PHYSICS-BASED POINT PLACEMENT BY PARTICLE DYNAMICS SIMULATION

By

Philip Wesley Fackler

PHYSICS-BASED POINT PLACEMENT BY PARTICLE DYNAMICS SIMULATION

By

Philip Wesley Fackler

A Thesis Submitted to the Faculty of the University
of Tennessee at Chattanooga in Partial
Fulfillment of the Requirements of the
Degree of Master of Science
in Engineering

The University of Tennessee at Chattanooga
Chattanooga, Tennessee

December 2013

ABSTRACT

A physics-based approach to point cloud distribution for mesh generation is investigated using inter-nodal attraction and repulsion forces based on the Lennard-Jones pair potential and a simplified particle dynamics simulation. This method produces smooth distributions of points which accurately correspond to desired scalar spacing fields. Resulting point distributions are triangulated using Lawson's algorithm and the quality of these resulting meshes is measured, demonstrating the effectiveness of the approach. Several features planned for future development for increasing the robustness and versatility of the proposed method are discussed.

DEDICATION

This work is dedicated to God, without whom everything is meaningless and before whom the plans of arrogant men will come to nothing. May I never forget You and may I do all things (even this!) as unto You.

ACKNOWLEDGEMENTS

I would like to thank my wife, Amanda Fackler, for her too-good-to-be-true-but-still-true love and support. She has sacrificed much to see me succeed.

Much gratitude goes to Dr. Steve Karman who has exhibited the availability, attention, and care that characterize a true teacher. I consider myself blessed to have been able to learn from him.

I would also like to thank Wally Edmondson for hours of help on my behalf in ways that I am sure are beyond his job description.

Appreciation must also be expressed for my (including my wife's) family who have never ceased encouraging me throughout this process.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Motivation

For many real-world computational continuum problems, the vast majority of man-hours involved in obtaining a solution goes into generating a suitable domain discretization (that is, a mesh or grid) that will allow the computational solver to compute accurate results. There is therefore a need for a method of generating meshes that is not only automated and universally applicable to complex geometries but also that has the following features:

- High level of quality in the resulting meshes
- Correspondence with desired spacing fields
- Ability to adapt to and resolve features of solution data in order to achieve greater spatial accuracy
- Ability to adapt to moving geometry maintaining temporal accuracy and quality elements (for time-dependent problems)
- Ability to use different types of geometry definitions (i.e., analytic or discrete)

Types of Meshes

A *structured* mesh is comprised of general quadrilateral-shaped cells (hexahedral in 3D) for which the connectivity between the vertices (nodes) is not explicitly stored because it is implied by the ordered indexing of the nodes. For a complex geometry multiple blocks of cells can be used for which the nodes on the shared faces of the blocks are duplicated but the node indexing does not continue across a face from one block to another.

When a mesh has no regular ordering of the node indices and, therefore, the node-to-node connectivity must be somehow explicitly defined, then, regardless of the shape of the elements, it is an *unstructured* mesh. Nodes in an unstructured mesh can be numbered arbitrarily, and the cells in an unstructured mesh can vary in shape one from another arbitrarily, polygons in 2D and polyhedra in 3D.

Methods for Generating Meshes

The process of generating a structured grid is mostly non-automated and usually quite labor-intensive, but it can be greatly aided by the use of commercial mesh generation software such as Pointwise [1]. Many methods have been proposed for generating unstructured meshes, some more automated than others. Cartesian methods make use of quadrilateral elements (in 2D) aligned with the Cartesian coordinates and recursively subdivide them to refine resolution in areas where needed. Cartesian hierarchical refinement is quite versatile and efficient [2, 3]. Extrusion methods march points from the boundaries along normal directions, producing a layer at a time of (generally) quadilateral elements with smooth outward gradation in element thickness. These methods are especially useful for producing viscous layer meshes for fluids applications [4].

Many have used the methods of Lawson [5] and of Bowyer [6] and Watson [7] for triangulating a given set of points that have already been distributed. Lawson's method is used in the proposed method for generating the final mesh after the dynamics simulation has distributed the points. When a distribution of points is not already available over a computational domain, point placement becomes a primary problem for generating a mesh. Standard point placement methods for generating triangular meshes are the Delaunay-based insertion methods, and the advancing front methods [8].

In the Delaunay-based algorithms, the boundary surfaces are initially meshed and then interior nodes are inserted sequentially. Triangles are reconnected with each node insertion to maintain the Delaunay criterion. A common strategy used for where to insert new points is at the centers of the existing elements' circumscribing circles [9, 10, 11, 12].

Advancing front algorithms march a layer (front) of elements away from each boundary. Ideal locations for new nodes are computed from the nodes of the element faces on each front. New elements are formed by joining front faces with either a recently inserted node or with an existing node where a new insertion was not necessary. When fronts intersect, elements in the intersection region must be adjusted or recreated so that the overlap is removed and the fronts are joined [13]. To control element shape and size, George [14] and Lee [15] use a background Delaunay-based mesh and metric tensors, respectively.

Hybrid methods have been proposed which take advantage of the strengths of multiple methods by applying different meshing algorithms to different regions of the computational domain [2, 3].

Physics-Based Node Placement

The Delaunay-based and advancing-front triangular mesh generation methods operate by inserting points incrementally. The physics-based methods reviewed in the next chapter as well as the proposed method begin by generating an initial distribution of points inserted at the same step and subsequently seek to smooth the distribution to obtain a desired configuration before generating the final mesh.

"Physics-based" means that the mesh or the set of points is treated to a degree as some physical entity and thus subjected to the corresponding physics equations in order to produce a desired configuration. In this research we will review some of the physics-based methods presented by others and also develop a method that gleans from these to study the usefulness of the approach. The proposed method also lays the preliminary groundwork for a method that will strive to achieve the goals listed above in future development.

Chapter Summaries

Chapter 2

The literature specifically relevant to physics-based node placement that was considered (to differing degrees) in the present research is reviewed.

Chapter 3

An overview of the particle dynamics simulation used in this study is presented along with explanations of some of the decisions made in the development process.

Chapter 4

Attention is given in greater detail to the necessary initial operations performed before the simulation begins. This includes reading and processing the given geometry and populating the domain (including the boundary curves) with an initial point distribution.

Chapter 5

The particle dynamics simulation process is presented in every aspect. This includes the computation of nodal force sums and local time step size, and the determination of a node's new location (which may involve interaction with a boundary). Also described in Chapter 5 are the process of dynamically adjusting the global maximum time step size and the inter-nodal force formula used.

Chapter 6

Experimental results are presented, demonstrating the quality of the resulting meshes compared with the meshes generated using the “Delaunay” and “Advancing Front” methods of Pointwise [1].

Chapter 7

Conclusions concerning the advantages and disadvantages of the proposed method are discussed. Goals for future work for improving the versatility and efficiency of the method as well as the quality of the resulting meshes are discussed.

Notation

The following notation will be used throughout the discussion of relevant literature as well as the description of the proposed method in order to maintain consistency and comparability. The desired spacing for node i will be denoted q_i, and the desired spacing between any pair of nodes i and j will be denoted

$$\sigma_{ij} = \frac{1}{2}\left(q_i + q_j\right) \tag{1.1}$$

The vector from node i to node j will be written as

$$\mathbf{r}_{ij} = \mathbf{x}_j - \mathbf{x}_i \tag{1.2}$$

with magnitude

$$r_{ij} = |\mathbf{r}_{ij}|$$

The pair force between two nodes, f_{ij}, will be applied to node i as a vector along the opposite direction of the vector $\mathbf{r}_{ij}$. That is,

$$\mathbf{F}_{ij} = f_{ij}\left(-\frac{\mathbf{r}_{ij}}{r_{ij}}\right) \tag{1.3}$$

And the net (resultant) force applied to node i will be denoted

$$\mathbf{F}_i = \sum_j \mathbf{F}_{ij} \tag{1.4}$$

CHAPTER 2
RELEVANT LITERATURE

Bubble Meshing

Shimada and Gossard [16, 17] presented a method for physically-based mesh generation by packing spheres on boundaries and interiors and smoothing with inter-bubble forces. They note, “the close packing of bubbles mimics a Voronoi diagram pattern, corresponding to well-shaped Delaunay triangles and tetrahedra” [17]. The force they use is a cubic interpolant constructed to behave somewhat like the van der Waals force but with further constraints. The variable for this force is the ratio (using our nomenclature)

$$w_{ij} = \frac{r_{ij}}{\sigma_{ij}} \tag{2.1}$$

where r_{ij} is the distance between nodes i and j (at the centers of the ith and jth bubbles, respectively), and

$$\sigma_{ij} = \frac{q_i}{2} + \frac{q_j}{2} \tag{2.2}$$

is the desired distance between them. This distance is the sum of the radii of bubbles i and j, and it is therefore the distance at which the two bubbles are “kissing” [17].

They enforce the following conditions for the interpolant:

$$\begin{cases} f'(0) = 0 \\ f(1) = 0 \\ f'(1) = -k_0 \\ f(1.5) = 0 \end{cases}$$

where k_0 represents the linear spring constant. The last condition shows that they take into account only nodes within $1.5\sigma_{ij}$ of node i when calculating node i's force sum. Thus their force calculation takes the form (using w_{ij} as defined in (2.1)):

$$f_{ij} = \begin{cases} k_0\left(1.25w_{ij}^3 - 2.375w_{ij}^2 + 1.125\right) & , \quad 0 \leq w_{ij} \leq 1.5 \\ 0 & , \quad 1.5 < w_{ij} \end{cases} \tag{2.3}$$

Therefore, when two nodes are closer than the equilibrium spacing (that is, $r_{ij} < \sigma_{ij}$ making $w_{ij} < 1$), there is a repulsive force between them, and when they are farther than the equilibrium spacing there is an attractive force. This force interpolant is shown in Figure 2.1 with σ set to 1. The idea of a spring between each node pair makes a bit more sense, since bubbles do not actually attract one another.

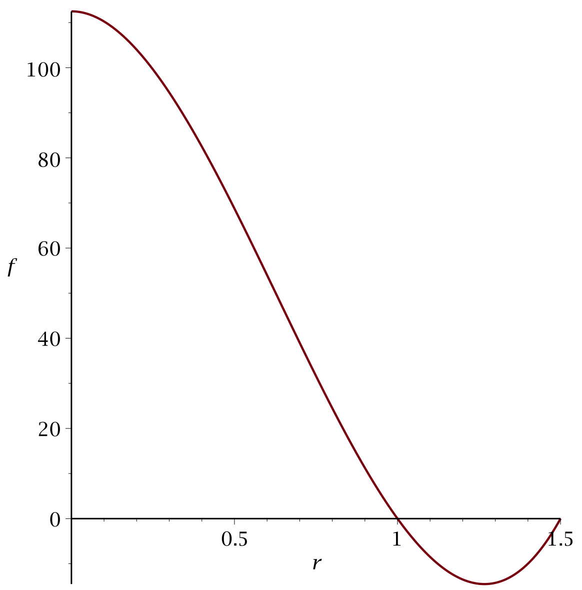


Figure 2.1 Bubble meshing force interpolant with $k_0 = 75$ and $\sigma = 1$

The bubble simulation also includes a damping force, $-c_i v_i$, on each node in order to ensure convergence to a stable configuration. They then use a numerical iterative solver to solve the applicable differential equations of motion, which can be written per node for their configuration as

$$m_i \mathbf{a}_i = \mathbf{F}_i - c_i \mathbf{v}_i \tag{2.4}$$

$$m_i \frac{d^2 \mathbf{x}_i}{dt^2} + c_i \frac{d\mathbf{x}_i}{dt} = \mathbf{F}_i \tag{2.5}$$

where $\mathbf{x}_i$ is the spatial position of the ith bubble and $\mathbf{F}_i = \sum_j f_{ij}\left(-\hat{\mathbf{r}}_{ij}\right)$.

They also use adaptive bubble population adjustment based on an overlap ratio for each node. Since the inter-bubble force function includes attraction as well as repulsion, then if there are too few nodes, there will be gaps in the resulting configuration, and if there are too many nodes, there will be areas in the domain in which nodes are significantly closer to other nodes than the equilibrium distance. This problem is overcome by adaptively controlling the bubble population, removing or adding bubbles based on an overlapping ratio defined as

$$\alpha_i = \frac{1}{q_i} \sum_j \left(2q_i + q_j - 2r_{ij}\right) \tag{2.6}$$

Nie, Zhang, Liu, and Wang [18] further researched the same bubble meshing method and proved its “convergence” to a stable configuration. Specifically, they proved that the average speed of the bubbles during the dynamic simulation tends to zero.

Monte Carlo Simulation

Zhang and Smirnov [19] proposed a physically-based mesh generation scheme using a Monte Carlo simulation to minimize the system’s total potential energy which they define as

$$U = \sum_i \sum_{j>i} \phi\left(r_{ij}\right) \tag{2.7}$$

where $\phi(r_{ij})$ is the Lennard-Jones pair potential between nodes i and j:

$$\phi_{ij} = \phi(r_{ij}) = 4a\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right] \tag{2.8}$$

Here again, r_{ij} is the actual distance between the two nodes and σ_{ij} is the distance between the pair of nodes at which the pair potential will be zero. In their research, like in the bubble meshing study, σ_{ij} is taken to be the arithmetic average of the spacing parameters of each node. That is,

$$\sigma_{ij} = \frac{1}{2}(q_i + q_j) \tag{2.9}$$

It can clearly be seen that this is the same as using the sum of the radii of the ith and jth bubbles, as in Equation (2.2), for the pair's desired spacing.

As noted by the authors, the equilibrium spacing for a given node pair is actually

$$\sigma_{0,ij} = 2^{1/6}\sigma_{ij} \approx 1.1225\sigma_{ij}$$

instead of σ_{ij}. This can clearly be seen when one differentiates the potential with respect to the distance r_{ij} to find the force acting between two nodes:

$$f_{ij} = -\frac{d\phi_{ij}}{dr_{ij}} = 4a\left[\frac{12\sigma_{ij}^{12}}{r_{ij}^{13}} - \frac{6\sigma_{ij}^{6}}{r_{ij}^{7}}\right] \tag{2.10}$$

which simplifies to

$$f_{ij} = 48a\left(\frac{\sigma_{ij}^{6}}{r_{ij}^{7}}\right)\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6} - \frac{1}{2}\right] \tag{2.11}$$

$$f_{ij} = 24a\left(\frac{\sigma_{ij}^{6}}{r_{ij}^{7}}\right)\left[\frac{2\sigma_{ij}^{6}}{r_{ij}^{6}} - 1\right] \tag{2.12}$$

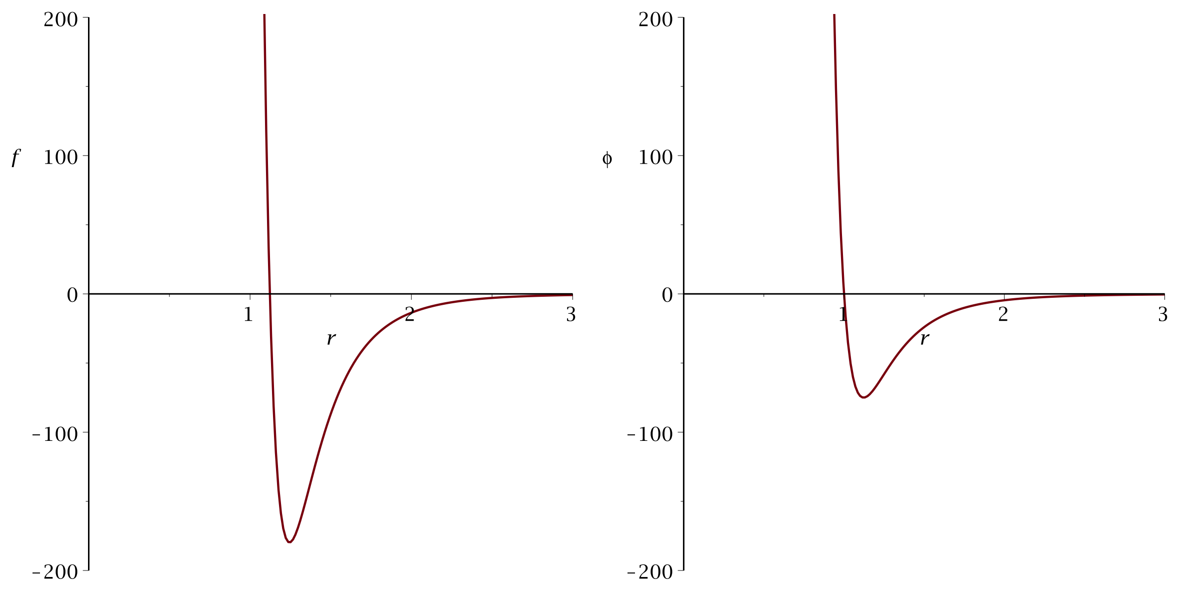


Figure 2.2 Lennard-Jones force (*left*) and pair potential (*right*) with $a = 75$ and $\sigma = 1$

As can be seen in Figure 2.2, this formula for the force is zero when the ratio $r_{ij}/\sigma_{ij} = 2^{1/6} \approx 1.1225$, whereas the potential is zero when this ratio is one. Of course, in their study, the formula for the force between the nodes is not used because they use a Monte Carlo simulation instead of a particle dynamics simulation.

During their simulation, a node is moved in a random direction and tested to see whether the node's potential energy sum has decreased or not. The test is based on the Boltzmann Distribution law and takes the form

$$\begin{aligned} &\text{accept if } e^{-\beta \Delta \phi_i} > R \\ &\text{reject if } e^{-\beta \Delta \phi_i} \leq R \end{aligned} \tag{2.13}$$

where

$$\Delta \phi_i = \sum_j \phi\left(r_{ij}^{n+1}\right) - \sum_j \phi\left(r_{ij}^{n}\right) \tag{2.14}$$

is the change in potential energy of node i from state n to state $n+1$, $R \in (0, 1)$ is a random number generated at each trial move, and $\beta = 1/kT$, where k is the Boltzmann constant and T is the temperature. In their case, T is not a real temperature and is selected such that the acceptable range of energy increase is reasonable for the system.

Here also, as in the bubble meshing study, the authors use adaptive node population control, but based on the total system potential energy. When the potential energy is negative, there are gaps in the mesh region and nodes should be added. When the potential energy is significantly greater than zero, the packing is too dense and nodes should be removed.

Molecular Dynamics Simulation

Zheleznyakova and Surzhikov [20] offered a physically-based method of mesh generation by molecular dynamics simulation using Coulomb's law as the particle interaction force. Every node is given a positive charge, and the force acting between a pair of nodes is given by

$$f_{ij} = C\frac{q_i q_j}{r_{ij}^k} \tag{2.15}$$

where $k \geq 2$ and C is a constant. The pair potential corresponding to this force is

$$\phi_{ij} = C\frac{q_i q_j}{(k-1)\, r_{ij}^{k-1}} \tag{2.16}$$

which is never negative but asymptotically approaches zero as r_{ij} grows. These functions are plotted in Figure 2.3.

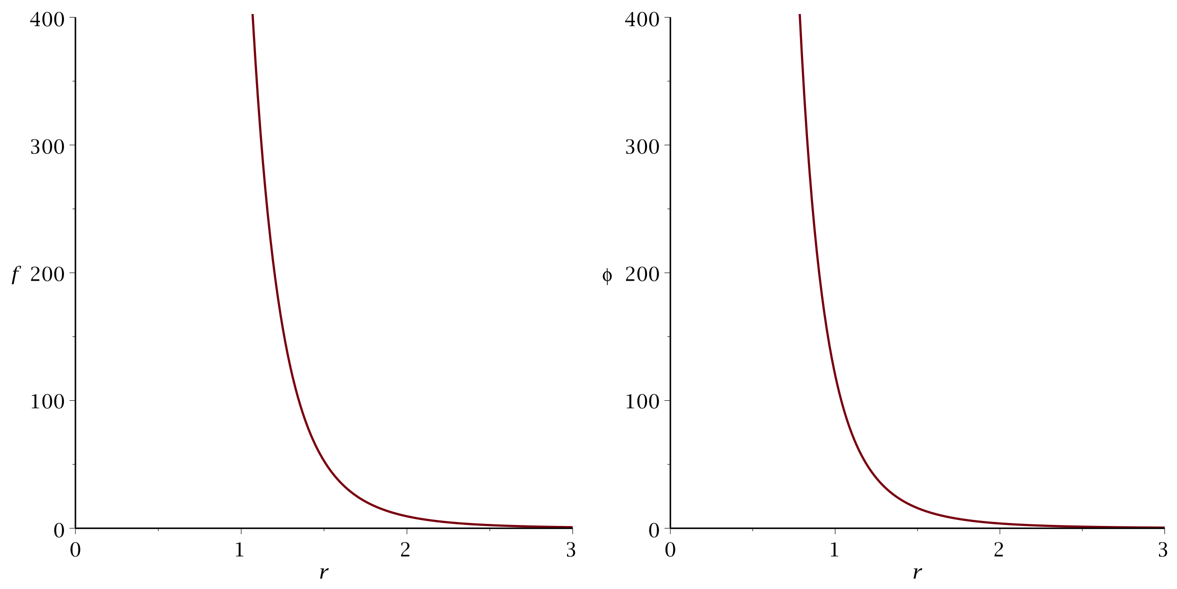


Figure 2.3 Coulomb force (*left*) and potential (*right*) with $C = 600$, $k = 6$, and $q_i = q_j = 1$

This force is only repulsive. There is no attraction between nodes. In order to accomplish proper spacing near boundaries, they add an attractive force in the interaction between boundary nodes and mobile particles in the meshing region. Thus when interior node i is interacting with a node on a wall (denoted wj) the total interaction force is described by

$$f_{iwj} = C_R \frac{q_i q_{wj}}{r_{iwj}^k} - C_A \frac{q_i q_{wj}}{r_{iwj}^m} \tag{2.17}$$

with $m < k$. The corresponding pair potential then for interaction with boundary nodes is

$$\phi_{iwj} = C_R \frac{q_i q_{wj}}{(k-1)\, r_{iwj}^{k-1}} - C_A \frac{q_i q_{wj}}{(m-1)\, r_{iwj}^{m-1}} \tag{2.18}$$

These modified functions are plotted in Figure 2.4.

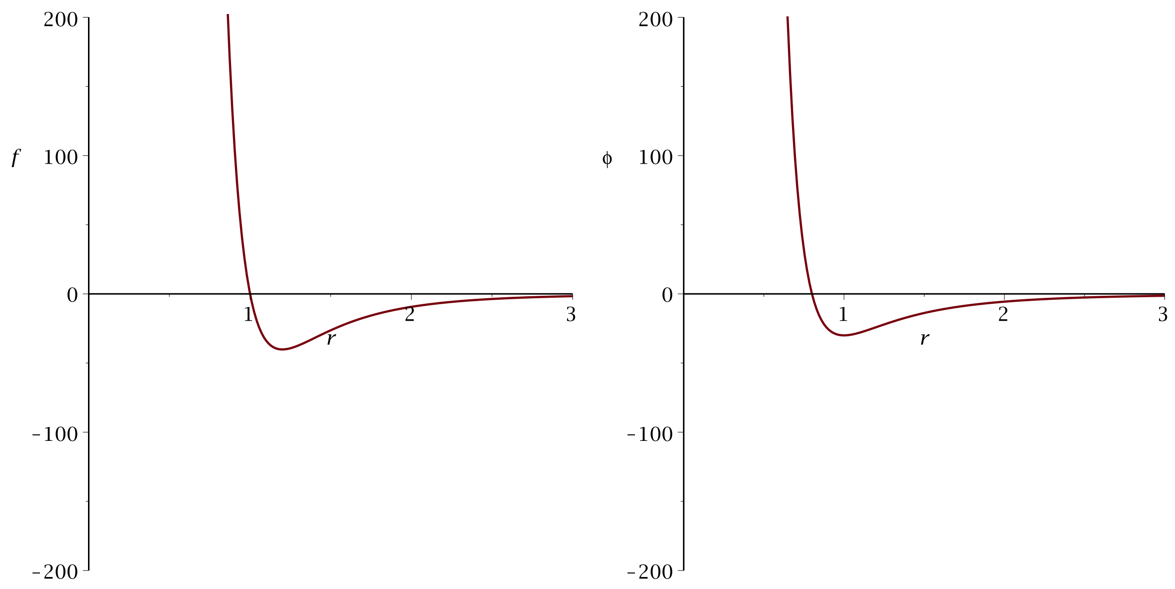


Figure 2.4 Modified Coulomb force (*left*) and potential (*right*) with $C_R = C_A = 600$, $k = 6$, $m = 5$ and $q_i = q_j = 1$

Thus, a mobile node is repulsed when it is within a desired distance from the boundary node, and it is attracted when it is farther away. In this method, as in the bubble mesh method, a drag force is applied to each node,

$$f_d = -Kv_i^p$$

where $p \geq 2$ and K is a constant. The forces are summed for each node and the equations of motion are numerically integrated to simulate the behavior of the nodes at each time interval as in equation (2.5).

Truss Equilibrium Method

The physically-based meshing method presented by Persson and Strang [21] is interesting as well. Their method re-triangulates the point system at every iteration and uses the connectivity as truss bars between nodes. A linear spring force function is used (strictly repulsive, see Figure 2.5)

$$f_{ij} = \begin{cases} k\left(\sigma_{ij} - r_{ij}\right) & , \ 0 \leq r_{ij} \leq \sigma_{ij} \\ 0 & , \ \sigma_{ij} < r_{ij} \end{cases} \tag{2.19}$$

to move the nodes toward force equilibrium. The nodes are re-triangulated in their new locations at every iteration to maintain the Delaunay properties. Thus at the beginning of each iteration there is a different set of truss edges from which internodal forces are computed.

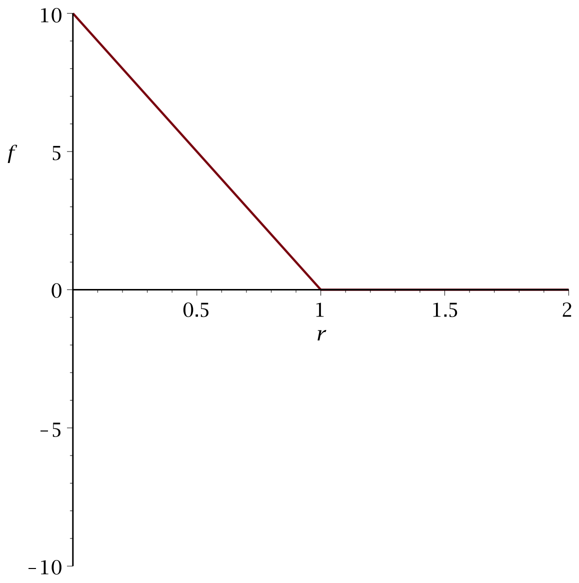


Figure 2.5 Linear spring force with $k = 10$ and $\sigma = 1$

Holm, Kaufmann, Heimsund, Øian, and Espedal [22] extend the algorithm of Persson and Strang to handle domains with complex geometries including internal boundaries.

CHAPTER 3
METHOD OVERVIEW

The proposed method treats each node as a particle, similar to what is done in the Monte Carlo simulation and the molecular dynamics simulation methods. We use a particle dynamics simulation rather than a Monte Carlo simulation to drive the nodes to force equilibrium. The bubble meshing method and the molecular dynamics method use a dynamics simulation approach very similar to one another. Differing significantly only in the choice of inter-nodal force formula used, they both sum the forces acting between node i and its surrounding neighbors, add a drag force dependent on node velocity, and numerically integrate the equations of motion to determine the movement of the nodes. We have chosen to simplify the simulation. For convenience, we rewrite equation (2.5) here

$$m_i \frac{d^2\mathbf{x}_i}{dt^2} + c_i \frac{d\mathbf{x}_i}{dt} = \mathbf{F}_i \tag{3.1}$$

First, we include no damping force. Instead, each node's velocity is reset to zero at the beginning of each iteration. Secondly, we do not numerically integrate the dynamics equations. Instead we take acceleration for each node to be constant over an iteration and apply the resulting motion formula directly, limiting the distance a node may travel each step. Thus (if we also take $m_i = 1 \ \forall i$), equation (3.1) above can be written as a scalar equation

$$\frac{d^2x_i}{dt^2} = F_i \tag{3.2}$$

in the direction of $\mathbf{F}_i$. This can be solved directly (taking F_i to be constant over the time interval) by integrating both sides of

$$d^2 x_i = F_i dt^2 \tag{3.3}$$

to obtain

$$\Delta x_i = \Delta t_i v_{i,0} + \frac{1}{2}\Delta t_i^2 F_i \tag{3.4}$$

But, as stated above, we set $v_{i,0} = 0$ for each iteration, resulting in

$$\Delta x_i = \frac{\Delta t_i^2}{2} F_i \tag{3.5}$$

The subscript on Δt_i indicates that a different time step size is used for each node. This value is globally bounded so that no node may travel more than 1/3 the distance to its nearest neighbor in the general direction of its resultant force.

Each node has a spacing value, q_i, associated with it which defines the desired distance from node i to each neighboring node. This parameter is computed from node i's physical location within the domain spacing field. The spacing field can be arbitrary, but for this study we have used a simple inverse distance weighting function. Given a 2D geometry defined by a set of segmented curves, the segment of each curve to which the node is closest contributes to the calculation of that node's spacing. The distance, d_{is}, from node i to each of these segments and the length of each segment, l_s, are used to calculate the spacing parameter using the formula

$$q_i = \frac{\sum_s \frac{l_s}{d_{is}}}{\sum_s \frac{1}{d_{is}}} \tag{3.6}$$

This spacing parameter defines the desired distance node i should be away from its nearest neighboring nodes. Note, this is the same as saying that the spacing parameter is the diameter of node i's circular bubble. That is, the spacing parameter is applied equally in all directions.

To determine the net force vector applied to node i the forces acting on it from each of its surrounding nodes are summed

$$\mathbf{F}_i = \sum_j \mathbf{F}_{ij} \tag{3.7}$$

The formula used in this study for the pair force magnitude between node i and each surrounding node j is the formula for the Lennard-Jones pair potential scaled by the pair spacing, σ_{ij}:

$$f_{ij} = 4a\sigma_{ij}\left(\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right) \tag{3.8}$$

Here, r_{ij} is the distance from node i to node j and σ_{ij} is the average of the two nodes' spacing parameters. This force is applied to node i along the negative of the vector from node i to node j:

$$\mathbf{F}_{ij} = f_{ij}\left(-\frac{\mathbf{r}_{ij}}{r_{ij}}\right) \tag{3.9}$$

The Lennard-Jones pair potential is used rather than the corresponding force formula because the force formula evaluates to zero at an equilibrium distance of $r_{ij} \approx 1.1225\sigma_{ij}$, while what we require is for σ_{ij} itself to be the distance at which force equilibrium is reached between the pair of nodes.

At the end of each iteration, once the new location of a node is determined, a new spacing parameter is computed for it based on its new physical location. The flowchart in Figure 3.1 outlines the major aspects of the proposed method which will be further explained in the succeeding chapters.

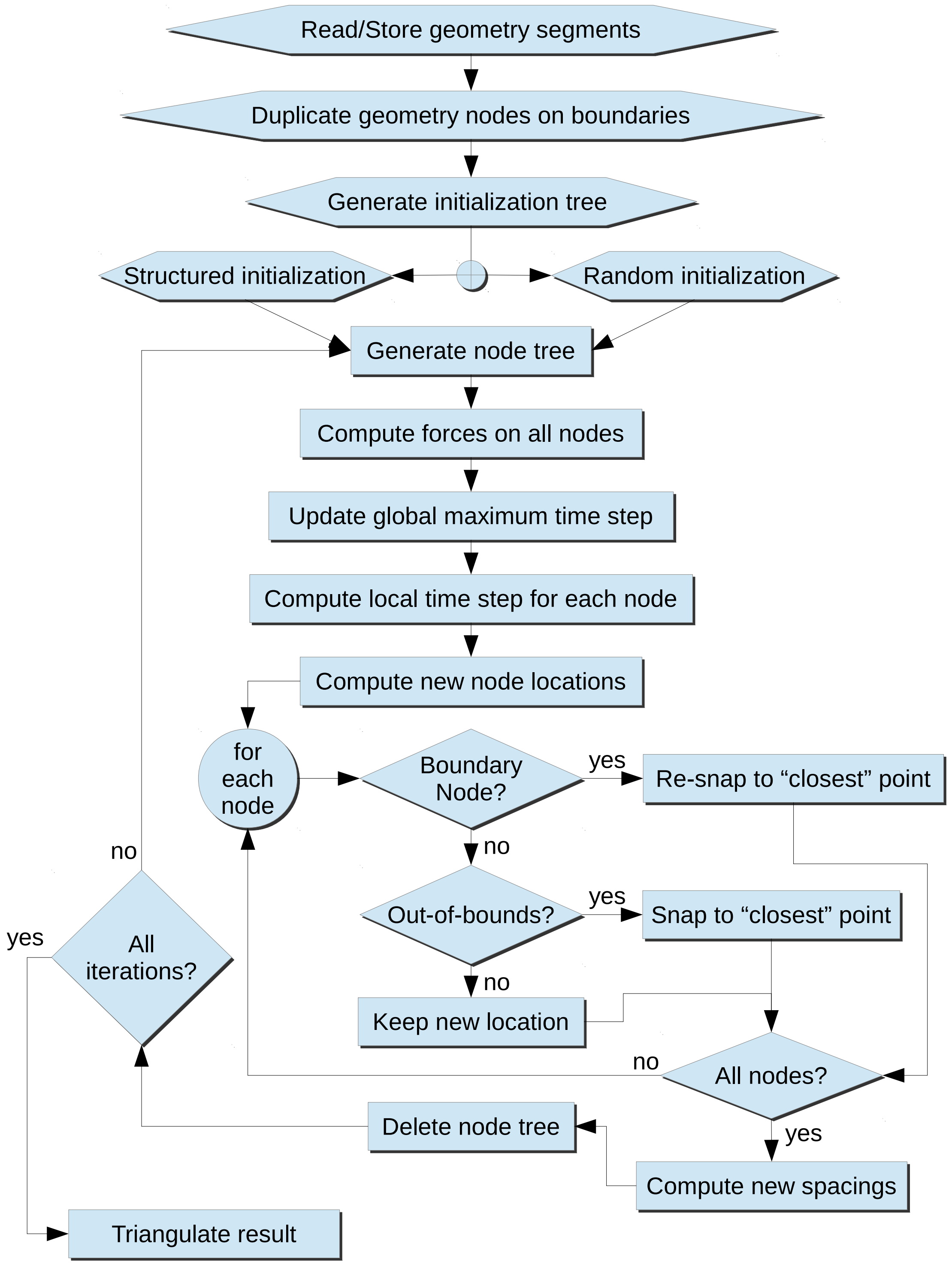

Read/Store geometry segments
Duplicate geometry nodes on boundaries
Generate initialization tree
Structured initialization
Random initialization
Generate node tree
Compute forces on all nodes
Update global maximum time step
Compute local time step for each node
Compute new node locations
for each node
Boundary Node?
yes
Re-snap to “closest” point
no
Out-of-bounds?
yes
Snap to “closest” point
no
Keep new location
no
All nodes?
yes
Compute new spacings
Delete node tree
no
All iterations?
yes
Triangulate result

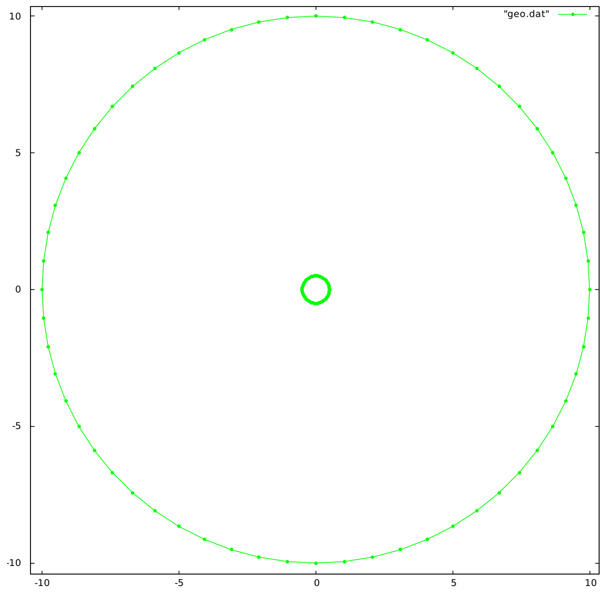
Figure 3.1 Flowchart of the major steps of the method

CHAPTER 4
INITIAL OPERATIONS

Geometry Processing

The proposed method takes as input file(s) containing segmented curves which define the 2D geometry. These geometry curves have to have been predefined and the points on them pre-populated and spaced. This is important because the proposed method currently uses a spacing field that is completely dependent on the geometry spacing.

After reading the file(s) and storing the geometry segments and node locations in memory, the geometry is analyzed and various data computed, such as segment vectors, body arc lengths, domain extents, and total interior domain area. Also determined at this step are the locations of the critical points–that is, points where (in 2D) two geometry curves join. Finally, the geometry segments are stored in a quad-tree data structure (referred to hereafter as the geometry tree) for efficient searching in other parts of the code. Examples of this geometry tree can be seen in Figures 4.1 through 4.3.



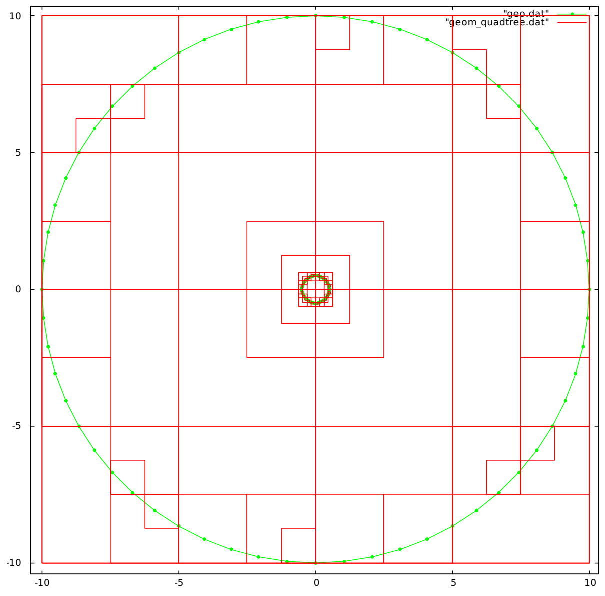
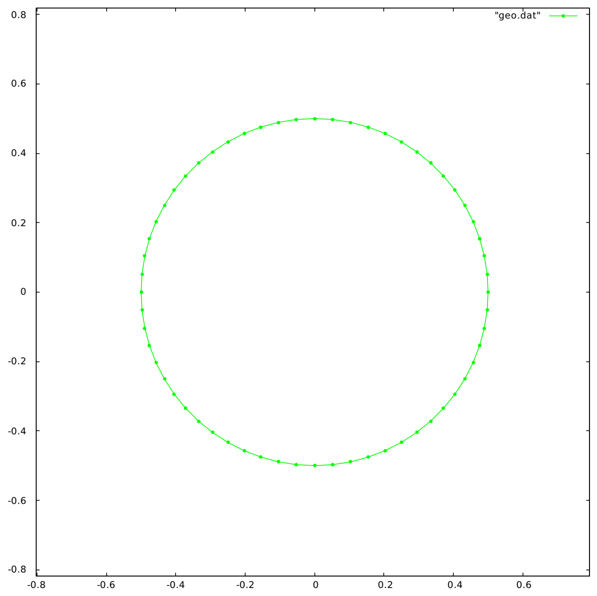


Figure 4.1 Circle within circle: Segmented geometry curves (*left*); Segmented geometry curves with geometry tree (*right*)



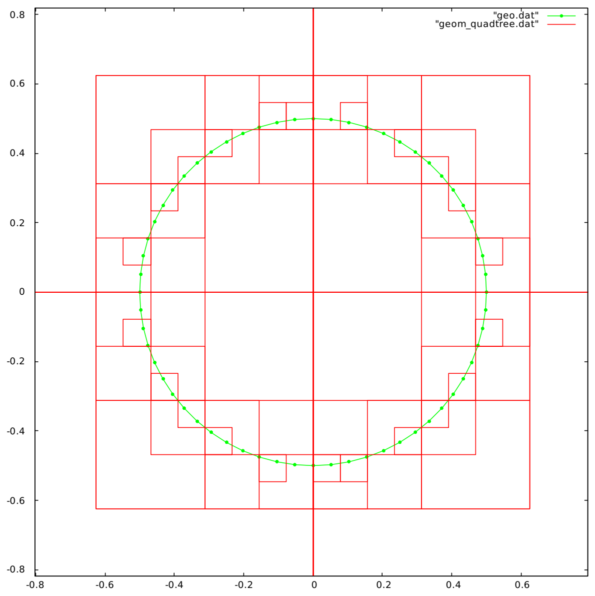


Figure 4.2 Circle within circle (zoomed to inner circle): Segmented geometry curves (*left*); Segmented geometry curves with geometry tree (*right*)

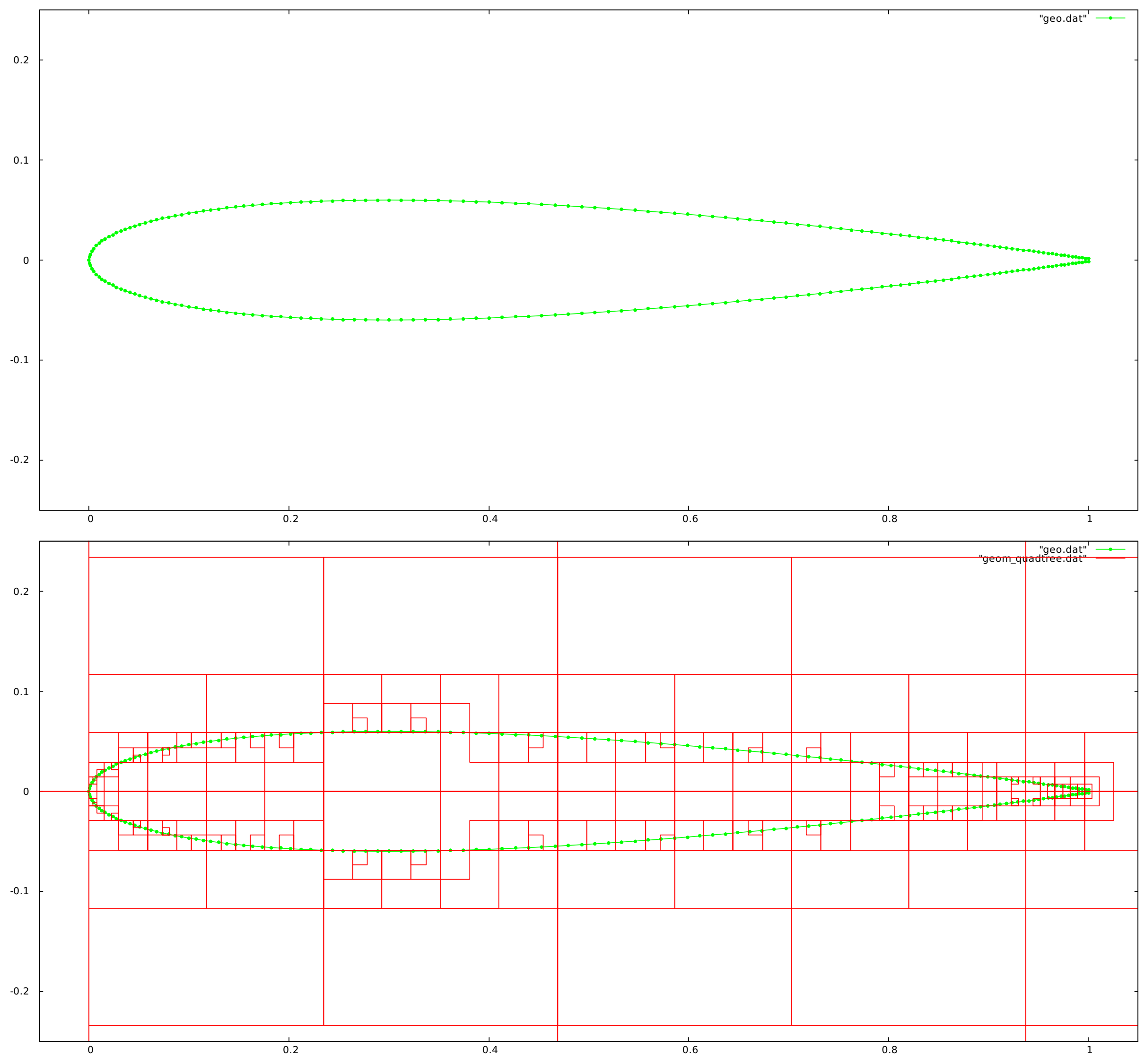


Figure 4.3 NACA 0012 airfoil (zoomed): Segmented geometry curves (*top*); Segmented geometry curves with geometry tree (*bottom*)

Initial Node Distribution

Boundary Nodes

The first mesh nodes to be placed are at the locations of the critical points of the geometry. These points will not be allowed to move throughout the simulation. Second, all the geometry nodes between the critical points are duplicated as mesh nodes. These points will be allowed to move along the boundaries. This is important since interior points will be adhered to the boundaries if/when they are forced outside of the domain. Thus all points on the boundaries (whether initially placed or added later) between the critical points will need to adjust their locations toward a state of force equilibrium along the local direction of the curve. Figure 4.4 shows an example boundary distribution in which the geometry nodes have been duplicated.

