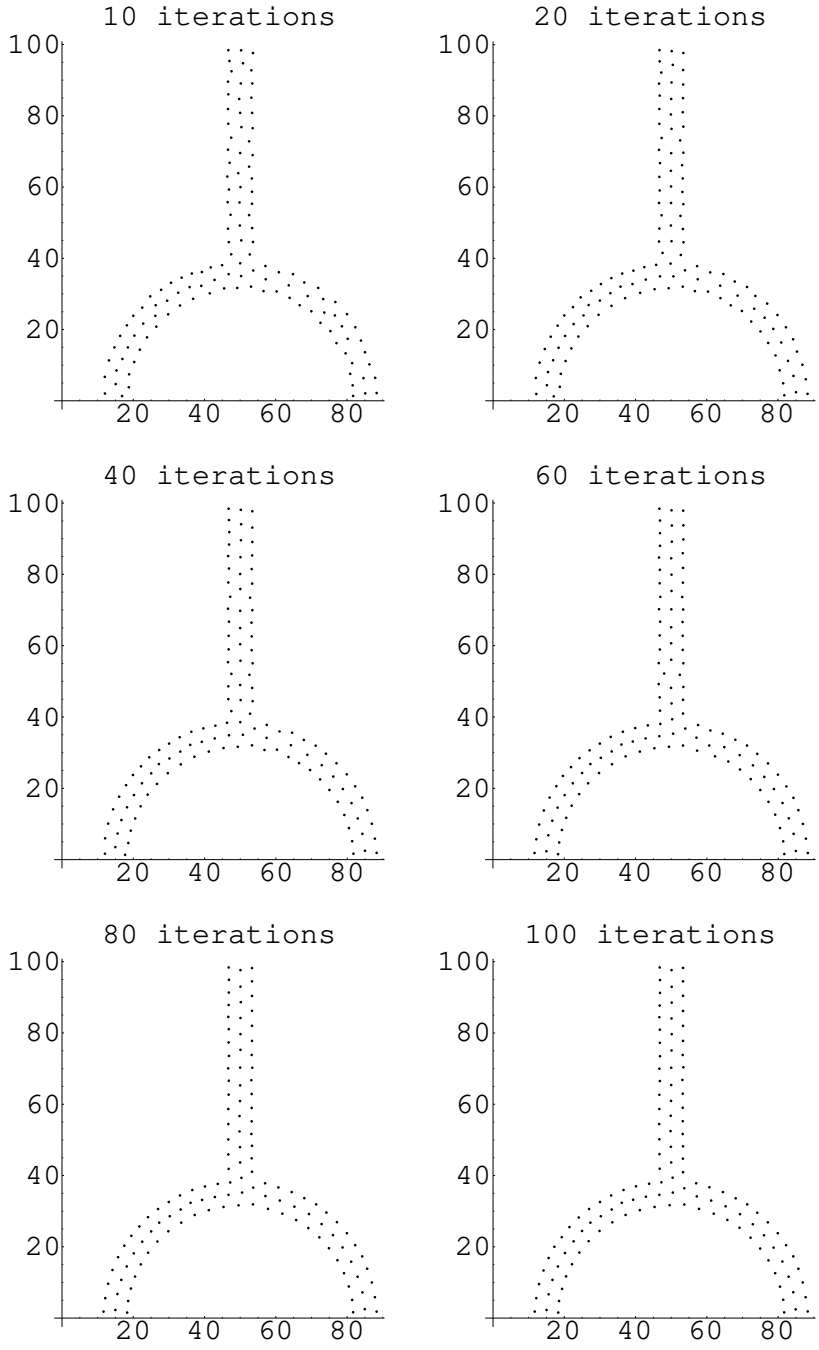

Figure 3.3: Position of 128 points after the indicated number of iterations of Algorithm 1 for determining a centroidal Voronoi tessellation.

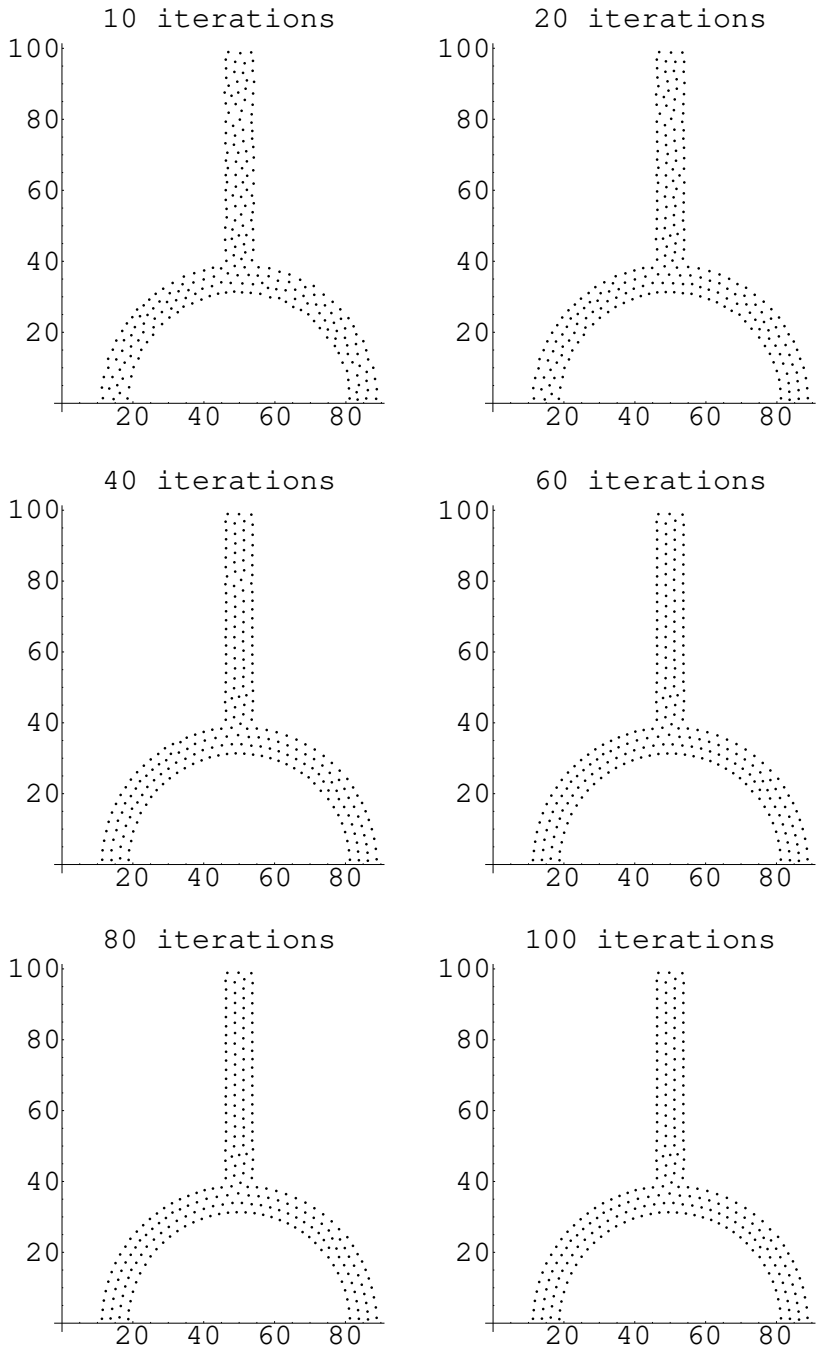

Figure 3.4: Position of 256 points after the indicated number of iterations of Algorithm 1 for determining a centroidal Voronoi tessellation.