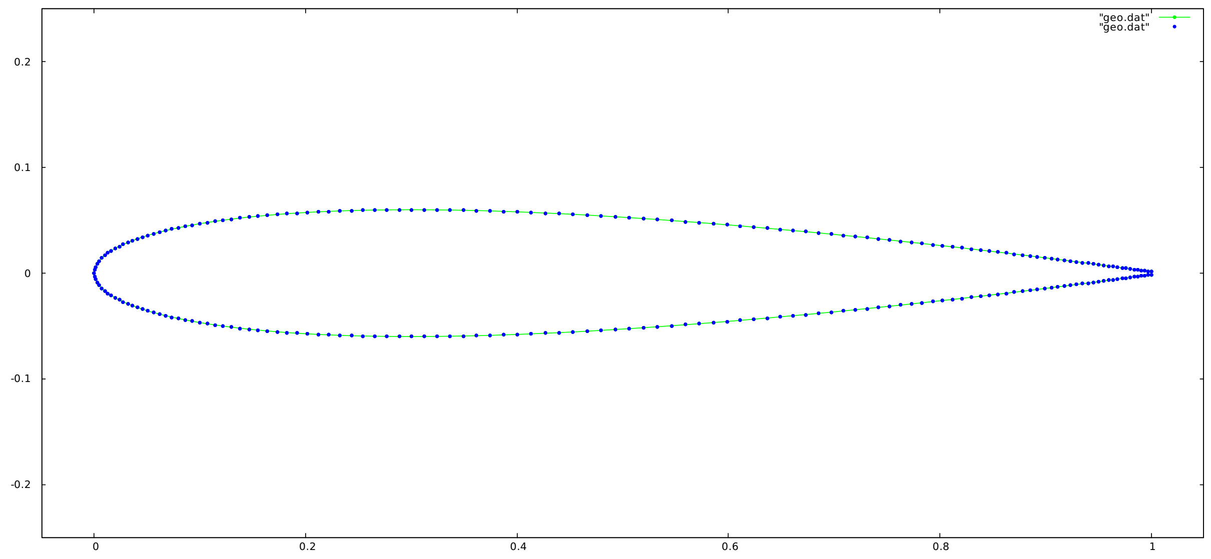
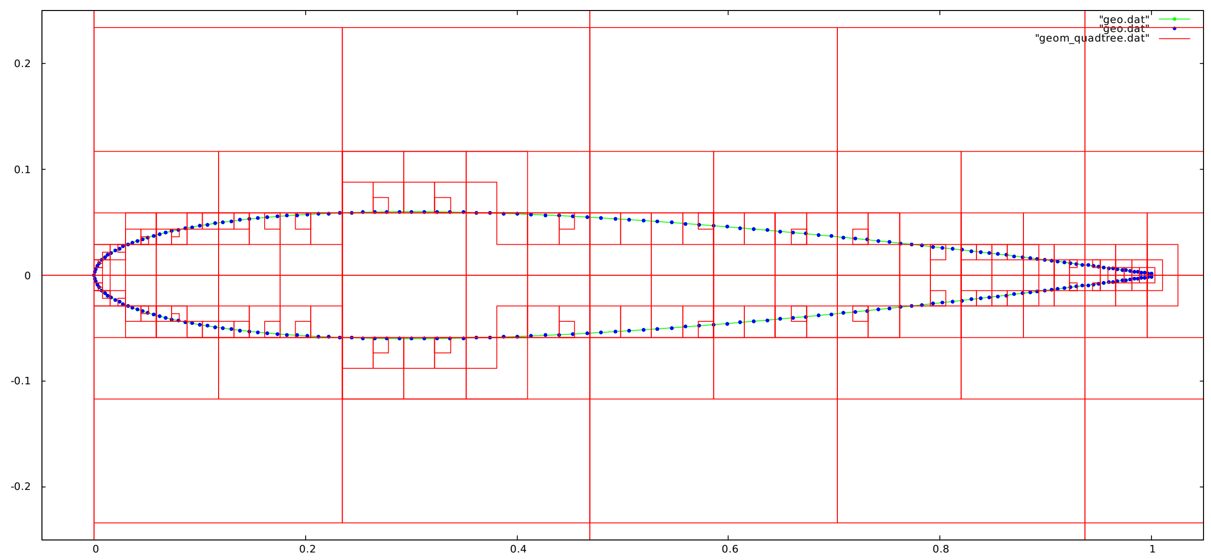


Figure 4.4 NACA 0012 airfoil (zoomed): Segmented geometry curves and initial boundary mesh points

Interior Nodes

A completely random initial distribution of interior nodes is inappropriate, especially when the spacing field is non-constant, since it will require far more iterations to pack points into areas of small spacing and spread points thin in areas of larger spacing. We need the initialization process to take the spacing field into account, not only so that points are clustered where they need to be clustered, but also so that the number of points used to fill the domain will be as close as possible to optimal. The following outlines the initialization method we have used.

In order to distribute nodes in the interior of the domain, another quad tree is built, again based on the geometry segments. This quad tree we will call the initialization tree. Whereas the geometry tree only stores the geometry segments efficiently, the initialization tree is processed recursively and filled out to provide a somewhat smoother gradation in the size of the quad elements. Also, as exemplified in the Figure 4.5 (*bottom*), elements of the initialization tree that are fully outside the domain are ignored for node initialization purposes. Figure 4.5 provides a comparative visualization of the geometry tree and the initialization tree for the same NACA 0012 airfoil as in Figure 4.4.



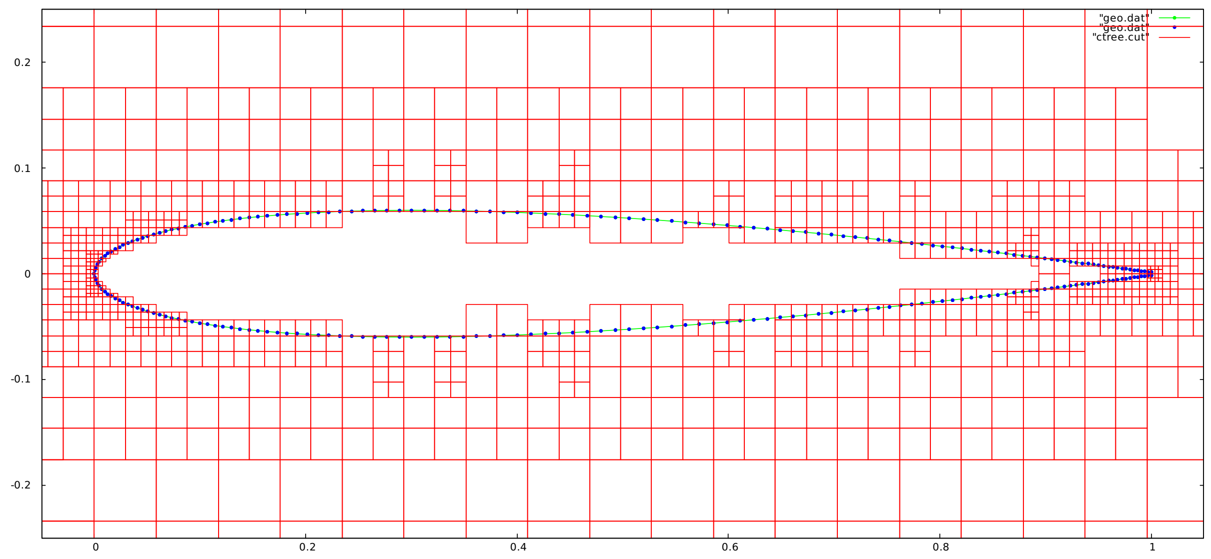


Figure 4.5 NACA 0012 airfoil (zoomed): Geometry tree (*top*) versus initialization tree (*bottom*)

Once the initialization tree is built, each quad element is visited and populated with one or more nodes. This process in each quad is also recursive. The spacing at the center location of the quad, q_{mid}, is computed. Then the ratio of the area of the quad to the area of the circle with diameter q_{mid} is computed. This ratio tells approximately how many of those circles will fit within the current quad and is thus used to determine how many points with which to populate the quad. The quad is populated with either a structured or random initial distribution, chosen ahead of time by the user.

Structured Quad Distribution Currently there are four possibilities for populating the quad with a structured distribution: 1) one node at the center of the quad; 2) four nodes interior to the quad, equally spaced as a square; 3) five nodes, four as a square and one at the center; and 4) nine nodes interior to the quad, equally spaced as a 3×3 square. Note, the use of the term “structured” here is referring to the fact that nodes are placed in a geometric pattern and is distinct from its use when defining “structured grids” as done in Chapter 1.

Random Quad Distribution The random populating option will insert the number of nodes corresponding to the area ratio, from one up to nine, using a pseudo-random coordinate pair within the quad.

Every node placed is tested to see whether it is outside the domain or too close to a boundary, and if it is, it is not kept. Recursion happens in either initialization process (structured or random) if the area ratio is greater than 9.5. In this case the quad is split into four sub-quads and each of these populated individually.

This process results in an initial distribution of points in which the area of the domain is filled appropriately according to the spacing field and all points inserted are only in the interior. The particle dynamics simulation will therefore be a means of smoothing this distribution toward a locally isotropic configuration, which if/when triangulated produces near equilateral triangles.

The user may specify an exact desired total number of points to insert that is greater than what the above process will use. If the user indicates that all of these points should be used, the current code will insert the remainder of the number given as random points within the domain. This is generally not recommended as it tends to “overload” the domain, but this is desired and useful in certain applications.

Result of Structured Initial Quad Distribution

Example initial point distributions, using the structured initialization procedure, are displayed (with and without the initialization tree) in Figures 4.6 through 4.9.

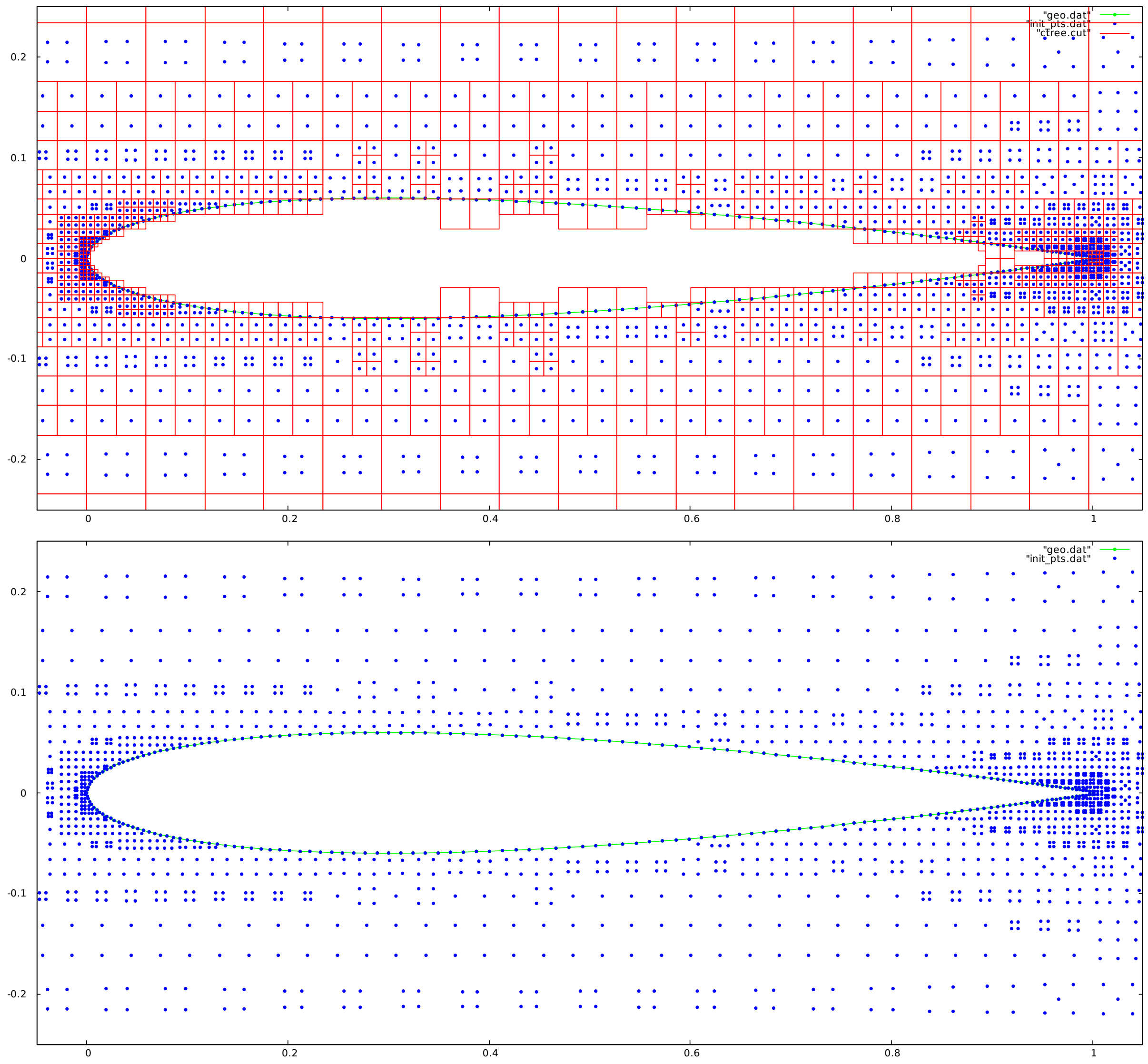


Figure 4.6 NACA 0012 airfoil (zoomed): Structured initial point distribution with (*top*) and without (*bottom*) initialization tree

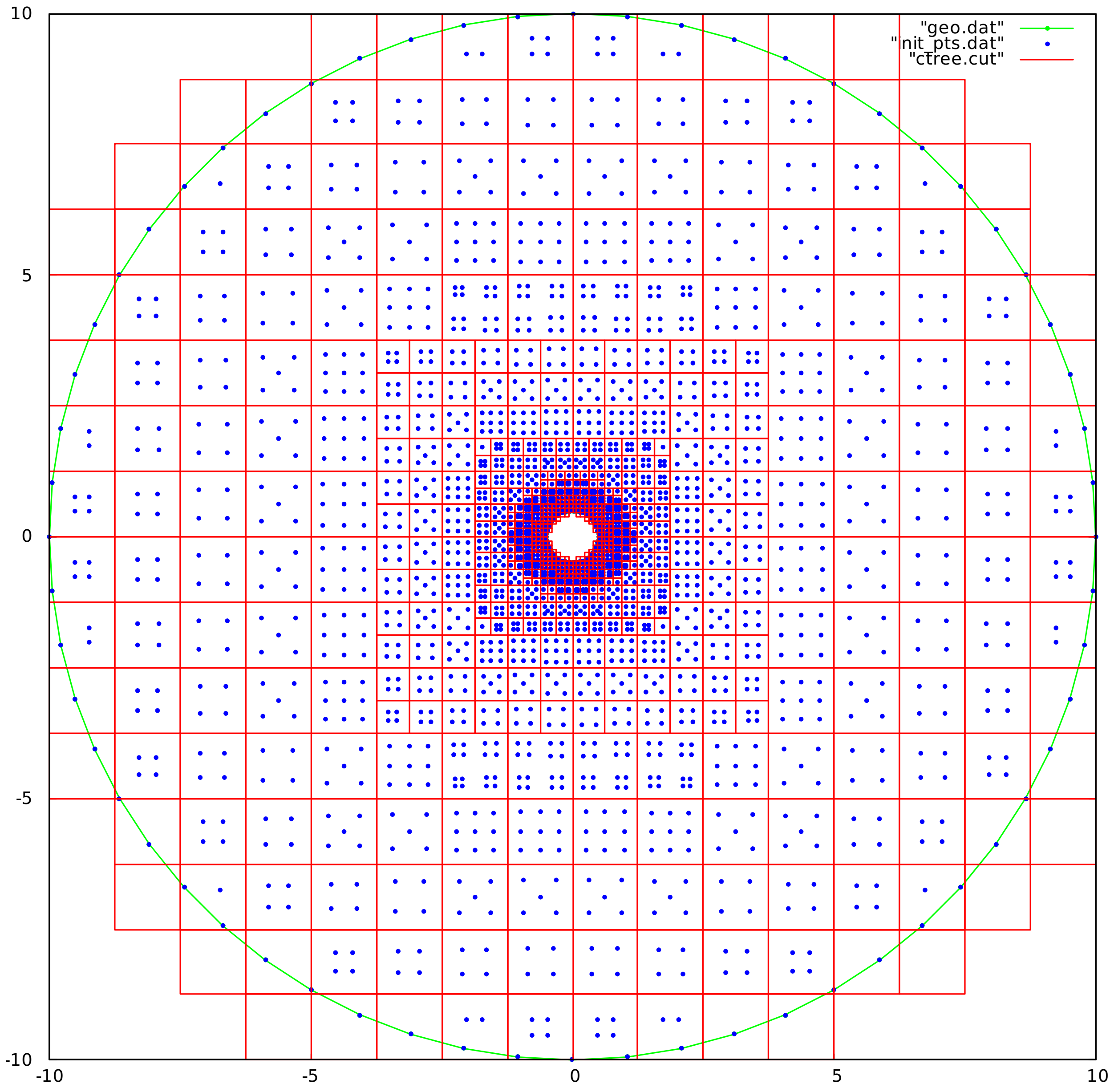


Figure 4.7 Circle within circle: Structured initial point distribution with initialization tree

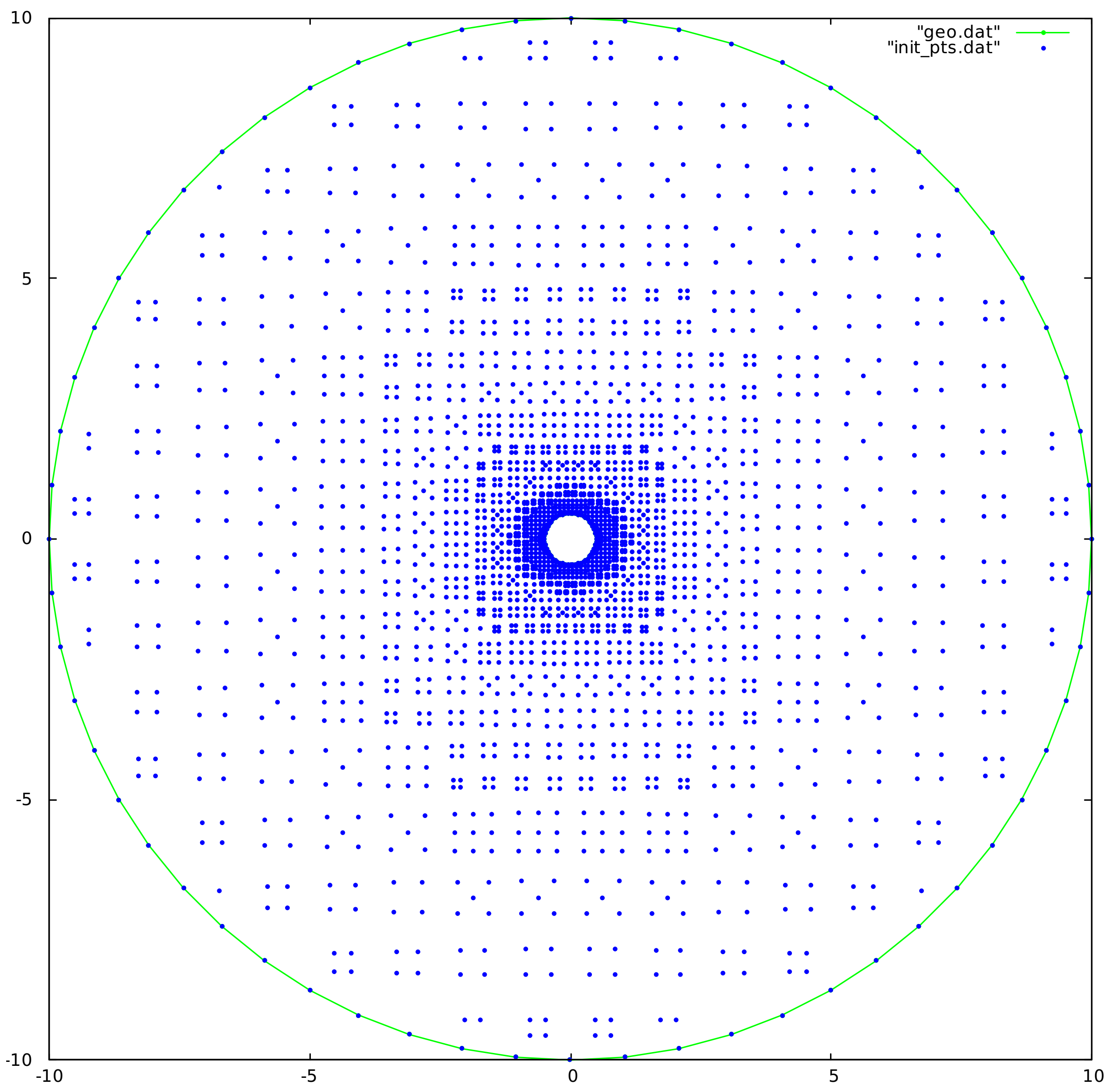


Figure 4.8 Circle within circle: Structured initial point distribution

In Figures 4.6 (*top*), 4.7, and 4.9, we can see several examples of the cases outlined above for structured initialization. We notice first the quad elements through which geometry segments "cut". In some of these elements nodes that were placed outside the domain (or too close to the boundary segment) have been removed. Second, we note the two elements in the upper left of Figure 4.9. These elements were further subdivided during the initialization process due to the ratio of the area of the representative bubble (determined using the spacing at the center of the element) to the area of the quad being greater than 9.5.

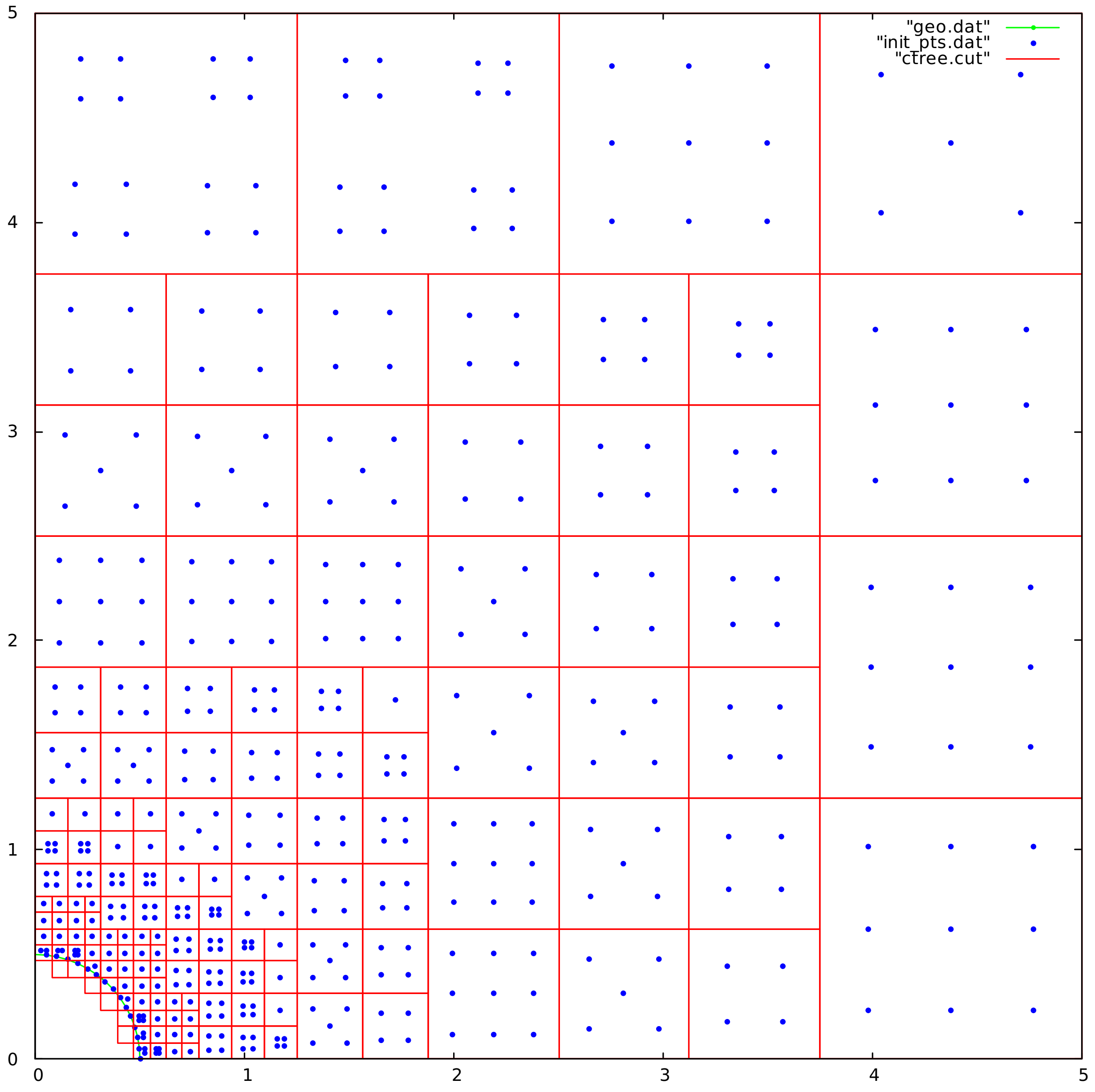
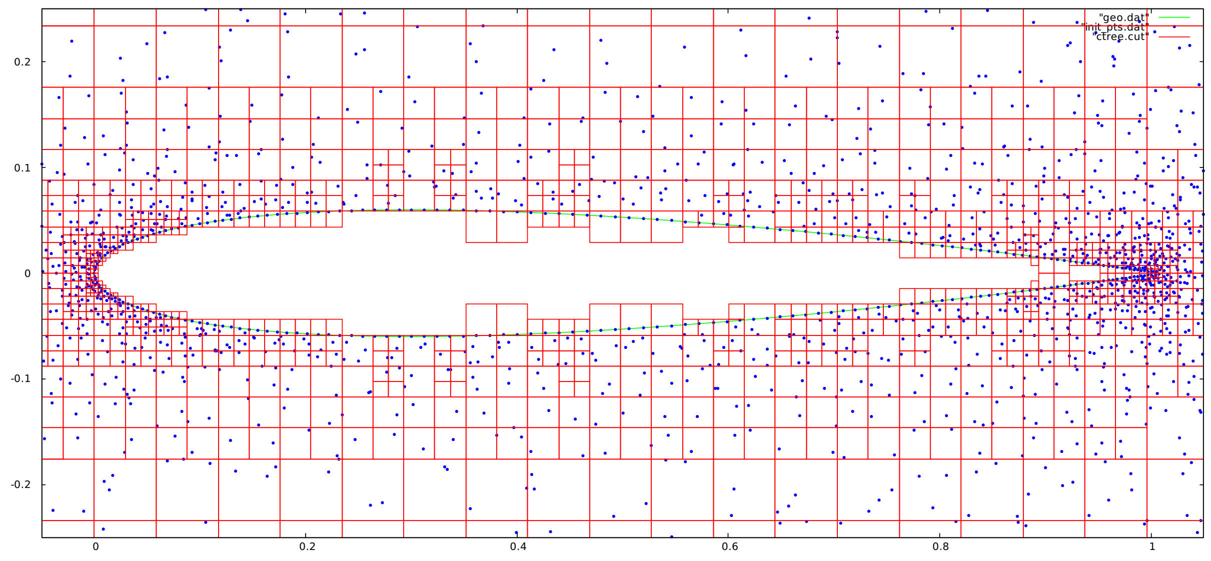


Figure 4.9 Circle within circle (zoomed to a portion of the upper right quadrant): Structured initial point distribution with initialization tree

Example Result of Random Initial Quad Distribution

Example initial point distributions, using the random initialization procedure, are displayed (with and without the initialization tree, with the same views as in the previous section) in Figures 4.10 through 4.13.



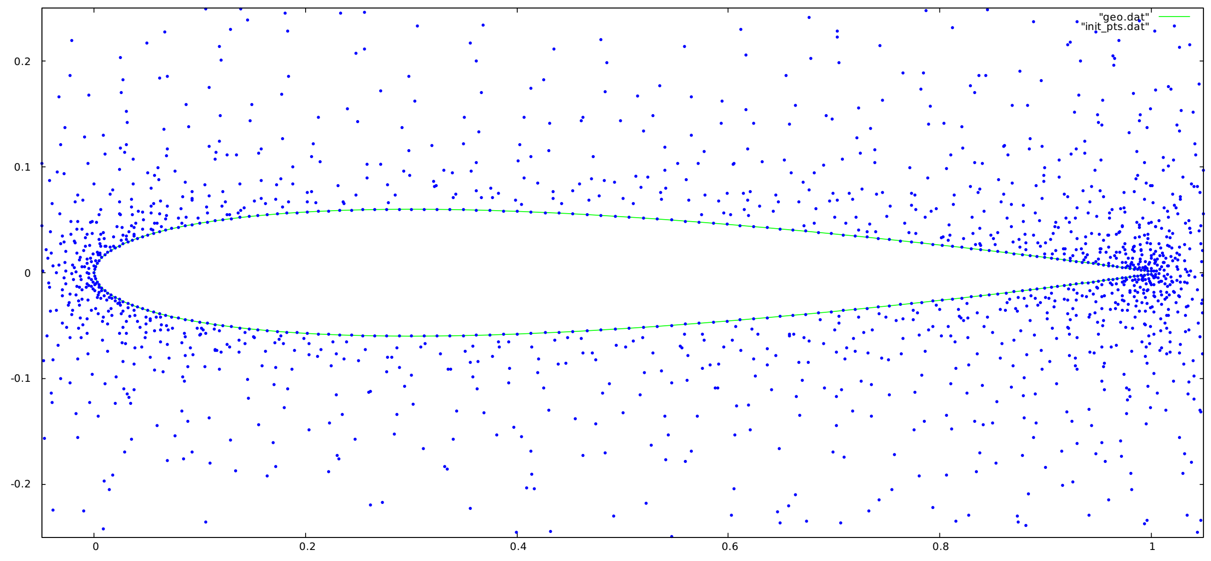


Figure 4.10 NACA 0012 airfoil (zoomed): Random initial point distribution with (*top*) and without (*bottom*) initialization tree

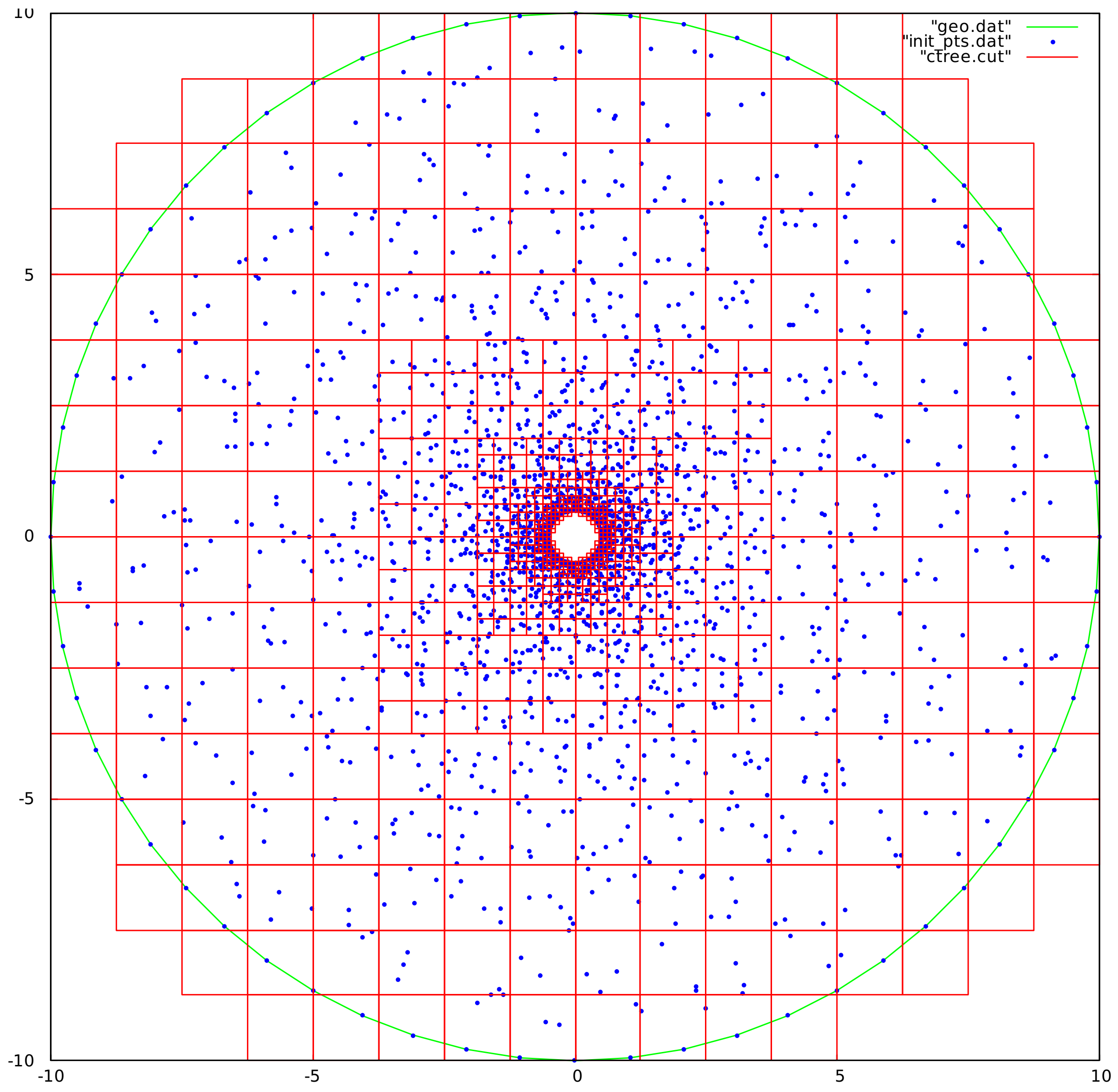


Figure 4.11 Circle within circle: Random initial point distribution with initialization tree

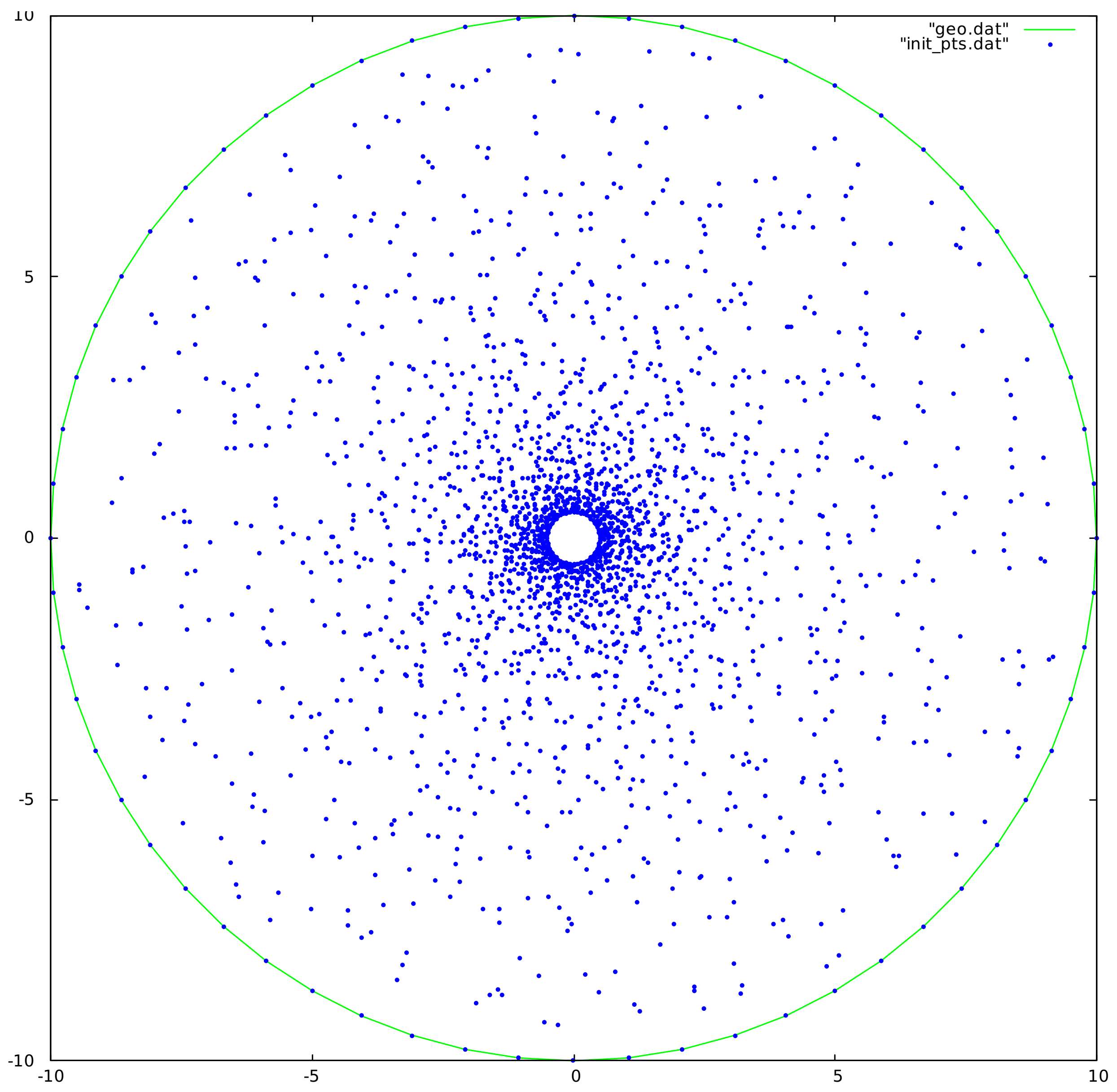


Figure 4.12 Circle within circle: Random initial point distribution

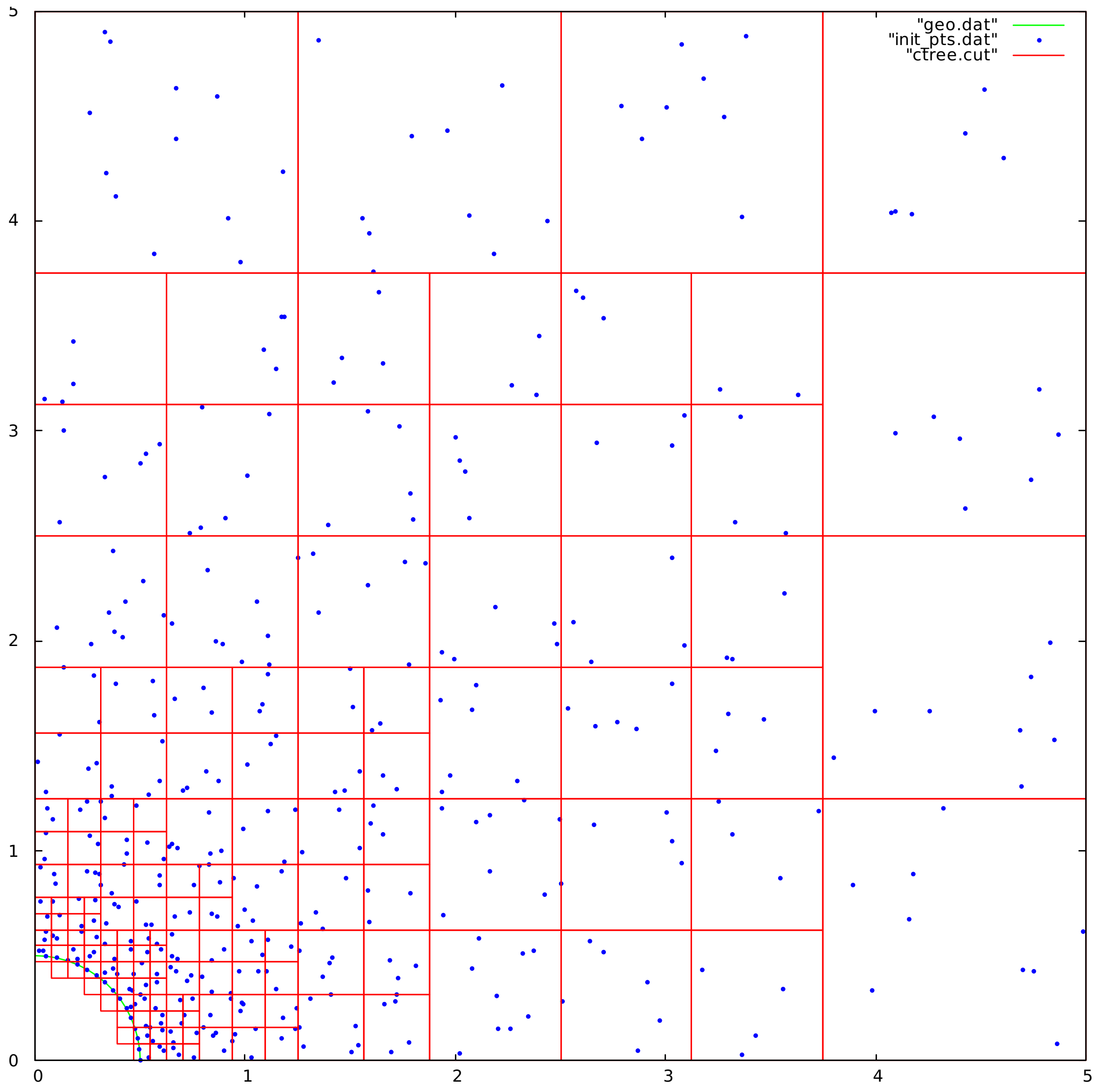


Figure 4.13 Circle within circle (zoomed to a portion of the upper right quadrant): Random initial point distribution with initialization tree

CHAPTER 5
PARTICLE DYNAMICS SIMULATION

Computing Forces

The nodes are all stored in a quad tree (this is now our third use of a quad tree, and we will refer to it as the node tree). Each node is stored as an extent square of side length q_i, which is node i's spacing parameter. The node tree is destroyed and recreated at each iteration because node locations change every iteration.

For an interior node, the sum of forces acting on that node is computed exclusively from its interaction with other nodes that are within a certain distance from it. That is, the influence of every node beyond this distance is neglected. The distance used here is a multiple of q_i, and the multiplication factor (called the cut-off distance factor) is a parameter set by the user prior to runtime (generally between 1.5 and 3).

For a boundary node the sum of forces acting on that node is contributed to only by the two boundary nodes immediately before and after that node on that boundary curve.

To compute the force contribution made by node j upon node i (the current node), a vector is computed, $\mathbf{r}_{ij}$, which is the vector from node i to node j. This vector and the local point cloud for node i are illustrated in Figure 5.1. Each node's "bubble" area is shown in order to visualize the spacing parameter q_i for each node, as shown in Figure 5.2. In these example figures, the spacing field is constant, and the radius of node i's circle of influence (transparent blue area with dashed boundary) is $1.5q_i$. (Interior nodes are colored in red and boundary nodes in blue.)

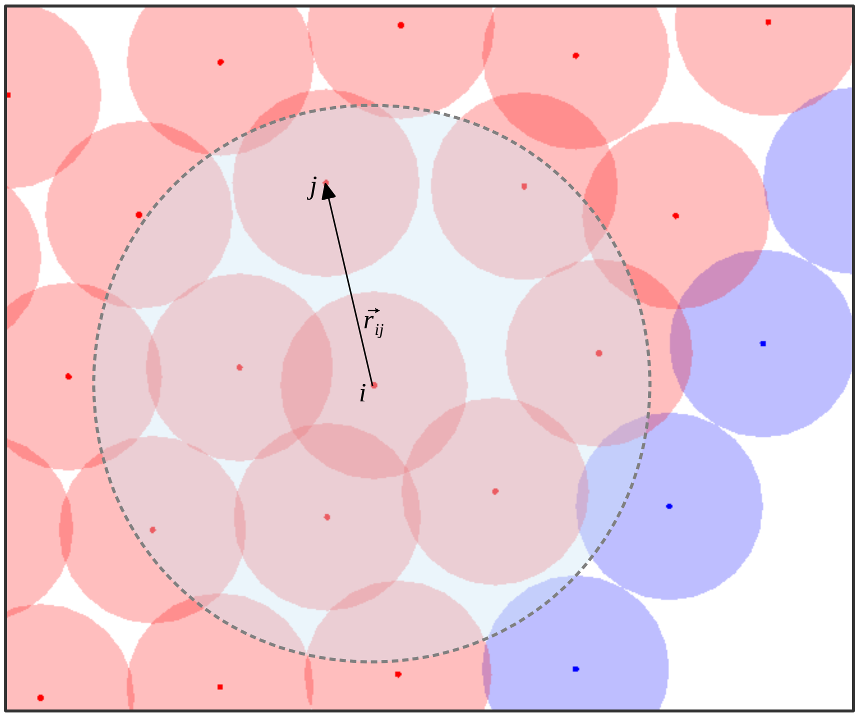


Figure 5.1 Vector from node i to node j

The vector, $\mathbf{r}_{ij}$, and the average of the spacing values of these two nodes,

$$\sigma_{ij} = \frac{q_i + q_j}{2} \tag{5.1}$$

are used to compute the force contribution from node j upon node i, $\mathbf{F}_{ij}$. The value of σ_{ij} is the distance desired between nodes i and j. This is the average of the two node spacings and can also be viewed as the sum of the radii of the two bubbles, for this is the distance these two nodes would be from one another were the edges of their bubbles just touching.

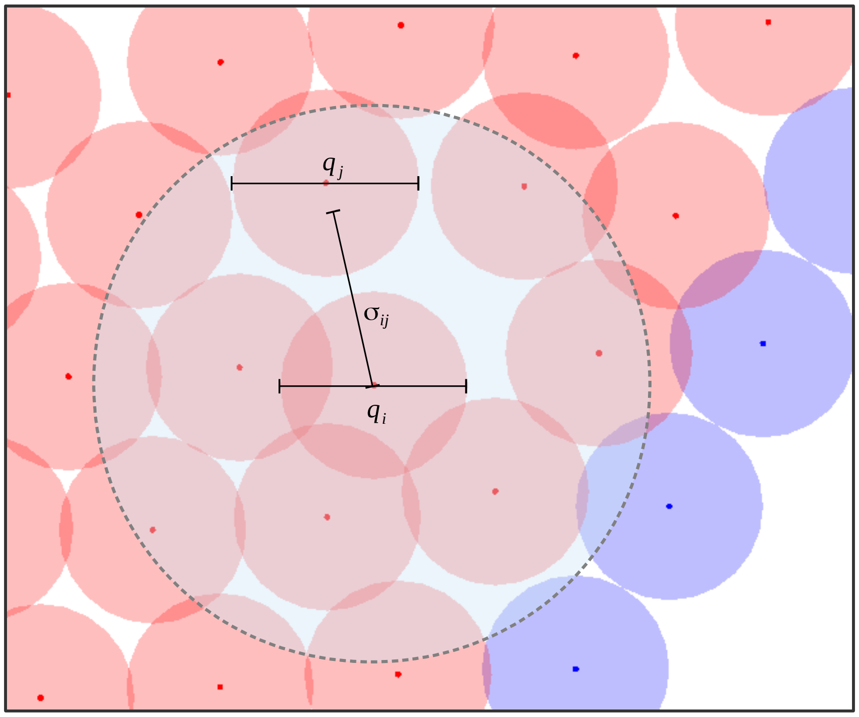


Figure 5.2 Ideal distance between a node pair is σ_{ij}: Node j in this example is farther from node i than σ_{ij}. Therefore node i will experience an attractive force toward node j

Figure 5.3 illustrates the forces exerted upon node i from each of the nodes in node i's local point cloud. (In this figure, nodes are numbered locally and force magnitudes are not drawn to scale.) Notice the direction of the force is dependent on the spacing. When node i's bubble overlaps with a neighbor's, node i is repelled from that neighbor. When there is a gap between the bubbles, node i is attracted to that bubble. For example, in Figure 5.3, we see that $\mathbf{F}_{i4}$ is pointing away from node 4 (repulsion) and $\mathbf{F}_{i7}$ is pointing toward node 7 (attraction).

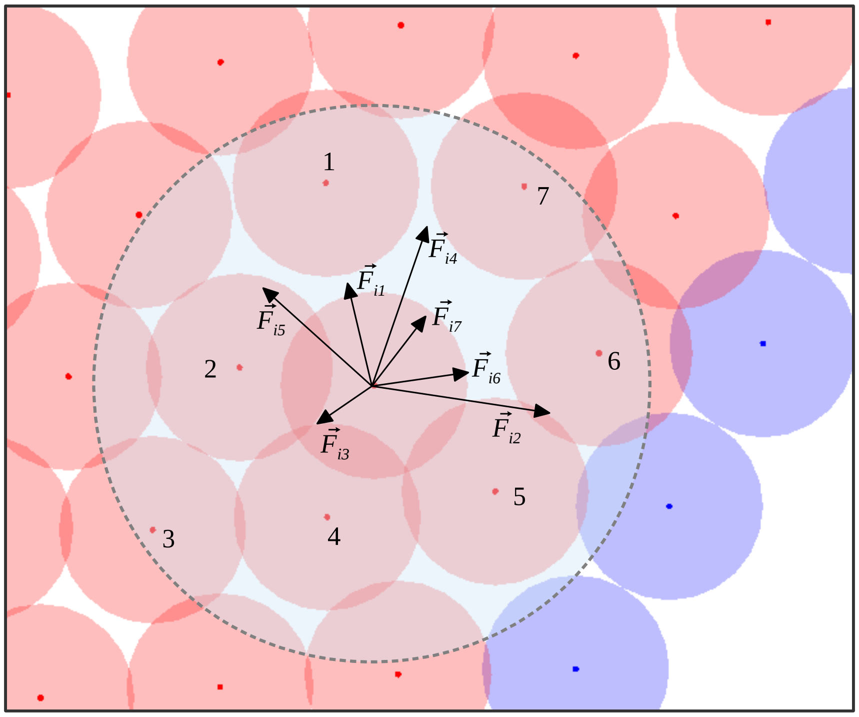


Figure 5.3 Vector forces exerted on node i from local cloud nodes

The net force acting on node i is then the sum of all these contributions

$$\mathbf{F}_i = \sum_j \mathbf{F}_{ij} \tag{5.2}$$

Figure 5.4 illustrates the resultant force vector sum for node i (again not drawn to scale).

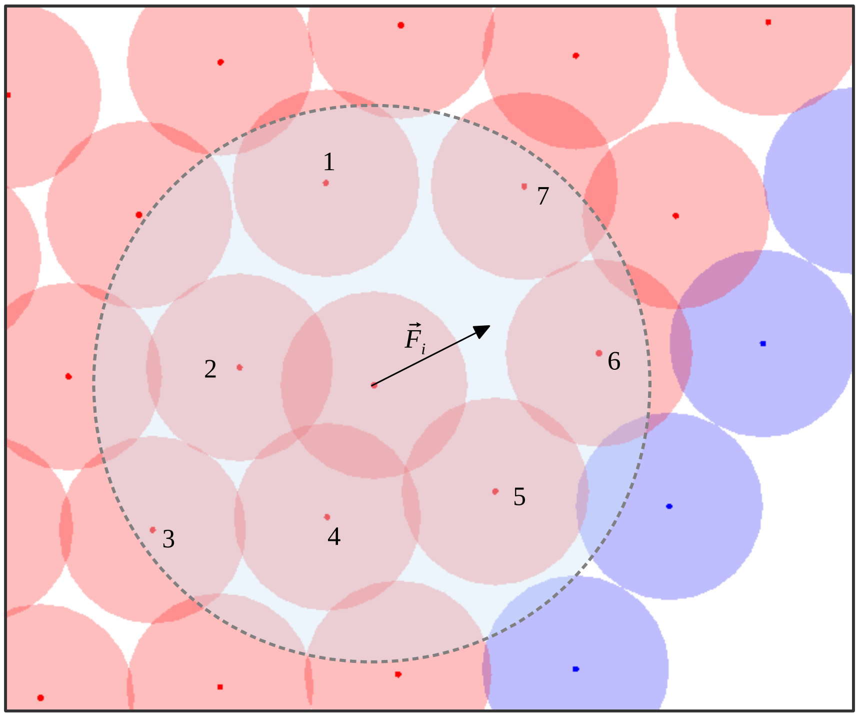


Figure 5.4 Resultant force vector sum for node i

Computing the Local Time Step

After summing the forces for node i, a distance is computed, $r_{0,i}$, which is the distance from the current node to its nearest neighboring node in the general direction of its force vector sum. That is,

$$r_{0,i} = \min_j \left\{ |\mathbf{r}_{ij}| : \hat{\mathbf{F}}_i \cdot \hat{\mathbf{r}}_{ij} > 0 \right\} \tag{5.3}$$

This is illustrated in Figure 5.5 and will be used when computing the time step for node i.

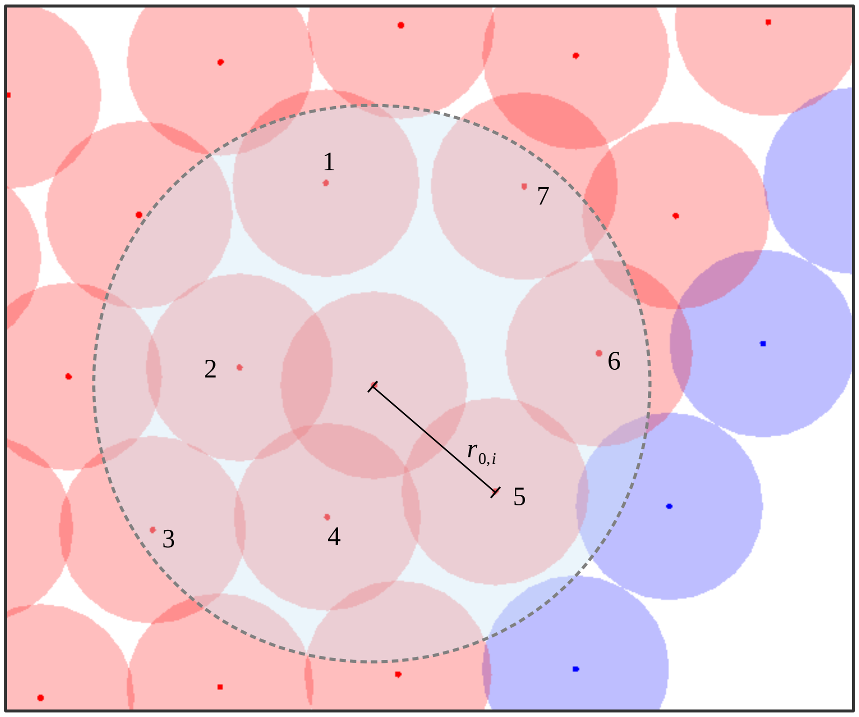


Figure 5.5 The distance to node i's nearest neighbor in the general direction of its vector force sum

For the motion of the particles we start with

$$\mathbf{F}_i = m_i \mathbf{a}_i \tag{5.4}$$

and, taking acceleration to be constant, we have that the change of position of a particle is described by

$$\Delta \mathbf{x}_i = \Delta t_i \mathbf{v}_{0,i} + \frac{\Delta t_i^2}{2} \mathbf{a}_i \tag{5.5}$$

where $\Delta \mathbf{x}_i = \Delta x_i \hat{\mathbf{i}} + \Delta y_i \hat{\mathbf{j}}$.

Since the nodes are not truly particles traveling in true space and time, there is no problem with altering the equations to obtain the desired results. Thus, taking $m_i = 1$ for every node, we have $\mathbf{a}_i = \mathbf{F}_i$. Also setting the velocity of each node to zero at the beginning of each iteration, and taking $\mathbf{F}_i$ to be constant throughout an iteration, (5.5) can be written as

$$\Delta \mathbf{x}_i = \frac{\Delta t_i^2}{2} \mathbf{F}_i \tag{5.6}$$

We want each node to move no farther in one iteration than 1/3 of the distance to its closest neighbor in the direction of its force sum ($r_{0,i}$ from above). To accomplish this, we substitute this fraction into (5.6) (that is, we set $|\Delta \mathbf{x}_{i,\max}| = \frac{1}{3} r_{0,i}$, shown in Figure 5.6) and solve as a scalar equation for Δt_i^2:

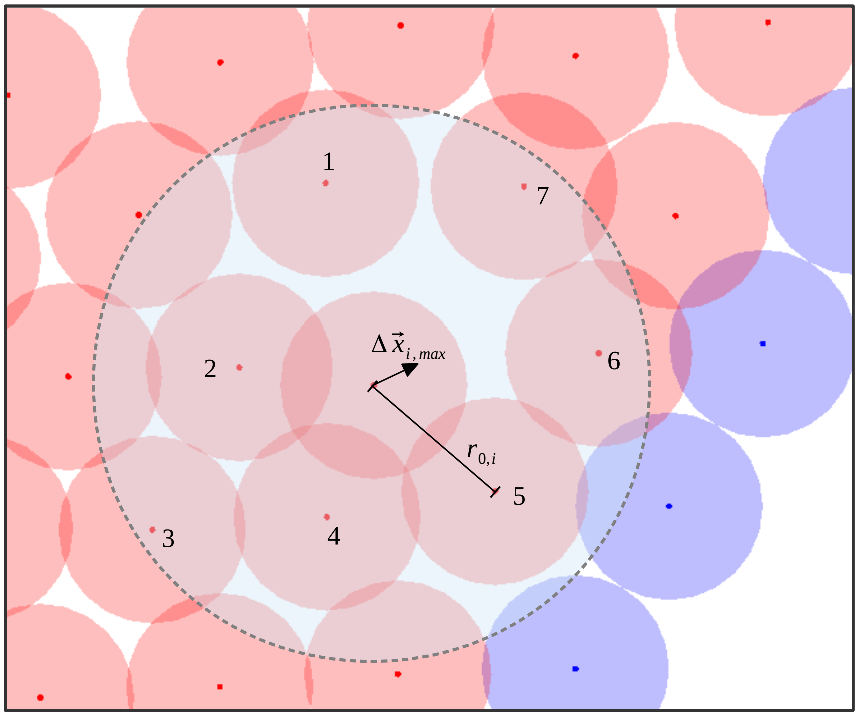


Figure 5.6 The maximum distance node i will be allowed to travel in one iteration

$$\frac{1}{3} r_{0,i} = \frac{\Delta t_i^2}{2} |\mathbf{F}_i| \tag{5.7}$$

$$\Delta t_i^2 = \frac{2 r_{0,i}}{3 |\mathbf{F}_i|} \tag{5.8}$$

This expression for Δt_i^2 could be substituted into (5.6) for efficient computation of a node's new location. However, in the interest of having the nodes settle out to a stable configuration (and not continue vacillating about), we instead limit the local node time step with a global maximum. Thus,

$$\Delta t_i^2 = \min\left(\Delta t_i^2, \Delta t_{\mathrm{GM}}^2\right) \tag{5.9}$$

In this way, a node with a relatively small force sum will travel a shorter distance than $\frac{1}{3} r_{0,i}$, since the time step required for the node to travel that distance would be greater than the global maximum.

New Node Locations

The final value of the local time step obtained from (5.9) is used in (5.6) to compute the change in node i's location. This vector, $\Delta \mathbf{x}_i$, is added to node i's original location vector to obtain the node's new spatial location. Each boundary mesh node is "re-snapped" to its boundary curve at the "closest point," the point from which a vector normal to the curve would reach the node's off-boundary location.

Each interior node is tested at its new location to see if it has moved outside the domain. If a node is "out-of-bounds," it is "snapped" to its "closest point" on whichever boundary curve it is closest to and its node number is incorporated into the boundary mesh node connectivity data structure. This process is illustrated in Figure 5.7.

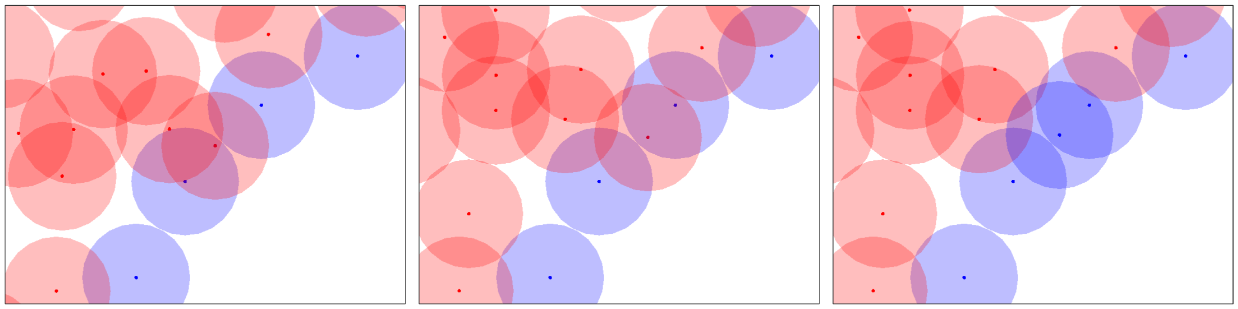


Figure 5.7 Example of a node being snapped to a boundary curve after crossing it

Once the new location for a node has been determined for the next iteration, its new spacing parameter is computed from its spatial location. Once again, the spacing function can be arbitrary, but the one used in the experimental cases in this study is an inverse distance weighting function based on the boundary spacing.

The Global Maximum Time Step

As described above, we limit the local time step size of each node with a global maximum in order for all of the nodes to settle out and keep them from endlessly jittering. The purpose of this can be understood as analogous to releasing and catching a marble along the inner surface of a bowl. Because our "marble" moves at a constant acceleration for all of one iteration, we need to catch and stop it after a certain amount of time so that it does not roll too far up the other side of the bowl even farther away from the bottom than at the beginning of the iteration.

At this point we do not have an automated time step calculation that will work well for different geometries and spacing fields and populations. In the current method, the user specifies an initial time step and this number is automatically incremented or decremented based on the stability of the point cloud configuration. If the maximum force sum decreases for five consecutive iterations, the maximum time step is increased by a factor of 1.01. On the other hand, if the value of the maximum force sum increases five times (not necessarily consecutively) before it decreases five times, the time step is decreased by a factor of 0.99.

Inter-Node Force Formula

As stated in Chapter 3, the formula used for the force magnitude between the i-j node pair is the Lennard-Jones pair potential (rather than the actual force). We use the potential formula because we want force equilibrium when $r_{ij} = \sigma_{ij}$, and the actual force formula derived from the potential has an equilibrium at $r_{ij} \approx 1.1225\sigma_{ij}$.

$$f_{ij} = 4a\left(\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right) \tag{5.10}$$

We can see the magnitude of force that node j will exert on node i based on the ratio of its distance away from node i to the desired spacing in Figure 5.8.

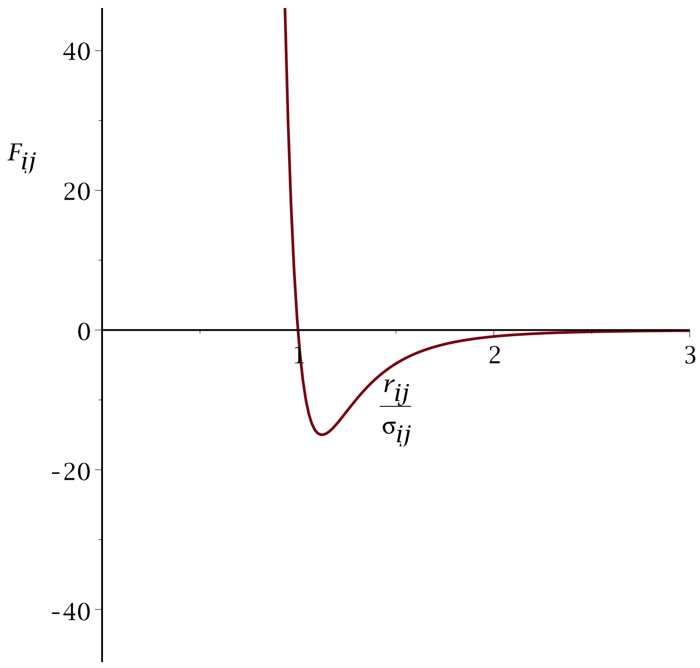


Figure 5.8 Force magnitude with respect to r_{ij}/σ_{ij} with $a = 15$: When $r_{ij} = \sigma_{ij}$, the magnitude of the force between nodes i and j is zero

In a variable spacing field, because of the globally limited time step, nodes in regions of larger spacing need to be able to move greater distances in the same amount of time as nodes in regions of smaller spacing move smaller distances. This means that the magnitude of the force sum needs to scale relative to the local spacing. With the formula in equation (5.10), the force magnitude is based only on the ratio of the desired spacing to the actual spacing. Therefore, we modify this formula with a scaling factor of the desired pair spacing, σ_{ij}. This force is then directed along $-\hat{\mathbf{r}}_{ij}$ to produce the inter-node vector force contribution:

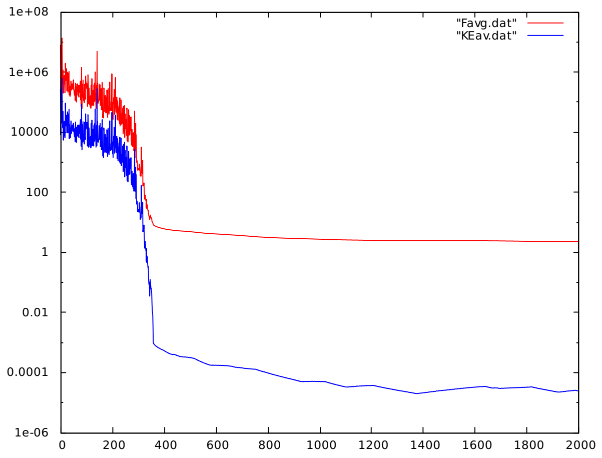
$$\mathbf{F}_{ij} = -4a\sigma_{ij}\left(\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right)\frac{\mathbf{r}_{ij}}{r_{ij}} \tag{5.11}$$

CHAPTER 6
EXPERIMENTAL RESULTS

In this chapter, the relevant results obtained from the method applied to different cases will be shown. These include the trends in average kinetic energy and net force over the course of the iterations, images of the resulting point distributions and triangulations, timing statistics for each run, and quality metrics of the resulting triangulation compared with those of meshes produced by the two unstructured initialization techniques of Pointwise [1] using the same geometry. (In point distribution images, the geometry segments are shown in green.) All of the following cases were run with a cut-off distance factor of 1.5 for 2000 iterations, so that the run times tabulated in the last section of this chapter are comparable.

Circle within a Circle

The geometry for this case is shown in Figures 4.1 and 4.2. The outer circle has a radius of 10 and the inner circle has a radius of 0.5. Each circle is split into a lower and an upper semicircle. Thus there are four boundary curves in all, each of which is populated with 31 equally-spaced points (including the endpoints). This case was run with an initial maximum time step of 0.01 for the structured initialization and 0.025 for the random initialization. In the following figures the results from the run starting with the structured initialization are shown on the left image of each figure and those from the random initialization are shown on the right image. Figure 6.1 shows the trend in the average nodal resultant force and the average nodal kinetic energy for each of the runs.



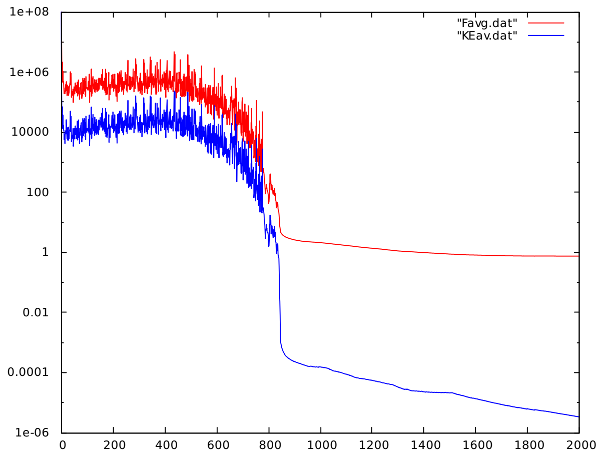


Figure 6.1 Circle within circle case: Average nodal net force (*red*) and kinetic energy (*blue*) for the structured (*left*) and random (*right*) initializations

Figures 6.2 and 6.3 together give a comparative view of the initial distribution to the final distribution for each run displaying the entire domain. Likewise, Figures 6.4 and 6.5 display the same configurations zoomed to the inner circle. Figures 6.6 and 6.7 show the triangulation of each resultant distribution.

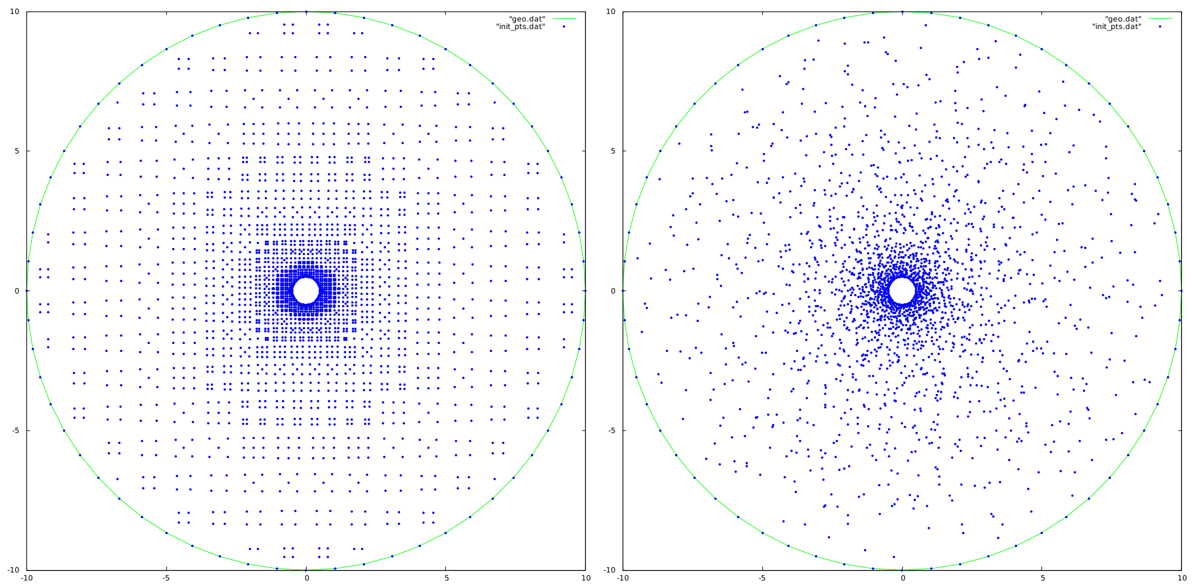


Figure 6.2 Circle within circle: Initial point distribution from structured (*left*) and random (*right*) initializations

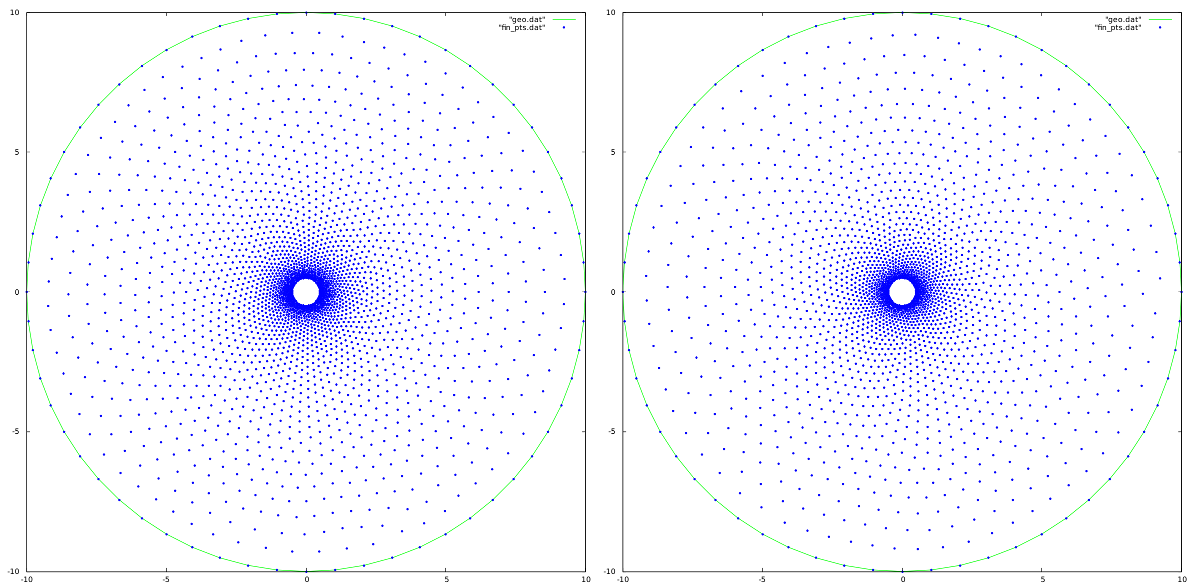


Figure 6.3 Circle within circle: Resulting point distribution from structured (*left*) and random (*right*) initializations

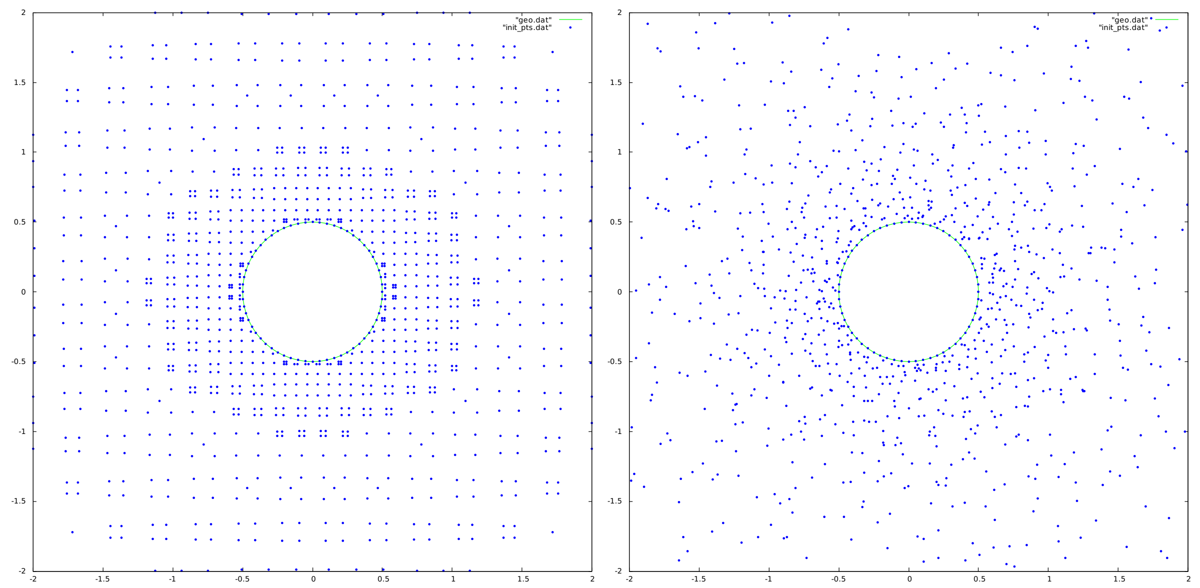


Figure 6.4 Circle within circle (zoomed to inner circle): Initial point distribution from structured (*left*) and random (*right*) initializations

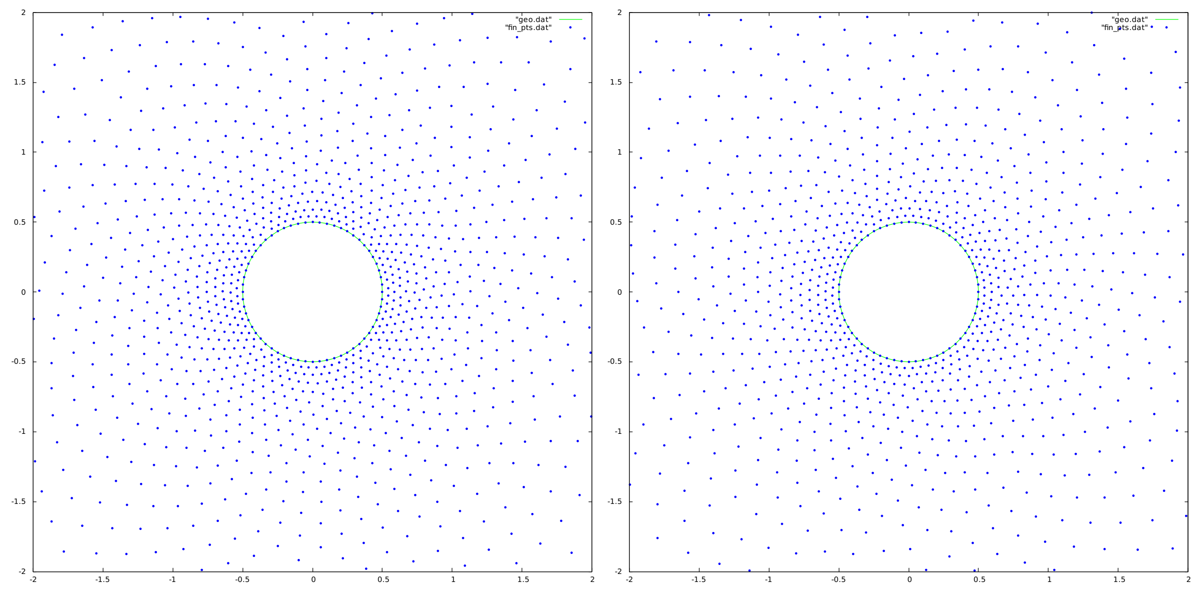


Figure 6.5 Circle within circle (zoomed to inner circle): Resulting point distribution from structured (*left*) and random (*right*) initializations

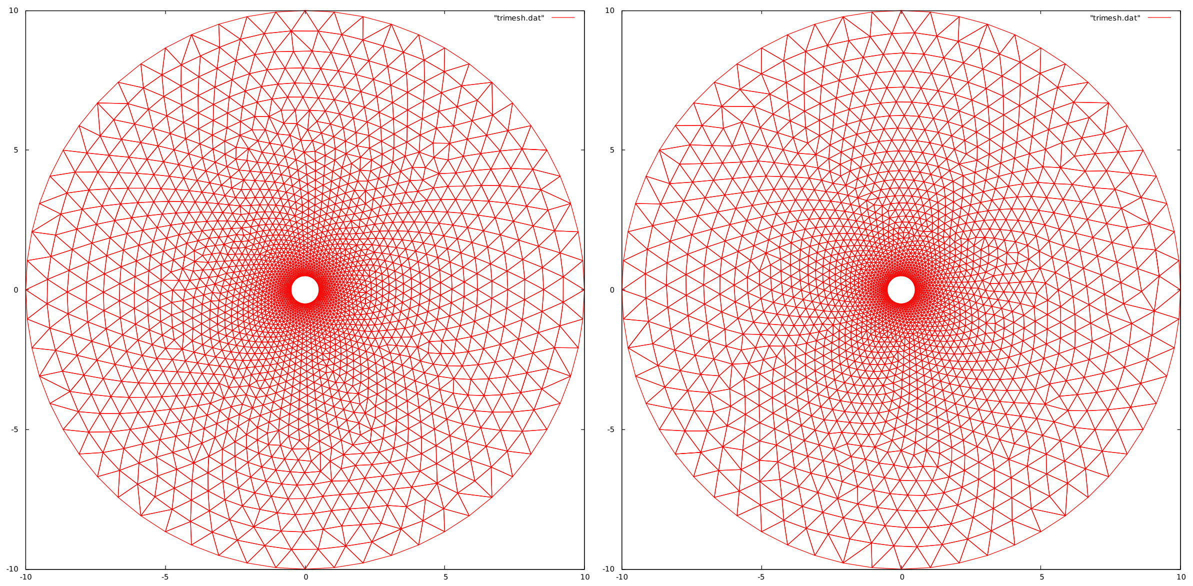


Figure 6.6 Circle within circle: Triangulated result from structured (*left*) and random (*right*) initializations

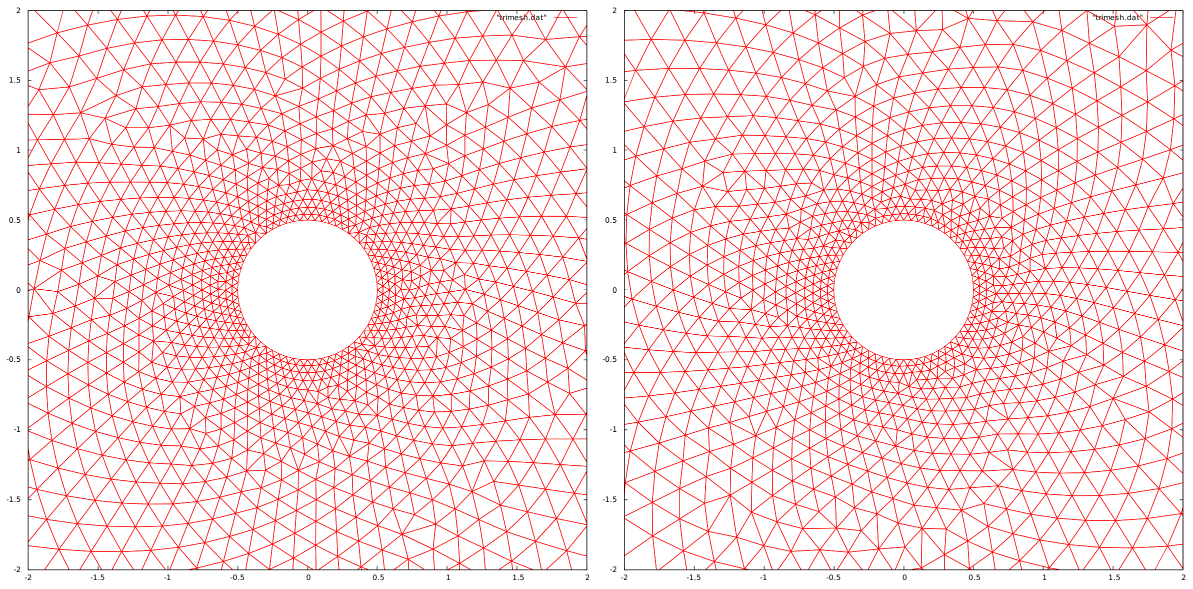


Figure 6.7 Circle within circle (zoomed to inner circle): Triangulated result from structured (*left*) and random (*right*) initializations

NACA 0012 Airfoil

The geometry for this case consists of a NACA 0012 airfoil of length 1 and a 30×30 square outer boundary. The spacing on the outer boundary is uniform. The spacing on the airfoil is smaller at the leading and trailing edges and coarser in the middle. This case was run with both the structured (tracked in the first section below) and random (tracked in the second section below) initialization schemes, using an initial maximum time step of 0.009 for both runs.

Structured Initialization

Figure 6.8 shows the trend in the average nodal resultant force and the average nodal kinetic energy from the run starting with the structured initialization.

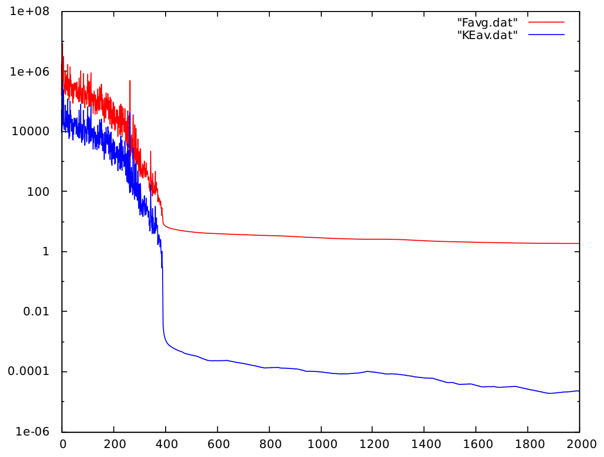


Figure 6.8 NACA 0012 airfoil case: Average nodal net force (*red*) and kinetic energy (*blue*) for the structured initialization

Figures 6.9 through 6.16 display the structured initial point distribution, the final point distribution, and the triangulation of the final point distribution for various regions of the domain.

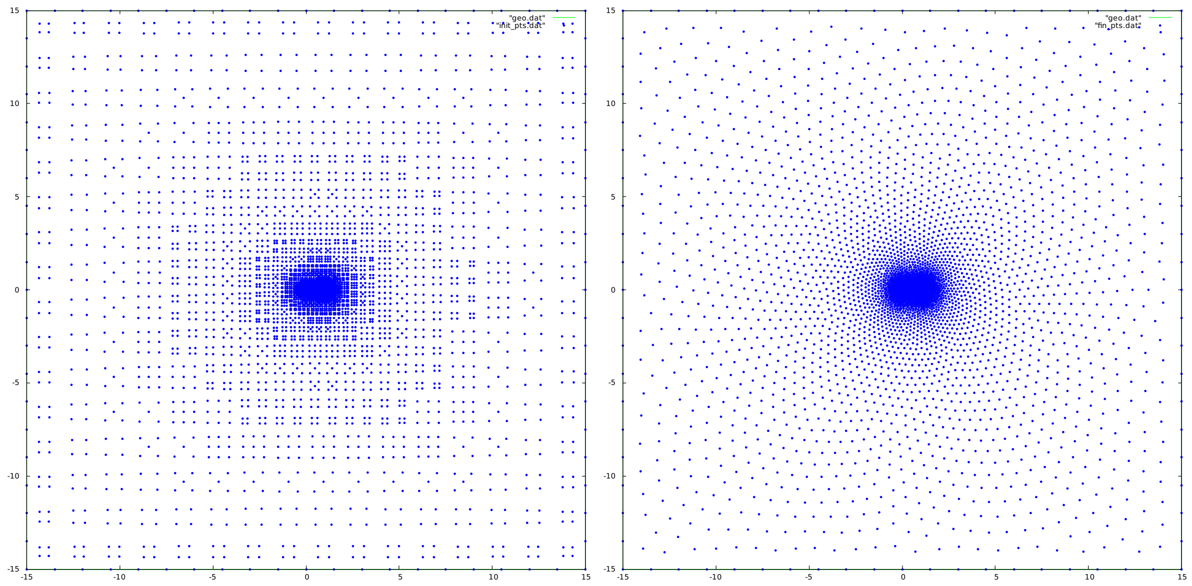


Figure 6.9 NACA 0012 airfoil: Initial point distribution (*left*) and resulting point distribution (*right*) from structured initialization

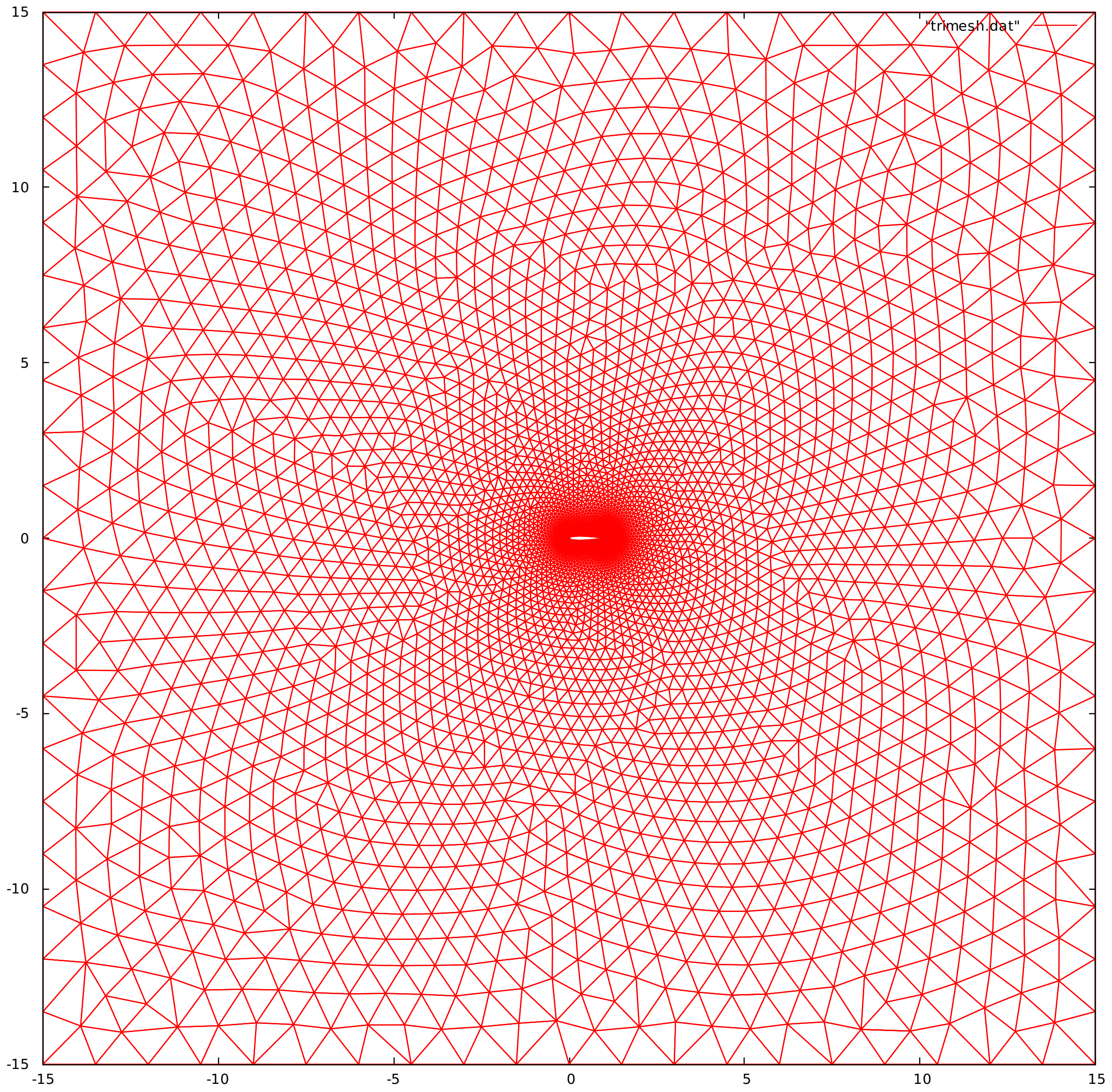
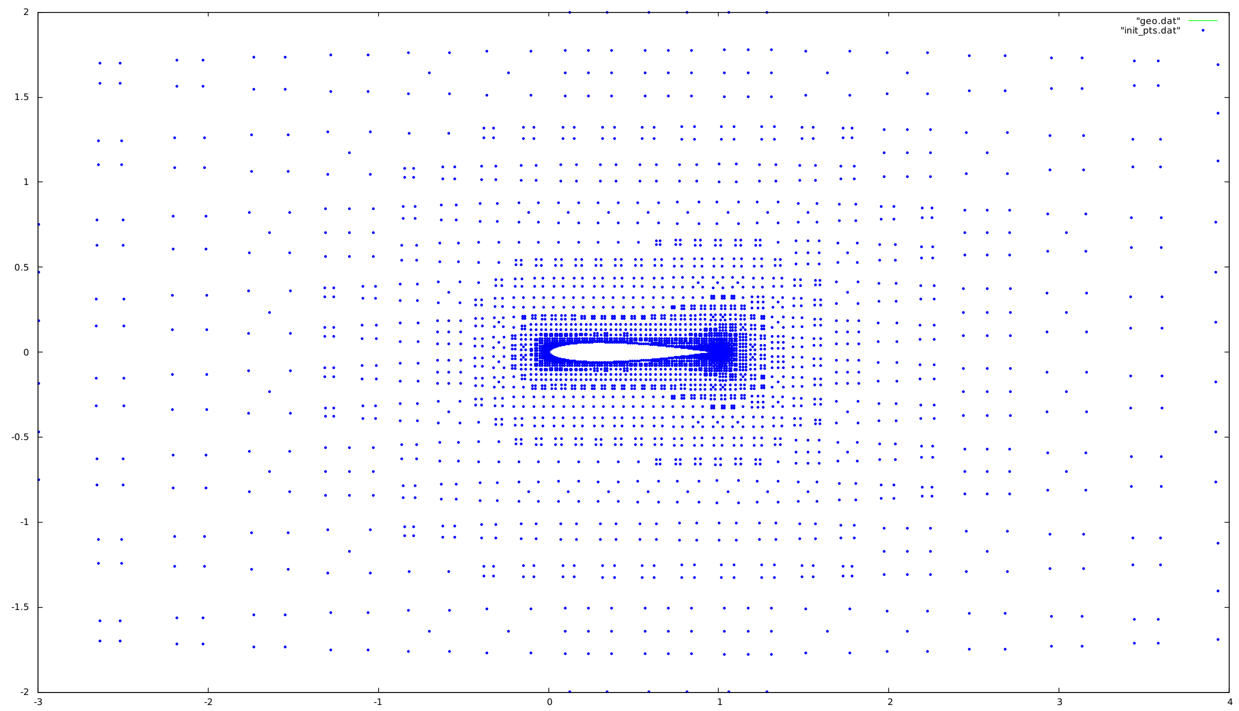