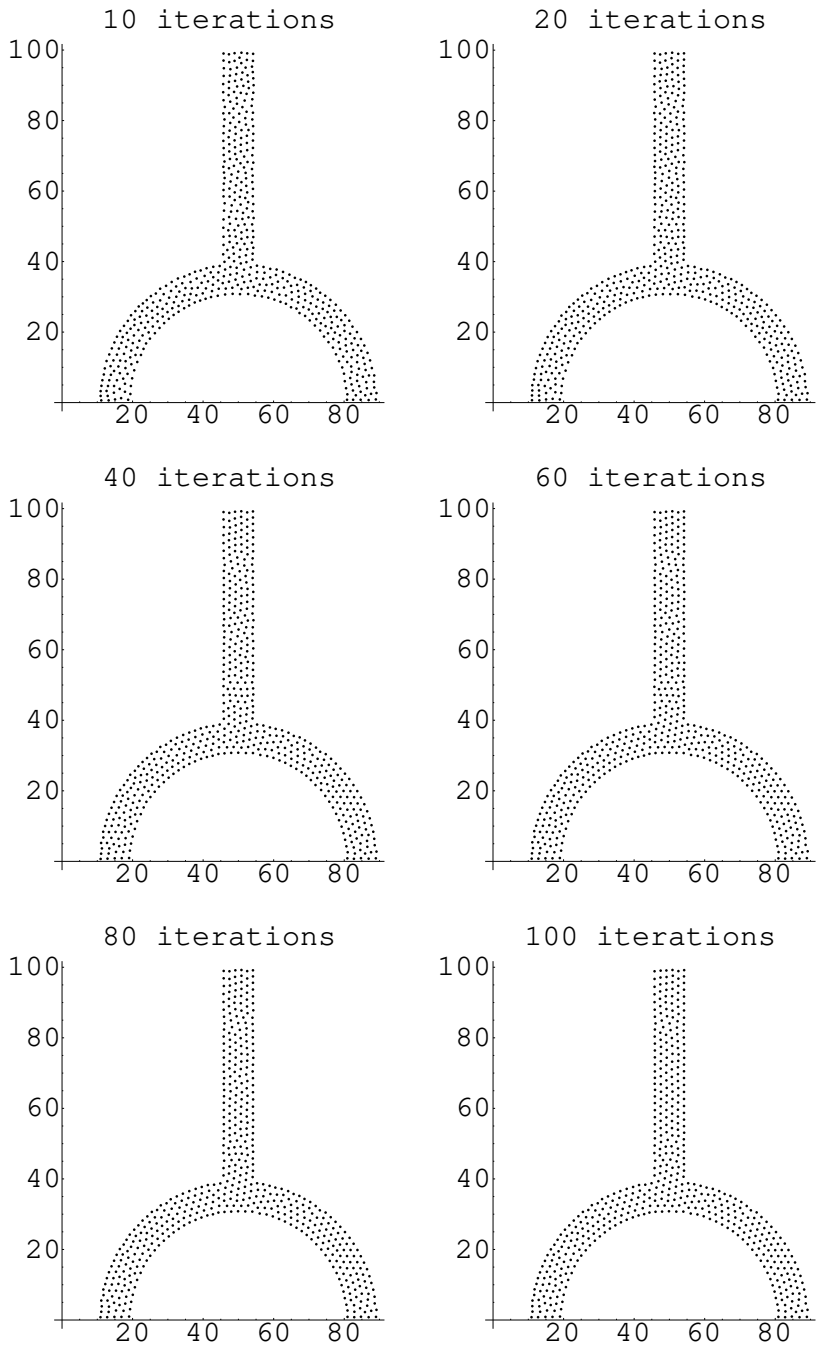

Figure 3.5: Position of 512 points after the indicated number of iterations of Algorithm 1 for determining a centroidal Voronoi tessellation.

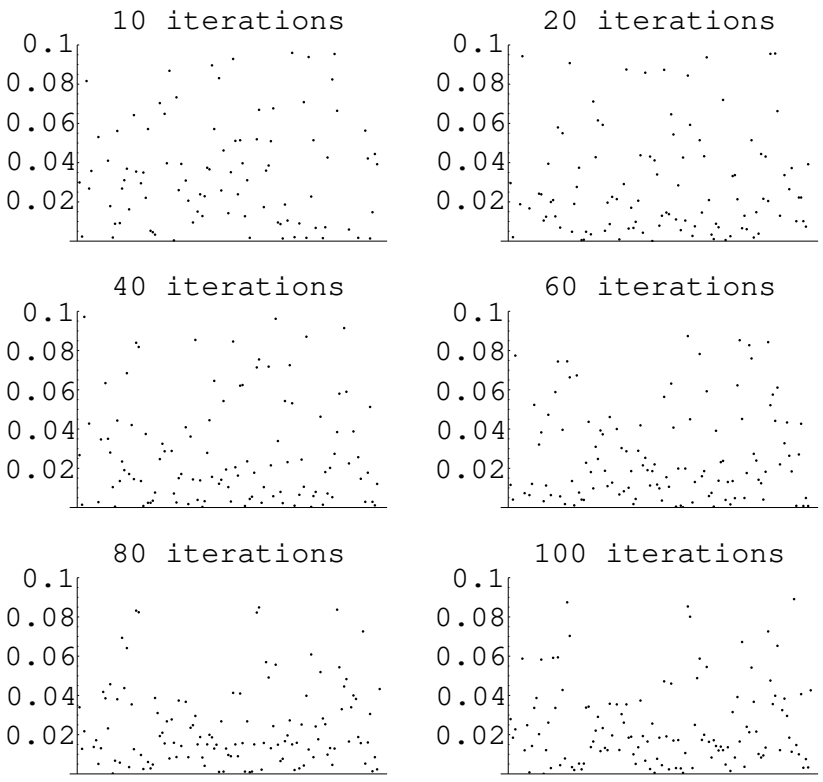

Figure 3.6: Magnitude of the determinant of the deviatoric matrix at each point after the indicated number of iterations for 128 generating points.

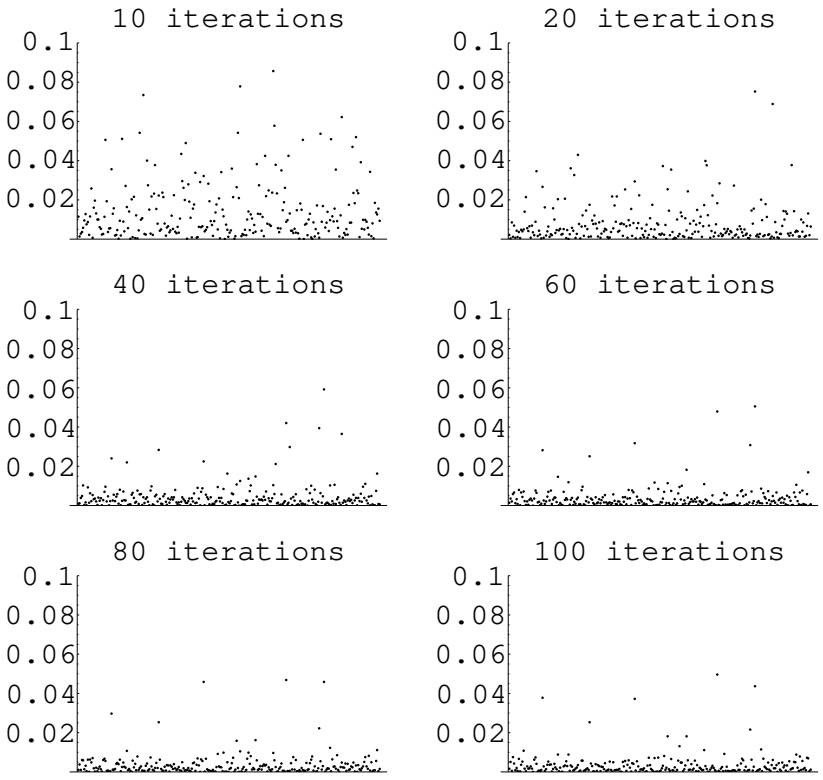

Figure 3.7: Magnitude of the determinant of the deviatoric matrix at each point after the indicated number of iterations for 256 generating points.

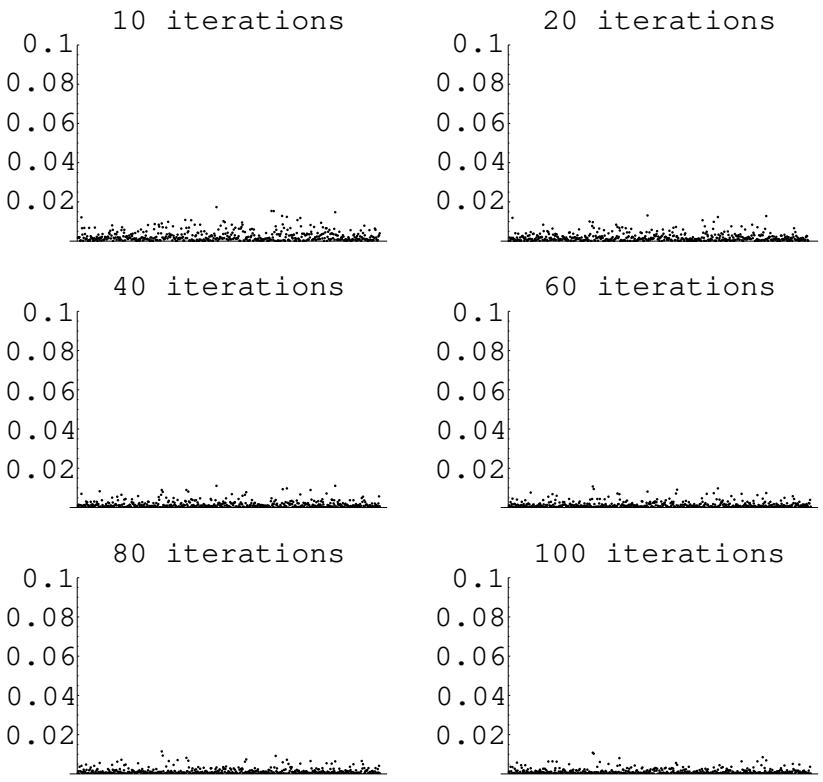

Figure 3.8: Magnitude of the determinant of the deviatoric matrix at each point after the indicated number of iterations for 512 generating points.