Figure 6.10 NACA 0012 airfoil: Triangulated result from structured initialization



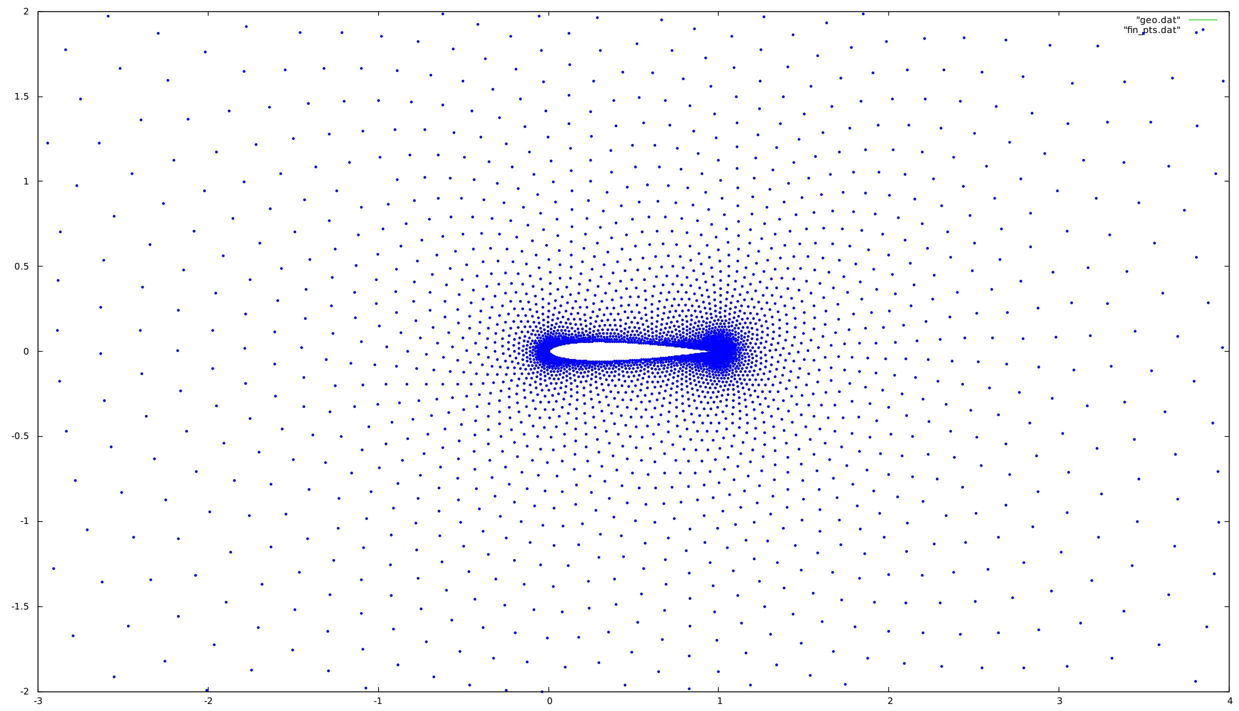


Figure 6.11 NACA 0012 airfoil (zoomed midway): Initial point distribution (*top*) and resulting point distribution (*bottom*) from structured initialization

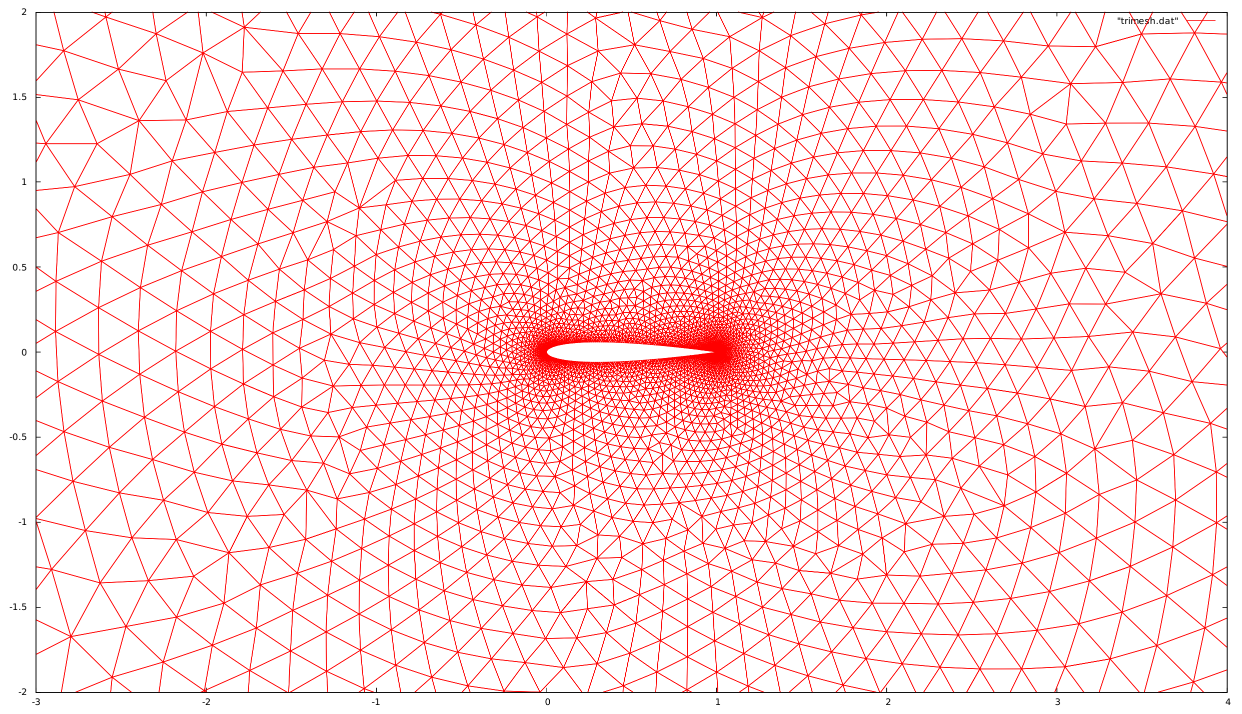


Figure 6.12 NACA 0012 airfoil (zoomed midway): Triangulated result from structured initialization

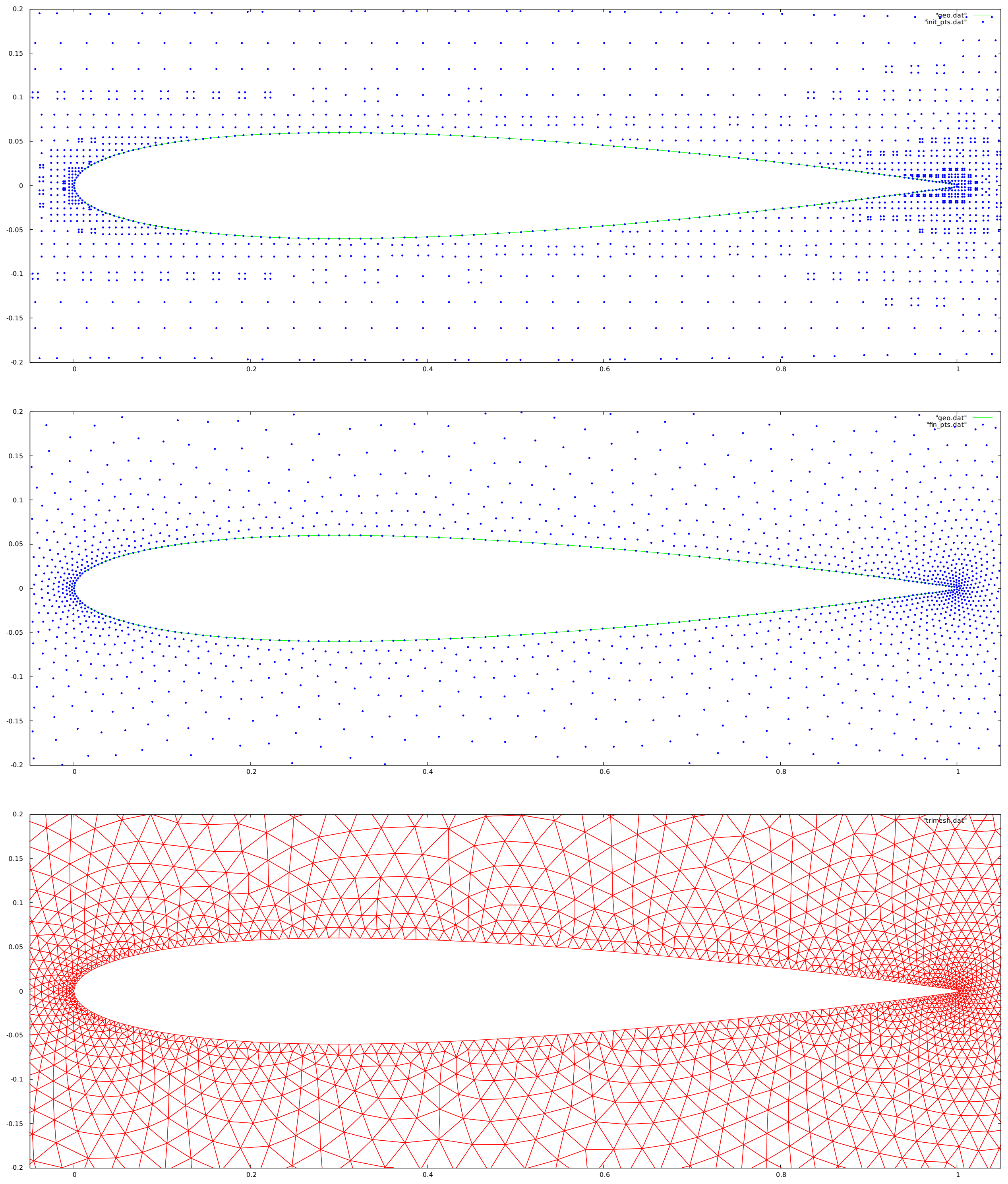
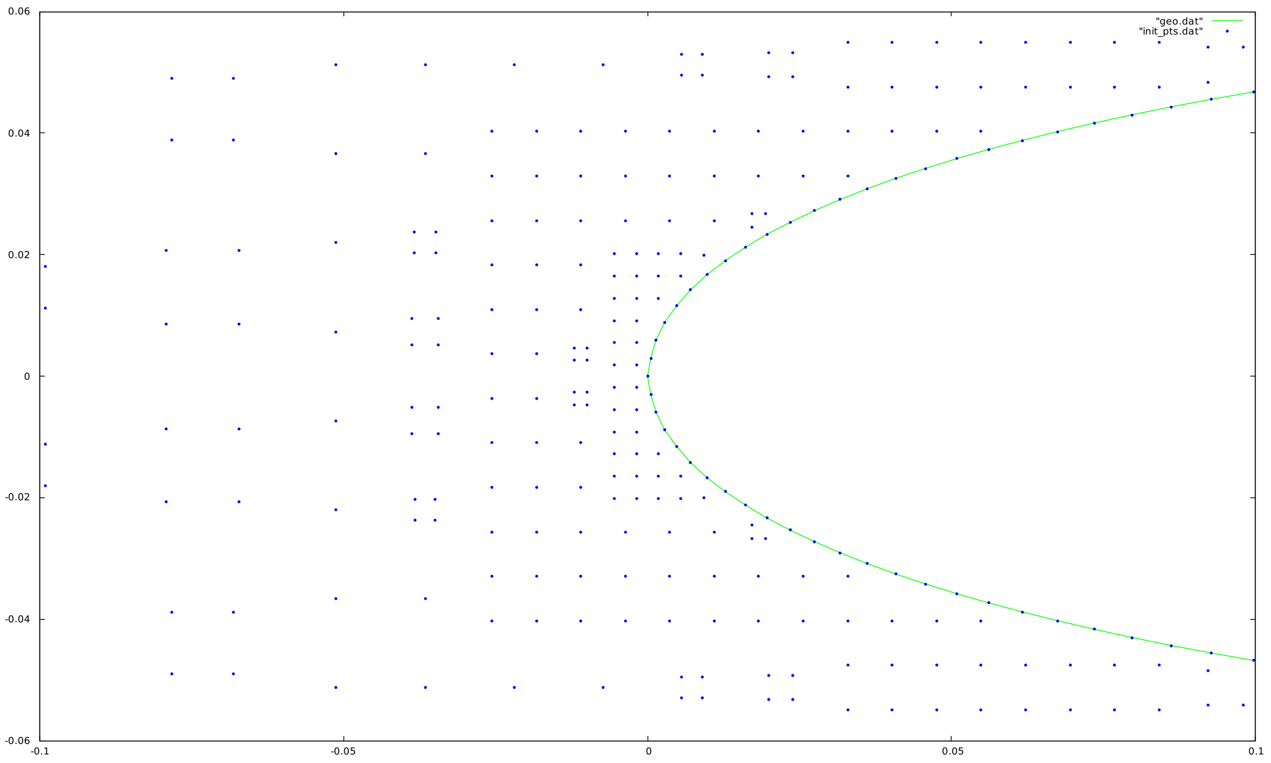


Figure 6.13 NACA 0012 airfoil (zoomed): Initial point distribution (*top*), resulting point distribution (*middle*), and triangulated result (*bottom*) from structured initialization



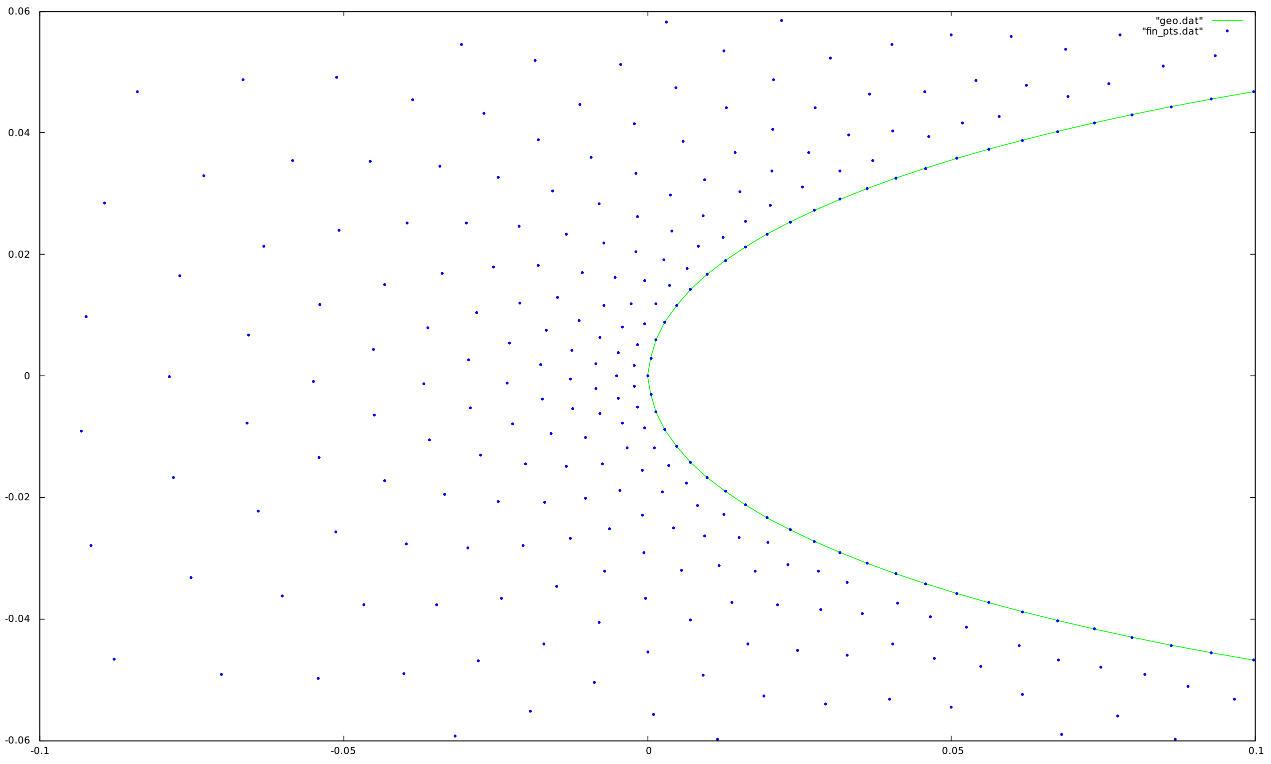
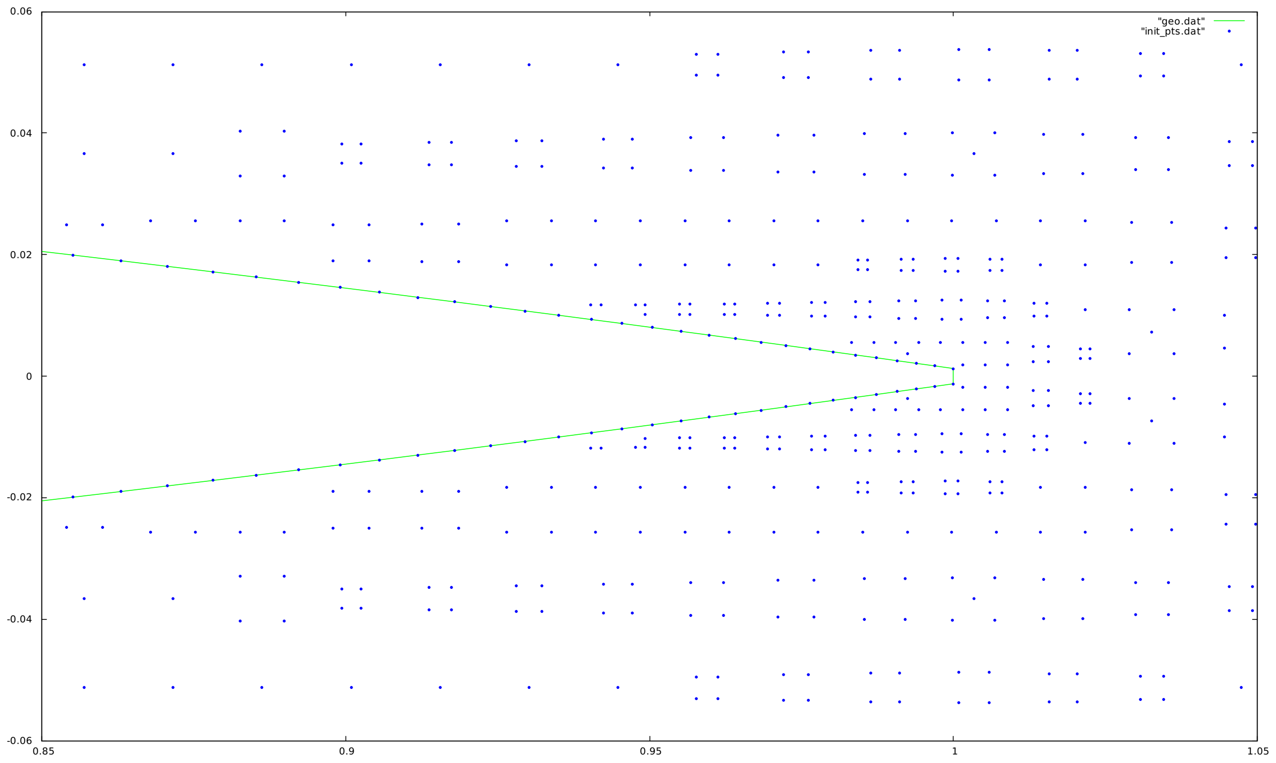


Figure 6.14 NACA 0012 airfoil (zoomed to leading edge): Initial point distribution (*top*) and resulting point distribution (*bottom*) from structured initialization



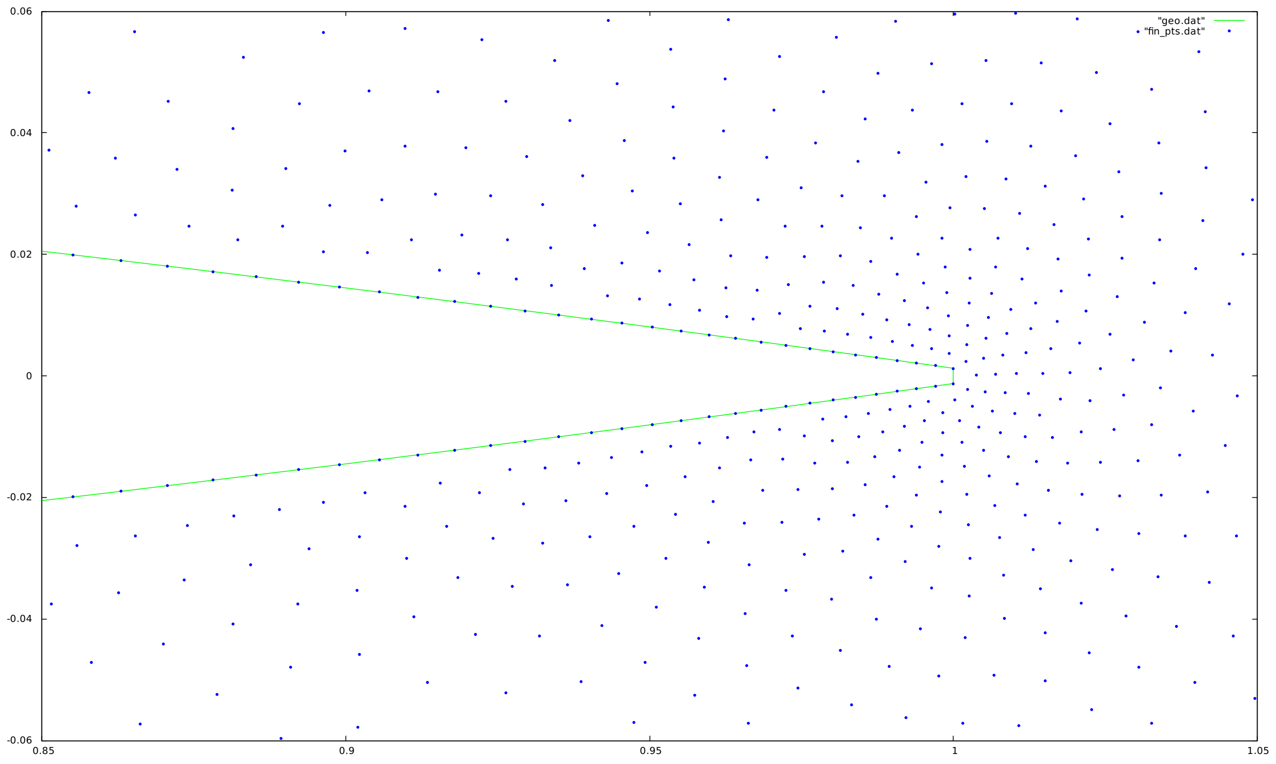
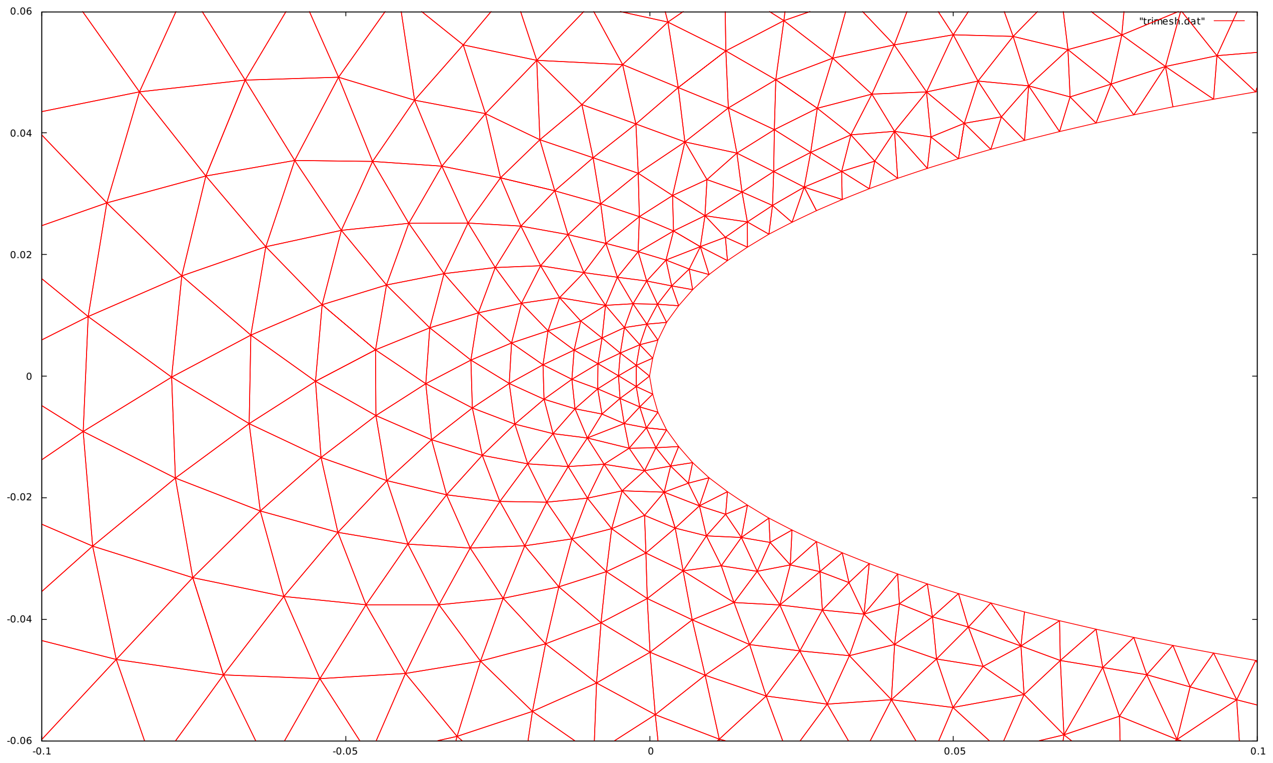


Figure 6.15 NACA 0012 airfoil (zoomed to trailing edge): Initial point distribution (*top*) and resulting point distribution (*bottom*) from structured initialization



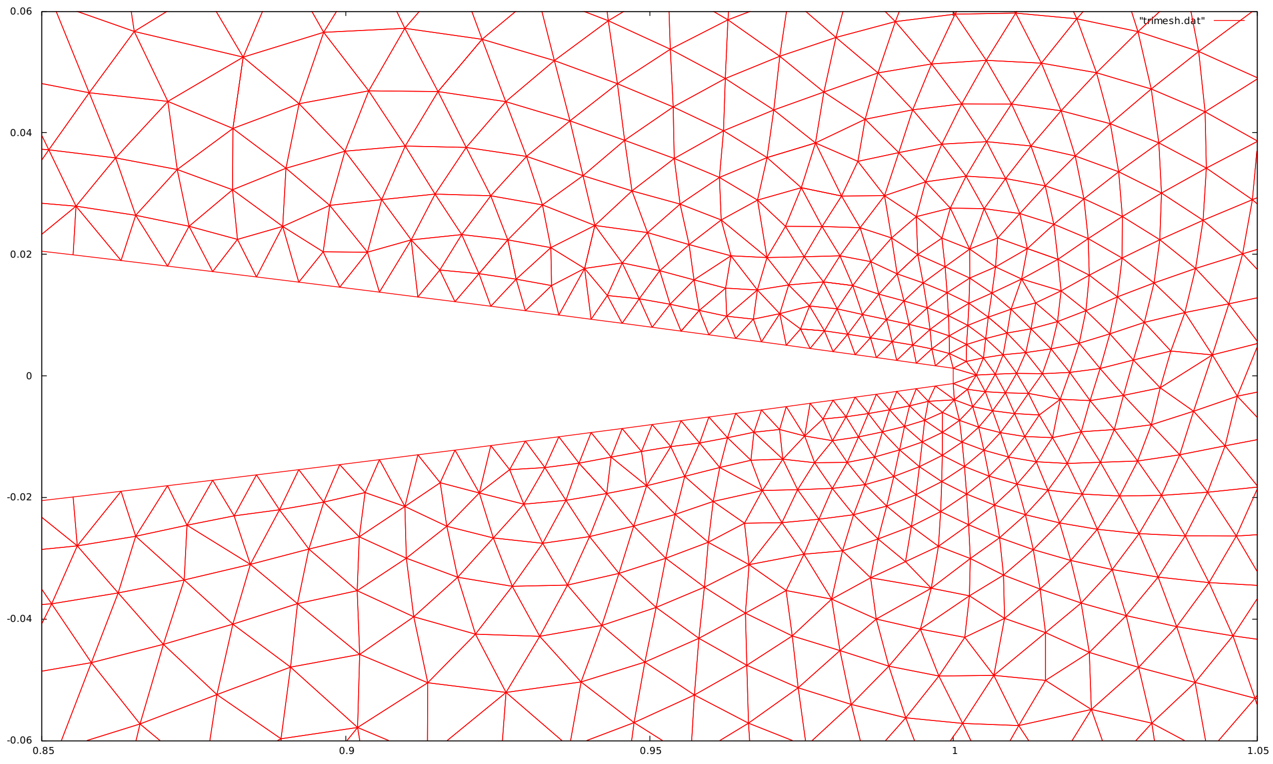


Figure 6.16 NACA 0012 airfoil: Triangulated result zoomed to leading edge (*top*) and trailing edge (*bottom*) from structured initialization

Random Initialization

Figure 6.17 shows the trend in the average nodal resultant force and the average nodal kinetic energy from the run starting with the random initialization.

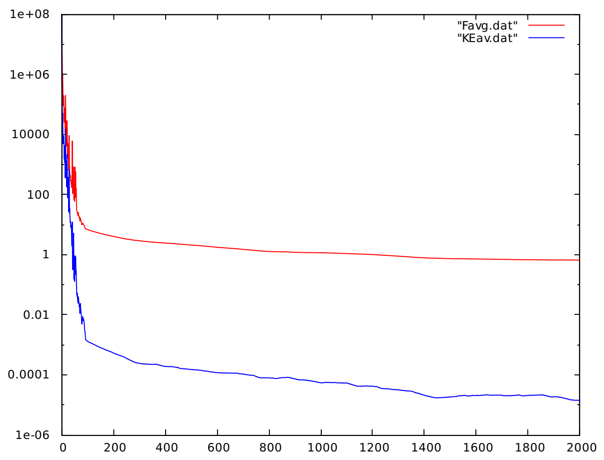


Figure 6.17 NACA 0012 airfoil case: Average nodal net force (*red*) and kinetic energy (*blue*) for the random initialization

Figures 6.18 through 6.25 display the randomized initial point distribution, the final point distribution, and the triangulation of the final point distribution for various regions of the domain, the same as shown in the previous section covering the results from the structured initialization.

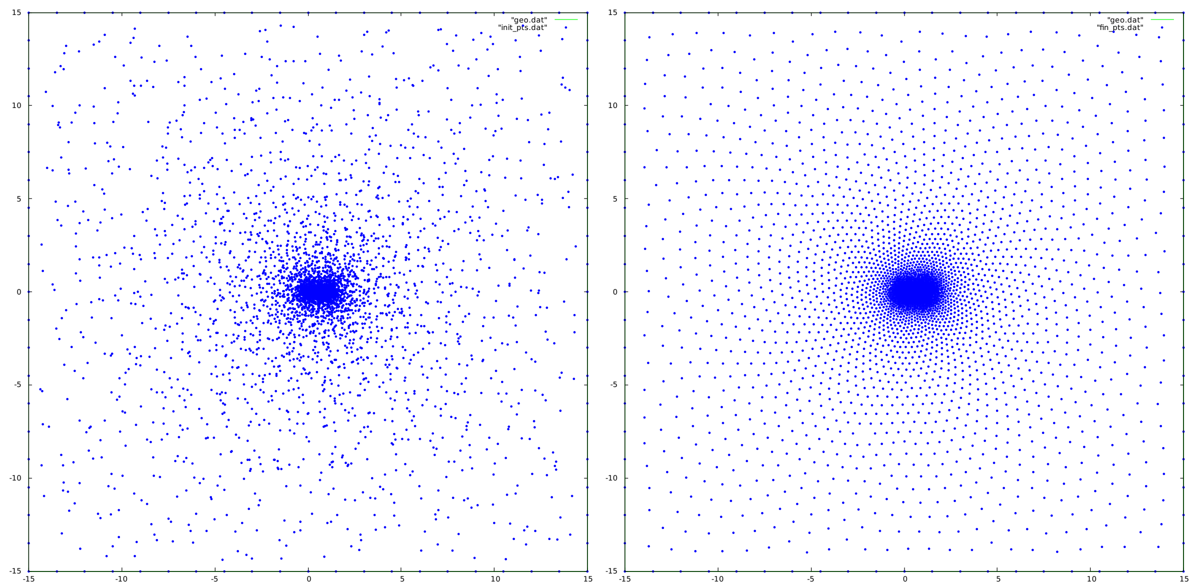


Figure 6.18 NACA 0012 airfoil: Initial point distribution (*left*) and resulting point distribution (*right*) from random initialization

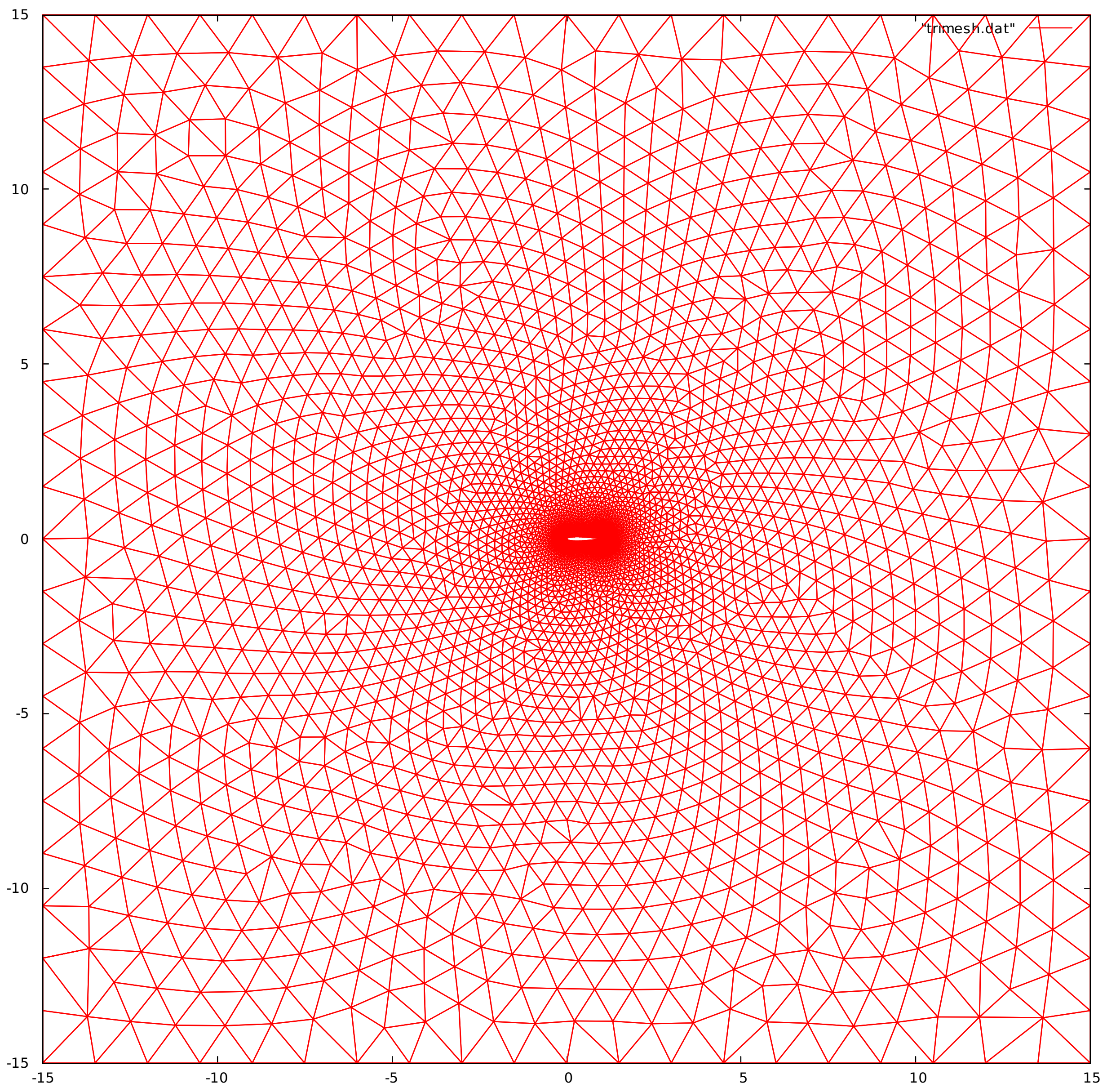


Figure 6.19 NACA 0012 airfoil: Triangulated result from random initialization

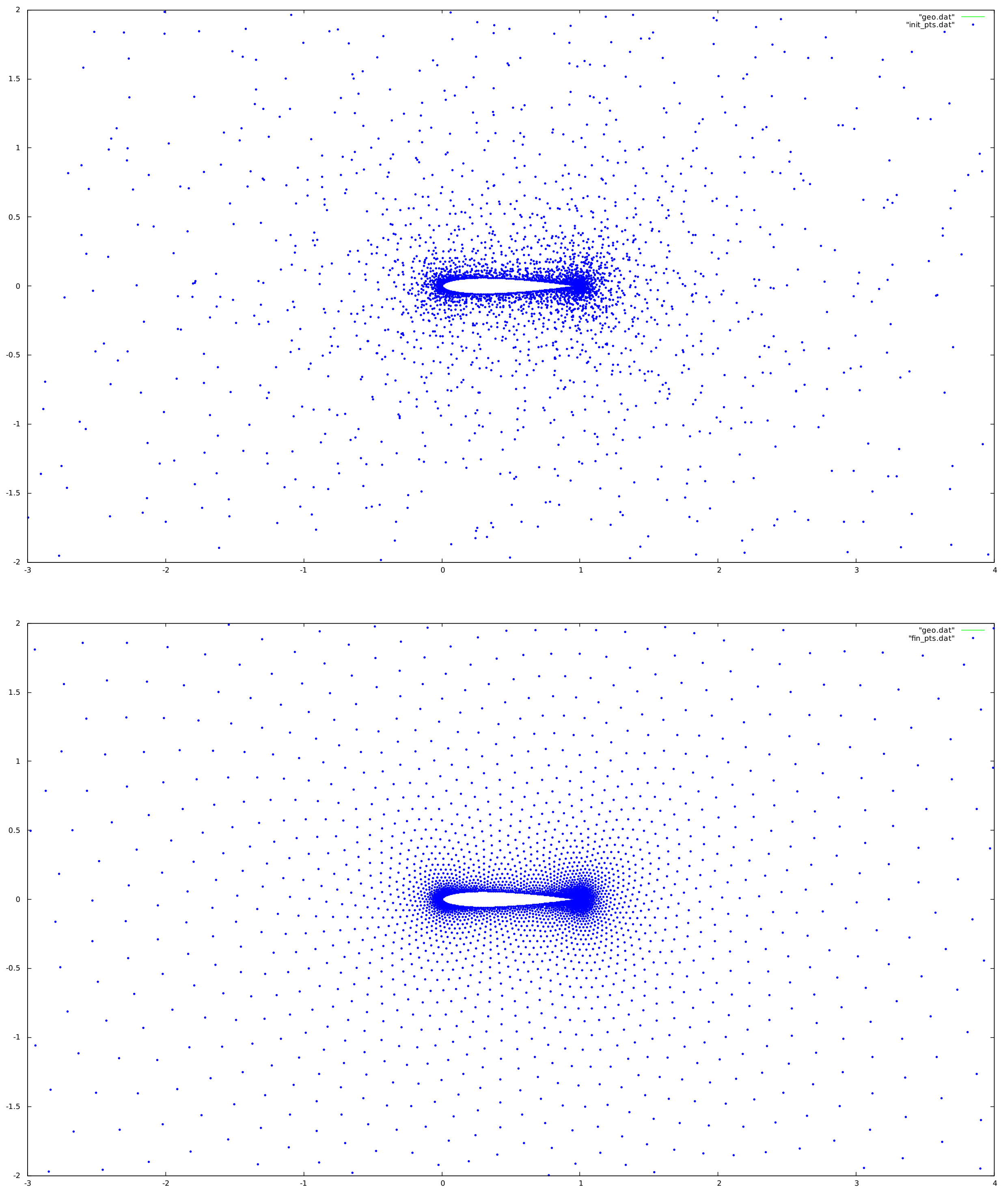


Figure 6.20 NACA 0012 airfoil (zoomed midway): Initial point distribution (*top*) and resulting point distribution (*bottom*) from random initialization

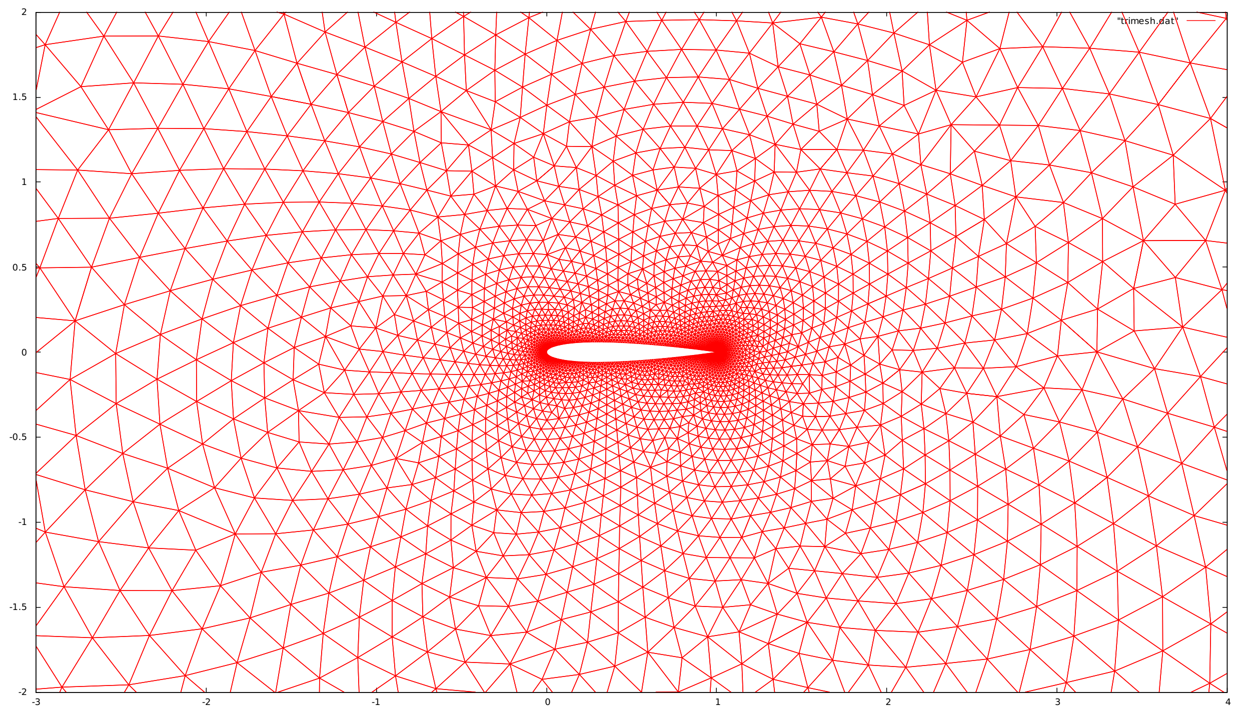


Figure 6.21 NACA 0012 airfoil (zoomed midway): Triangulated result from random initialization