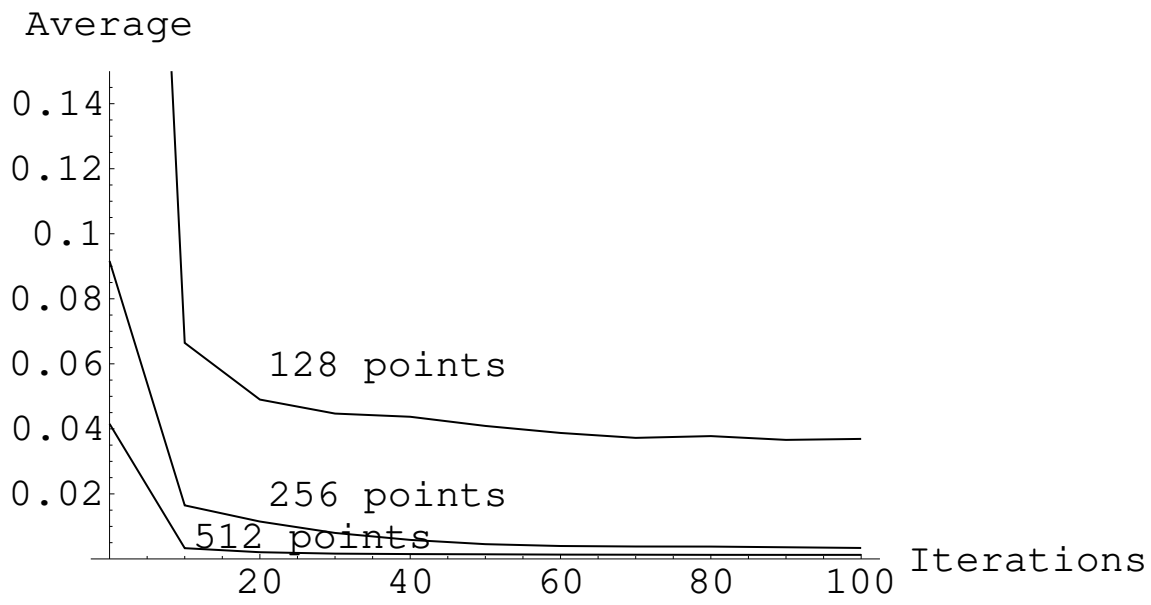

Figure 3.9: Average of the magnitude of the determinant of the deviatoric matrix vs. number of iterations for 128, 256, and 512 generating points.

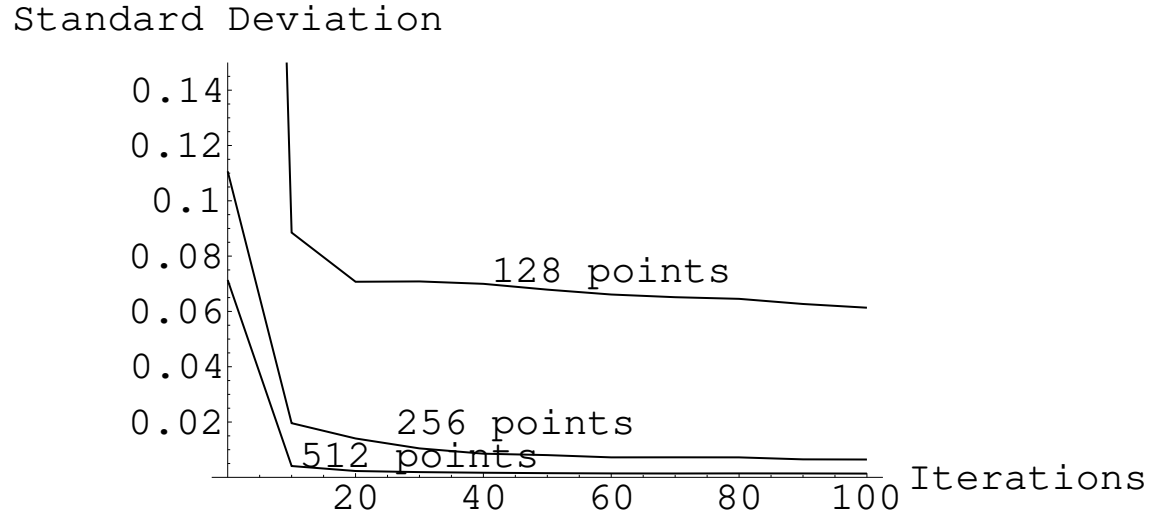

Figure 3.10: Standard deviation of the magnitude of the determinant of the deviatoric matrix vs. number of iterations for 128, 256, and 512 generating points.

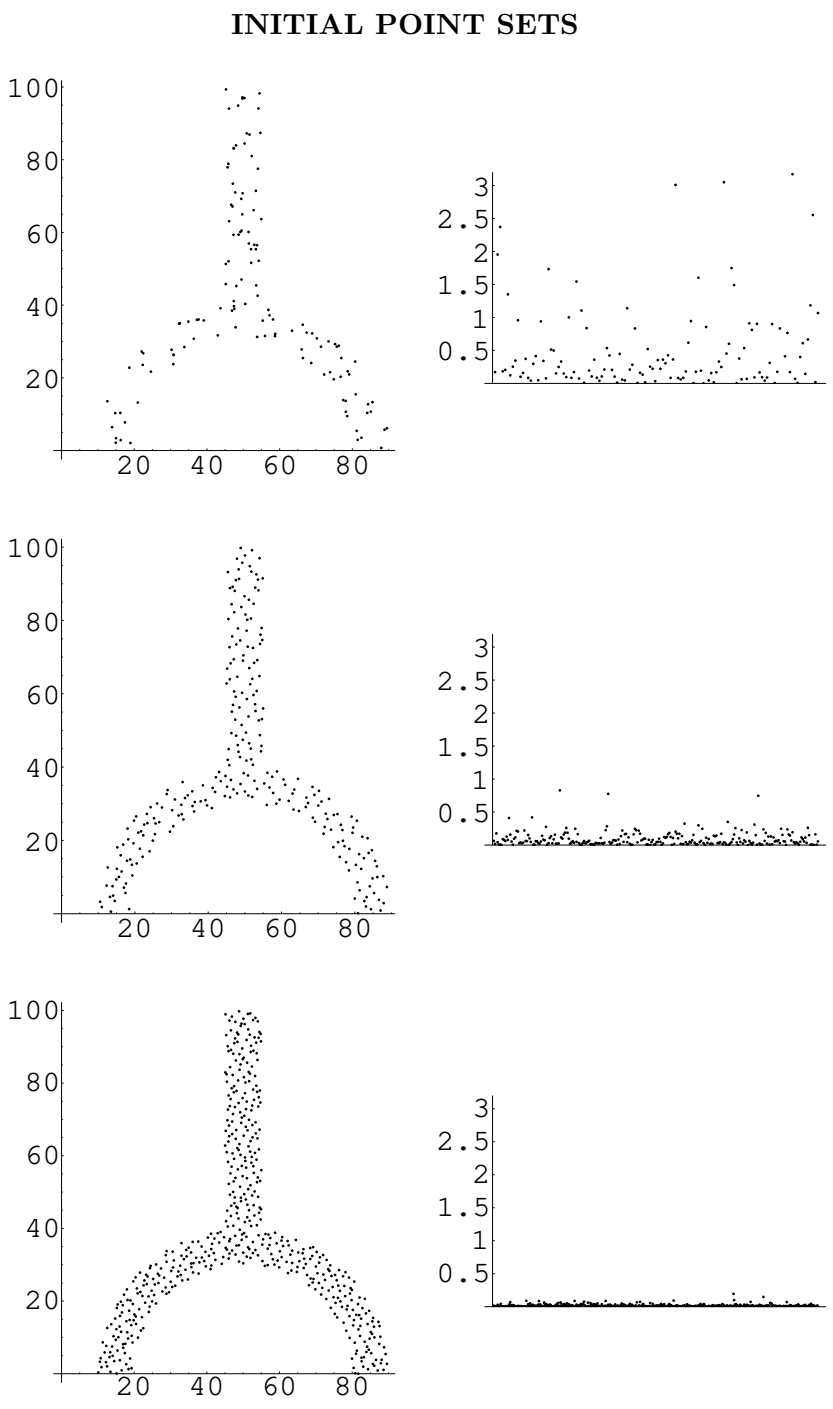

Figure 3.11: The *initial* distribution of points, i.e., Halton points, and the magnitude of the determinant of the deviatoric matrix at each point for 128, 256, and 512 generating points. The final centroidal Voronoi point distribution may be viewed in Figures 3.3 to 3.5.

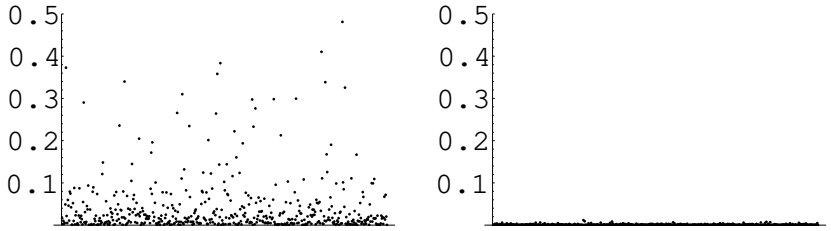

Figure 3.12: The magnitude of the determinant of the deviatoric matrix for the initial (Halton) and converged (CVT) 512 point sets.

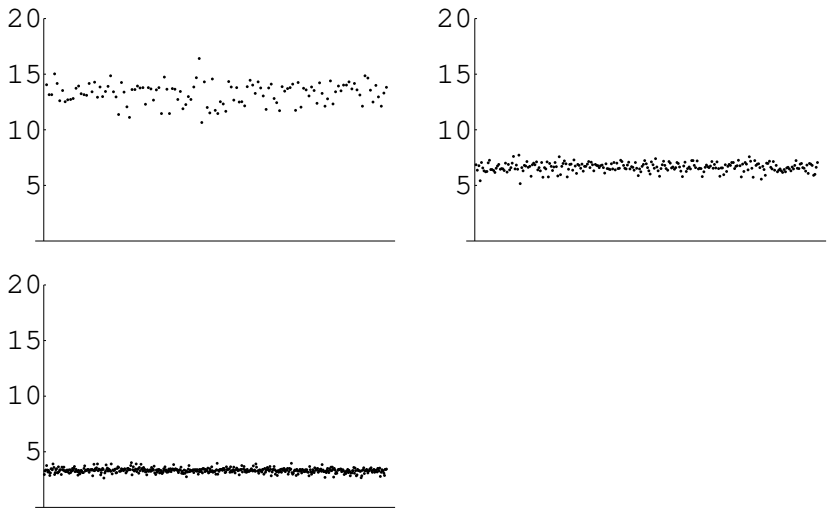

Figure 3.13: The volumes (zeroth moments) of the associated Voronoi regions after 100 iterations for 128, 256, and 512 generating points.