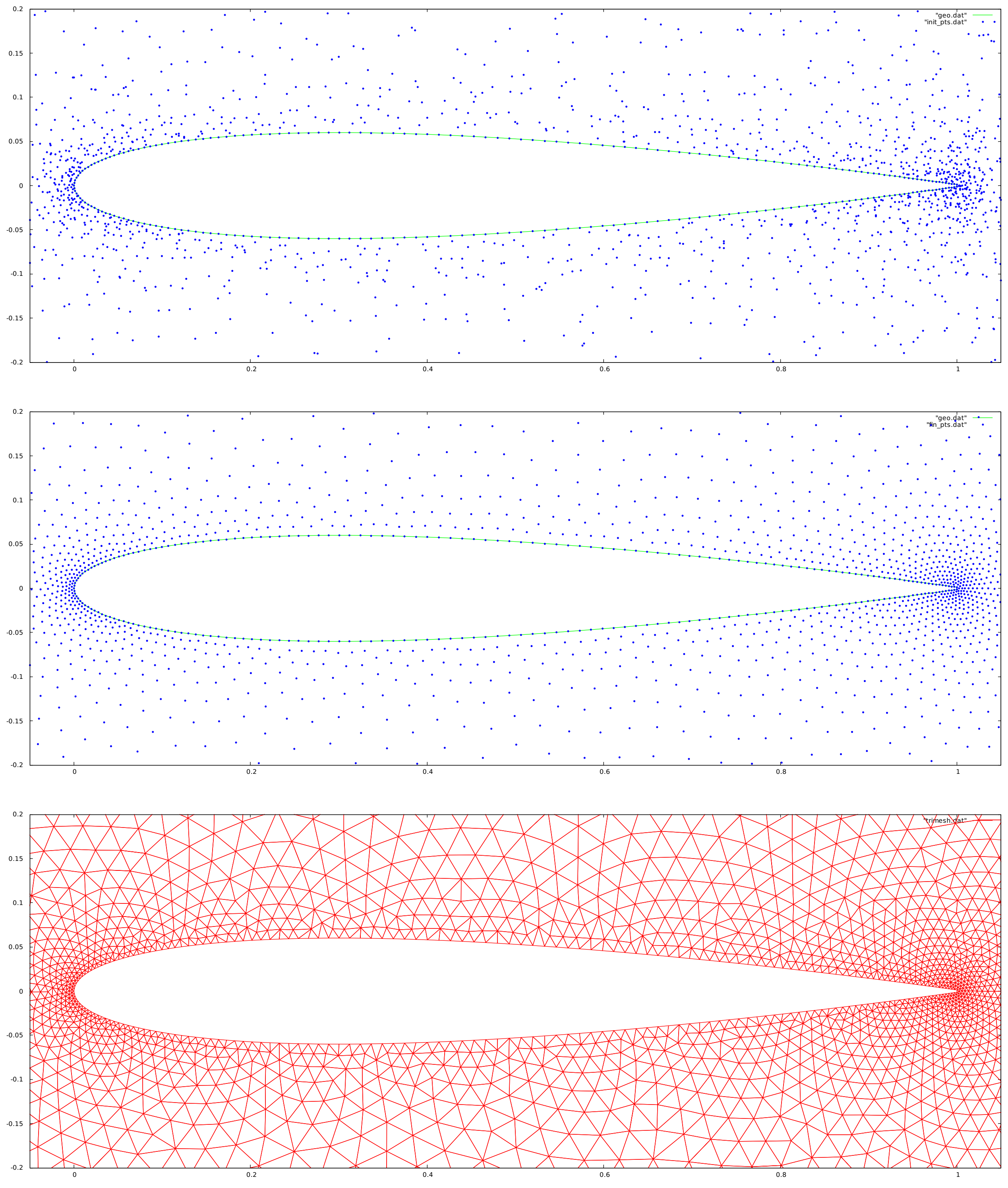
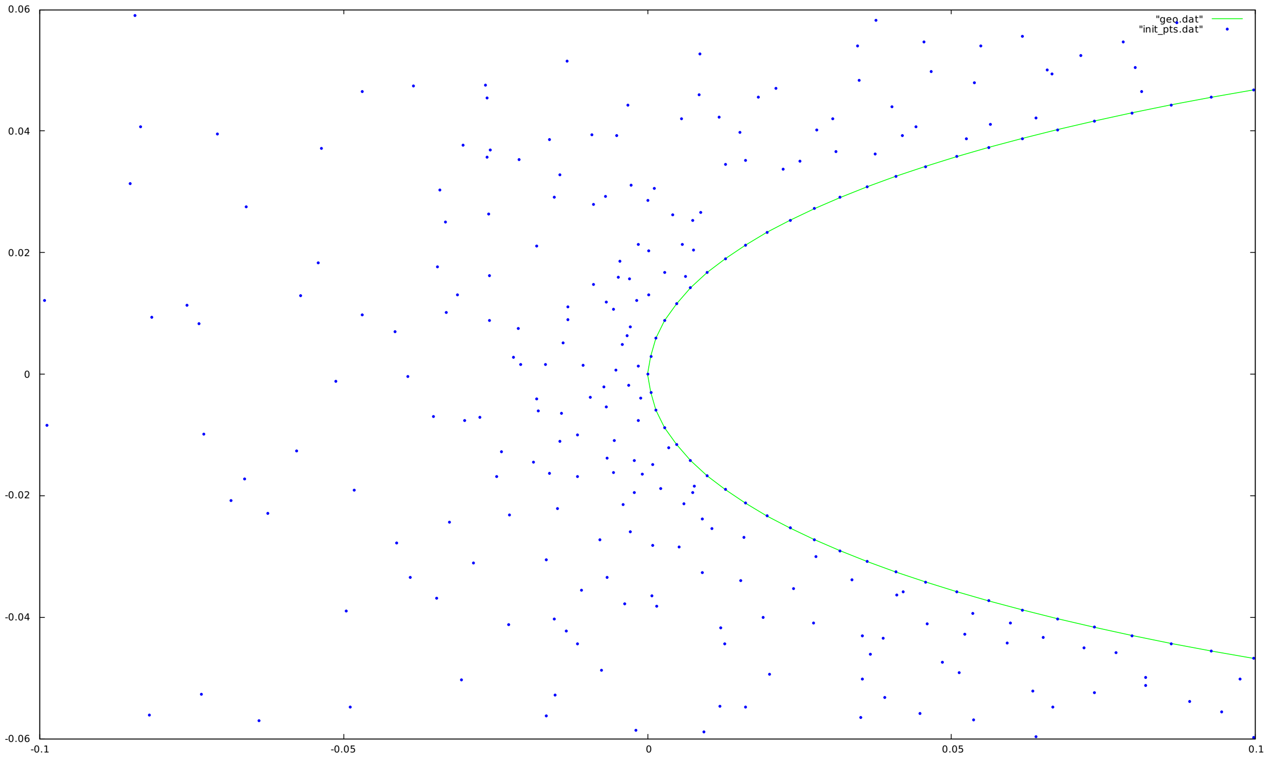


Figure 6.22 NACA 0012 airfoil (zoomed): Initial point distribution (*top*), resulting point distribution (*middle*), and triangulated result (*bottom*) from random initialization



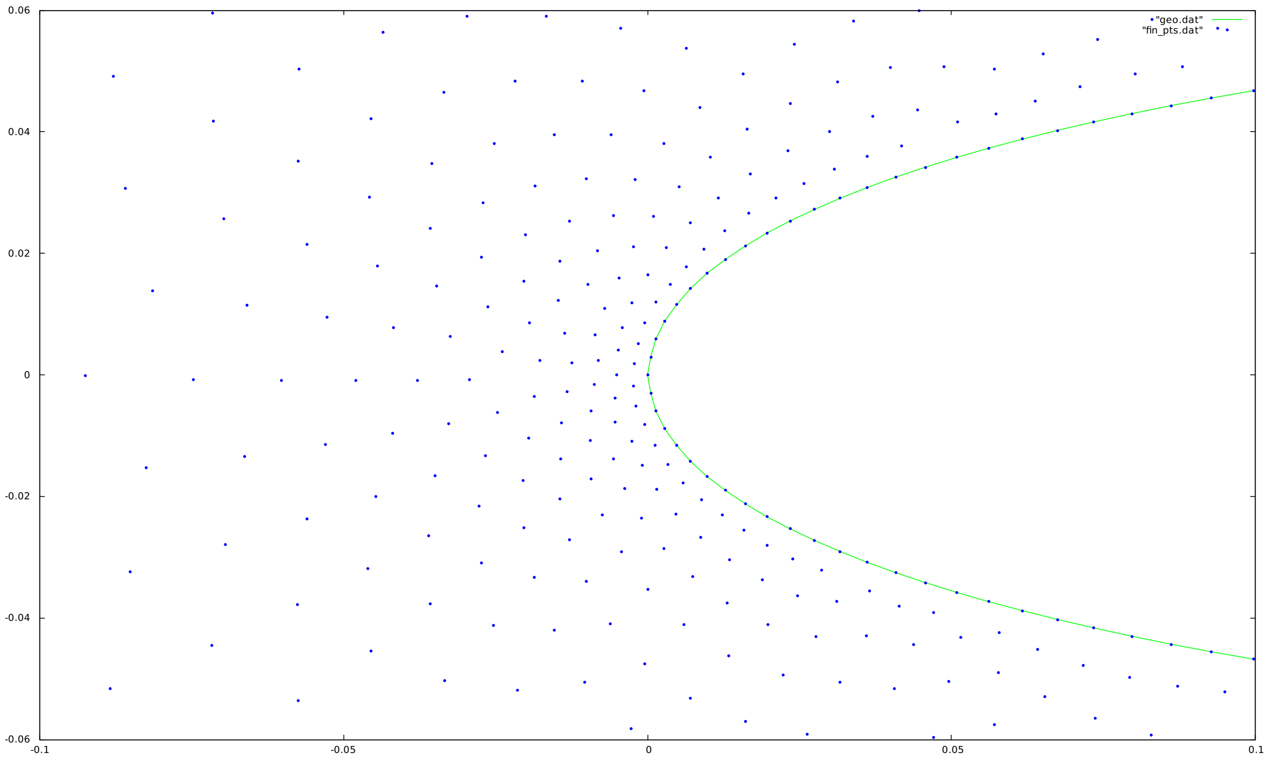
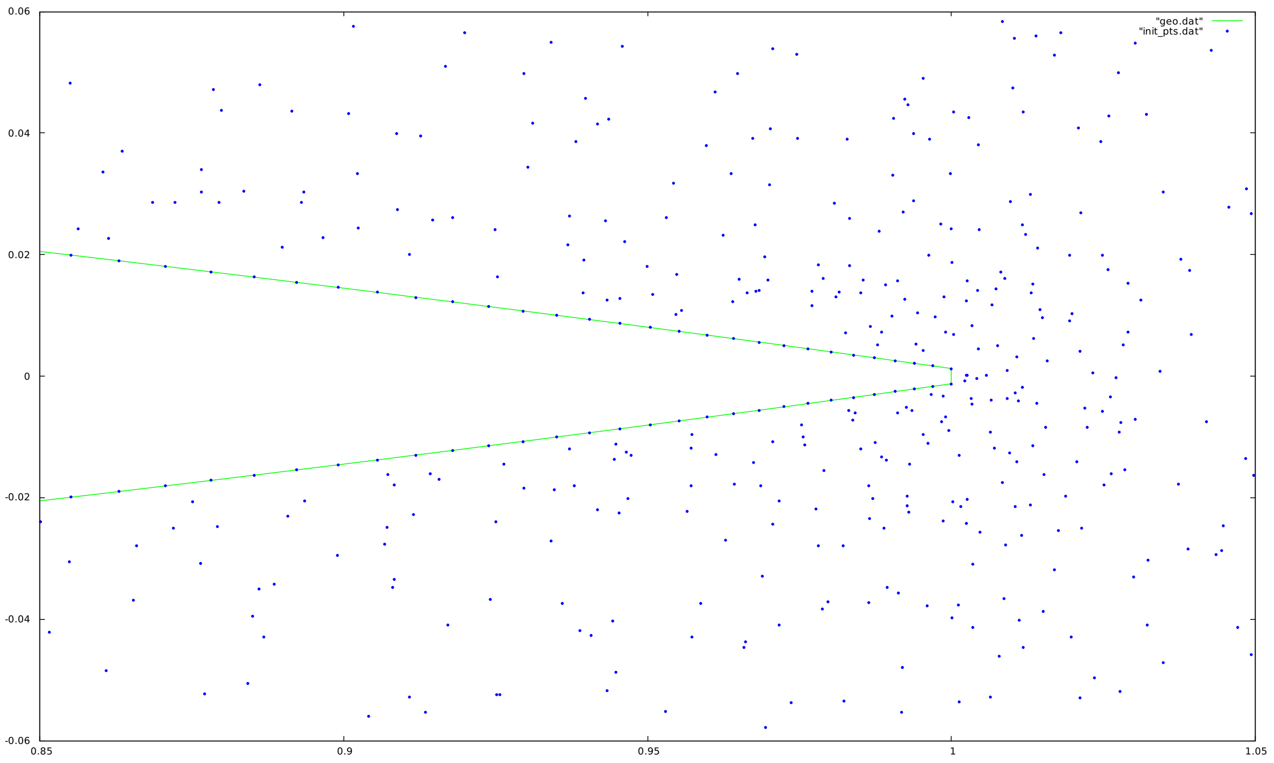


Figure 6.23 NACA 0012 airfoil (zoomed to leading edge): Initial point distribution (*top*) and resulting point distribution (*bottom*) from random initialization



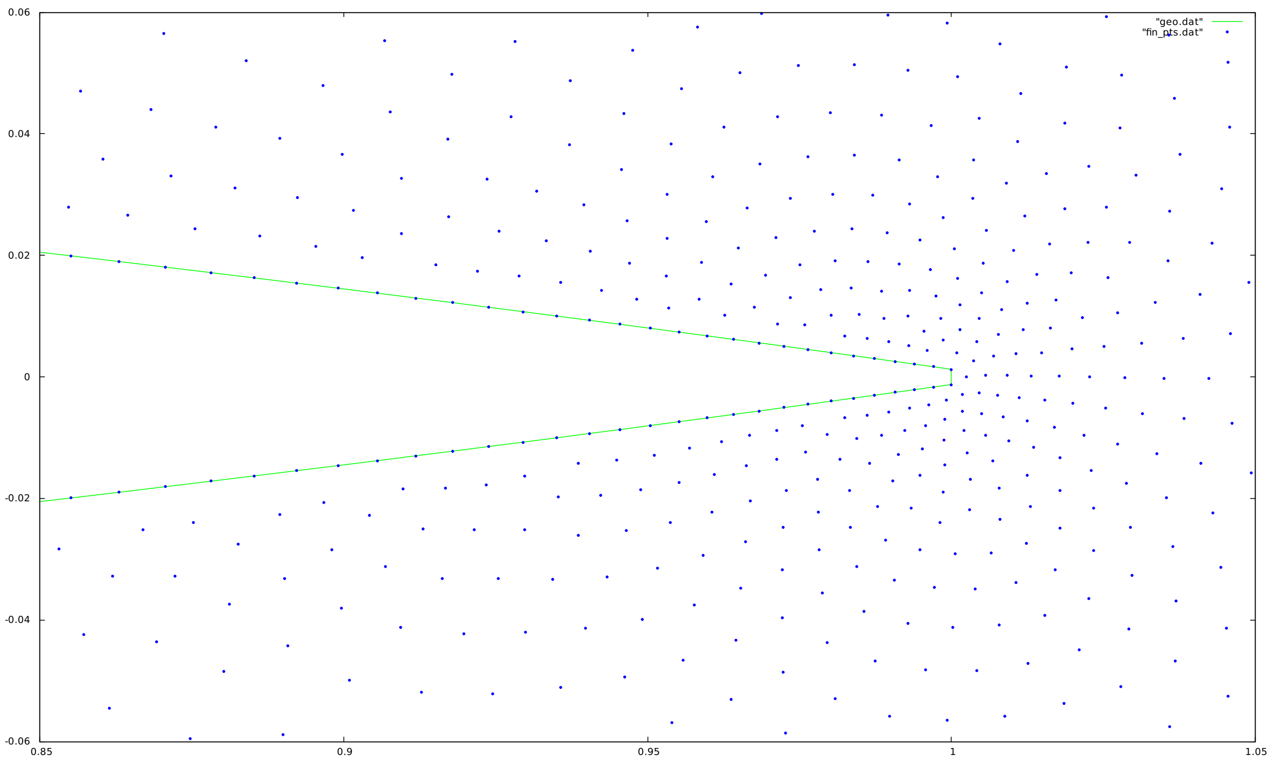


Figure 6.24 NACA 0012 airfoil (zoomed to trailing edge): Initial point distribution (*top*) and resulting point distribution (*bottom*) from random initialization

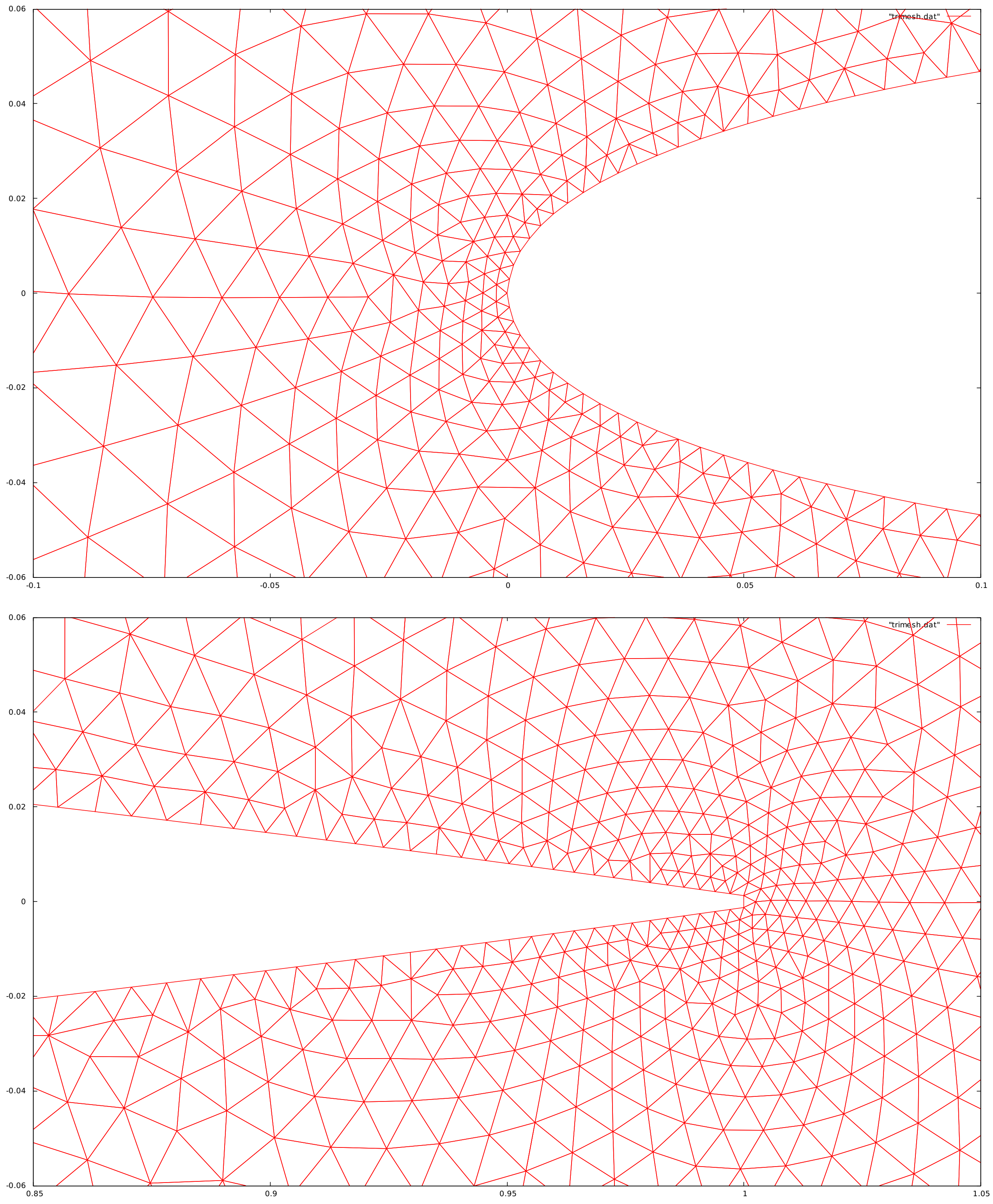


Figure 6.25 NACA 0012 airfoil: Triangulated result zoomed to leading edge (*top*) and trailing edge (*bottom*) from random initialization

30P/30N Multi-element Airfoil

The geometry of this case consists of a 30P/30N multi-element airfoil with a total length of approximately 1.2 surrounded by a 20×20 square outer boundary. This case was run with both the structured (tracked in the first section below) and random (tracked in the second section below) initialization schemes, using an initial maximum time step of 0.009 for both runs.

Structured Initialization

Figure 6.26 shows the trend in the average nodal resultant force and the average nodal kinetic energy from the run starting with the structured initialization.

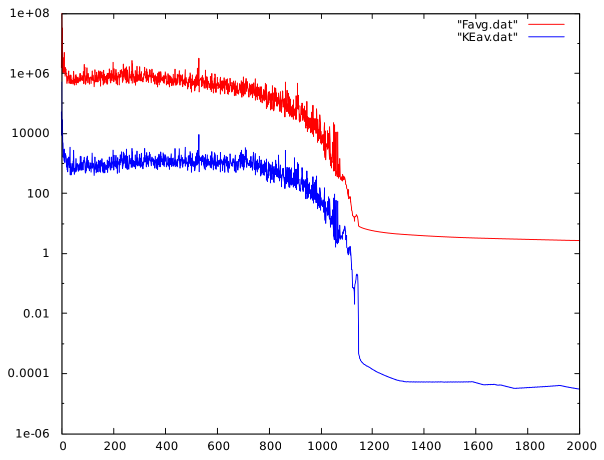
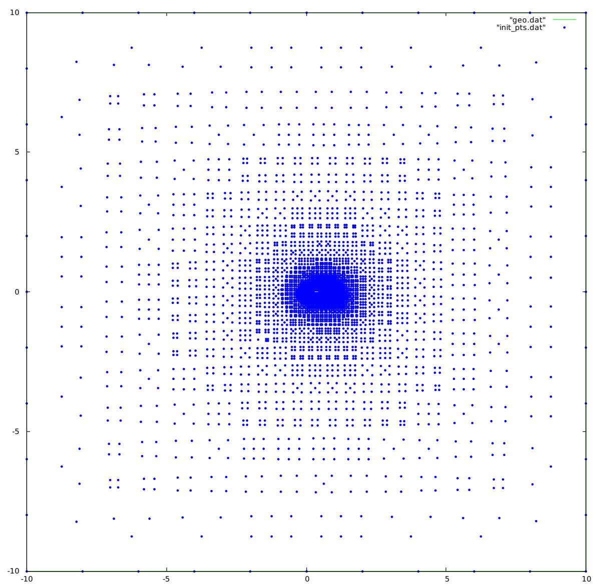


Figure 6.26 30P/30N multi-element airfoil case: Average nodal net force (*red*) and kinetic energy (*blue*) for the structured initialization

Figures 6.27 through 6.36 display the structured initial point distribution, the final point distribution, and the triangulation of the final point distribution for various regions of the domain.



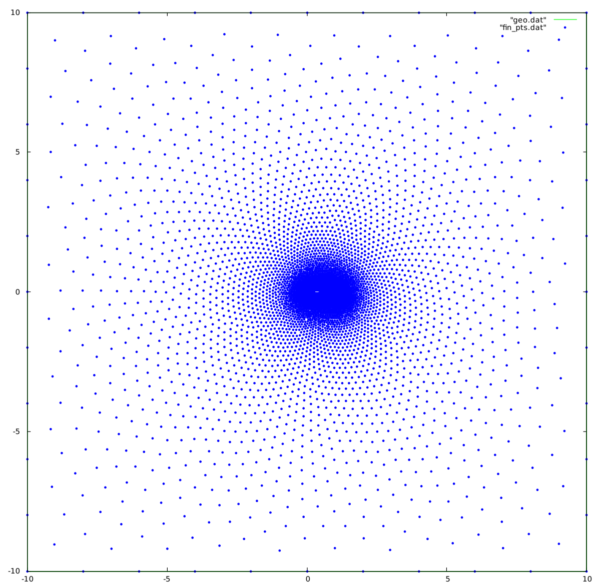


Figure 6.27 30P/30N multi-element airfoil: Initial point distribution (*left*) and resulting point distribution (*right*) from structured initialization

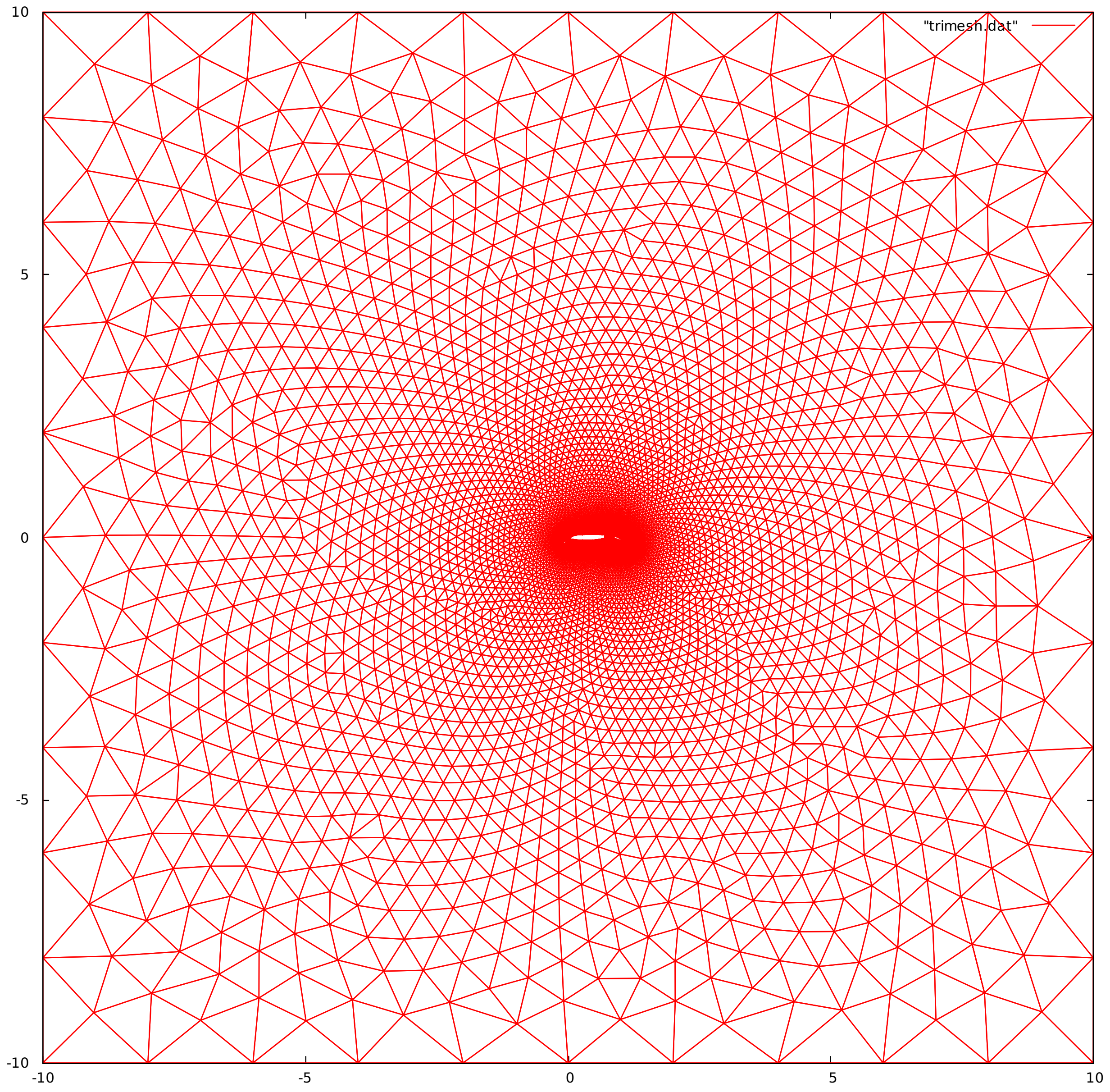
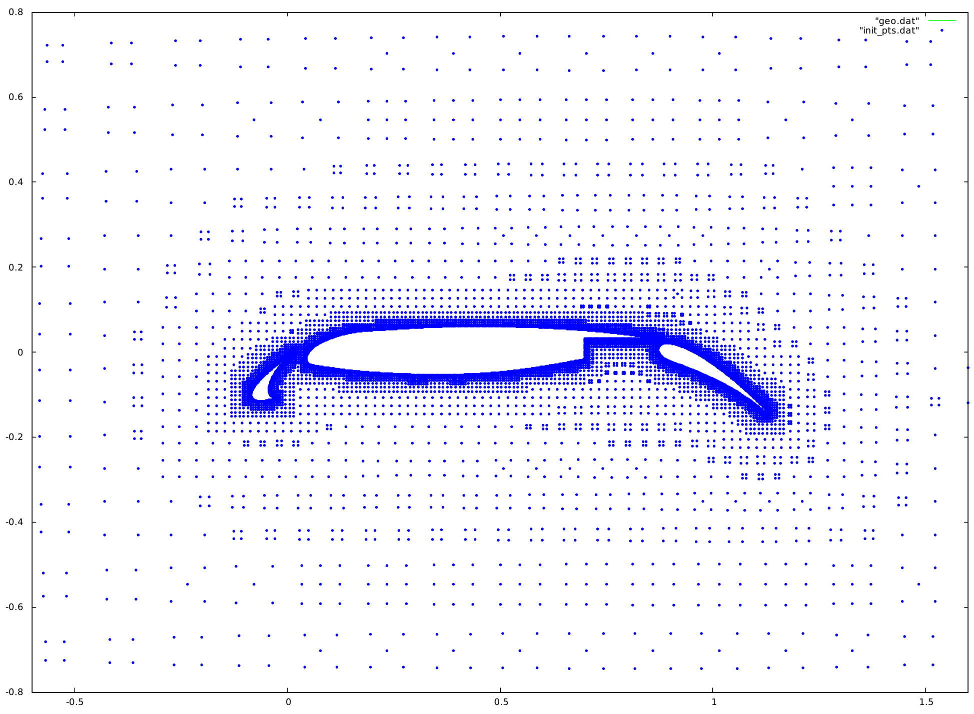


Figure 6.28 30P/30N multi-element airfoil: Triangulated result from structured initialization



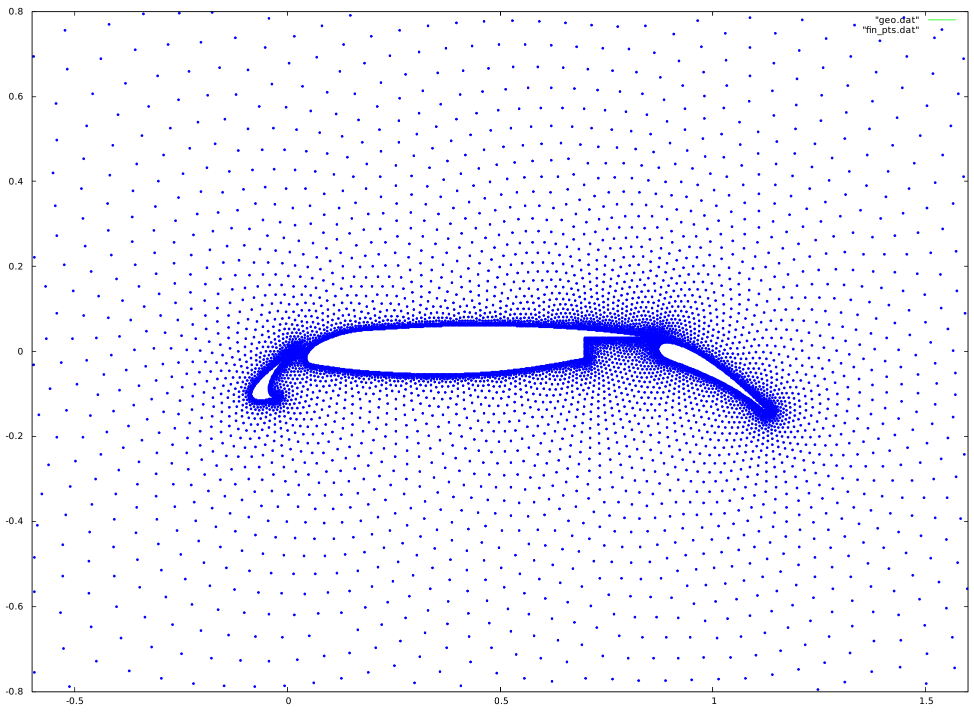


Figure 6.29 30P/30N multi-element airfoil (zoomed midway): Initial point distribution (*top*) and resulting point distribution (*bottom*) from structured initialization

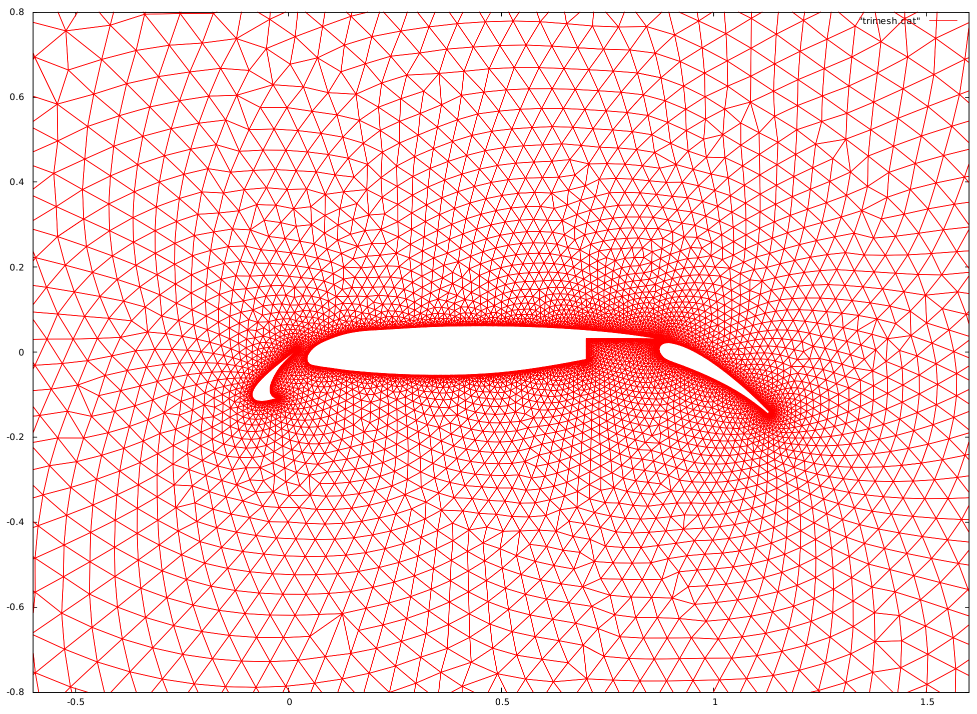
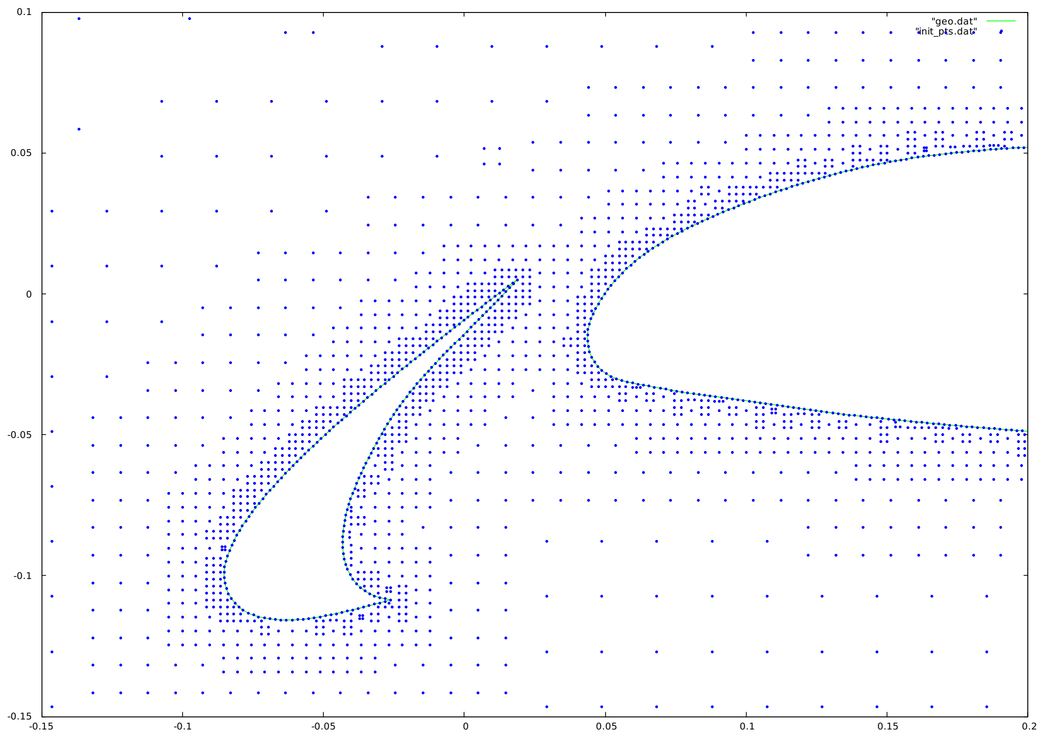


Figure 6.30 30P/30N multi-element airfoil (zoomed midway): Triangulated result from structured initialization



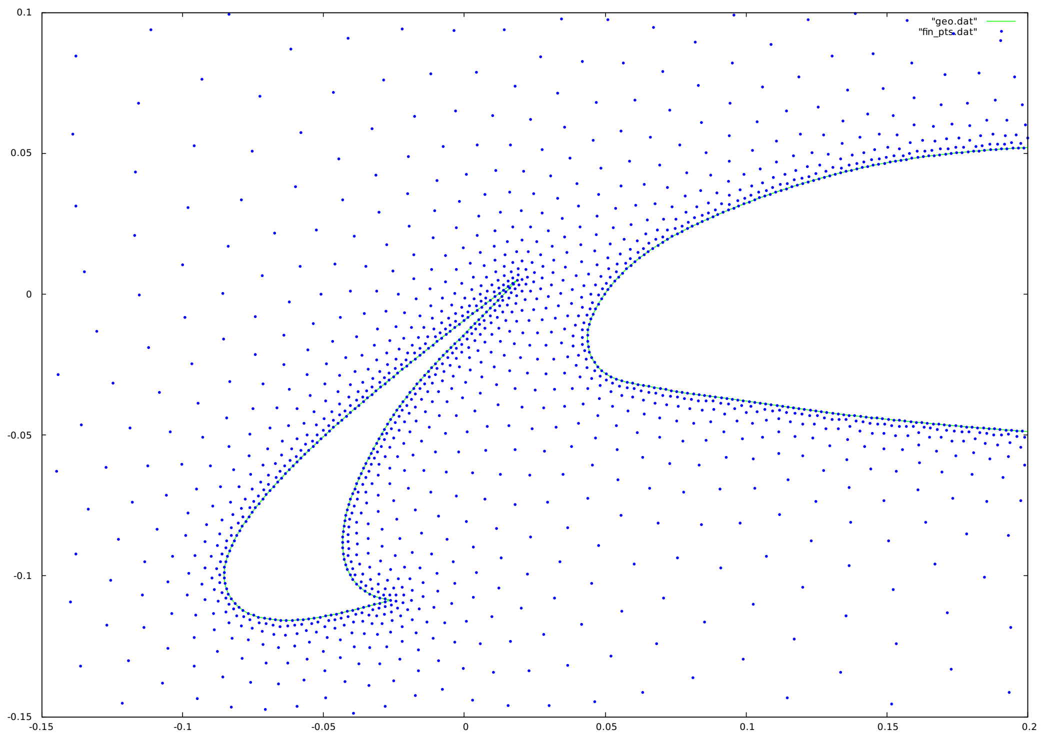


Figure 6.31 30P/30N multi-element airfoil (zoomed to leading edge): Initial point distribution (*top*) and resulting point distribution (*bottom*) from structured initialization

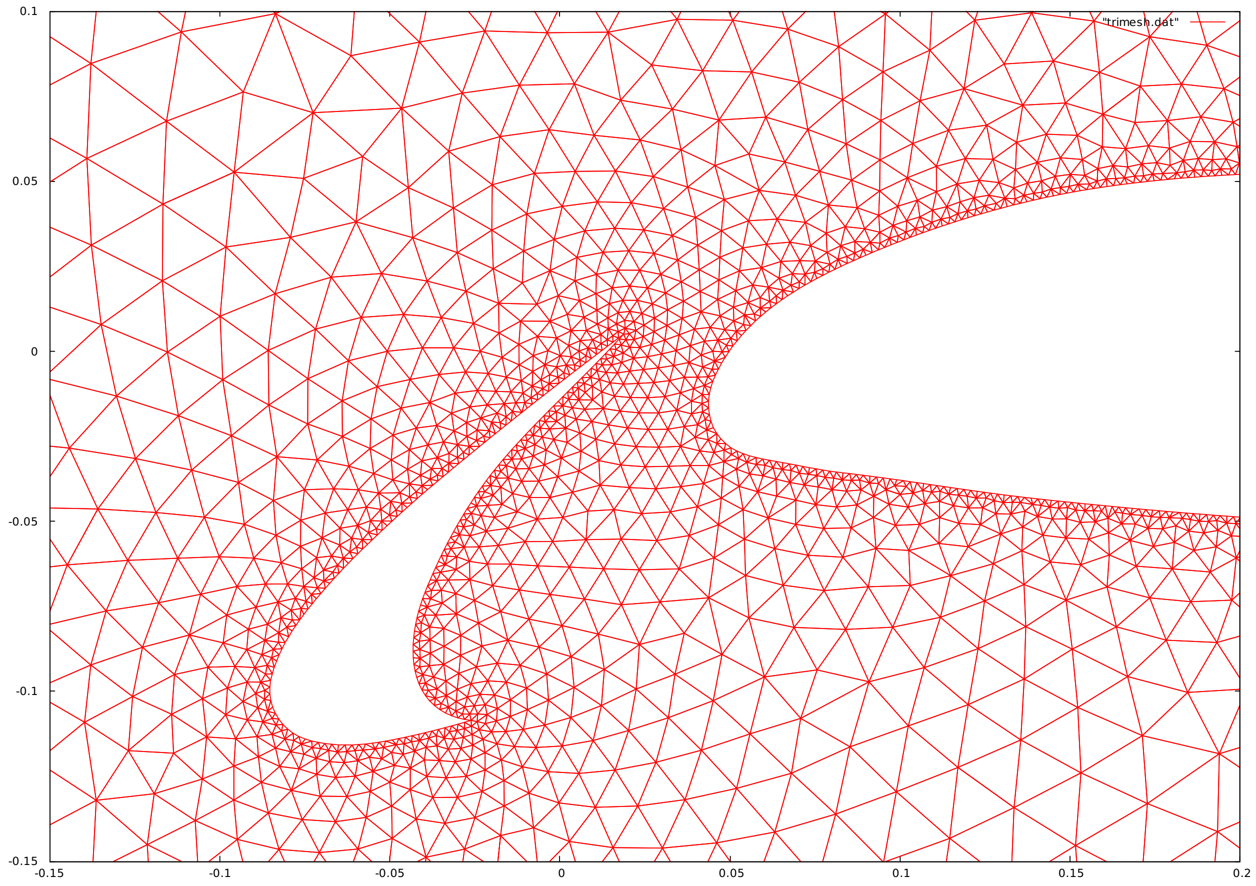


Figure 6.32 30P/30N multi-element airfoil (zoomed to leading edge): Triangulated result from structured initialization

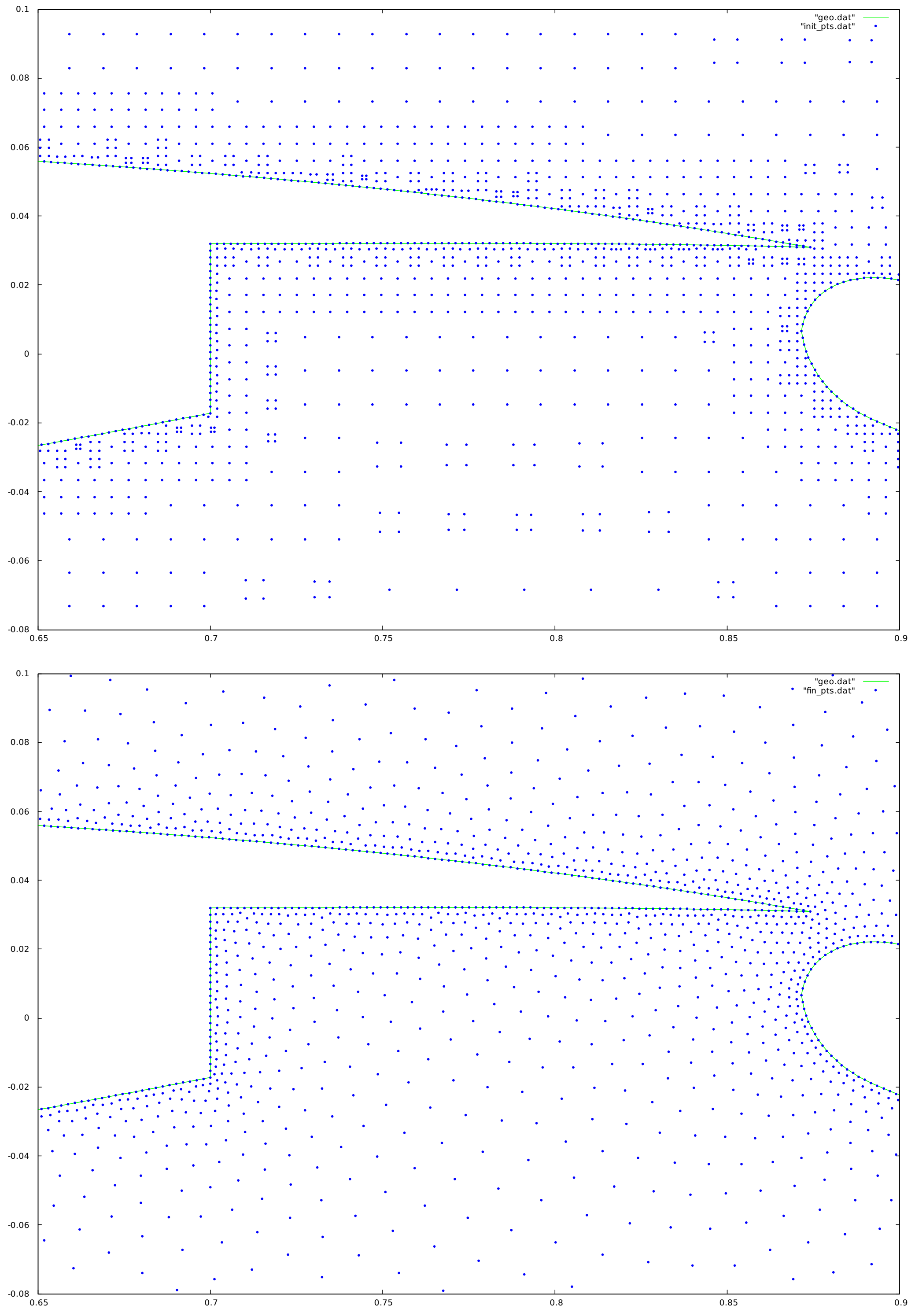


Figure 6.33 30P/30N multi-element airfoil (zoomed to inter-element space): Initial point distribution (*top*) and resulting point distribution (*bottom*) from structured initialization