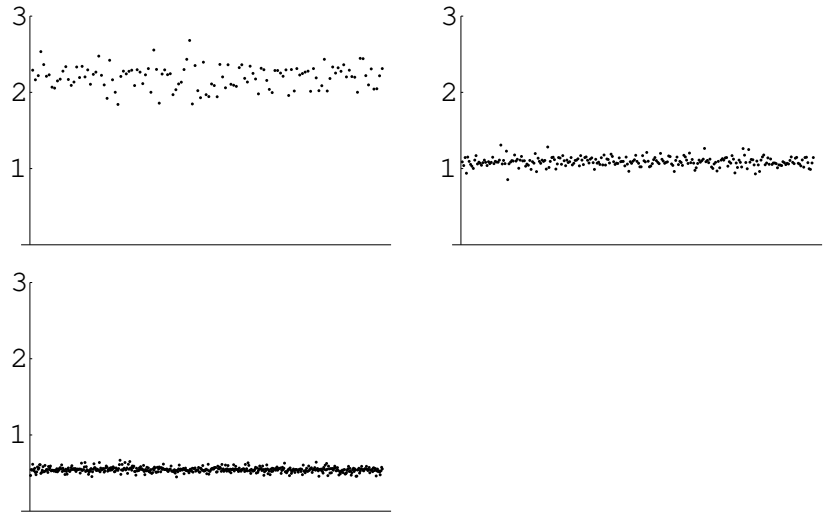

Figure 3.14: The trace of the second moment matrix after 100 iterations for 128, 256, and 512 generating points.

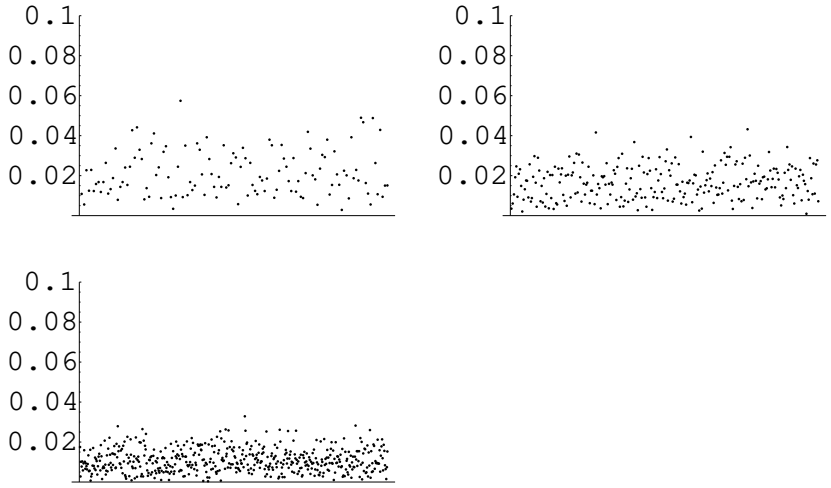

Figure 3.15: Distance between generators and first-moment vectors after 100 iterations for 128, 256, and 512 generating points.

3.2 An example in three dimensions

As an example in three dimensions, we extended the two-dimensional domain used in Section 3.1 to three dimensions by allowing the z-component to range between 0 and 20. Specifically, we use the region

$$\begin{aligned} 45 \le x \le 55, \qquad & 40 \le y \le 100, \qquad & 0 \le z \le 20 \\ 30 \le (x-50)^2 + y^2 \le 40, \qquad & 0 \le y < 40, \qquad & 0 \le z \le 20\,. \end{aligned}$$

As before, we illustrate the convergence and accuracy of the method by plotting the magnitude of the determinant of the deviatoric matrix. Results for 256, 512, and 1024 point sets are presented in Figures 3.16–3.18, respectively. From a three dimensional view of the points, it is not easy to discern how uniform the distribution of points is; however, in Figure 3.19, for completeness, we provide a three-dimensional plot of the point distribution. In Figure 3.20 we provide the traces of the second-order moment matrix for 256, 512, and 1024 points. All plots are for results obtained after 100 iterations.

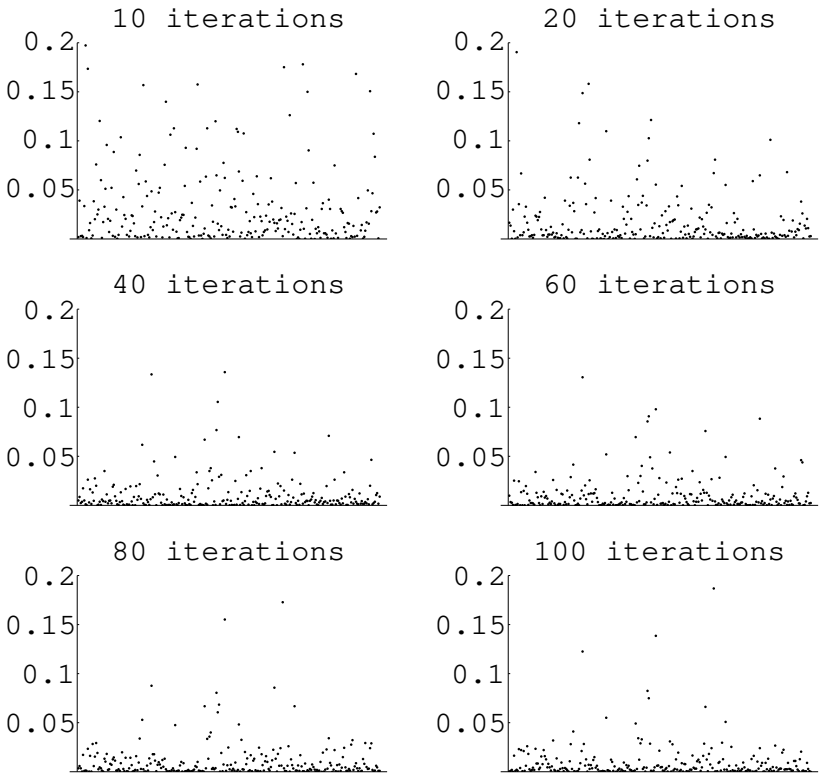

Figure 3.16: Magnitude of the determinant of the deviatoric matrix for 256 points for the three-dimensional example.

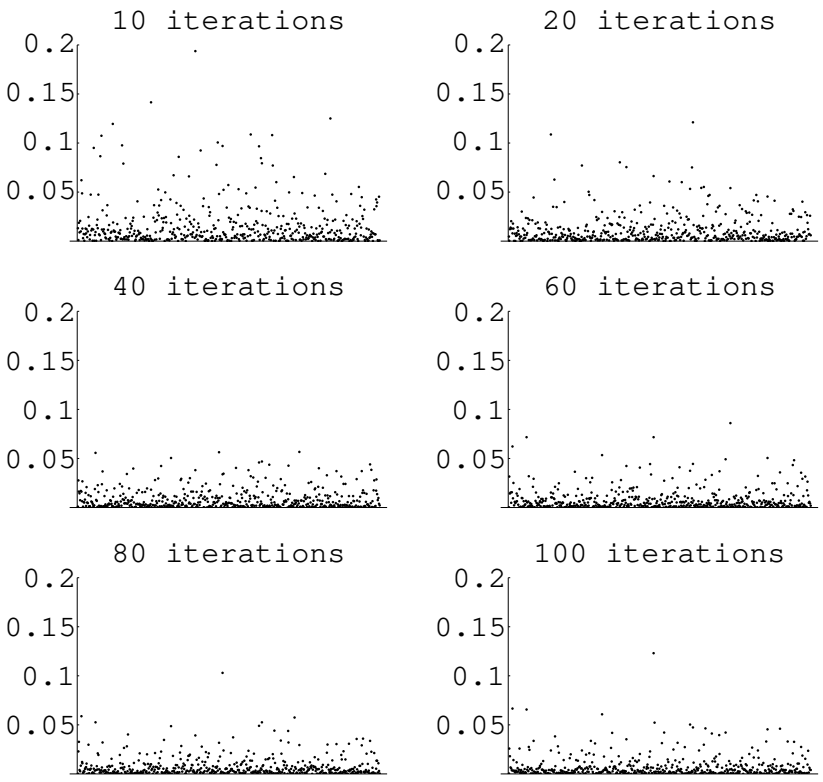

Figure 3.17: Magnitude of the determinant of the deviatoric matrix for 512 points for the three-dimensional example.

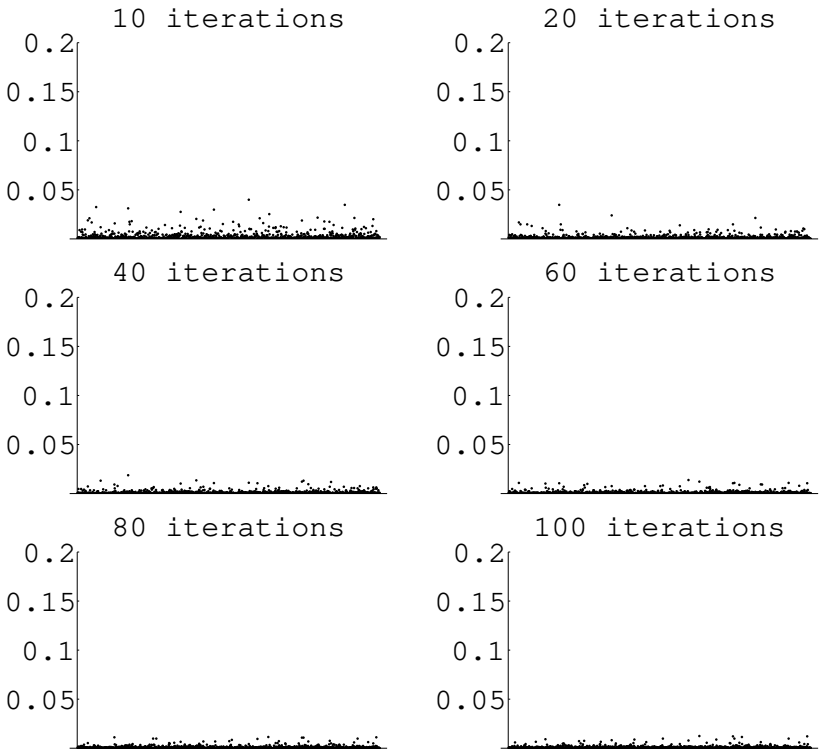

Figure 3.18: Magnitude of the determinant of the deviatoric matrix for 1024 points for the three-dimensional example.

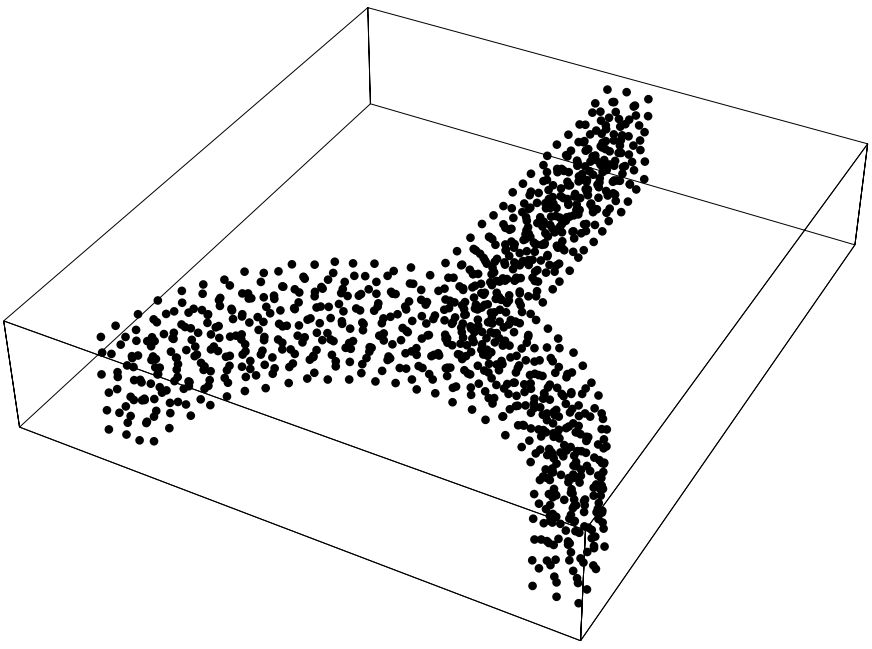

Figure 3.19: CVT point distribution for 1024 points for the three-dimensional example.

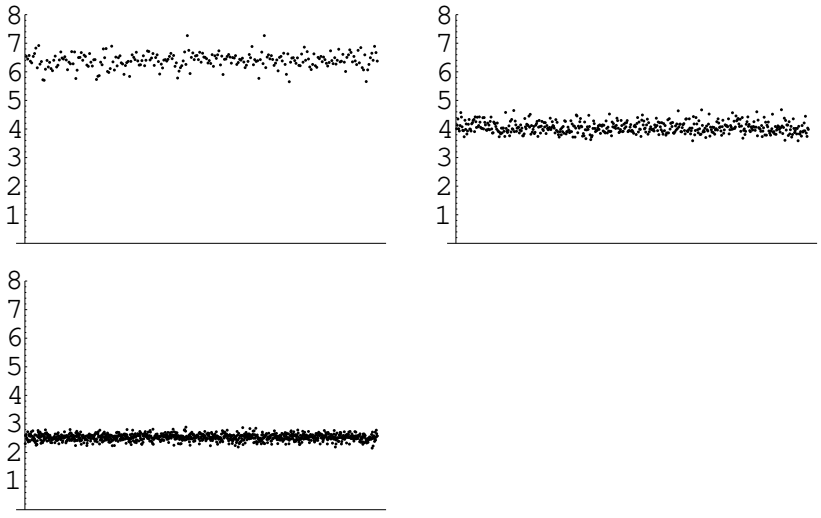

Figure 3.20: Traces of second-order moment matrix for 256, 512, and 1024 points for the three-dimensional example.

3.3 Timings

There are several factors which affect the time it takes the code to generate a centroidal Voronoi point set. The most important input factors which affect the timing are

N - the number of points or generators;

N_s - the number of sampling points per generator;

N_{iter} - the number of iterations.

In addition to these input parameters, the other factors that significantly contribute to the timing are

– the routine to test whether a sampled point is inside the region Ω;

– the routine to find the centroidal Voronoi generator closest to a given point.

The total number of sampling points used at each iteration is $N \cdot N_s$, i.e., the average number of sampling points per generator times the number of generators.

The increase in time due to an increase in N_s, the number of sampling points per generator, is roughly linear as can be seen from Table 3.1. In that table, we compare the time it takes to complete ten iterations for the two-dimensional example of Section 3.1. In Table 3.2, we see that when we fix the number of sampling points per generator N_s and double the number of generators N, then the timing increases by a factor of roughly four. For Tables 3.1 and 3.2, we used the exhaustive approach to finding the closest generator; we saw in Section 2.3 that this approach requires N^2 work, which is confirmed in the tables. Note that an increase in N_{iter}, the number of iterations, increases the timing linearly since each iteration takes approximately the same amount of time when all other parameters are fixed.

Number of generators	*Number of sample points per generator*	*CPU time*	*Increase in CPU time*
256	50	24	
256	500	249	10.4
256	5000	2419	9.71
512	50	92	
512	500	911	9.9
512	5000	9133	10.0
1024	50	362	
1024	500	3626	10.0
1024	5000	35608	9.8

Table 3.1: Comparison of timings for 10 iterations in two dimensions.

There are two aspects of the code which take the majority of the time required to complete an iteration. The first is, after a point is randomly sampled in an enclosing box,

Number of generators	*Number of sample points per generator*	*CPU time*	*Increase in CPU time*
256	5000	2419	
512	5000	9133	3.78
1024	5000	35608	3.90

Table 3.2: Comparison of timings as the number of generators increase.

determining if the point is inside or outside the region; the second is, given a point in the region, finding the closest generator.

The code allows two options for finding the generator nearest to a given point. The first option is to use a routine which is simply an exhaustive search among all generators whereas the second uses "bins" to reduce the amount of work; see Section 2.3. In Table 3.3, some timings are presented to compare the two methods. In all cases, 5000 sampling points per generator are used and the times given are for the completion of 10 iterations. We see that the "bins" approach results in a linear increase in the times as the number of generators increases while, as we already have seen, the exhaustive search approach results in a quadratic increase in the time. Thus, it is usually recommended to use the "bin approach" but, as already mentioned, the code allows the user to choose either approach. Note that the relative size of the advantage of the "bin" approach can also depend on things like compiler options.

Number of generators	*CPU time for exhaustive search*	*CPU time for bin search*	*Factor*
256	2419	442	5.5
512	9133	894	10.2
1024	35608	1847	19.3

Table 3.3: Timings to compare methods to find a generator nearest a given point.

The other portion of the code that can significantly contribute to CPU time is the test to determine if a point is inside or outside the given region. In the above calculations, we have used an analytic description of the region (instead of DIATOM) to determine if a given point is inside the region. In two dimensions, we have not seen a significant difference in timing when DIATOM or the analytic description of the region is used. However, in three dimensions, we have seen a significant difference. For example, using 256 generators and 5000 sampling points per generator, we saw an increase of a factor of approximately 15 when DIATOM was used. After running some experiments, we believe that DIATOM can quickly determine if a point is or is not in the region but when the point is in the region, DIATOM calculates a quantity MDENS which evidently is costly to compute. Since this quantity is not needed by our codes, it seems that one could probably modify DIATOM to quickly determine if a point is inside the region.

Chapter 4

Avenues for future work

There are a number of directions for the possible improvement or generalization of the algorithms discussed in this report. Here, we list a few of the possibilities.

- Instead of using randomly distributed sample points, one could sample points on a regular lattice. This is not only less expensive to do, but allows for the use of Richardson extrapolation, i.e., running simulations using two different sampling lattices, one finer than the other, and then combining the results to obtain higher accuracy.
- Similarly, instead of using Halton sequences or random sampling for determining initial point sets, one could use a regular hexagonal grid of points. This may be less expensive to determine and, since the final arrangement is hexagonal away from boundaries, it may require fewer iterations for the convergence of algorithms to determine a centroidal Voronoi tessellation.
- It would be useful to allow the user to specify the orientation of the hexagonal lattice in the interior of the domain. One potential application is the generation of point sets for laminates.
- For some applications, it would be useful for the codes to be able to treat point distributions which are nonuniformly distributed according to a user specified density function.
- For some applications, it would be useful to determine particles on the boundary or particles that are near the boundary. Likewise, it would be useful to determine the boundary normal at boundary points or the normal vector from nearby interior points to the boundary.
- For irregularly shaped domains, it may be beneficial to use a recursively refined quadtree (or oct-tree in three dimensions) lattice to locate the boundary. In this approach, refinement would occur as one approaches the boundary. As a result, except near the boundary, cells that are totally within the domain would be large. The DIATOM interface built into the codes can particularly take advantage of this approach since it is designed to return volume fractions.