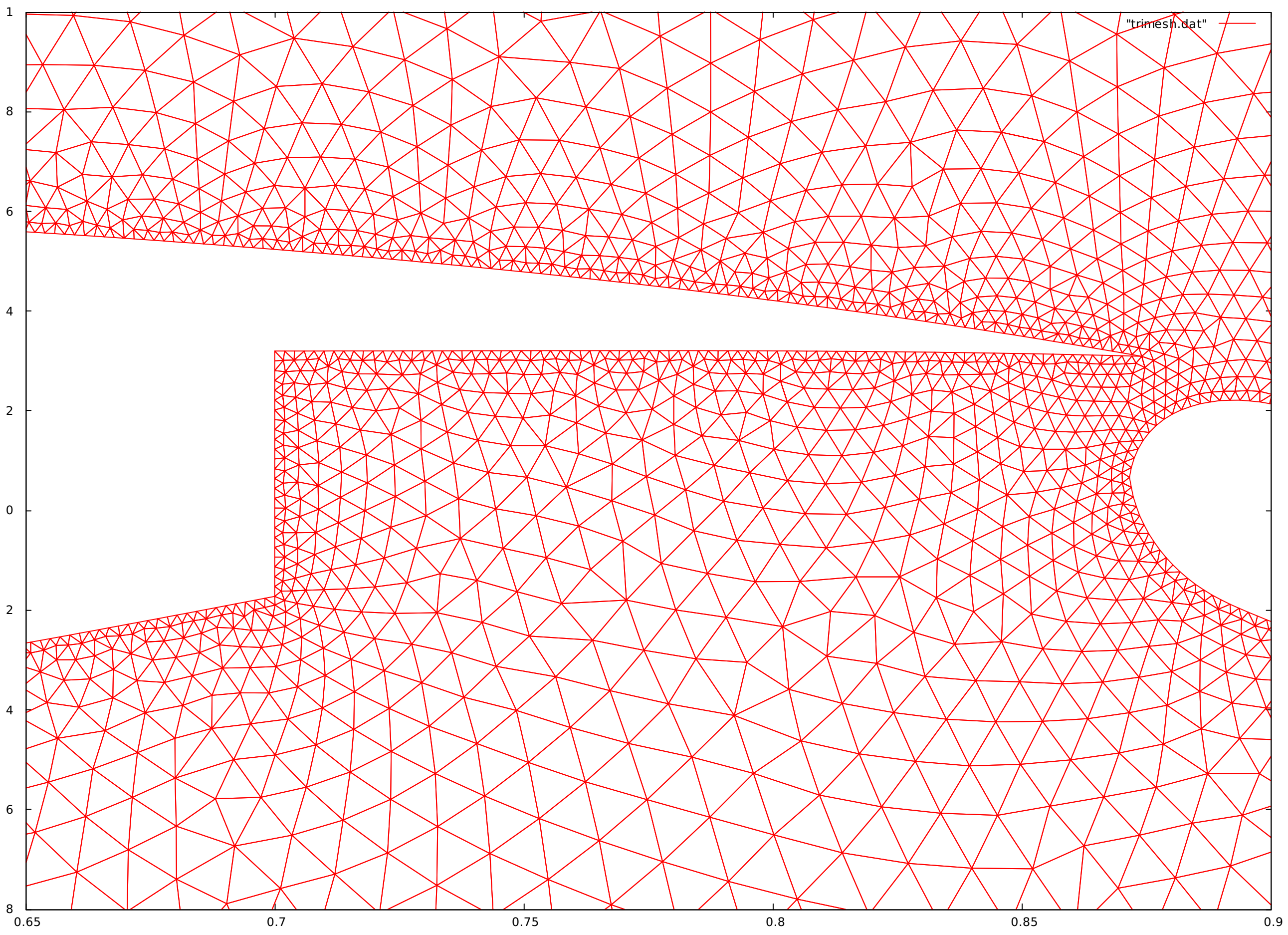
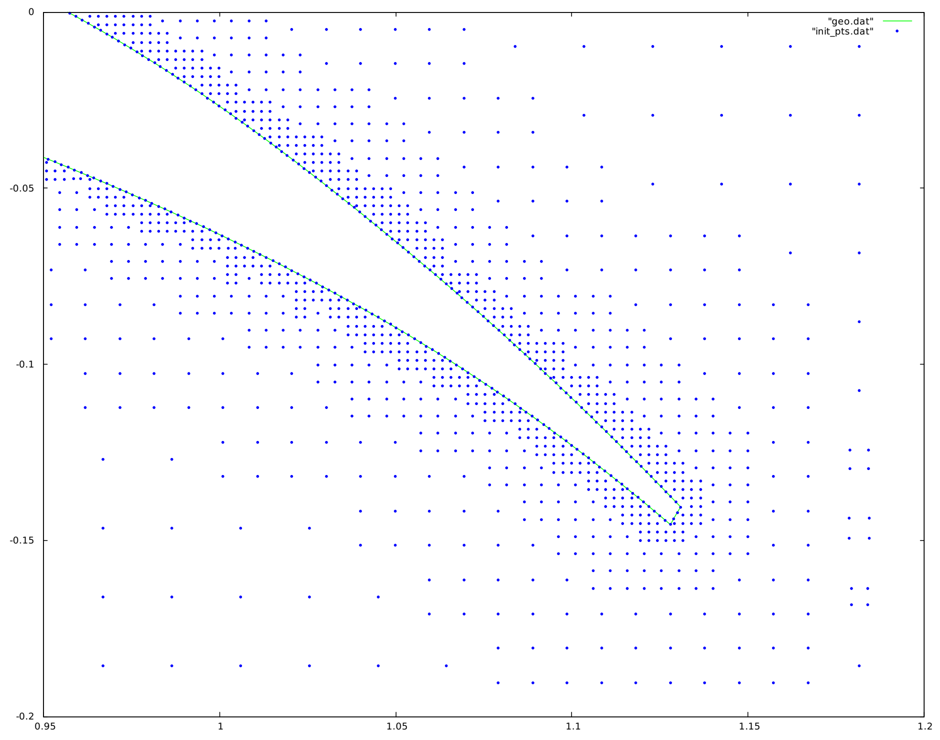


Figure 6.34 30P/30N multi-element airfoil (zoomed to inter-element space): Triangulated result from structured initialization



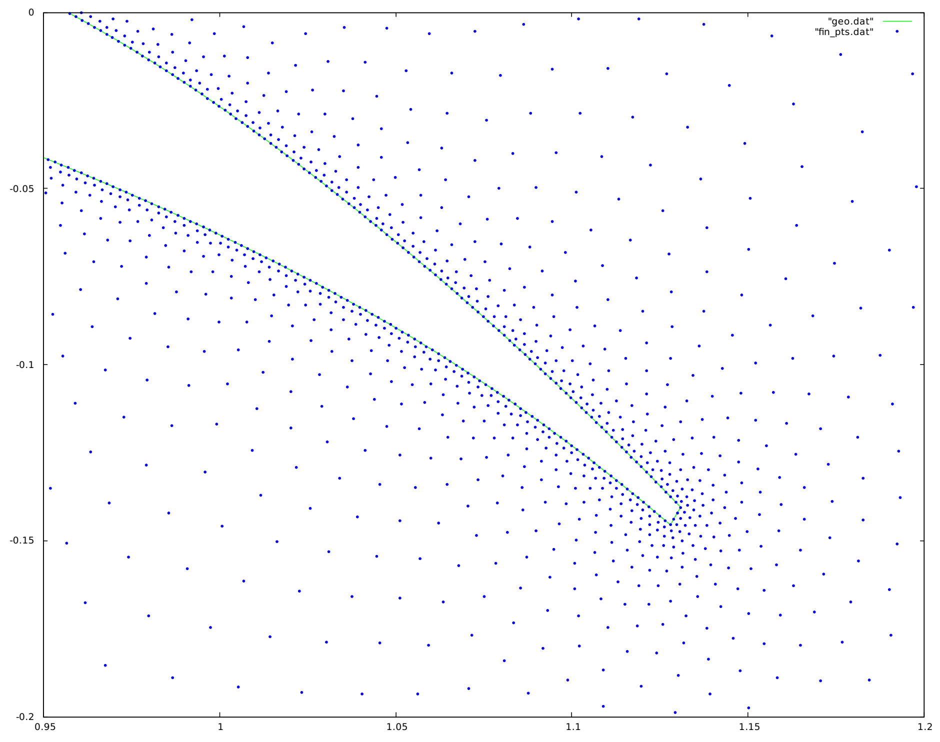


Figure 6.35 30P/30N multi-element airfoil (zoomed to trailing edge): Initial point distribution (*top*) and resulting point distribution (*bottom*) from structured initialization

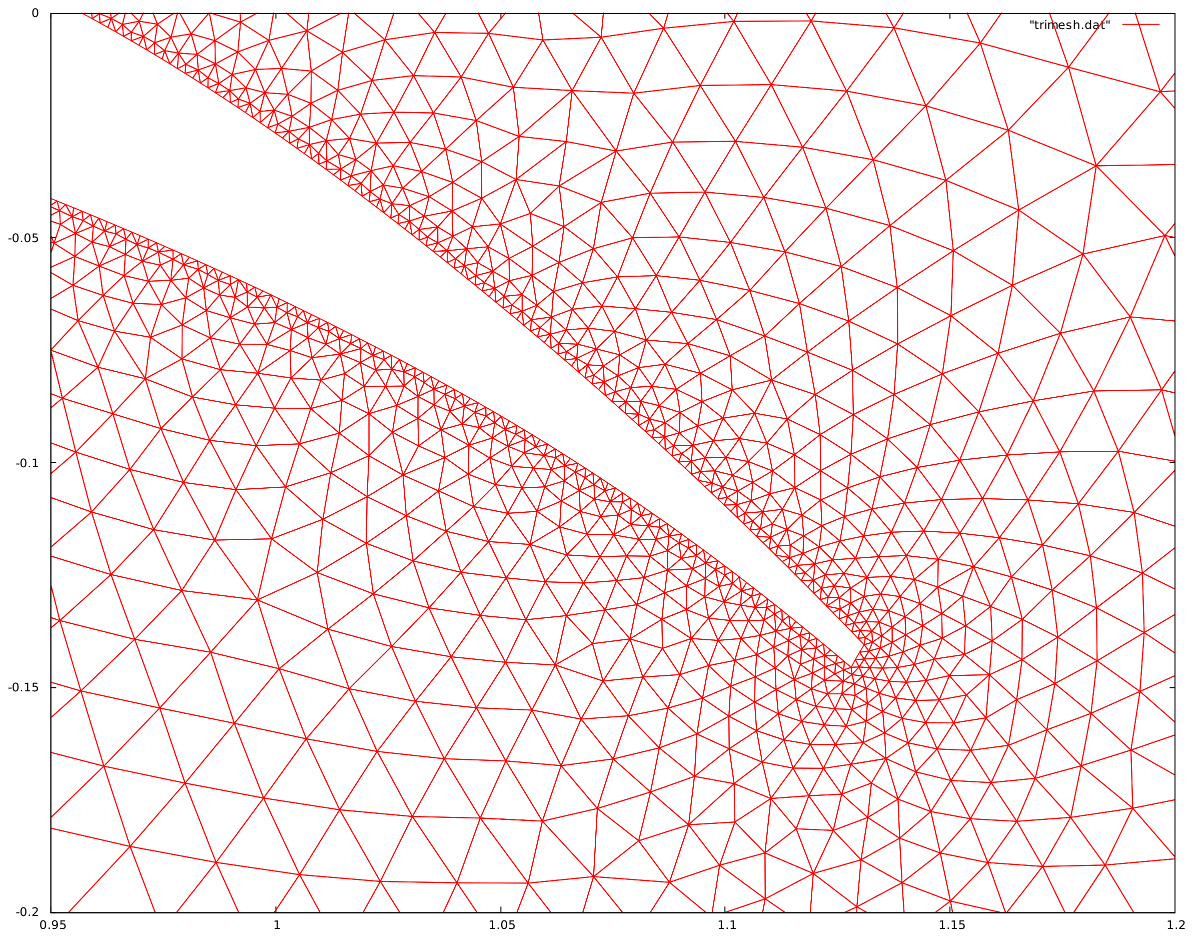


Figure 6.36 30P/30N multi-element airfoil (zoomed to trailing edge): Triangulated result from structured initialization

Random Initialization

Figure 6.37 shows the trend in the average nodal resultant force and the average nodal kinetic energy from the run starting with the random initialization.

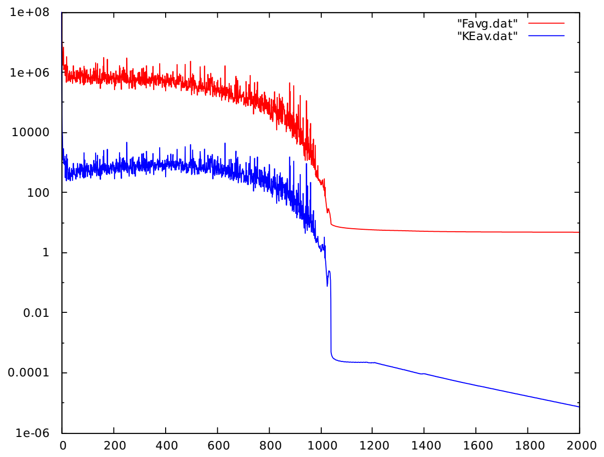


Figure 6.37 30P/30N multi-element airfoil case: Average nodal net force (*red*) and kinetic energy (*blue*) for the random initialization

Figures 6.38 through 6.47 display the randomized initial point distribution, the final point distribution, and the triangulation of the final point distribution for various regions of the domain, the same as shown in the previous section covering the results from the structured initialization.

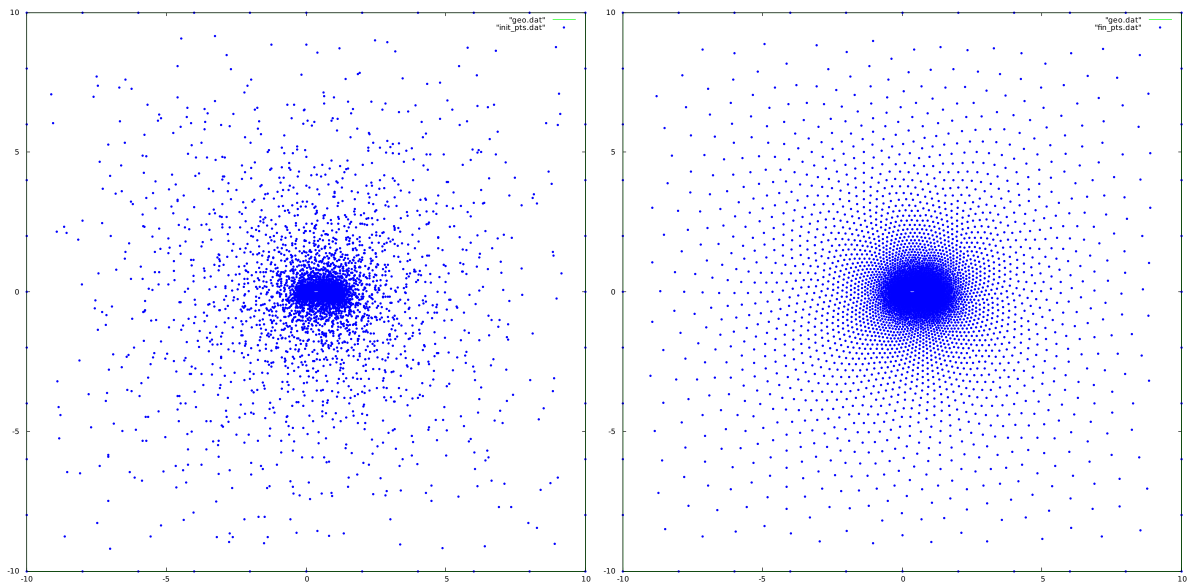


Figure 6.38 30P/30N multi-element airfoil: Initial point distribution (*left*) and resulting point distribution (*right*) from random initialization

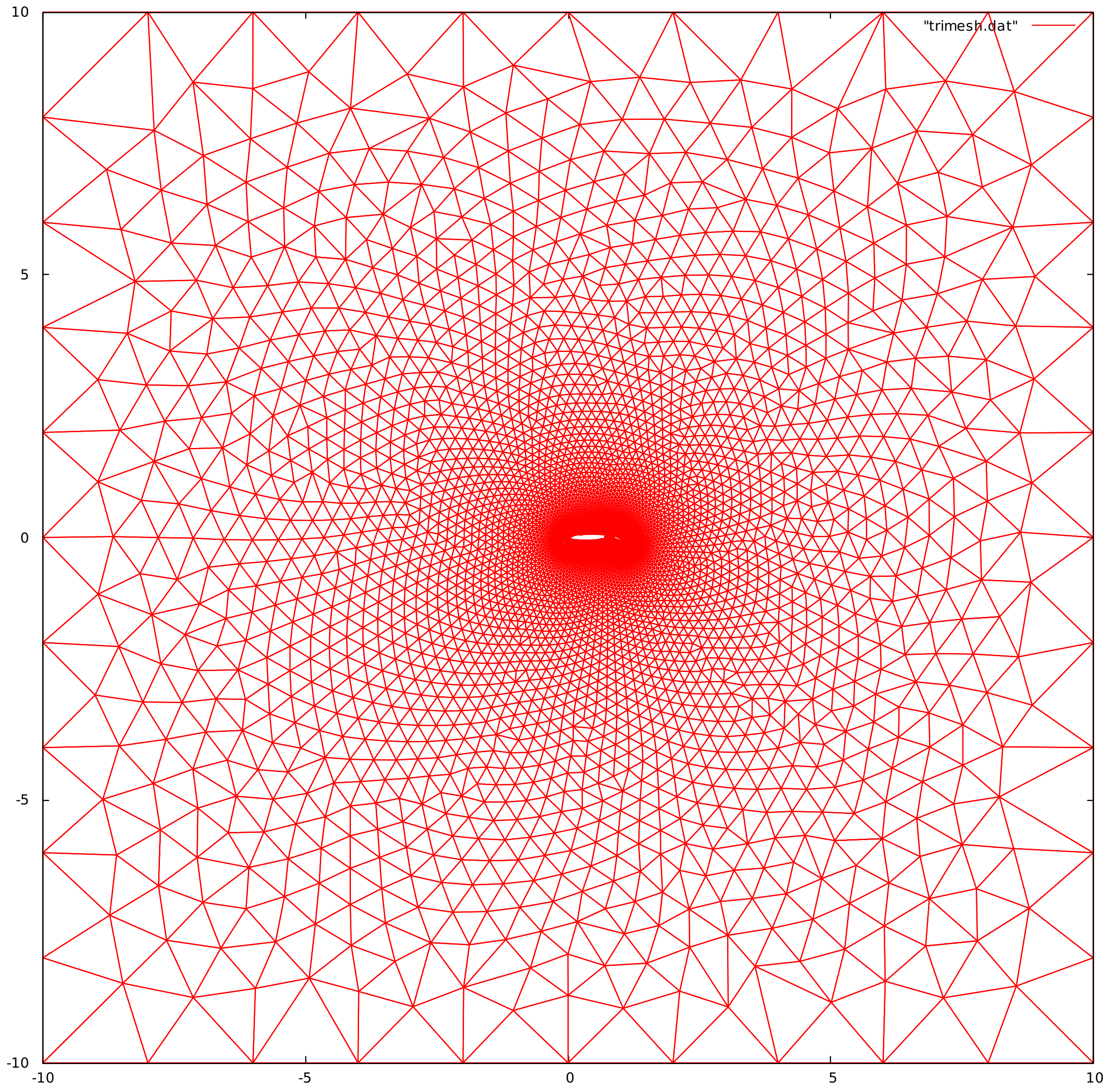
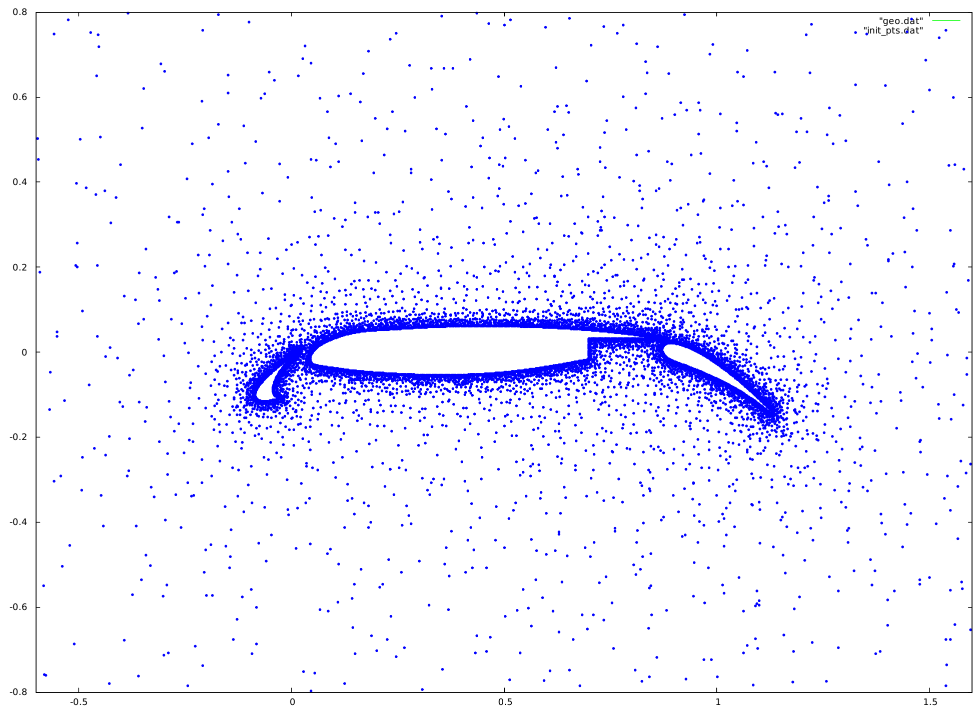


Figure 6.39 30P/30N multi-element airfoil: Triangulated result from random initialization



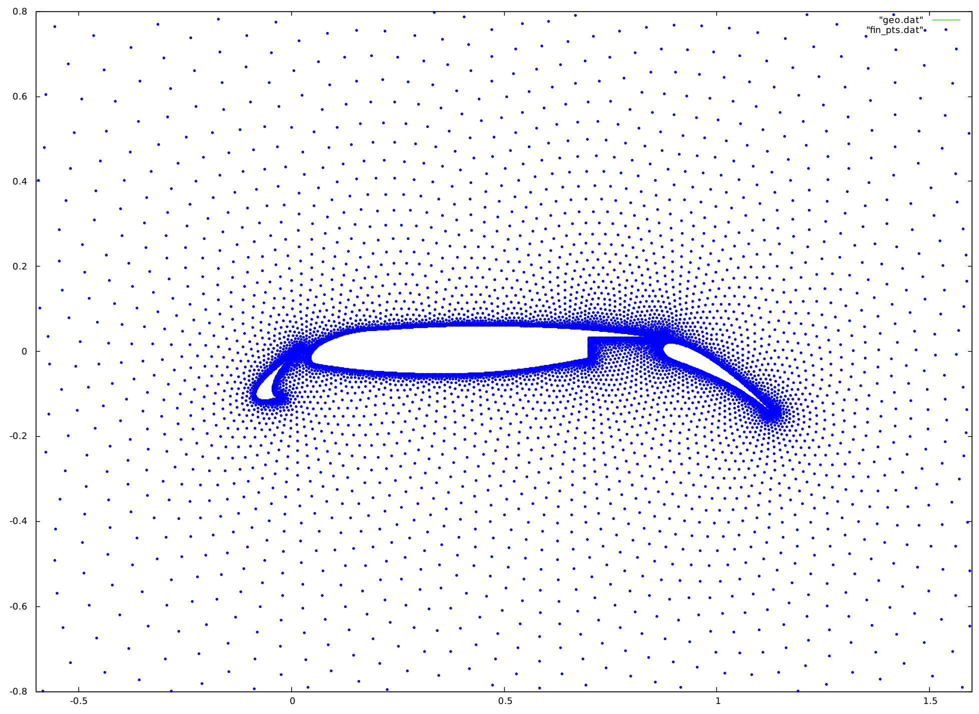


Figure 6.40 30P/30N multi-element airfoil (zoomed midway): Initial point distribution (*top*) and resulting point distribution (*bottom*) from random initialization

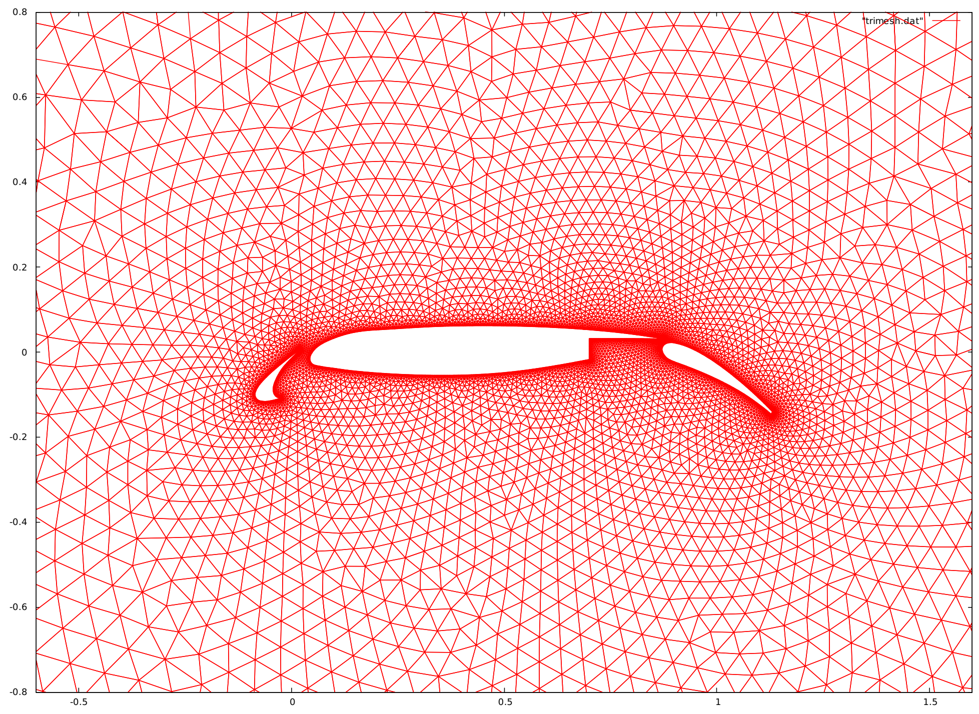
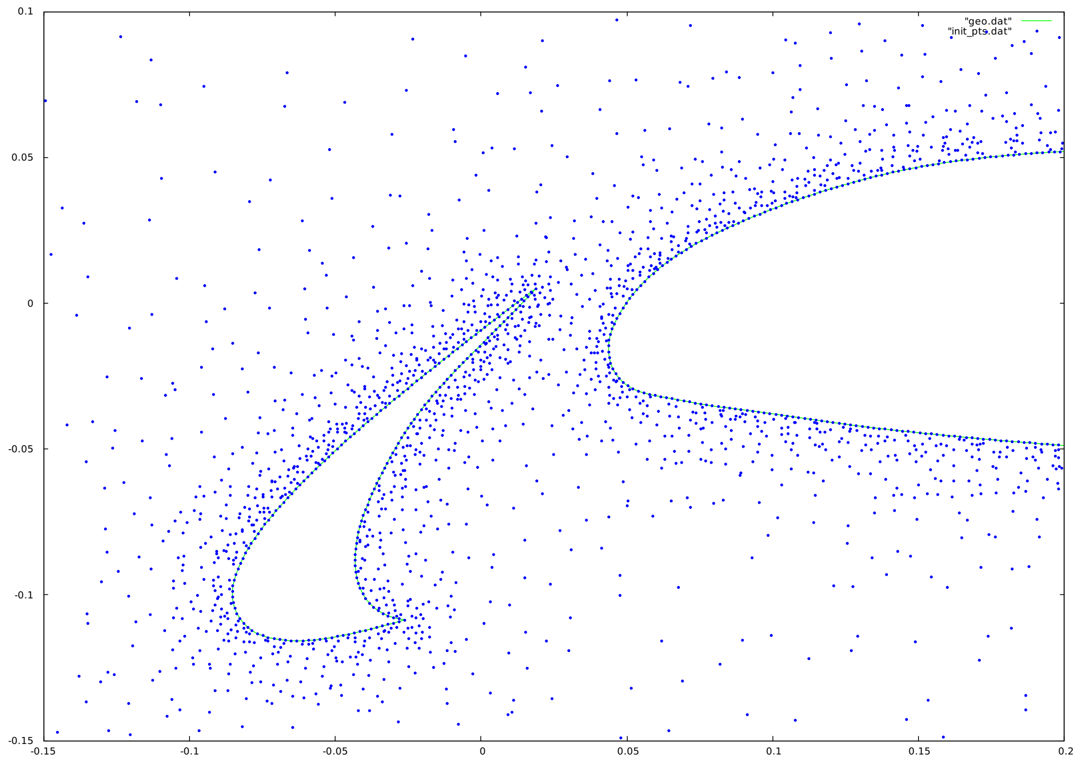


Figure 6.41 30P/30N multi-element airfoil (zoomed midway): Triangulated result from random initialization



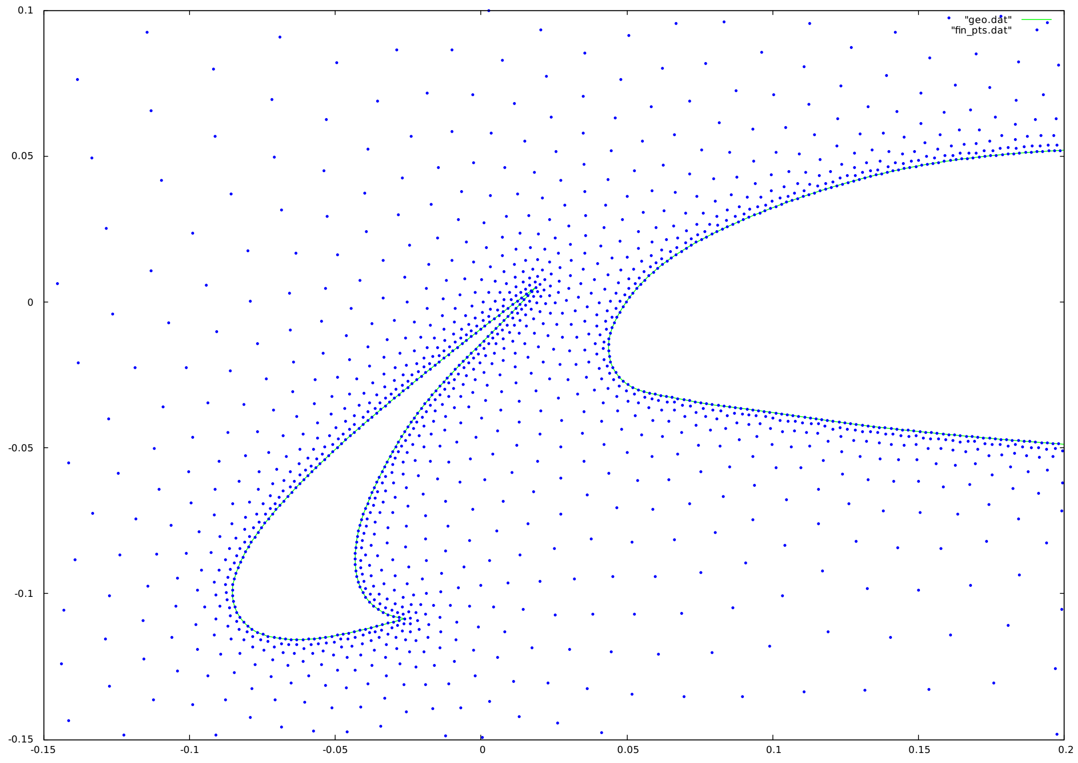


Figure 6.42 30P/30N multi-element airfoil (zoomed to leading edge): Initial point distribution (*top*) and resulting point distribution (*bottom*) from random initialization

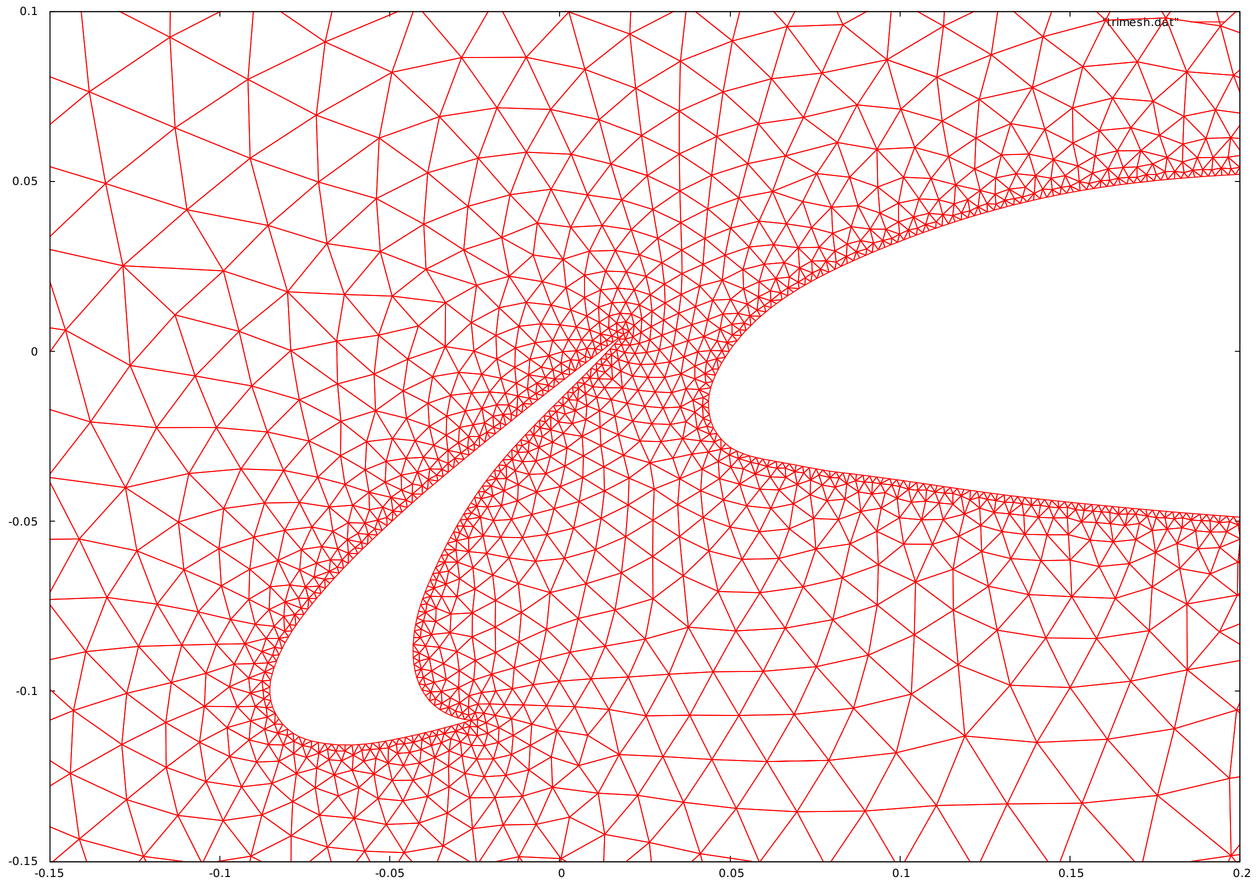
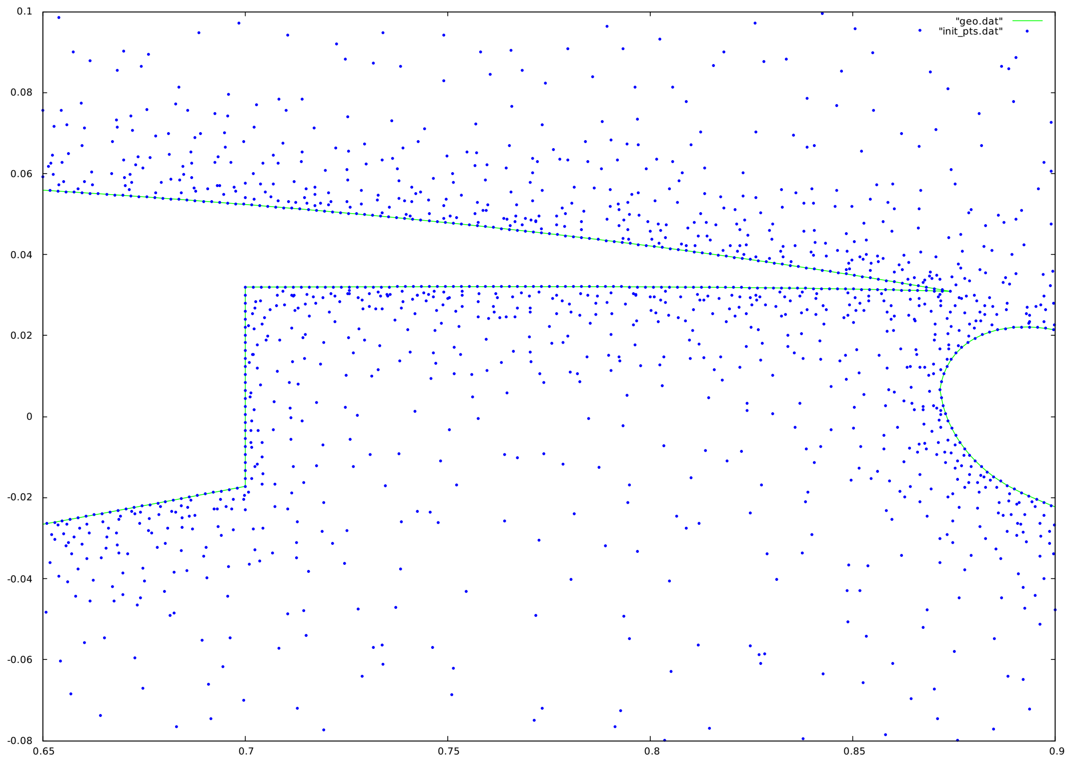


Figure 6.43 30P/30N multi-element airfoil (zoomed to leading edge): Triangulated result from random initialization



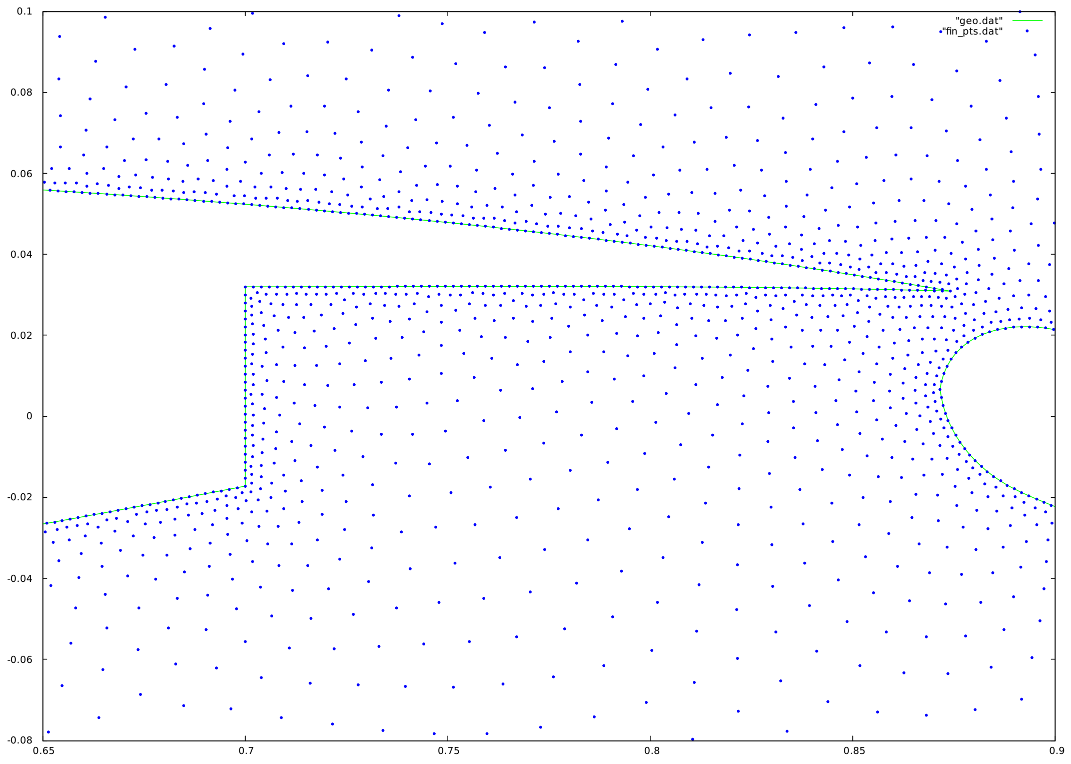


Figure 6.44 30P/30N multi-element airfoil (zoomed to inter-element space): Initial point distribution (*top*) and resulting point distribution (*bottom*) from random initialization

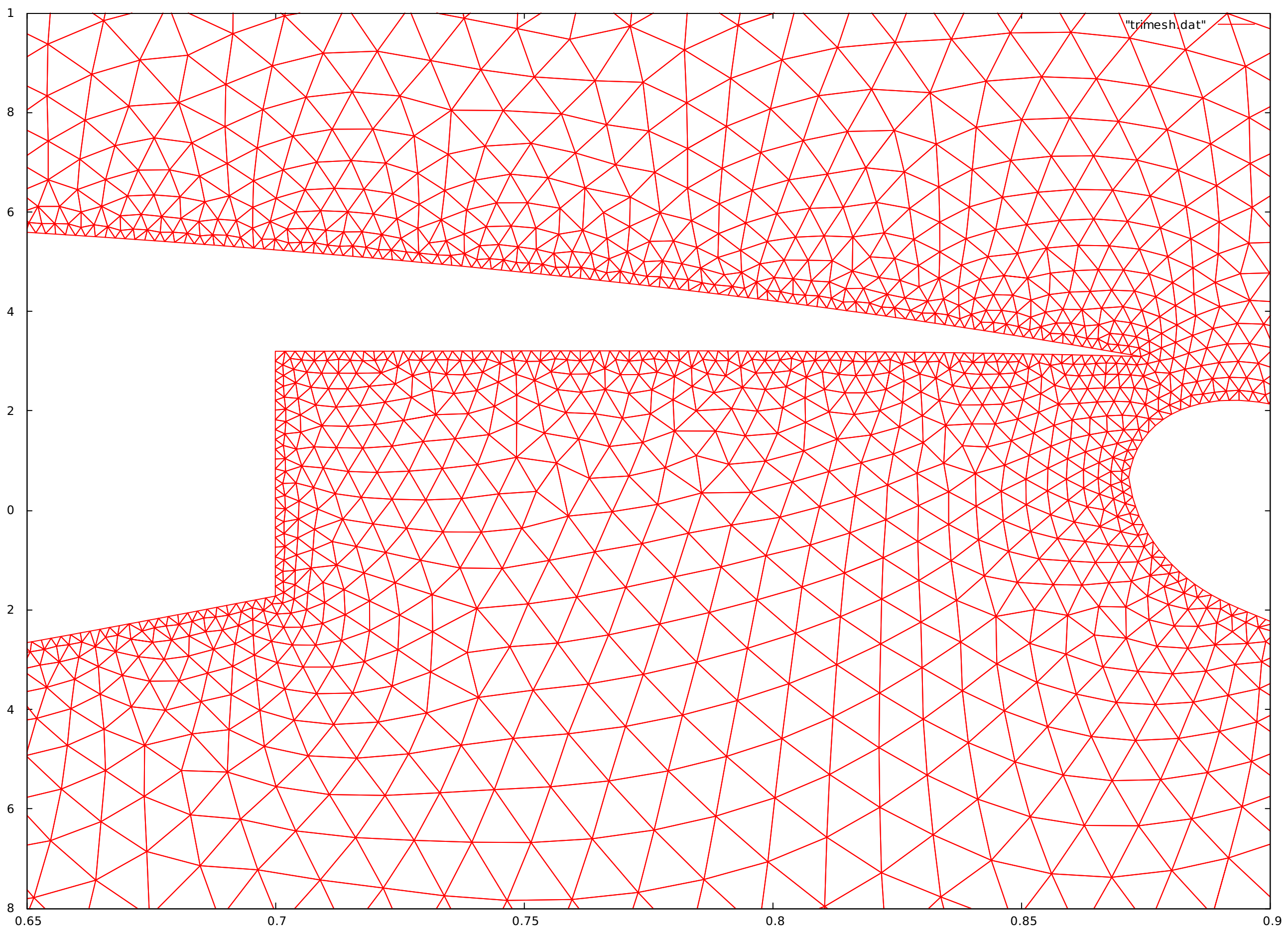
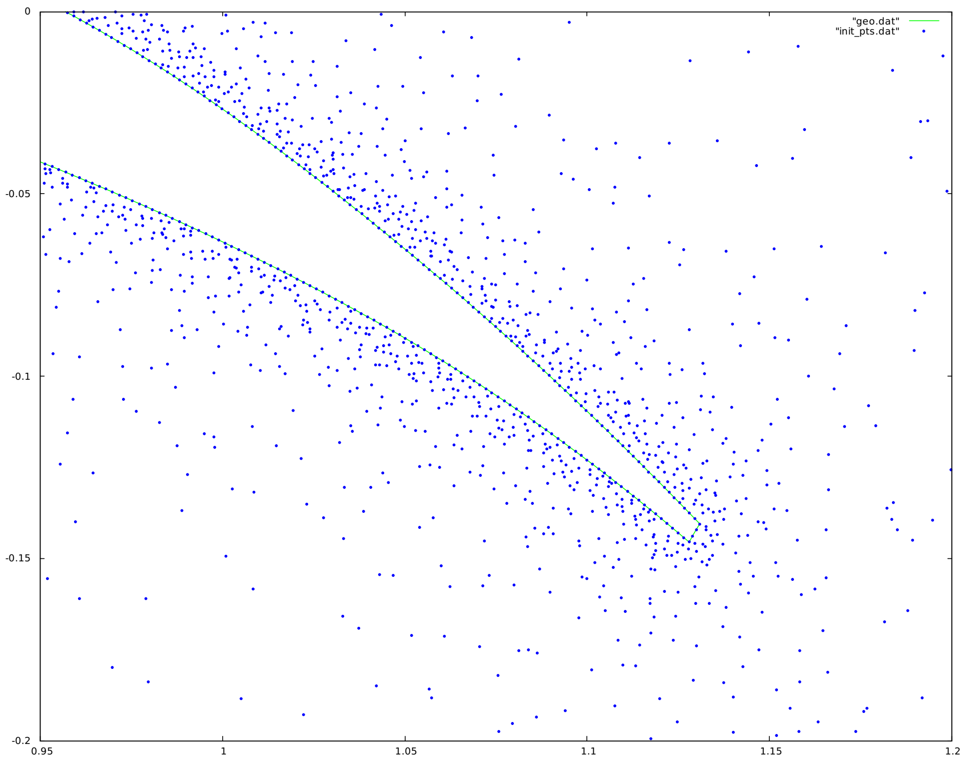