Bibliography

[1] R. Bell, M. Baer, R. Brannon, R. Cole, D. Crawford, M. Elrick, E. Hertel, Jr. S. Silling, and P. Taylor; *CTH User's Manual and Input Instructions*, Version 6.00 (Internal Report), CTH Development Project, Sandia National Laboratories, Albuquerque.

[2] J. Bentley, B. Weide, and A. Yao; Optimal expected time algorithms for closest point problems, *ACM Trans. Math. Soft.* **6** 1980, pp. 563-580.

[3] E. Boucheron, K. Budge, D. Carrol, S. Carrol, R. Drake, T. Haill, J. Peery, S. Petney, A. Robinson, R. Summers, T. Trucano, J. Weatherby, and M. Wong; ALEGRA: User Input and Physics Descriptions, *Sandia National Laboratories report No.* SAND99-3012 Sandia National Laboratories, Albuquerque, 1999.

[4] T. Cormen, C. Leiserson, and R. Rivest; *Introduction to Algorithms*, MIT Press, Cambridge.

[5] Q. Du, V. Faber, and M. Gunzburger; Centroidal Voronoi tessellations: Applications and algorithms, *SIAM Review* **41** 1999, pp. 637-676.

[6] Q. Du and M. Gunzburger; Grid generation and optimization based on centroidal Voronoi tessellations; submitted.

[7] Q. Du, M. Gunzburger, and J. Lili; Probabilistic methods for centroidal Voronoi tessellations and their parallel implementation; submitted.

[8] Q. Du, M. Gunzburger, and J. Lili; Meshfree, probabilistic determination of point sets and support regions for meshless computing; submitted.

[9] S. Lloyd; Least squares quantization in PCM, *IEEE Trans. Infor. Theory* **28** 1982, pp. 129-137.

[10] J. MacQueen; Some methods for classification and analysis of multivariate observations, *Proc. Fifth Berkeley Symposium on Mathematical Statistics and Probability*, Vol I, Ed. by L. Le Cam and J. Neyman, University of California, 1967, pp. 281-297.

[11] A. Okabe, B. Boots, K. Sugihara, and S. Chui; *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*, 2nd Edition, Wiley, Chichester, 2000.

Appendix A

User manual

In this chapter, we discuss the use of a FORTRAN 90 (F90) library of routines that implement Algorithm 1 (see page 8) for determining the generators of a centroidal Voronoi tessellation of a user-specified region and Algorithm 4 (see page 15) for determining the zeroth, first, and second-order moments of the associated Voronoi regions. Algorithm 2 (see page 9) for random sampling in a general region and Algorithm 3 (see page 12) for a bin-based closest point search are also implemented in the library, as are algorithms for Halton sequence-based point placement (see page 10) and for exhaustive closest point search (see page 11).

The user is required to supply the following.

- A calling program in which input variables are specified, storage space for several internal variables is reserved, and desired outputs are saved to files; see Section A.2.6. The user supplied calling program also calls the library routines in an appropriate order.

- A means for defining the geometry of the physical region. This task is simplified, however, if the user already has a DIATOM input file. The library has an interface with DIATOM and can be instructed to have DIATOM read that file and participate in the treatment of the region. Otherwise, the user has to supply a routine that specifies the region.

These requirements are discussed in detail in the remainder of this chapter. In particular, guidance about the meaning of program parameters of user interest is provided below (see Section A.2.6) as are descriptions of the purpose and use of the most important routines that make up the library (see Sections A.2.2–A.2.5). A sample calling program (see Sections A.2.1 and A.2.7) and a DIATOM input file (see Section A.1) will also be briefly described; this may provide the quickest way to understanding how the library is intended to work.

In order to facilitate comparisons between the algorithmic descriptions in Chapters 1 and 2 and how they are realized in the codes, we provide, in Table A.1, a list of variables defined in Chapters 1 and 2 and the corresponding names used in the codes.

Variable name		*Description of variable*	*Page reference*
in chapters 1 and 2	*in codes*		
d	NDIM	number of space dimensions	3
N	N	number of generators and Voronoi regions	3
N_s	NS_CVT	average number of points sampled per generator per iteration of the CVT algorithm (Algorithm 1)	8
a_1, a_2, a_3	BOX_MIN	minimal coordinates of box enclosing the region	9
b_1, b_2, b_3	BOX_MAX	maximal coordinates of box enclosing the region	9
N_{iter}	MAXIT	number of iterations used in CVT determination (Algorithm 1)	11
N_m	NS_MOM	average number of points sampled per Voronoi region for the approximation of the moments (Algorithm 4)	15

Table A.1: Correspondence between variable names used in Chapters 1 and 2 and those appearing in the codes. The "Page references" column gives the number of the page in Chapter 1 or 2 in which the variable is defined.

A.1 Region specification

The library allows, at the user's discretion, for two ways of specifying the geometry of the region of interest. First, the library contains routines that interface with DIATOM, and second, it allows the user to supply a routine called TEST_REGION for domain specification. We will not discuss TEST_REGION any further; however, a sample which corresponds to an analytic description of the DIATOM test case (see Figure 3.19) is included with the code distribution.

A.1.1 The DIATOM geometry file

If, in the main program, the user sets the logical parameter USE_DIATOM to TRUE, then DIATOM_SETUP is called before the CVT routines are invoked and the code expects that the region is defined by a DIATOM input file. The general structure of such input files can be found in [3] and [1]. However, we display below the three-dimensional version of a DIATOM test region, which is the union of a box and a cylindrical shell; see Figure 3.19. For this example, the DIATOM input file is simply given as follows.

```
diatom

  package 'box'
```

```
  density 1
  insert box
    p1 = 45,35,0
    p2 = 55,90,20
  endinsert
endpackage

package 'cylinder'
  density 1
  insert cylinder
    ce1 = 50,0,0
    ce2 = 50,0,20
    r = 40
    ri = 30
  endinsert
endpackage
```

A.2 Library subroutines and code variables

Starting from a typical user supplied main program, we look at the more interesting routines that make up the library of codes. We will also look at the input and output variables, as well as other variables of interest; see Section A.2.6 for a complete list. A schematic of a sample main calling program is given in Section A.2.7; it may be referred to to facilitate understanding of the purposes of variables and library routines.

A.2.1 The user main program

The user supplied main program is responsible for setting parameters, defining data, calling and controlling the library routines, and processing output. It generally calls the following routines:

- RANDOM_INITIALIZE which initializes the F90 random number seed;
- DIATOM_SETUP which initializes DIATOM and reads a DIATOM input file, if DIATOM is to be used for region definition;
- GENERATOR_INIT which initializes the Voronoi generators;
- BIN_PREPROCESS which sorts generators into bins, if bins are used for closest point searches;
- CVT_ITERATION which takes one step of the CVT iteration;
- VCM which calculates the zeroth moments (volumes), first moments (centroids), and second moments of the Voronoi regions.

The main program embeds the calls to the BIN_PREPROCESS and CVT_ITERATION routines inside of a loop to effect multiple iterations of the CVT generation algorithm. The BIN_PREPROCESS routine also has to be called before the call to the VCM routine.

A.2.2 CVT construction

The CVT calculation (Algorithm 1), is embodied in the routines GENERATOR_INIT and CVT_ITERATION.

- GENERATOR_INIT uses sampling to produce a set of N points to be used as the initial positions for the generators; it repeatedly calls the routine
  - REGION_SAMPLER which produces a random point X in the region.
- CVT_ITERATION is called repeatedly, each time applying one step of the probabilistic Lloyd's method (Algorithm 1). In doing so, it repeatedly calls the routine
  - REGION_SAMPLER which produces a random point X in the region

  and then, to find the closest generator to the point X, it calls *one* of the following two routines:
  - FIND_CLOSEST which uses a naive algorithm for closest point search;
  - POINTS_NEAREST_POINT_BINS3_3D which uses bins for a more efficient closest point search.

A.2.3 Calculation of moments

The calculation of the zeroth, first, and second moments (Algorithm 4) is performed by the routine VCM.

- VCM uses random sampling and averaging to estimate the integrals in the definitions of the three moments; it repeatedly calls the routine:
  - REGION_SAMPLER which produces a random point X in the region

  and then, to find the nearest generator to the point X, it calls *one* of the following two routines:
  - FIND_CLOSEST which uses a naive algorithm for closest point search;
  - POINTS_NEAREST_POINT_BINS3_3D which uses bins for a more efficient closest point search.

A.2.4 Sampling

A recurring task in the code is the sampling of points in the region. Sampling is done by the routine REGION_SAMPLER which chooses a random point in the enclosing box and determines if the point lies inside or outside the given region. If it is inside, the point is accepted; if it is outside it is rejected and another point is sampled and tested.

- REGION_SAMPLER supervises the sampling procedure; a random point within the enclosing box is chosen by calling *one* of the following two routines:

- RANDOM_NUMBER which uses the F90 random number generator as a basis for sampling points;
- I_TO_HALTON_VECTOR which uses Halton sequences as a basis for sampling points.

Then, the sample point is tested to determine whether it is within the physical region by calling *one* of the following two routines:

- TEST_REGION which determines if a point is within a region defined by formulas;
- DIATOM_POINT_TEST2 which determines if a point is within a region defined by DIATOM input.

The generation and testing process is repeated, if necessary, until the sampling routine can return a suitable point.

The Halton sequence choice for generating a sampling point in the enclosing box is only available for the initialization of the Voronoi generators (routine GENERATOR_INIT of Section A.2.2). The other two places that require point sampling (routines CVT_ITERATION and VCM of Sections A.2.2 and A.2.3, respectively) use only point sampling based on the the F90 random number generator.

Note that the F90 random number generator is used to sample random points in the enclosing box, i.e., by the RANDOM_NUMBER routine. The RANDOM_INITIALIZE routine called by the main program initializes the seed for the F90 random number generator, either by using a user supplied seed or by using the system clock to generate a seed. The first option should be used if reproducible results are of interest.

A.2.5 Closest neighbor search

Given a point x in the region (which is the output of one of the sampling routines of Section A.2.4) and the current set of CVT generators, both the CVT_ITERATION and VCM routines need to determine the generator that is closest to x. The user chooses which of the following routines is called to accomplish this task:

- FIND_CLOSEST which finds the closest generator to x through an naive exhaustive search;
- POINTS_NEAREST_POINT_BINS3_3D which finds the closest generator to x more efficiently using bins as in Algorithm 3.

If the bins approach is used for the closest neighbor calculation, then the generators must be assigned to bins. This should be done after the generators are initialized and whenever they are adjusted, i.e., whenever they may have moved from one bin to another. In the main program, the routine BIN_PREPROCESS is called for this purpose. For a three-dimensional problem, this preprocessing routine calls the following three routines:

- D3VEC_BIN_EVEN3 which places the elements of an array of three-dimensional vectors into evenly spaced bins;

- D3VEC_BINNED_REORDER2 which reorders the elements of an array by bin;
- D3VEC_BINNED_SORT_A2 which sorts the elements within each bin.

The bin preprocessing and computation routines call some lower level routines which will not be discussed here.

Note that the bin routine for closest point determination includes a two-dimensional version.

A.2.6 Glossary of variables

In Tables A.2–A.6, we list the names and meanings of the input variables that the user must specify. In Table A.7, the output variables that the user can access are listed. In Table A.8, internal variables for which storage space must be reserved are listed.

Initialization of random number generator		
Name	*Type*	*Description*
SEED	integer	if set to zero, the seed for the F90 random number generator is determined from the system clock; if set to a nonzero number, it is used as the seed for the F90 random number generator; this option should be used if reproducible results are desired.

Table A.2: The input variable used to determine how the F90 random number generator is initialized.

Geometric input variables		
Name	*Type*	*Description*
NDIM	integer	the spatial dimension, either 2 or 3.
BOX_MIN	double precision (NDIM)	the minimum coordinate values of a box enclosing the physical region.
BOX_MAX	double precision (NDIM)	the maximum coordinate values of a box enclosing the physical region.
USE_DIATOM	logical	set to TRUE if DIATOM is called to determine if a point is inside in the physical region; set to FALSE if a user-supplied subroutine TEST_REGION is called for this purpose.
DR	double precision	if USE_DIATOM is TRUE, a tolerance used by DIATOM when testing if a point is inside the physical region.

Table A.3: The input variables used to define the physical region being analyzed.

CVT algorithm input variables		
Name	*Type*	*Description*
N	integer	the number of points to be generated.
MAXIT	integer	the number of iterations used for the CVT determination.
NS_CVT	integer	the average number of points sampled per generator at each step of the CVT iteration.
RANDOM_GENERATOR	integer	set to 0 if the F90 random number generator is used for determining the initial position of the generators; set to 1 if Halton sequence-based sampling is used for this purpose (preferred choice).

Table A.4: The input variables needed by the CVT determination algorithm.

Moment algorithm input variables		
Name	*Type*	*Description*
NS_MOM	integer	the average number of points sampled per generator for the approximation of the moments.
REGION_VOLUME_GIVEN	logical	set to TRUE if the region volume is input in REGION_VOLUME; set to FALSE if the region volume is determined internally by the library.
REGION_VOLUME	double precision	if REGION_VOLUME_GIVEN is TRUE, then REGION_VOLUME is the region volume input by the user.

Table A.5: The input variables needed for the determination of the zeroth, first, and second moments.

Closest neighbor calculation input variables		
Name	*Type*	*Description*
USE_BINS	logical	set to TRUE if the enclosing box is divided up into bins to speed up the nearest neighbor search; set to FALSE if the nearest neighbor search is done through exhaustive search.
NBIN	integer (3)	the number of bins to use in each direction; for 2D problems, set NBIN$(3) = 1$. For efficiency, these values should be set in such a way that the bins are nearly square or cubical.

Table A.6: The input variables needed for closest neighbor calculation.

Output variables		
Name	*Type*	*Description*
CELL_GENERATOR	double precision (NDIM,N)	the CVT generating points.
CELL_VOLUME	double precision (N)	the volume (zeroth moment) of the Voronoi regions.
CELL_CENTROID	double precision (NDIM,N)	the centroids (first moments) of the Voronoi regions.
CELL_MOMENT	double precision (NDIM,NDIM,N)	the second moments of the Voronoi regions.

Table A.7: The output variables of the CVT and moment determination library.

Internal variables for which storage must be reserved		
Name	*Type*	*Description*
BIN_START	integer (NBIN(1),NBIN(2),NBIN(3))	the index of the first cell center in the bin, or -1 if none.
BIN_LAST	integer (NBIN(1),NBIN(2),NBIN(3))	the index of the last cell center in the bin, or -1 if none.
BIN_NEXT	integer (N)	the index of the next cell center in the bin containing this cell center.

Table A.8: Variables which are internally set but for which the user has to reserve storage space.

A.2.7 Schematic of the main program

We provide a schematic version of a typical main program, emphasizing the input and output variables of primary interest to the user, and the order in which the CVT library routines should be called. For clarity, we suppress the display of less important data and routines, particularly those used for bin searches. For a full working example, refer to the main program included with the code distribution. For an explanation of the purpose of the individual subroutines invoked here, see Sections A.2.2–A.2.5 and A.1. For an explanation of the meaning and purpose of the variables, see Section A.2.6.

```
!    Inputs

  integer, parameter :: seed = 0
  integer, parameter :: n = 1024
```

```
  integer, parameter :: ndim = 3
  real, dimension ( ndim ) :: box_min = (/   0.0,   0.0, 0.0 /)
  real, dimension ( ndim ) :: box_max = (/ 100.0, 100.0, 5.0 /)
  integer, parameter :: maxit = 10
  integer, parameter :: ns_cvt = 5000
  integer, parameter :: ns_mom = 5000
  logical, parameter :: use_bins = .false.
  logical, parameter :: use_diatom = .true.

!    Output variables

  double precision cell_centroid(ndim,n)
  double precision cell_generator(ndim,n)
  double precision cell_moment(ndim,ndim,n)
  double precision cell_volume(n)

!    Initialize the random number generator.

  call random_initialize ( seed )

!    If DIATOM is to be used, then DIATOM must be intialized.

  call diatom_setup ( )

!    Initialize the position of the Voronoi generators.

  call generator_init ( ndim, box_min, box_max, n, cell_generator, &
    use_diatom, ... )

!    Carry out the CVT iteration.

  do it = 1, maxit

    if ( use_bins ) then

       call bin_preprocess ( ndim, box_min, box_max, n, cell_generator, &
          nbin, ... )

    end if

    if ( it <= maxit ) then

      call cvt_iteration ( ndim, box_min, box_max, n, cell_generator, &
          ns_cvt, use_diatom, use_bins, dr, ... )
```

```
    end if

  end do

!    Compute moments.

  call vcm ( ndim, box_min, box_max, n, cell_generator, ns_mom, use_diatom, &
    use_bins, ... , cell_volume, cell_centroid, cell_moment )

!    Output of results would occur here.

end
```

A.3 Summary of user control

The library includes a great deal of flexibility for the user. The user can specify the problem in several ways and has available several options for the way the CVT and moment determination algorithms are carried out. In many cases, a great deal of code is available which the user can access by simply setting a switch or parameter.

The most critical information that the code needs is the geometry of the region. The user must specify NDIM, the spatial dimension of the region, and BOX_MIN and BOX_MAX, arrays which specify the minimum and maximum coordinate values, respectively, of a box that encloses the region. The code also must be able to query the region, i.e., determine whether a given point is inside or outside the region. The user may set the value of USE_DIATOM to TRUE to have DIATOM answer these queries based on a user input file; alternately, the user can supply a subroutine to answer the queries.

For the CVT iteration (Algorithm 1), the user specifies the number of generators N and may specify how the generators are initialized by setting RANDOM_GENERATOR. The user then calls the CVT iteration step routine a number of times, with control returning to the user after each step. In the sample main program, the iteration is simply carried out MAXIT times, with no other iteration control. The CVT iteration requires the random sampling of points in the region. The user may specify the value of NS_CVT, the average number of points sampled for each Voronoi region at each iteration. As part of the CVT iteration, closest neighbor calculations are carried out. For problems with a large number of generators, this calculation can dominate the problem execution time. In such cases, the code offers an alternative method, using bins, to the naive closest neighbor calculation via exhaustive search. The user can use this option by setting USE_BINS, and then specifying in the NBINS array the number of bins to use in each dimension.

The calculation of moments also requires random sampling. The user may specify the value of NS_MOM, the average number of points sampled for each Voronoi region for the moment calculation. Again, the user may choose to do closest neighbor determination by exhaustive searches or by the use of a bin-based method.

A.4 Summary of the overall structure of the running codes

The complete running version of the codes is made up of five logical groups.