Figure 6.45 30P/30N multi-element airfoil (zoomed to inter-element space): Triangulated result from random initialization



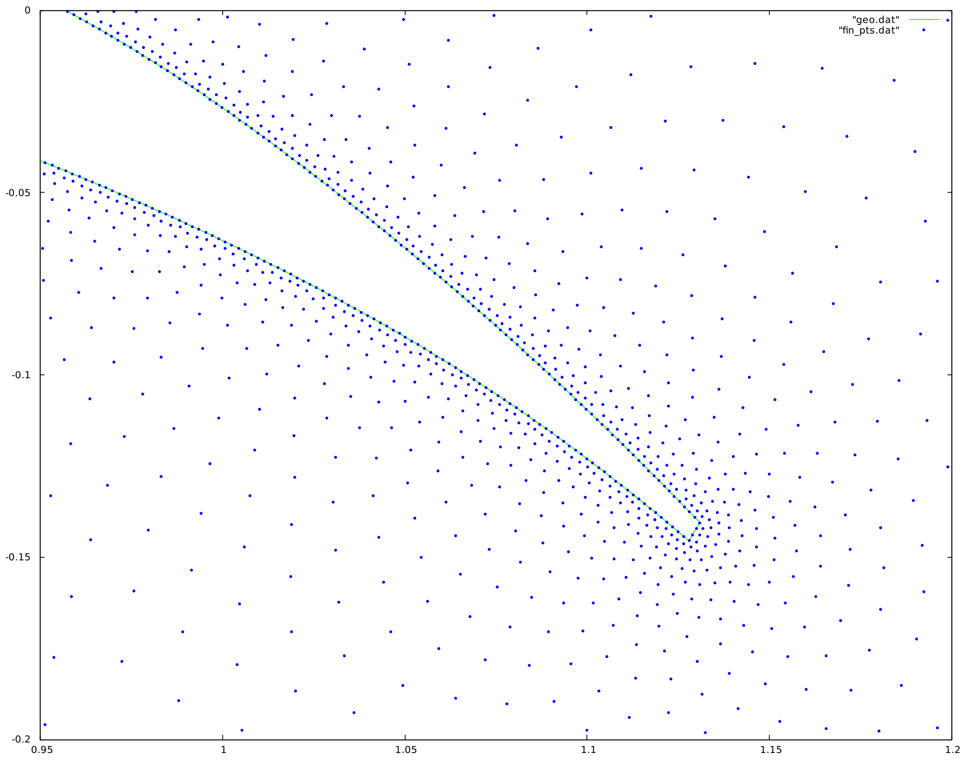


Figure 6.46 30P/30N multi-element airfoil (zoomed to trailing edge): Initial point distribution (*top*) and resulting point distribution (*bottom*) from random initialization

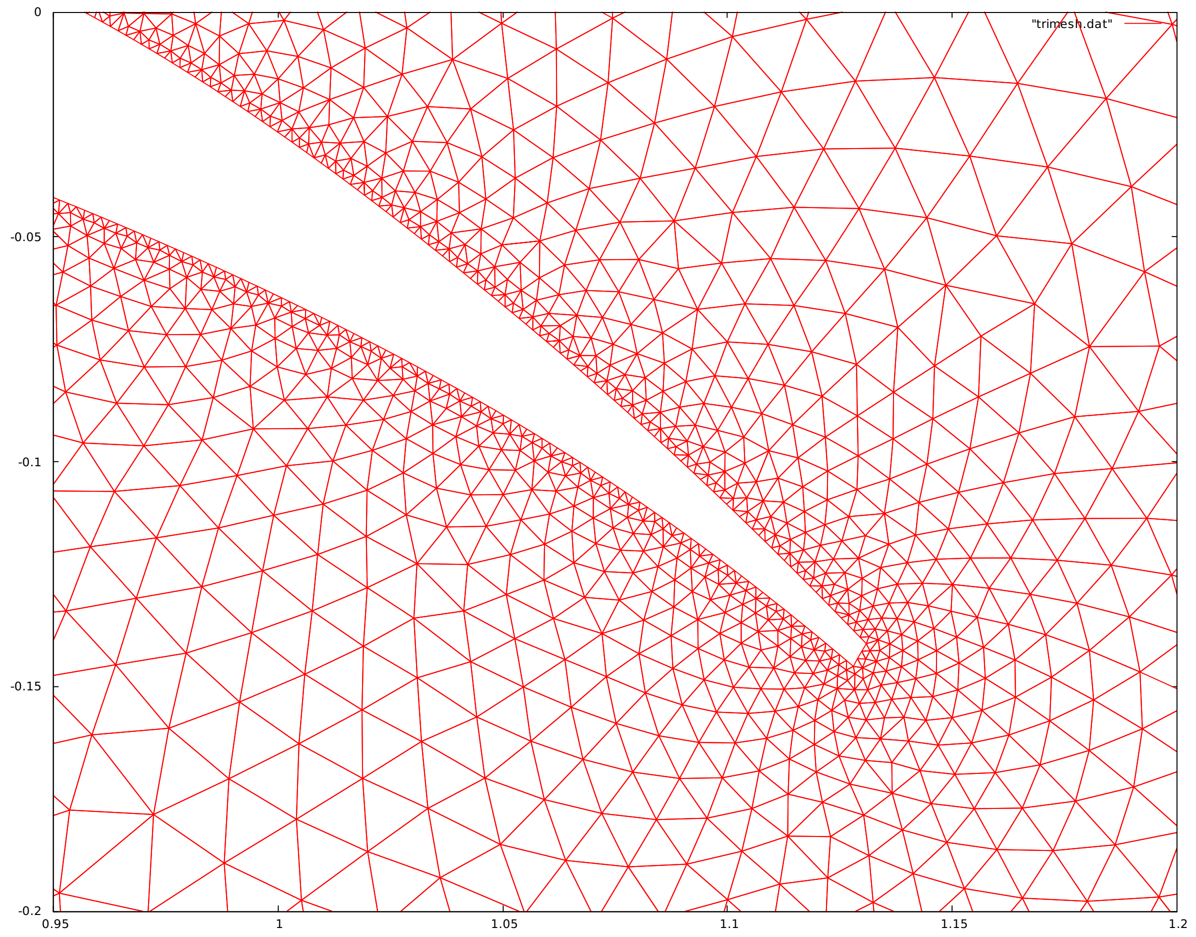


Figure 6.47 30P/30N multi-element airfoil (zoomed to trailing edge): Triangulated result from random initialization

Nuclear Reactor Rod Assembly

This geometry is comprised of 25 rod cross-sections in a 5×5 array inside a square outer boundary. For this geometry we ran two cases, one with a coarse boundary distribution and one with a fine boundary distribution (recall that the spacing function used in this research is entirely dependent upon the spacing on the boundaries).

Coarse Version

The coarse version of this geometry has 20 segments on each circle and 40 on each side of the outer square. This case was run with an initial maximum time step of 0.0025 for the structured initialization and 0.005 for the random initialization. Figure 6.48 shows the trend in the average nodal resultant force and the average nodal kinetic energy for each of the runs.

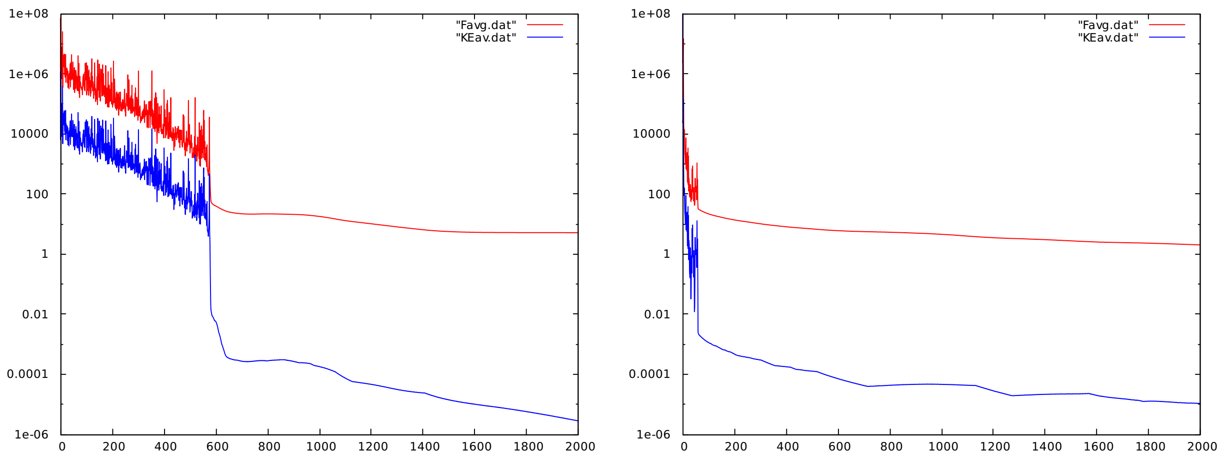


Figure 6.48 Reactor rod assembly (coarse) case: Average nodal net force (*red*) and kinetic energy (*blue*) for the structured (*left*) and random (*right*) initializations

Figures 6.49 through 6.54 provide comparative views of the initial distribution to the final distribution for each run for various regions of the domain. Figures 6.55 through 6.57 show the triangulation of each resultant distribution for the same regions of the domain.

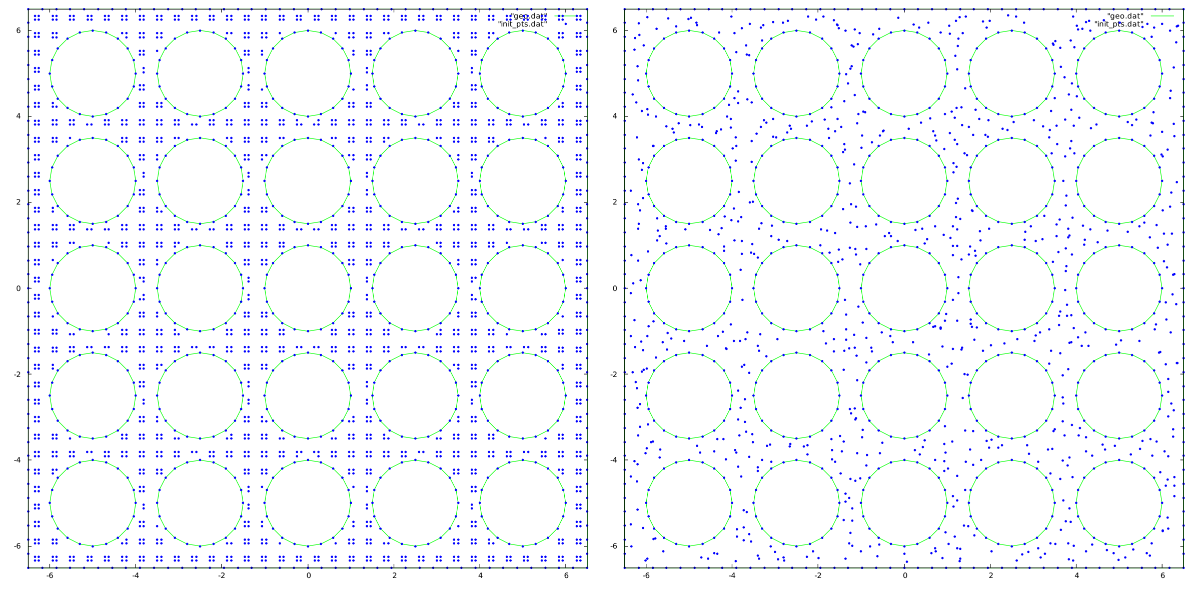


Figure 6.49 Reactor rod assembly (coarse): Initial point distribution from structured (*left*) and random (*right*) initializations

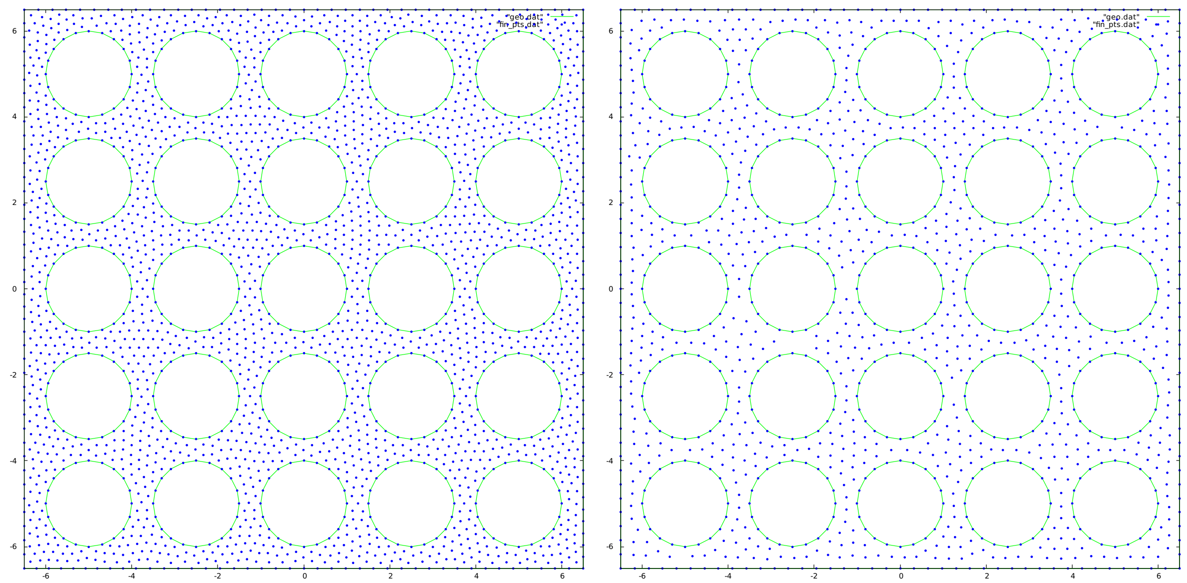


Figure 6.50 Reactor rod assembly (coarse): Resulting point distribution from structured (*left*) and random (*right*) initializations

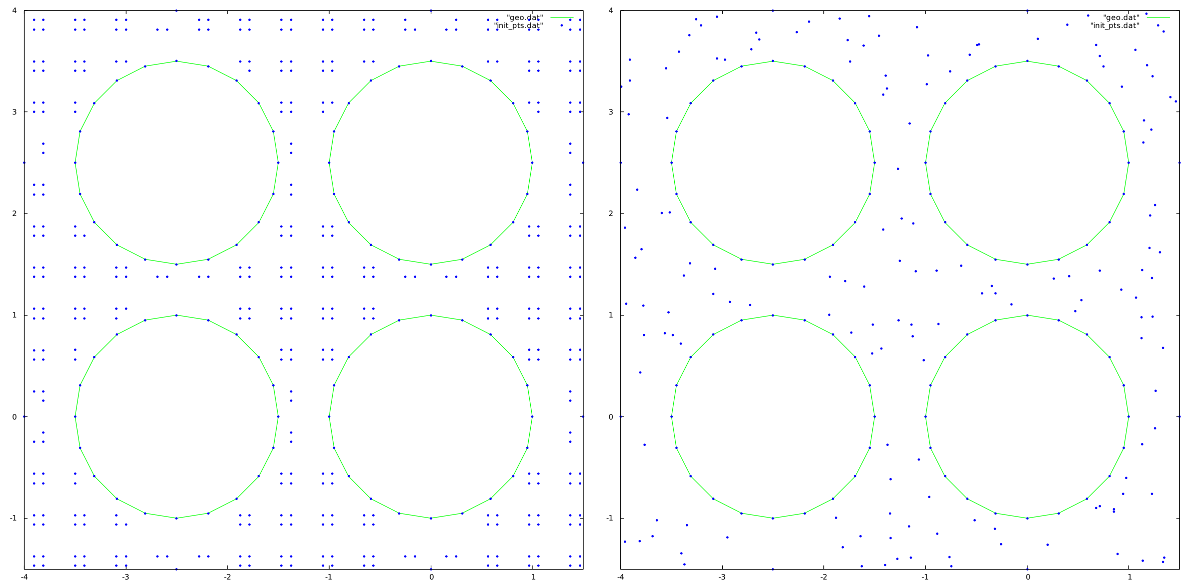


Figure 6.51 Reactor rod assembly (coarse) (zoomed): Initial point distribution from structured (*left*) and random (*right*) initializations

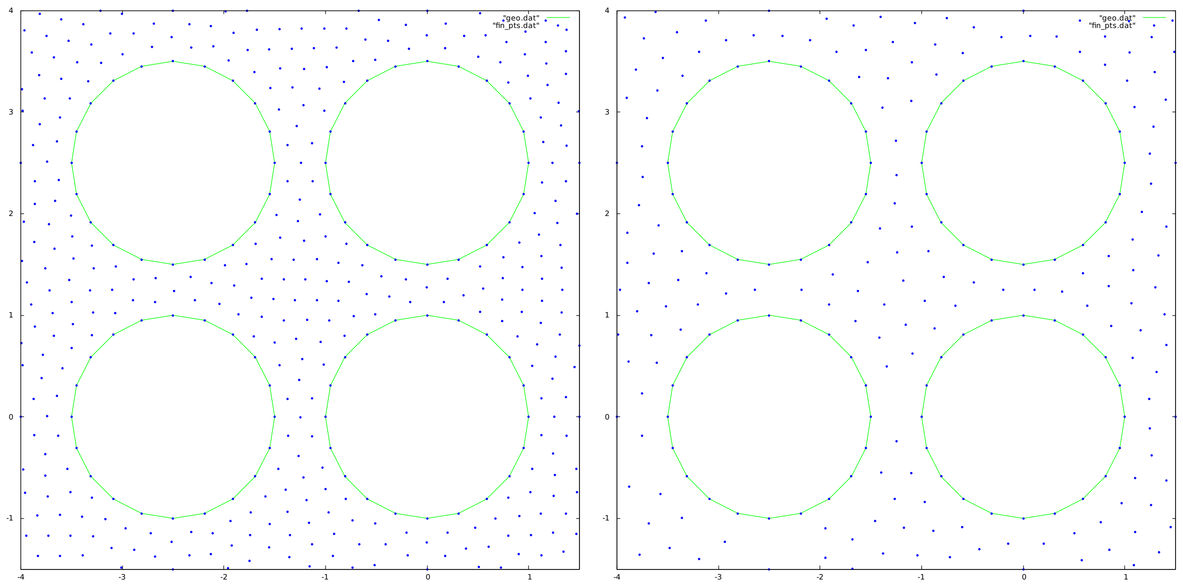


Figure 6.52 Reactor rod assembly (coarse) (zoomed): Resulting point distribution from structured (*left*) and random (*right*) initializations

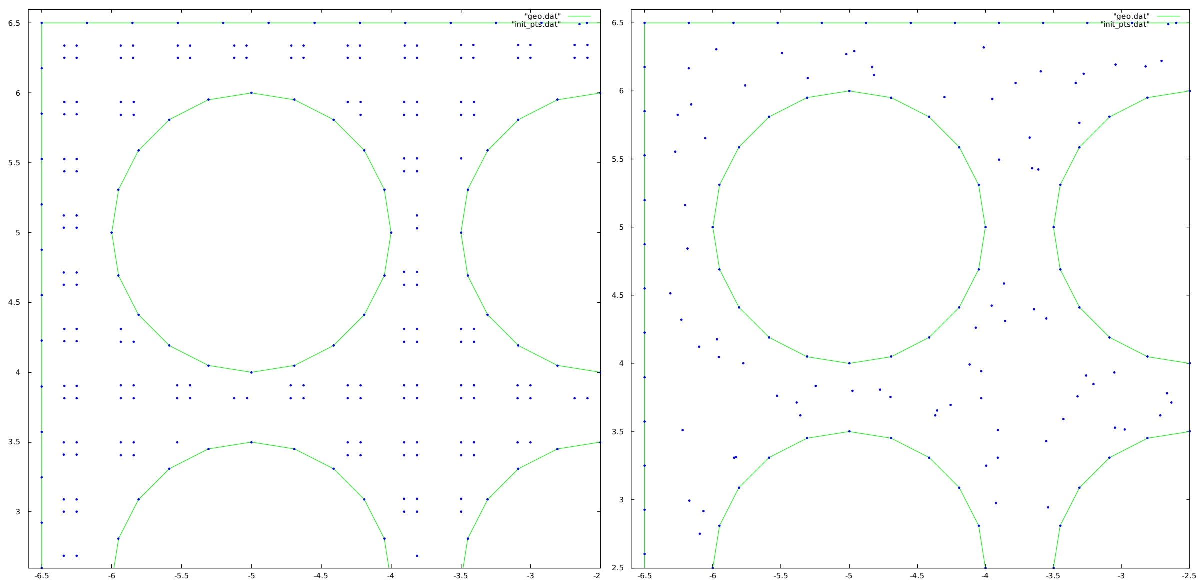


Figure 6.53 Reactor rod assembly (coarse) (zoomed to upper-left corner): Initial point distribution from structured (*left*) and random (*right*) initializations

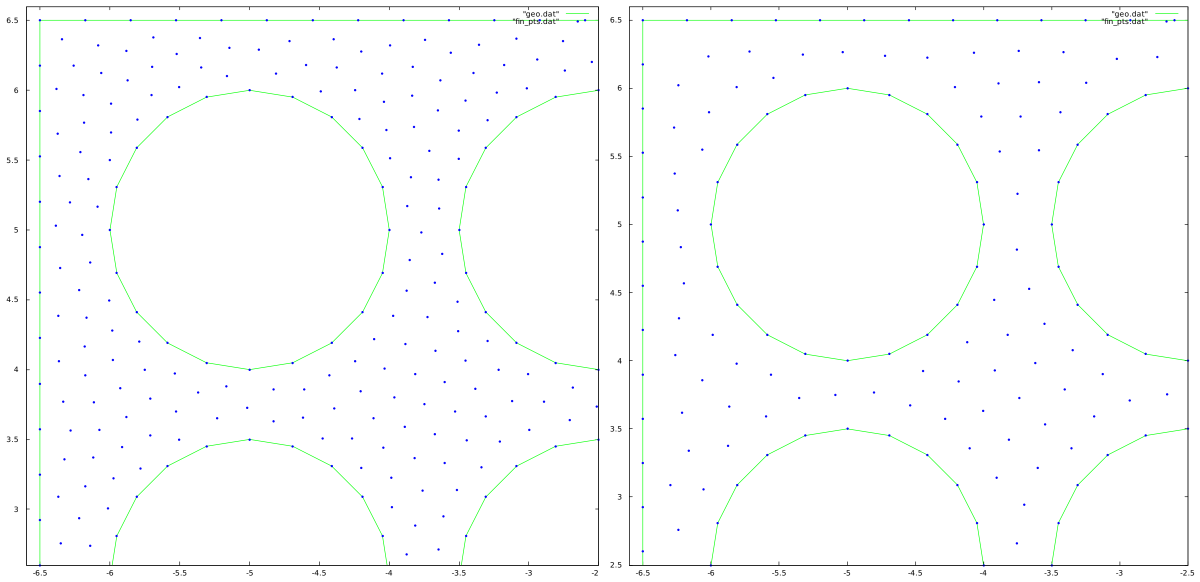


Figure 6.54 Reactor rod assembly (coarse) (zoomed to upper-left corner): Resulting point distribution from structured (*left*) and random (*right*) initializations

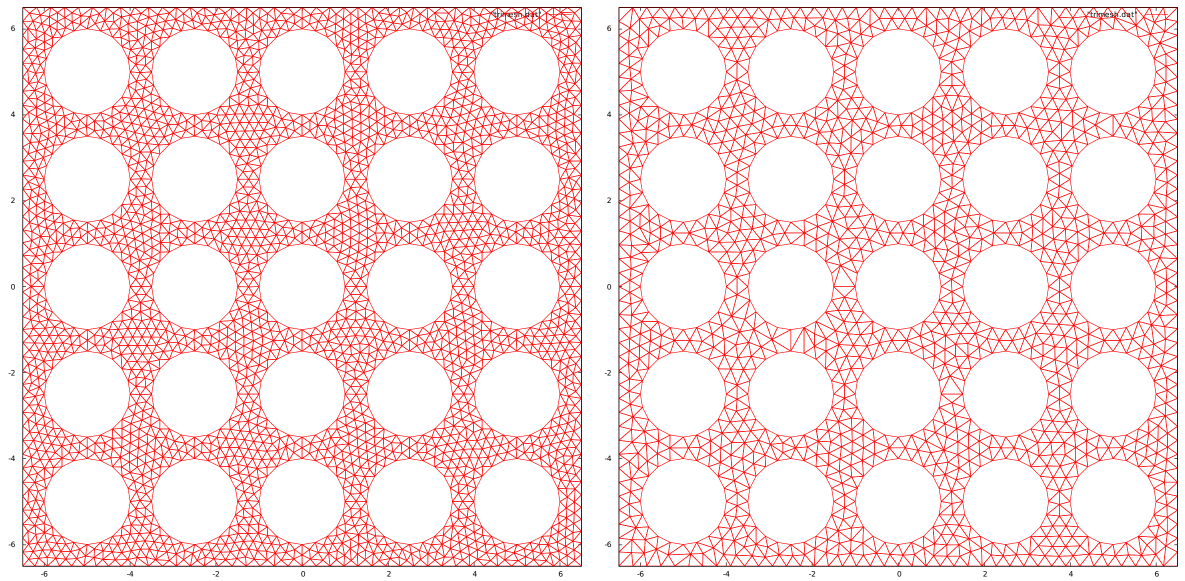


Figure 6.55 Reactor rod assembly (coarse): Triangulated result from structured (*left*) and random (*right*) initializations

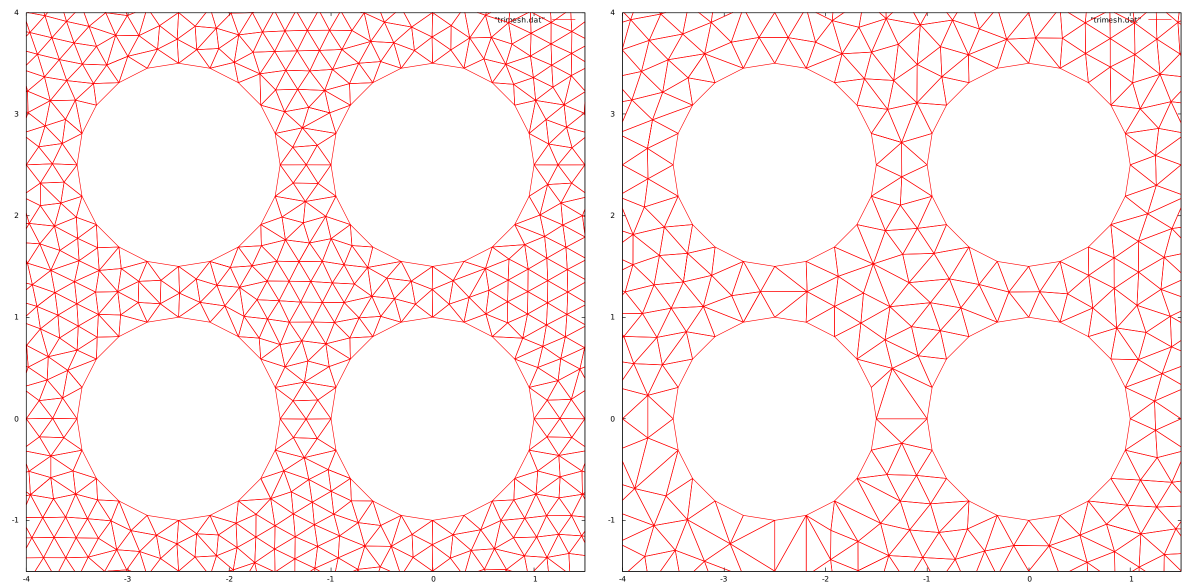


Figure 6.56 Reactor rod assembly (coarse) (zoomed): Triangulated result from structured (*left*) and random (*right*) initializations

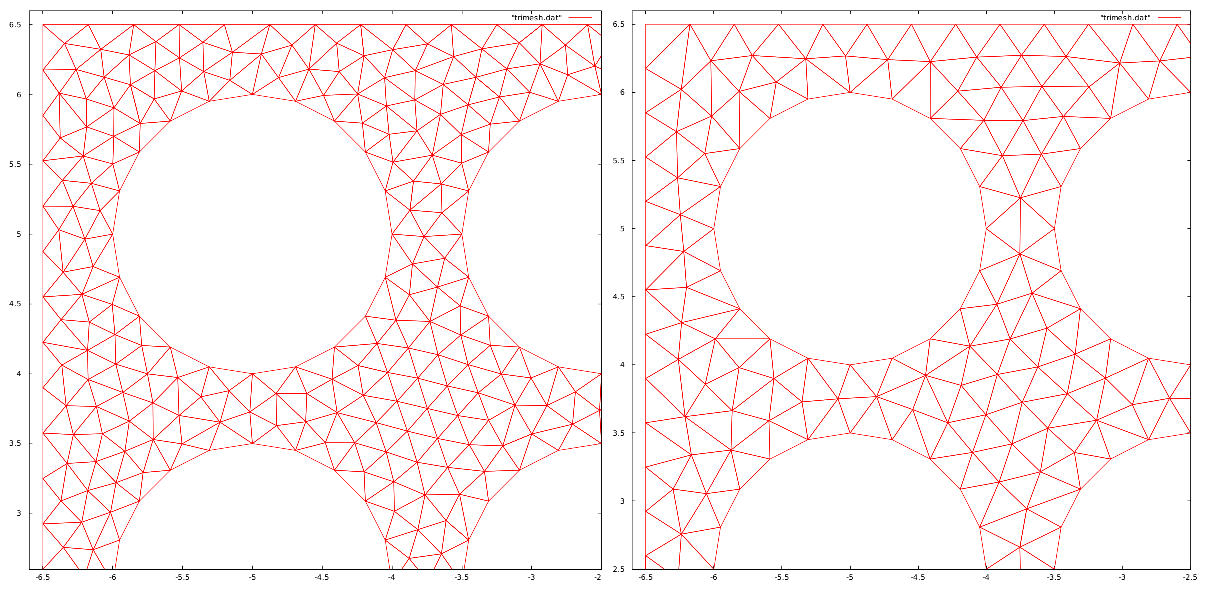
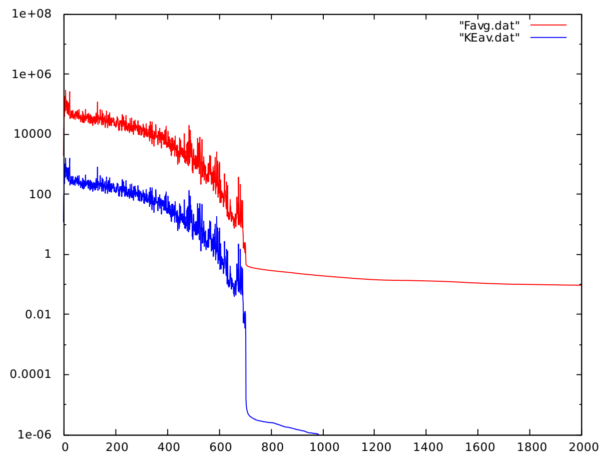


Figure 6.57 Reactor rod assembly (coarse) (zoomed to upper-left corner): Triangulated result from structured (*left*) and random (*right*) initializations

Fine Version

The fine version of this geometry has 60 segments on each circle and 130 on each side of the outer square. Therefore, the spacing will be much smaller everywhere than in the coarse version. This case was run with an initial maximum time step of 0.02 for both initialization methods. Figure 6.58 shows the trend in the average nodal resultant force and the average nodal kinetic energy for each of the runs.



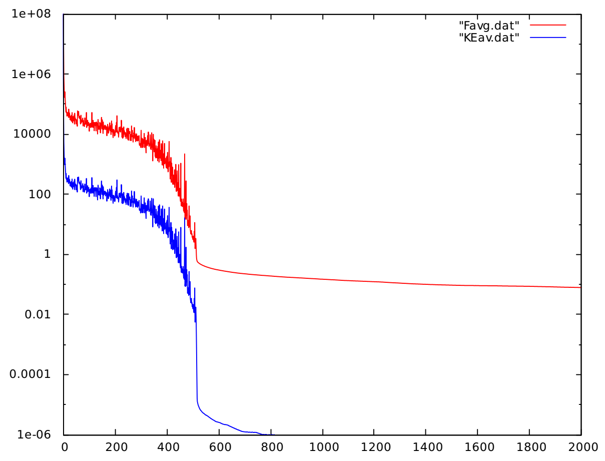


Figure 6.58 Reactor rod assembly (fine) case: Average nodal net force (*red*) and kinetic energy (*blue*) for the structured (*left*) and random (*right*) initializations

Figures 6.59 through 6.64 provide comparative views of the initial distribution to the final distribution for each run for various regions of the domain. Figures 6.55 through 6.67 show the triangulation of each resultant distribution for the same regions of the domain.

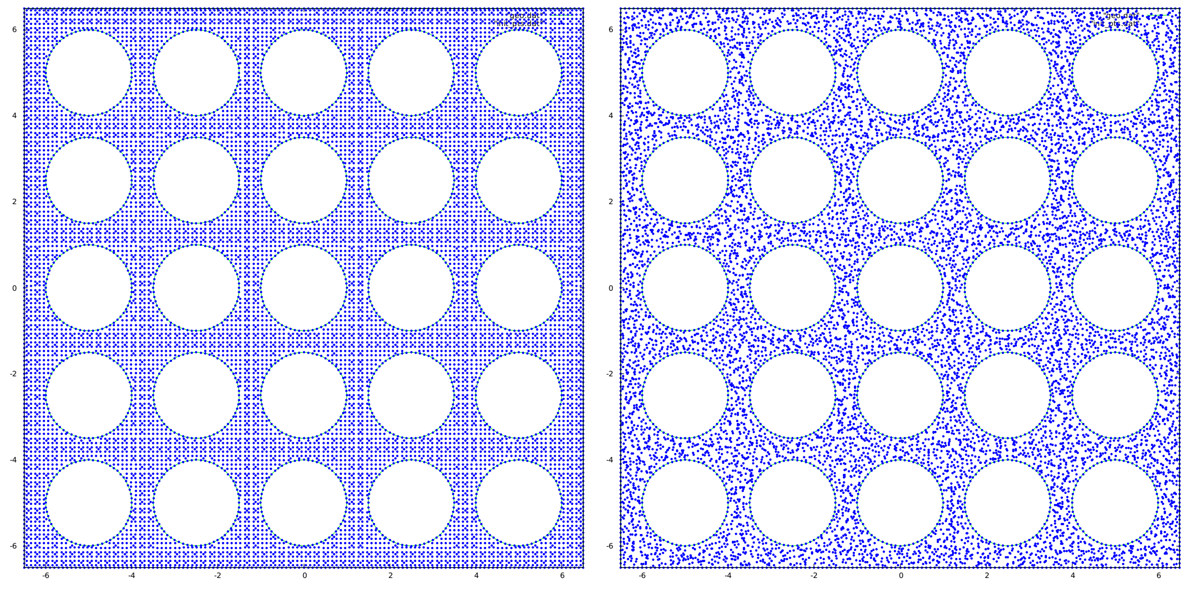


Figure 6.59 Reactor rod assembly (fine): Initial point distribution from structured (*left*) and random (*right*) initializations

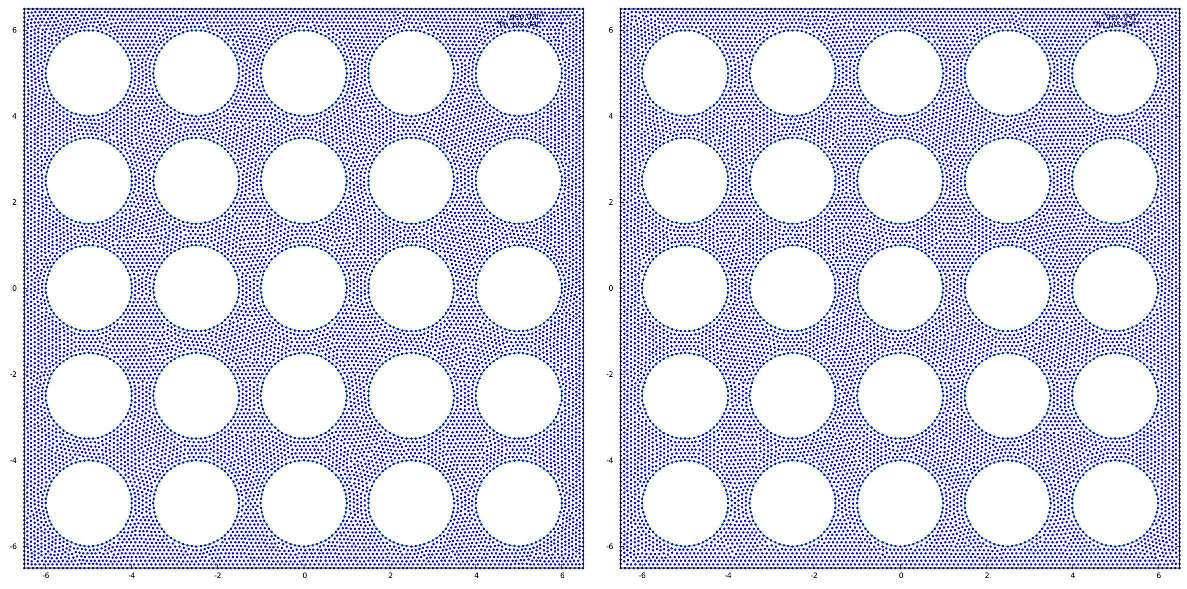


Figure 6.60 Reactor rod assembly (fine): Resulting point distribution from structured (*left*) and random (*right*) initializations

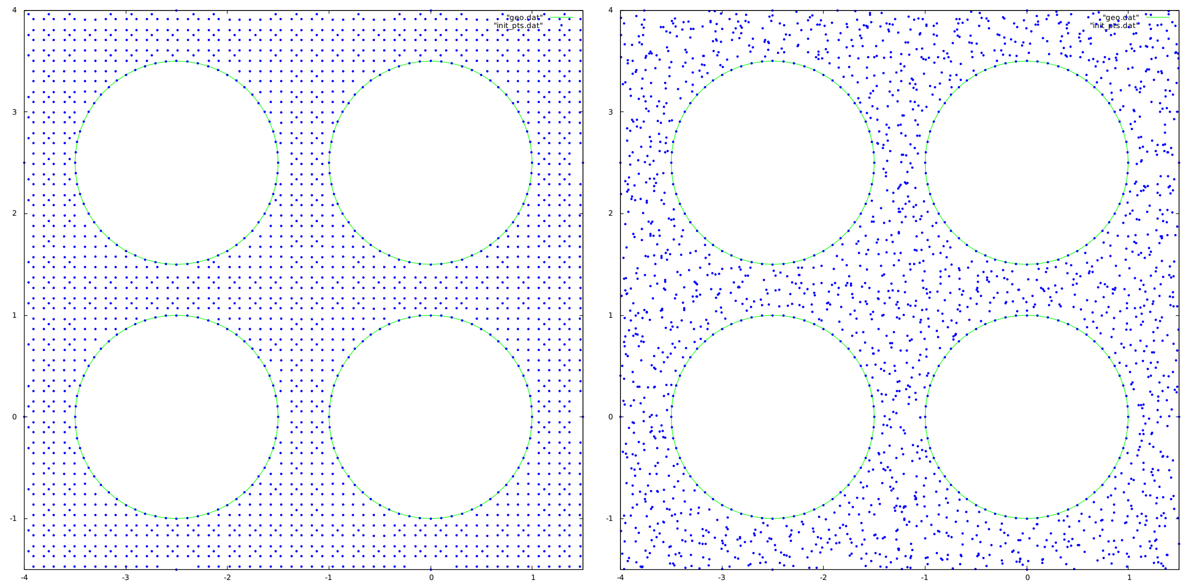


Figure 6.61 Reactor rod assembly (fine) (zoomed): Initial point distribution from structured (*left*) and random (*right*) initializations

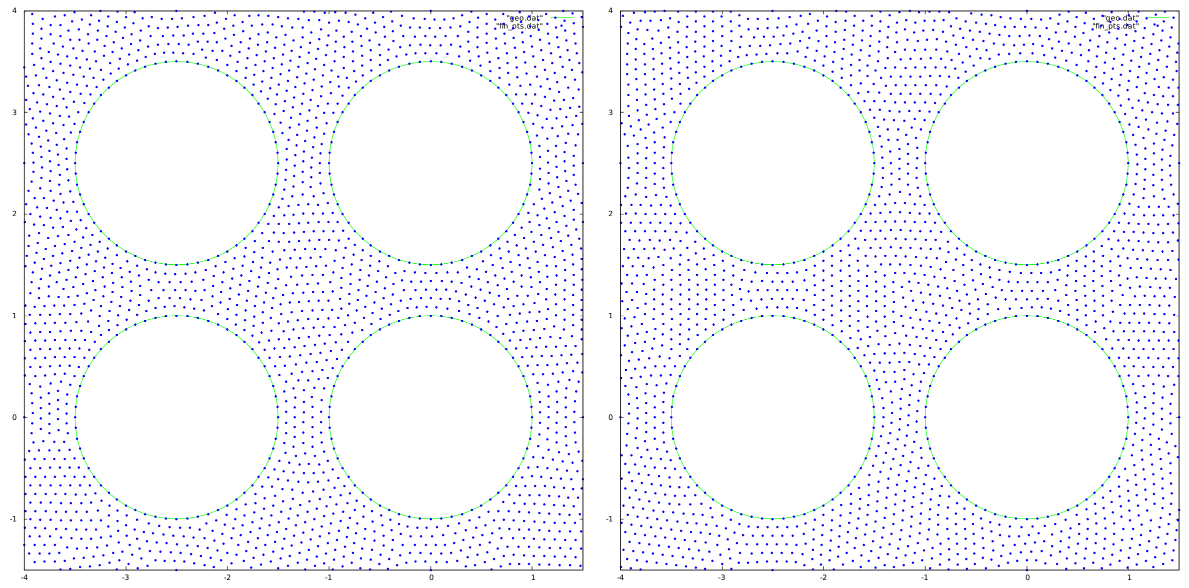


Figure 6.62 Reactor rod assembly (fine) (zoomed): Resulting point distribution from structured (*left*) and random (*right*) initializations

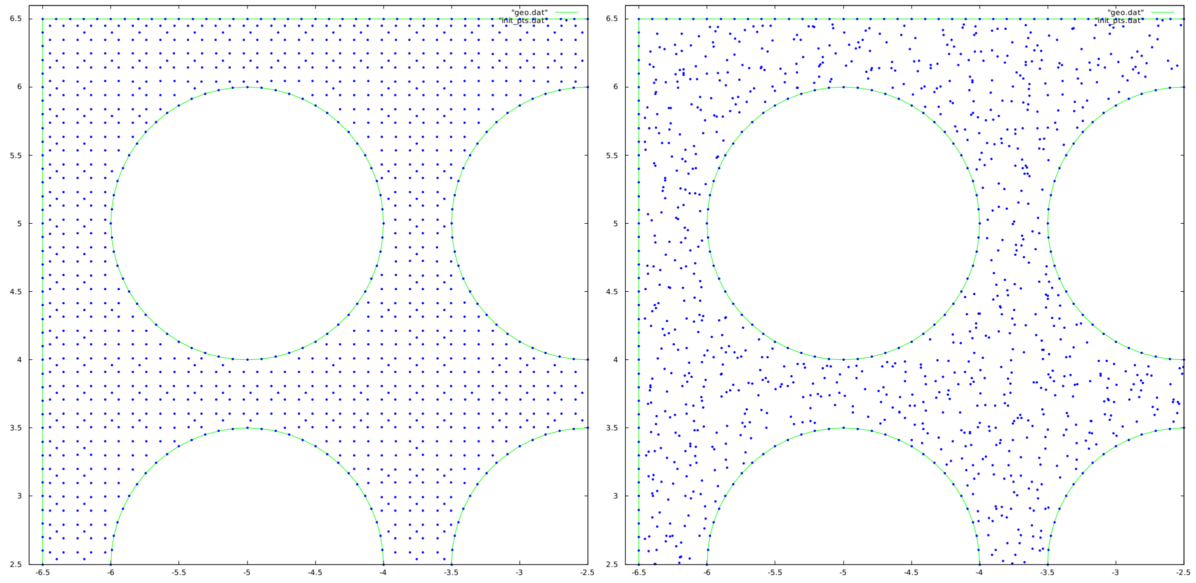


Figure 6.63 Reactor rod assembly (fine) (zoomed to upper-left corner): Initial point distribution from structured (*left*) and random (*right*) initializations