1. The master routine, a user supplied main program which defines parameters, sets up data structures, calls library routines to carry out the algorithms, and writes out important data.

2. The CVT library. This comprises a number of F90 routines that carry out the CVT construction algorithm and the determination of moments of the associated Voronoi regions. Many of these routines have optional switches that the user can set in order to vary the nature of the algorithm.

3. DIATOM, a library of routines in C and FORTRAN, supplied by Sandia.

4. Two interface routines, based on code supplied by Sandia; both are written in C and currently included in the DIATOM library. The first routine, DIATOM_SETUP, causes a DIATOM input file to be read and stored in the DIATOM geometry structure. Geometrical information about the region of interest is set in this routine. For example, for the DIATOM test case, the routine currently sets the geometry to be three dimensional, rectilinear, and contained in the bounding box [0,100] by [0,100] by [0,20]. This information is also needed by the CVT library, and some of it must be also set in the main program that runs the CVT and moment calculations. It would be much better to have these choices made by the user and passed from the main program to DIATOM_SETUP as arguments, instead of being set twice in separate routines. However, we have not effected this change; in order to avoid possibly disastrous conflicts in data, we recommend that the DIATOM_SETUP routine be changed to accept, through its calling sequence, input from the main program. The second routine,[1] DIATOM_POINT_TEST2, allows the CVT routines to query the physical region stored in DIATOM, with a response of +1, -1, or 0 depending on whether the point is inside, on the boundary, or outside the region.

5. A DIATOM input file that defines the geometry of the region of interest. In order to have DIATOM read this file, the main program must call the interface routine DIATOM_SETUP. As currently written, DIATOM expects the input file to have the name DIATOM_TEST.IN, another hardwired feature that should be made more flexible.

A.5 Miscellaneous remarks

- The CVT library is written in FORTRAN 90.

- Floating point quantities are declared DOUBLE PRECISION. The DIATOM library declares floating point quantities using the symbolic type REAL, which we

[1] See Section A.5 for comments about the necessity for the changes that were made to the Sandia supplied DIATOM_POINT_TEST routine.

currently have set to "double." The F90 code and DIATOM must always agree in this way on the size of floating point values.

- UNIX FORTRAN compilers have a tradition of storing FORTRAN symbolic names with an appended underscore. C compilers do not do this. Thus, if a FORTRAN routine wants to call a C procedure named SOLVE, a problem arises because the FORTRAN compiler will, by default, convert this into a call to SOLVE_. Most FORTRAN compilers have a way, during compilation, of requesting that underscores not be appended to symbolic names and this was necessary in the development of the CVT library in order to interface with DIATOM.

- Another issue is that FORTRAN procedure calls are by address. The interface routine DIATOM_POINT_TEST, supplied by Sandia, had a call by value. This is the reason that a slightly revised version, called DIATOM_POINT_TEST2 had to be written for use as an interface between the F90 CVT library and DIATOM.

- DIATOM has some internal source code settings that specify the spatial dimension, bounding box, and geometry type. These quantities are currently hard-wired in DIATOM_SETUP. This means that a user may have to enter certain information in two places in order to get a correct run. A proper approach would allow the user to choose these values in the main program and pass them to DIATOM_SETUP by its argument list, avoiding the possibility of a catastrophic disagreement in problem definition.

- During development of the code, CPU and real time measurements were made. There is an F90 standard routine called CPU_TIME for CPU time measurement. The implementation of this routine on the DEC Alpha was unreliable for CPU times that were more than about 30 minutes. Apparently, an internal integer would wrap around, resulting in negative CPU time intervals. Hence, CPU timings were also done with the UNIX routine ETIME. However, because the compiler was instructed not to append an underscore to symbolic names, but the compiled version of ETIME actually has an underscore, the calls to ETIME had to be explicitly made to ETIME_. Real time measurements were made by calls to the F90 routine SYSTEM_CLOCK, for which wrap-around problems did not seem to arise.

EXTERNAL DISTRIBUTION

Ted Belytschko
Department of Mechanical Engineering
Northwestern University
2145 Sheridan Road
Evanston, IL 60208

Jiun-Shyan Chen
Associate Professor
University of California, Los Angeles
Civil & Environmental Engineering Department
5731G Boelter Hall
Box 951593
Los Angeles, CA 90095

G. Filbey
Army Research Laboratory,
AMSRL-WM-TA
Aberdeen Proving Ground, MD 21005-5066

Max D. Gunzburger **(15)**
Professor and Chair
Iowa State University of Science and Technology
Department of Mathematics
400 Carver Hall
Ames, Iowa 5001-2064

Gordon Johnson
MN11-1614
Alliant Techsystems, Inc.
600 2nd Street NE
Hopkins, MN 55343-8384

Wing Kam Liu
Northwestern University
Dept. of Mechanical Engineering
2145 Sheridan Road
Evanston, IL 601MS 08-3111

Patrick McMurtry
University of Utah
Department of Mechanical Engineering
50 S. Central Campus Drive,
Room 2202
Salt Lake City, UT 84112-92MS 08

Louis Moresi and Hans-Bernd Mhlhaus
Australian Geodynamics Cooperative
Research Centre CSIRO division of Exploration and Mining
PO Box 437
Nedlands WA 6009, Australia

A.M. Rajendren
Army Research Laboratory,
AMSRL-MA-PD
Aberdeen Proving Ground MD 21005-5066

Glenn Randers-Pehrson
Army Research Laboratory,
AMSRL-WM-TD
Aberdeen Proving Ground MD 21005-5066

H. L. Schreyer
Department of Mechanical Engineering
University of New Mexico
Albuquerque, NM 87131

Steven Segletes
Army Research Laboratory,
AMSRL-WM-TD
Aberdeen Proving Ground, MD 21005-5066

Deborah Sulsky
Department of Mechanical Engineering
University of New Mexico
Albuquerque, NM 87131

Allen York
Applied Research Associates, Inc.
811 Spring Forest Road, Suite 100
Raleigh, NC 27609

Los Alamos National Laboratory
Mail Station 5000
PO Box 1663
Los Alamos, NM 87545

Attn: Frank Addessio, MS B216
Group T-3
Attn: Jerry Brackbill, MS B216
Attn: D. L. Crane, MS P946
Attn: Gary A. Dilts, MS D413
CCS-2: Scientific Computing
Attn: Y. Horie, MS D413,
X-7 Applied Physics Div.
Attn: L. Libersky, MS D413, X-3

MS 0151 T.O. Hunter, 09100
MS 0310 P. Yarrington, 09230
MS 0310 G.S. Heffelfinger, 09209
MS 0316 J.B. Aidun, 09235
MS 0316 M.D. Rintoul, 09235
MS 0316 M.J. Stevens, 09235
MS 0321 W.J. Camp, 09200
MS 0525 T.V. Russo, 01734
MS 0751 L.S. Costin, 06117
MS 0751 A.F. Fossum, 06117
MS 0819 E.A. Boucheron, 09231
MS 0819 K.H. Brown, 09231
MS 0819 T.E. Voth, 09231
MS 0819 M.K. Wong, 09231
MS 0820 T.J. Bartel, 09232
MS 0820 P.F. Chavez, 09232
MS0820 D.A. Crawford, 09232
MS 0820 M.E. Kipp, 09232
MS 0820 S.A. Silling, 09232
MS 0826 H.C. Edwards, 09131
MS 0826 W.L. Hermina, 09113
MS 0826 J.R. Stewart, 09131
MS 0826 J.D. Zepper, 09131
MS 0834 A.C. Ratzel, 09112
MS 0835 S.N. Kempka, 09141
MS 0835 J.S. Peery, 09142
MS 0836 M.R. Baer, 09100
MS 0836 E.S. Hertel, 09116
MS 0841 T.C. Bickel, 09100
MS 0847 S.W. Attaway, 09142
MS 0847 M.L. Blanford, 09121
MS 0847 S.N. Burchett, 09132
MS 0847 A.S. Gullerud, 09121
MS 0847 M. W. Heinstein, 09121
MS 0847 S.W. Key, 09121
MS 0847 J.A. Mitchell, 09121
MS 0847 H.S. Morgan, 09120
MS 0847 D. J. Segalman, 09124
MS 0847 J. W. Swegle, 09142
MS 0893 R.S. Brannon, 09123 **(5)**
MS 0893 R. S. Chambers, 09123
MS 0893 D.C. Hammerand, 09123
MS 0893 C.S. Lo, 09123
MS 0893 M.K. Neilsen, 09123
MS 0893 E.D. Reedy, 09123
MS 0893 W.M. Scherzinger, 09123
MS 0893 G.W. Wellman, 09123
MS 9042 M.L. Chiesa, 08727
MS 9042 J.J. Dike, 08727
MS 9042 J.M. Hruby, 08702
MS 9042 V.D. Revelli, 08727
MS 9161 E.P. Chen, 08726
MS 9161 P.A. Klein, 08726
MS 9405 D.J. Bammann, 08726

MS 9405 J.W. Foulk, 08726

MS 9405 M.F. Horstemeyer, 08726

MS 9405 D.A. Hughes, 08726

MS 1033 D.S. Drumheller, 06211

MS 1076 S.C. Hwang, 01745

MS 1152 M.L. Kiefer, 01642

MS 1411 H.E. Fang, 01834

MS 1411 E.A. Holm, 08134

MS 1411 V. Tikare, 01834

MS 0612 Review and Approval Desk, 9612 **(1)**
for DOE/OSTI

MS 0899 Technical Library, 9616 **(2)**

MS 9018 Central Technical Files,
8945-1