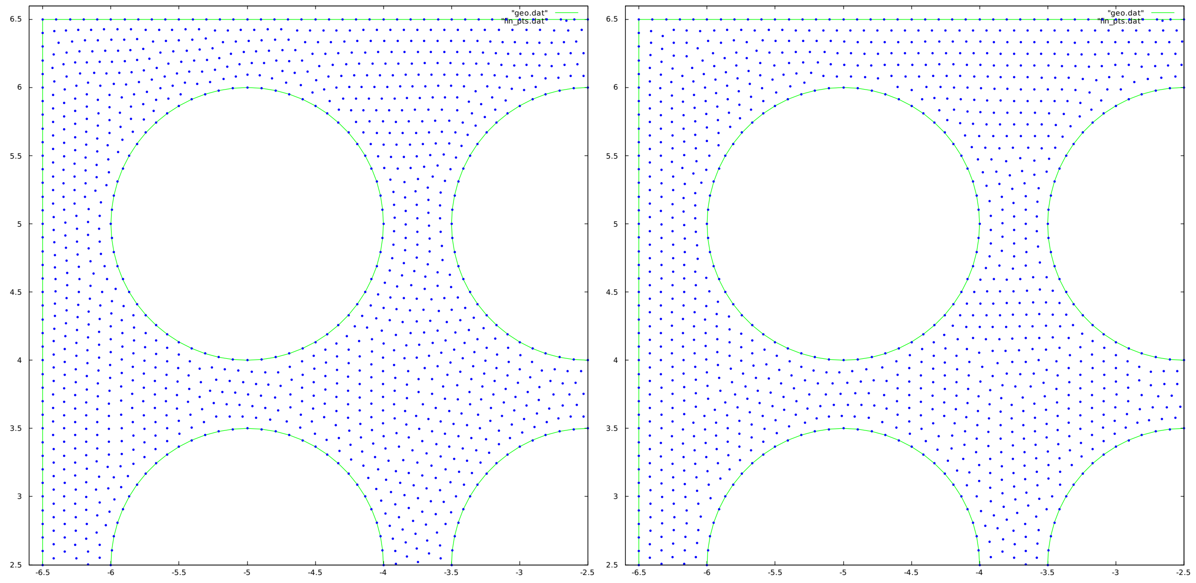


Figure 6.64 Reactor rod assembly (fine) (zoomed to upper-left corner): Resulting point distribution from structured (*left*) and random (*right*) initializations

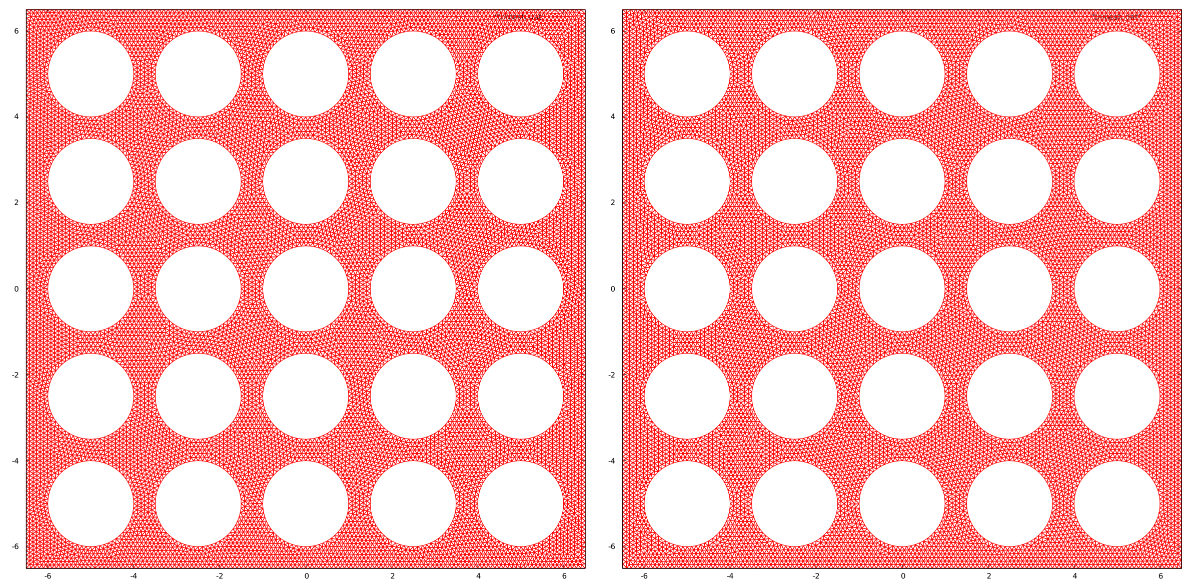


Figure 6.65 Reactor rod assembly (fine): Triangulated result from structured (*left*) and random (*right*) initializations

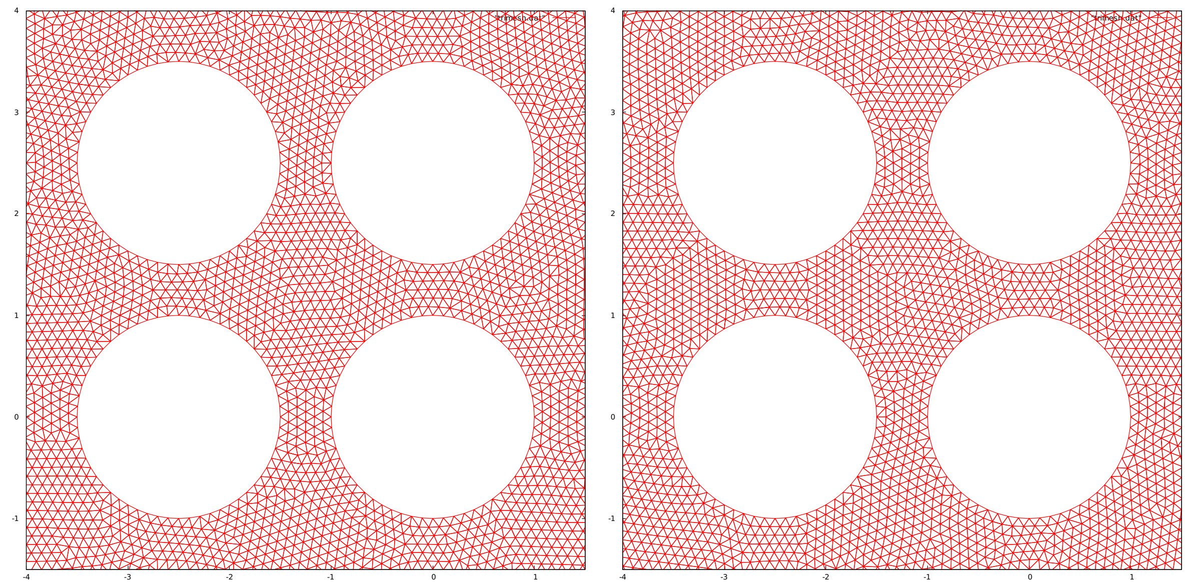


Figure 6.66 Reactor rod assembly (fine) (zoomed): Triangulated result from structured (*left*) and random (*right*) initializations

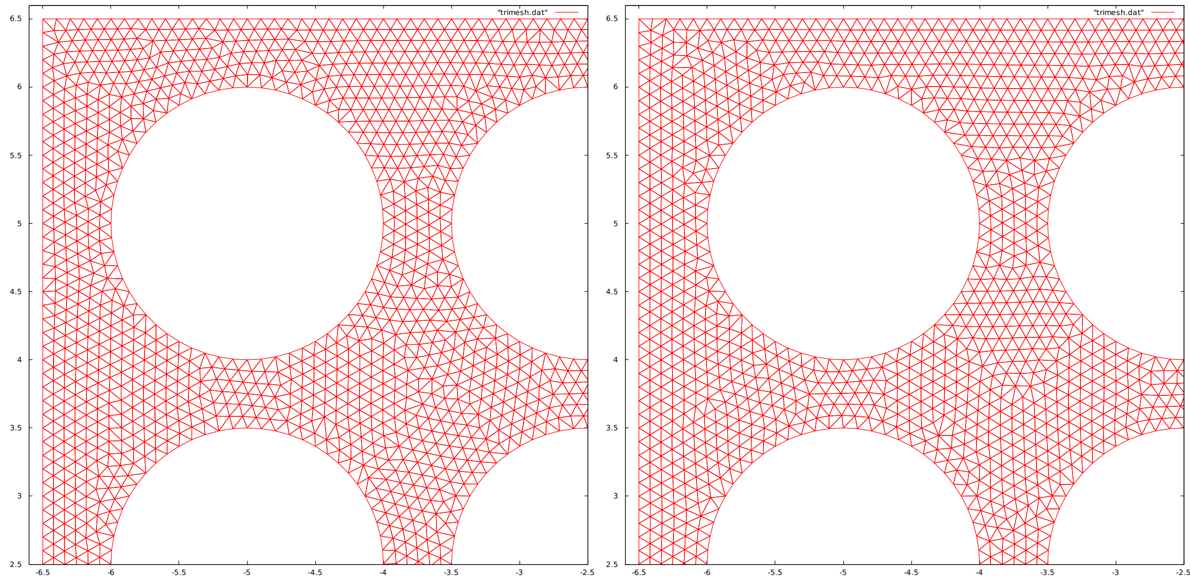


Figure 6.67 Reactor rod assembly (fine) (zoomed to upper-left corner): Triangulated result from structured (*left*) and random (*right*) initializations

Timing and Quality Results

All of the above test cases were run on a Dell Vostro with an Intel Core i5 (2.67GHz × 4) processor and 4GB of RAM (SimCenter Workstation). Table 6.1 presents the time taken to run each case. This information is split into three parts: initialization time, simulation time, and triangulation time, all measured in seconds.

Geometry	Initialization	No. of Nodes	Initialization	Simulation	Triangulation
Circle-Circle	Structured	2536	0.03	37.97	0.03
	Random	2257	0.02	32.85	0.02
NACA 0012	Structured	5852	0.13	156.99	0.11
	Random	5272	0.13	136.51	0.09
30P/30N	Structured	11219	1.07	872.67	0.40
	Random	10733	1.03	828.23	0.37
NRRA (coarse)	Structured	2364	0.10	87.63	0.03
	Random	1409	0.06	38.65	0.02
NRRA (fine)	Structured	11712	1.15	1189.63	0.43
	Random	11478	1.11	1148.07	0.40

Table 6.1 Timing results

The triangulation used is Lawson's algorithm and is not part of the research, but only used to generate connectivity for the resultant point distribution after all the iterations for the simulation are completed. Tables 6.5 through 6.4 compare some quality metrics computed for the results of each of the above test cases and two meshes generated by Pointwise [1] using the same geometry and boundary distribution. These metrics are described in the Appendix. The column heading abbreviations reference the following:

JP_Str	Mesh resulting from the proposed method using the structured initialization
JP_Ran	Mesh resulting from the proposed method using the randomized initialization
PW_Del	Mesh produced by Pointwise using the "Delaunay" method
PW_AF	Mesh produced by Pointwise using the "Advancing Front" method

		JP_Str	JP_Ran	PW_Del	PW_AF
Number of Triangles		4952	4394	2420	2222
Included Angle	MIN	35.07	35.37	26.28	23.42
(degrees)	MAX	106.15	105.32	105.51	112.20
Aspect ratio	Average	1.019	1.017	1.079	1.066
$(1, \infty)$	MAX	1.563	1.535	1.613	1.780
Skewness	Average	1.104	1.102	1.313	1.258
$(1, \infty)$	MAX	1.719	1.698	2.213	2.516
Weighted Condition Number	Average	1.015	1.014	1.067	1.058
$(1, \infty)$	MAX	1.374	1.356	1.499	1.585
Minimum Corner Jacobian		0.575	0.579	0.443	0.397

Table 6.2 Quality results: Circle within circle case

In Table 6.2 we can see that for the circle within circle case, all of the metrics for the meshes produced by the proposed method are better than those of the meshes produced by Pointwise (some more so than others). For example the minimum included angle is above 35 degrees in the proposed method's resulting meshes, but not for those from Pointwise. Also, the maximum skewness is significantly smaller for the JP meshes than for the Pointwise meshes.

		JP_Str	JP_Ran	PW_Del	PW_AF
Number of Triangles		11383	10223	6609	5881
Included Angle	MIN	29.57	30.49	24.55	24.93
(degrees)	MAX	117.61	113.01	107.48	112.67
Aspect ratio	Average	1.022	1.025	1.081	1.062
$(1, \infty)$	MAX	2.022	1.805	1.634	1.836
Skewness	Average	1.118	1.127	1.313	1.240
$(1, \infty)$	MAX	1.928	1.961	2.388	2.332
Weighted Condition Number	Average	1.018	1.020	1.068	1.054
$(1, \infty)$	MAX	1.606	1.501	1.548	1.547
Minimum Corner Jacobian		0.494	0.507	0.416	0.421

Table 6.3 Quality results: NACA 0012 airfoil case

For the NACA 0012 airfoil case, we can see from Table 6.3 that starting from the random initialization resulted generally in slightly better metrics than Pointwise, and starting from the structured initialization resulted generally in slightly worse metrics.

		JP_Str	JP_Ran	PW_Del	PW_AF
Number of Triangles		21026	20054	25416	23652
Included Angle	MIN	27.71	24.74	26.58	23.82
(degrees)	MAX	120.90	121.39	112.31	115.98
Aspect ratio	Average	1.040	1.043	1.081	1.064
$(1, \infty)$	MAX	2.218	2.253	1.785	1.940
Skewness	Average	1.165	1.177	1.315	1.262
$(1, \infty)$	MAX	2.089	2.387	2.232	2.474
Weighted Condition Number	Average	1.032	1.035	1.068	1.057
$(1, \infty)$	MAX	1.699	1.717	1.524	1.591
Minimum Corner Jacobian		0.465	0.419	0.447	0.404

Table 6.4 Quality results: 30P/30N Multi-element airfoil case

From Table 6.4 we can see that the proposed method resulted in meshes with quality metrics quite similar overall to those from Pointwise but generally worse in the extrema (the averages are still slightly better).

		JP_Str	JP_Ran	PW_Del	PW_AF
Number of Triangles		4113	2206	2776	2864
Included Angle	MIN	33.85	26.60	29.99	30.39
(degrees)	MAX	110.15	123.23	105.80	103.77
Aspect ratio	Average	1.085	1.065	1.120	1.095
$(1, \infty)$	MAX	1.695	2.373	1.551	1.504
Skewness	Average	1.243	1.237	1.382	1.341
$(1, \infty)$	MAX	1.742	2.060	2.000	1.955
Weighted Condition Number	Average	1.065	1.052	1.099	1.082
$(1, \infty)$	MAX	1.444	1.768	1.366	1.363
Minimum Corner Jacobian		0.557	0.448	0.500	0.506

Table 6.5 Quality results: Reactor rod assembly (coarse) case

On the coarse version of the reactor rod assembly, starting from the random initialization scheme resulted in some gaps in the domain and, therefore, some poorly shaped triangles in the resulting mesh. This can be seen here in Table 6.5 as well. The minimum included angle quite smaller than we would like to have and the extrema of the other metrics are not ideal. On the other hand, the structured case did quite well, with the resulting metrics being similar or slightly better than those from Pointwise.

		JP_Str	JP_Ran	PW_Del	PW_AF
Number of Triangles		21452	20984	26252	20452
Included Angle	MIN	30.79	36.18	28.13	27.90
(degrees)	MAX	117.90	104.88	111.80	116.34
Aspect ratio	Average	1.022	1.019	1.079	1.052
$(1, \infty)$	MAX	2.037	1.521	1.768	1.962
Skewness	Average	1.116	1.107	1.314	1.210
$(1, \infty)$	MAX	1.951	1.640	2.112	2.079
Weighted Condition Number	Average	1.018	1.016	1.067	1.043
$(1, \infty)$	MAX	1.612	1.348	1.487	1.580
Minimum Corner Jacobian		0.512	0.590	0.471	0.468

Table 6.6 Quality results: Reactor rod assembly (fine) case

With the fine version of this geometry, we have an opposite result. The structured case resulted in poorer metrics than the random case, which produced better numbers here than the Pointwise meshes.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

In this research, we have validated the usefulness of the physics-based approach for generating smooth, quality point distributions (and meshes) that match a desired spacing field for complicated geometries. Much of what was done in this work was reproducing the work of others (see Chapter 2) in a simplified fashion in order to test this approach on complex geometries. From the above results we can see that the proposed method, though having several limitations, has shown to be capable of producing high quality resulting meshes, and we can generally agree with Zhang and Smirnov [19] that "liquefying" (overloading) the domain results in better distributions.

There is much work to be done in developing the proposed method, adding and improving features to increase its versatility, reliability, and speed. The following sections address some of the problems with the current method and outline the major goals for future development, in keeping with the goals stated in Chapter 1.

Restart Capability

This will be beneficial in the cases of large point clouds which do not reach equilibrium within the given number of iterations. The node data will be written to a file and that file given as input when restarting the program. The case could thus be given more iterations starting from the configuration in which it previously stopped.

This will also include the capability of reading in a generic mesh file generated elsewhere. The connectivity would be removed and the program would be used as smoothing routine for the nodes already populated from the given mesh file.

Better Initial Node Distribution Method

The initialization method needs to be improved in order to 1) reduce the number of iterations required to reach a stable configuration and 2) better match the desired spacing field. For example, in the 30P/30N multi-element airfoil case, the domain was significantly overloaded especially in the regions between and around the airfoil elements. That is, the spacing field was not matched well by the initialization.

Shimada and Gossard point out that a hexagonal pattern (in 2D) gives the ideal packing of bubbles, since the connectivity of nodes at the centers of those bubbles produces equilateral triangles [17]. In their method, they therefore initialize the (2D) domain by packing the bubbles in a hexagonal pattern. (In 3D the ideal packing is a pattern of icosahedra. That is, each node should be surrounded by 12 immediate neighbors.) This initialization method will need to be implemented and tested in future work.

Automatic Global Maximum Time Step Calculation

For every case, there is a value for the global maximum time step size that is both small enough to stabilize the motion of the nodes (i.e., nodes do not continue to jitter indefinitely) and large enough to allow the nodes to move enough to smooth the distribution. This value depends on many factors, including the total domain area (or volume), the number of nodes in the domain, the spacing field, and the inter-node force formula being used. It would be preferable to have a method to automatically calculate the right value for the maximum time step size for a given case from the data specific to that case. Various methods have been attempted throughout the course of this research, but none have worked equally well for all cases tested.

Automatic Node Population Control

Since the force formula we are using includes attraction as well as repulsion, this means that when there are too few nodes in the system, there will be gaps in the domain in which there are no nodes. On the other hand, if there are too many nodes in the domain, most node pairs will be at a distance that is smaller than their ideal spacing, σ_{ij}. In some cases, especially when the geometry is complex, a resulting configuration may simultaneously have regions in the domain with gaps and other regions in which the nodes are too tightly packed.

Examples of this problem can be seen in the figures in Chapter 6. A simple example of gaps in a domain with too few points is shown in Figures 7.1 and 7.2. The geometry is a simple 200×200 square with 40 points on the boundary. Thus the spacing field is constant ($q_i = 20, \ \forall i$). Using the structured initialization method, the domain is initialized with 64 interior nodes. Figure 7.1 shows the initial and resulting bubble configurations. Figure 7.2 shows the initial and resulting point distributions as well as the triangulated result.

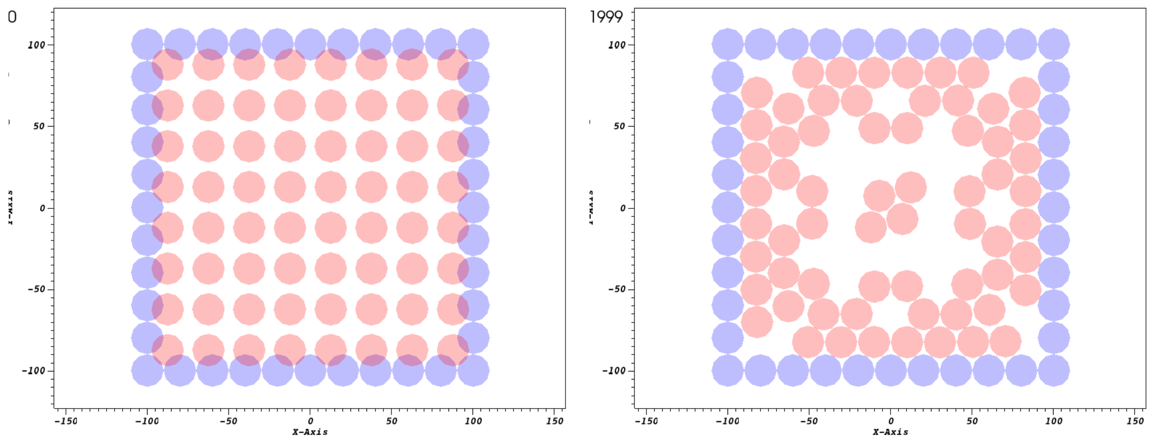


Figure 7.1 Example initialization (*left*) and result (*right*) with too few points (bubble view)

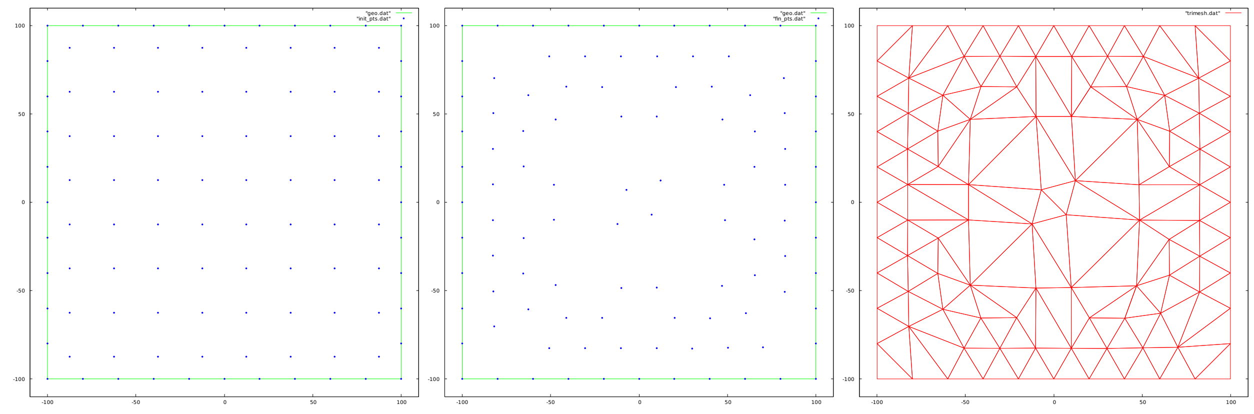


Figure 7.2 Example initialization (*left*), result (*middle*), and triangulated result (*right*) with too few points

There is therefore the necessity of being able to both insert points into gaps and remove points from areas of excess concentration. Shimada and Gossard [17] proposed a method for accomplishing this by measuring an “overlap ratio” for each node. Using our notation, the formula for this ratio is:

$$\alpha_i = \frac{1}{q_i} \sum_j (2q_i + q_j - r_{ij}) \tag{7.1}$$

This ratio corresponds to how tightly the nodes are packed in the domain. For an interior node in 2D, this ratio should have a value of 6 (in 3D, it should be 12). If the ratio is smaller than ideal, the node is considered “open” and if it is greater, the node is considered “excess”. An excess node is removed and one or more points are inserted around an open node.

The results of a preliminary implementation of this approach used on the example above are shown in Figures 7.3 through 7.5. In this implementation, the simulation is allowed to progress until it reaches the criterion that the maximum force magnitude decreases for 20 consecutive iterations. At this point the population control function is called which tests every interior node’s overlap ratio. It will then delete excess nodes and search around each open node for empty spaces and fill them. Figure 7.3 shows the intertion of 23 points from iteration 106 to 107 after the nodes have significantly slowed their movement.

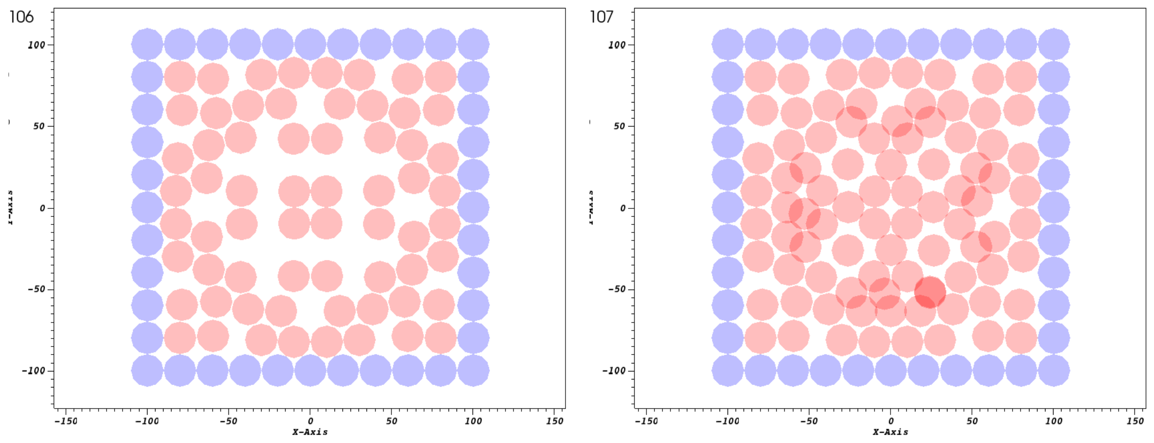


Figure 7.3 Example of inserting nodes into gap regions

In all, this function is called 27 times throughout the course of the simulation. Figure 7.3 shows the first call. Two other calls result in one node being inserted each. The other 24 calls result in no change to the population. At the end of 2000 iterations then the result is much more uniformly packed than before, as is shown in Figure 7.4.

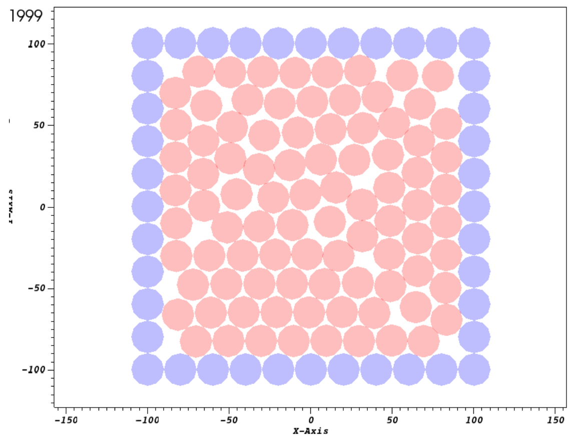


Figure 7.4 Example resulting bubble configuration after using population control

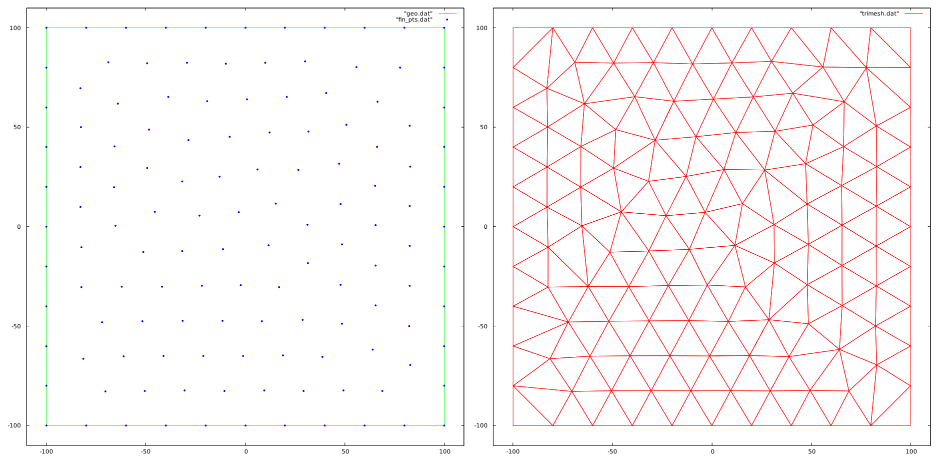


Figure 7.5 Example resulting point distribution and triangulation after using population control

So far this implementation works well only for simple cases, especially with constant spacing. With more complex geometries, too many points are deleted near inner boundaries, resulting in more gaps than would result without using it.

Automatic Boundary Population

The current method requires the user to have distributed points on the boundaries prior to runtime. For complex geometries (especially in 3D), just the task of distributing points on the boundaries can be a very time-consuming process. There is therefore a need for an automatic boundary population method to be implemented as part of the initialization process.

The spacing field used (prior to flow solution) is also affected by the choice of method used here. One option is to require the user to specify the spacing only at the critical points and use an inverse-distance weighted average formula based only on the critical points to define the spacing at all other locations on the boundaries and in the interior of the domain.

Non-Spherical Packing Methods for Viscous Layers

In the current method, the spacing parameter for each node is a scalar, and therefore each node's bubble is circular. That is, the same spacing value is enforced in every direction from the node. This is not suitable for creating viscous layer meshes, in which generally the spacing in the direction normal to a boundary surface is significantly smaller than the spacing tangential to the surface. In order to create viscous meshes, nodes near a (viscous) boundary surface need to have bubbles (spacing implementations) that are elliptical (or perhaps quadrilateral) in shape, with the longer dimension tangential to the boundary surface. Shimada [23] and Yamakawa [24] have proposed a method of ellipsoid packing that may be useful for this purpose. This technique may also be useful for refining a mesh in regions of high gradients with anisotropic elements.

Spacing Field Adapted to Flow Solution

At the current state of development, the proposed method can adapt well to a variable spacing field. The spacing field used can be arbitrary as long as it is a scalar function over the entire domain. In future development, we will implement the capability to adapt to a spacing field defined as a tensor function. Of course, the primary reason for this capability is to adapt the mesh to solution data.

Using the same geometry as shown in Figure 7.1 (*left*) and the random initialization, we define an analytic spacing field specific to that case which "smoothly" mimics a spacing field adapted to a vertical shock region:

$$q_i = 0.5 + 19.5 \left(\tanh \left(\frac{x}{16} \right) \right)^2 \tag{7.2}$$

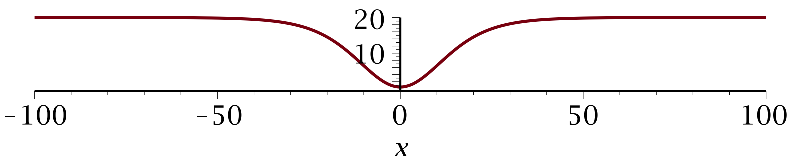


Figure 7.6 Plot of Equation (7.2)

Figures 7.7 through 7.9 display the resulting point distribution and triangulation after running this case with the population control function described above. On the bottom boundary, points were pushed onto the boundary so that the resulting boundary distribution matches the spacing function. On the top boundary, however, not enough points were pushed onto the boundary, and therefore the spacing function is not matched and ill-shaped triangles result.

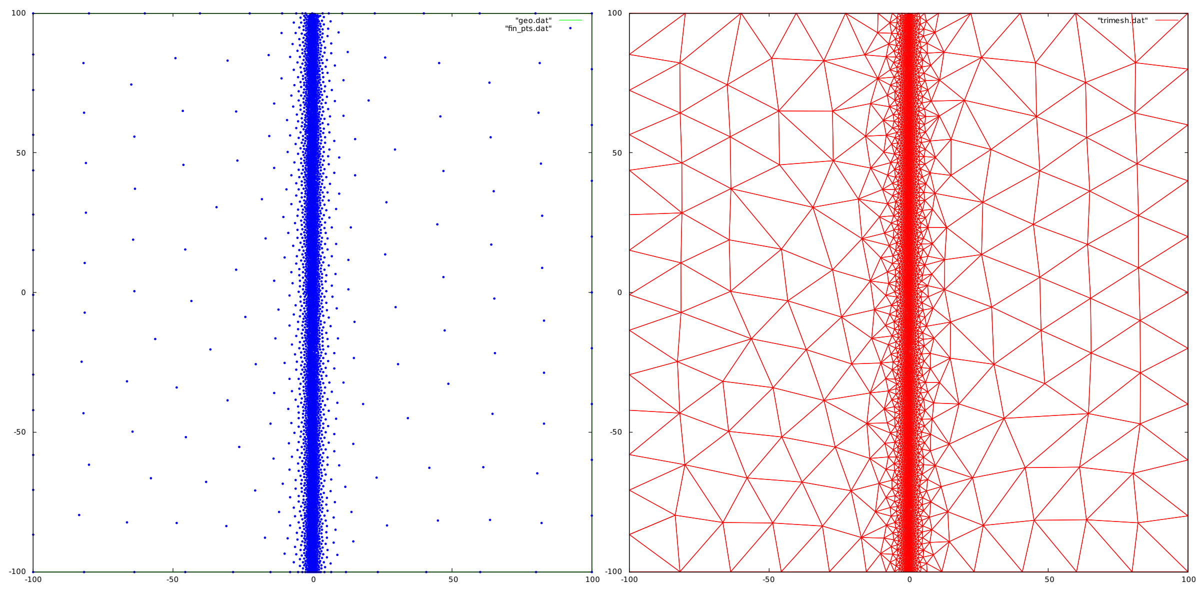


Figure 7.7 Example resulting point distribution and triangulation after using population control with analytic spacing field

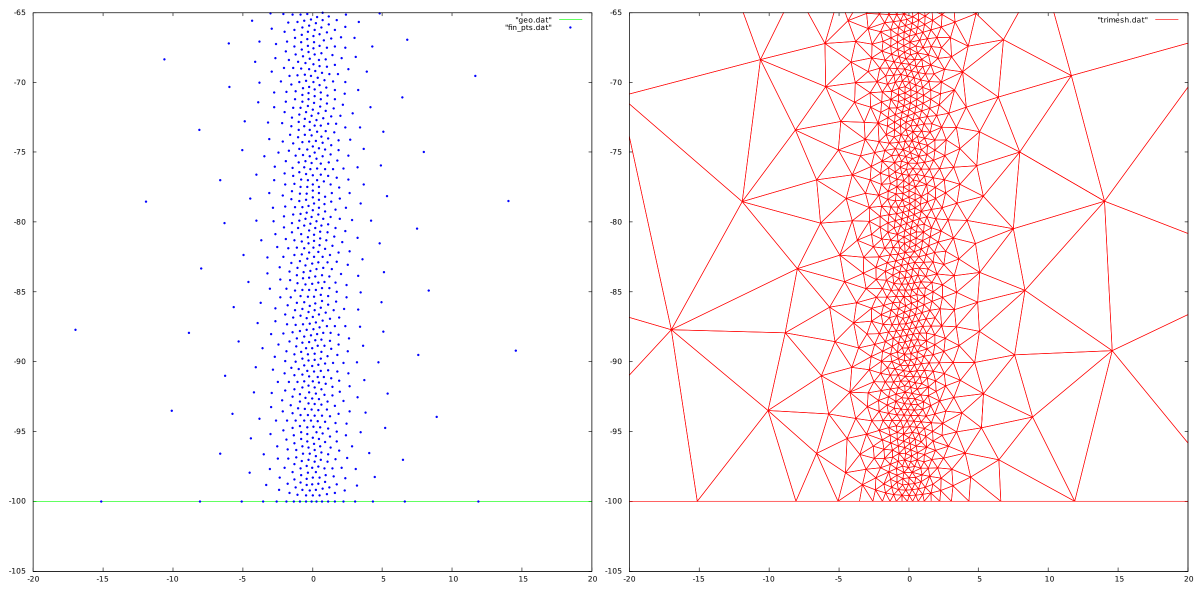


Figure 7.8 Example resulting point distribution and triangulation after using population control with analytic spacing field (bottom boundary)

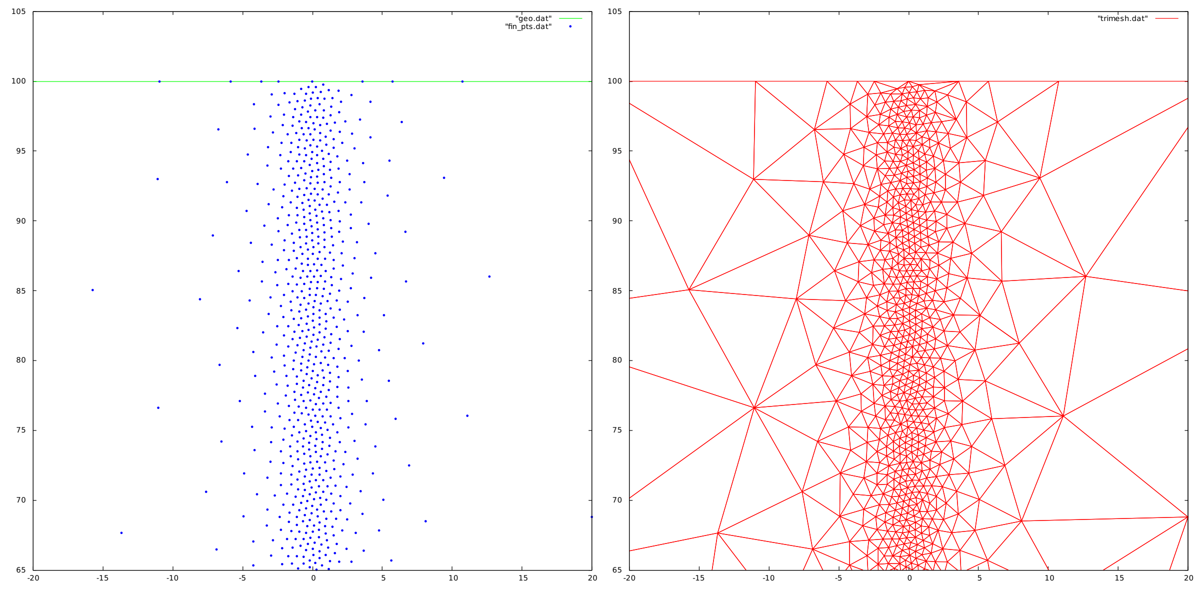


Figure 7.9 Example resulting point distribution and triangulation after using population control with analytic spacing field (top boundary)

It is worth noting that the initialization for this example did not take the analytic spacing field into account. The random initialization was used and therefore in the first iteration, q_i was equal to 20 everywhere. More work will need to go into initializing (from restart as well) the boundaries and interior with a new spacing field.

Moving Geometry

The proposed method is foreseeably useful for adapting to moving geometry. It will be implemented to pair with either a finite volume solver or a meshless solver to produce time-accurate solution results for moving geometry cases.

3D

The proposed method along with all of the above future capabilities must be implemented in three dimensions. Perhaps the greatest challenge with this will be dealing with the geometry surface meshes, which will be significantly more complicated than dealing with segmented curves in two dimensions.

Parallel

Implementing the method in parallel will clearly be necessary for scalability.

REFERENCES

[1] Pointwise Inc. www.pointwise.com, 2013. 2, 5, 47, 104

[2] Vincent C. Betro. *Fully Anisotropic Split-tree Adaptive Refinement Mesh Generation using Tetrahedral Mesh Stitching.* PhD thesis, The University of Tennessee at Chattanooga, 2010. 2, 3

[3] Cameron T. Druyor Jr. An adaptive hybrid mesh generation method for complex geometries. Master's thesis, The University of Tennessee at Chattanooga, 2011. 2, 3

[4] Satish Chalasani and David Thompson. Quality improvements in extruded meshes using topologically adaptive generalized elements. *International Journal for Numerical Methods in Engineering*, 60(6):1139–1159, 2004. 2

[5] Charles L. Lawson. Software for C^1 surface interpolation. In *Mathematical Software III*, pages 161–194. Academic Press: New York, 1977. 2

[6] Adrian Bowyer. Computing dirichlet tessellations. *The Computer Journal*, 24(2):162–166, 1981. 2

[7] David F. Watson. Computing the n-dimensional delaunay tessellation with application to voronoi polytopes. *The Computer Journal*, 24(2):167–172, 1981. 2

[8] Steven J. Owen. A survey of unstructured mesh generation technology. In *Proceedings, 7th International Meshing Roundtable*, pages 239–267, October 1998. 2

[9] L. Paul Chew. Guaranteed-quality triangular meshes. Technical Report TR89-983, Department of Computer Science, Cornell University, April 1989. 3

[10] Jim Ruppert. A new and simple algorithm for quality 2-dimensional mesh generation. Technical Report CSD92-694, Computer Science Division, University of California at Berkeley, 1992. 3

[11] Jim Ruppert. A delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18:548–585, 1995. 3

[12] Jonathan Richard Shewchuk. Delaunay refinement algorithm for triangular mesh generation. *Computational Geometry: Theory and Applications*, 22(1-3):21–74, May 2002. 3

[13] Rainald Löhner and Paresh Parikh. Generation of three-dimensional unstructured grids by the advancing front method. *International Journal for Numerical Methods in Fluids*, 8:1135–1149, 1988. 3

[14] Paul Louis George and Eric Seveno. The advancing-front mesh generation method revisited. *International Journal for Numerical Methods in Engineering*, 37:3605–3619, 1994. 3

[15] C. K. Lee. Automatic adaptive mesh generation using metric advancing front approach. *Engineering Computations*, 16(2):230–263, 1999. 3

[16] Kenji Shimada. *Physically-Based Mesh Generation: Automated Triangulation of Surfaces and Volumes via Bubble Packing.* PhD thesis, Massachusetts Institute of Technology, May 1993. 7

[17] Kenji Shimada and David C. Gossard. Bubble mesh: Automated triangular meshing of non-manifold geometry by sphere packing. In *ACM Symposium on Solid Modeling and Applications*, pages 409–419. ACM, 1995. 7, 111, 113

[18] Yufeng Nie, Weiwei Zhang, Ying Liu, and Lei Wang. A node placement method with high quality for mesh generation. *IOP Conference Series: Materials Science and Engineering*, 10(1):012218, 2010. 9

[19] Hanzhou Zhang and Andrei V. Smirnov. Node placement for triangular mesh generation by monte carlo simulation. *International Journal for Numerical Methods in Engineering*, 64:973–989, 2005. 9, 110

[20] A. L. Zheleznyakova and S. T. Surzhikov. Triangular-mesh generation for aerodynamics problems by molecular-dynamics simulation. *Doklady Physics*, 56(7):385–390, 2011. 12

[21] Per-Olof Persson and Gilbert Strang. A simple mesh generator in matlab. *SIAM Review*, 46(2):329–345, 2004. 14

[22] Randi Holm, Roland Kaufmann, Bjørn-Ove Heimsund, Erlend Øian, and Magne S. Espedal. Meshing of domains with complex internal geometries. *Numerical Linear Algebra with Applications*, 13:717–731, 2006. 15

[23] Kenji Shimada, Atsushi Yamada, and Takayuki Itoh. Anisotropic triangular meshing of parametric surfaces via close packing of ellipsoidal bubbles. In *Proceedings, 6th International Meshing Roundtable*, pages 375–390, October 1997. 116

[24] Soji Yamakawa and Kenji Shimada. High quality anisotropic tetrahedral mesh generation via ellipsoidal bubble packing. In *Proceedings, 9th International Meshing Roundtable*, pages 263–273, October 2000. 116

APPENDIX

OVERVIEW OF MESH QUALITY METRICS

Herein we define the metrics used to compare meshes in Chapter 6 and show how they are computed.

Included Angle

An included angle is the angle between any two edges of a triangle. For any triangle with positive (and nonzero) area, all three included angles will be less than 180 degrees and greater than 0 degrees. A good mesh triangle should have included angles between 35 and 145 degrees.

Aspect Ratio

Aspect ratio is defined for a triangle as

$$\rho_{\mathrm{AR}} = \frac{R}{2r}$$

where R is the radius of the triangle's circumscribing circle and r is the radius of its inscribing circle. The division by 2 serves to normalize the equation so that equilateral triangles have an aspect ratio of $\rho_{\mathrm{AR}} = 1$. This metric is thus bounded below by 1. The greater the aspect ratio the "skinnier" (or more ill-shaped) the triangle.

Skewness

Another measure of how much a triangle deviates from a standard triangle shape is called the skewness. This is computed as a ratio of the maximum edge length over the minimum edge length:

$$\rho_{\mathrm{skew}} = \frac{\max\{l_1, l_2, l_3\}}{\min\{l_1, l_2, l_3\}}$$

where l_1, l_2, and l_3 are the lengths of the three edges of the triangle. This metric is bounded below, like aspect ratio, by 1.

Weighted Condition Number

This metric uses the condition number of a matrix as another measure of how far the triangle deviates from a standard shape. In this case the weight matrix

$$W = \begin{bmatrix} 1 & \frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} \end{bmatrix}$$

defines the standard shape to be a right triangle. We define two vectors $\mathbf{u} = u_x\mathbf{e}_1 + u_y\mathbf{e}_2$ and $\mathbf{v} = v_x\mathbf{e}_1 + v_y\mathbf{e}_2$ as the vectors pointing from one vertex of the triangle to the other two vertices, then we can define a matrix

$$A = \begin{bmatrix} u_x & v_x \\ u_y & v_y \end{bmatrix}$$

and compute the weighted condition number of this matrix as

$$K_W = \frac{\|AW^{-1}\| \, \|WA^{-1}\|}{2}$$

using as a matrix norm $\|M\| = \sqrt{\operatorname{tr}(M^T M)}$. This number is again bounded below by 1, and is the same regardless of the vertex upon which the computation is based.

Corner Jacobian

In two dimensions the corner Jacobian is the cross product of the two (normalized) vectors pointing away from one vertex.

$$J\mathbf{e}_3 = \frac{\mathbf{u}}{\|\mathbf{u}\|} \times \frac{\mathbf{v}}{\|\mathbf{v}\|}$$

with **u** and **v** defined as above. This number will be different for each corner and is related to area giving a measure of how well shaped each corner is. A valid element (non-inverted) will have a positive value for the Jacobian at all corners. Elements with negative Jacobians are not considered valid. This value is bounded above by 1 (since the vectors **u** and **v** are normalized, the value of J is simply the sine of the angle between them), and the larger the value is, the better the shape of the corner.

VITA

Philip Wesley Fackler was born in Bowling Green, Kentucky, on February 16th, 1986, the youngest of three sons of David and Teale Fackler. He attended Madisonville North Hopkins High School of Madisonville, Kentucky, graduating and receiving the Commonwealth diploma in May of 2004. He graduated with honors from Asbury College (now University) of Wilmore, Kentucky, in May of 2008 with a Bachelor of Arts degree in Mathematics.