A PHYSICS-BASED ADAPTIVE POINT DISTRIBUTION
METHOD FOR COMPUTATIONAL DOMAIN DISCRETIZATION

By

Philip Wesley Fackler

Dr. C. Bruce Hilbert
Algorithms/Optimization
Branch Technology, Inc.
(Chair)

Dr. Steve L. Karman, Jr.
Staff Specialist
Pointwise, Inc.
(Committee Member)

Dr. Kidambi Sreenivas
Associate Professor of Mechanical Engineering
(Committee Member)

Dr. Robert Webster
Associate Professor of Mechanical Engineering
(Committee Member)
A PHYSICS-BASED ADAPTIVE POINT DISTRIBUTION

METHOD FOR COMPUTATIONAL DOMAIN DISCRETIZATION

By

Philip Wesley Fackler

A Dissertation Submitted to the Faculty of the University of Tennessee at Chattanooga in Partial Fulfillment of the Requirements of the Degree of Doctor of Philosophy in Computational Engineering

The University of Tennessee at Chattanooga
Chattanooga, Tennessee

December 2017
ABSTRACT

Two algorithms are presented which together generate well-spaced point distributions applied to curves, surfaces, and the volume of a computational domain. The first is a force equilibrium simulation which applies a simplified direct solution of the equations of motion at each node. Inter-nodal pair forces are computed based on the desired spacing between nodes and summed to provide a net force on each node. The nodes are allowed to travel a restricted distance with each locally distinct time step. The motion of the point distribution is stabilized by applying friction to each node from its neighboring nodes as well as globally restricting the time step size over the series of iterations. Second, an algorithm for node population adaptation is presented which deletes nodes or inserts new nodes depending on how well the local concentration of nodes matches a desired local spacing prescription, or spacing field. Experimental results are provided which demonstrate the ability of these algorithms to generate smooth distributions of points matching various spacing field function definitions.
DEDICATION

This work is dedicated to Amanda Fackler, my wife and my best friend forever. She is the only person on the ground at this time who has the right to destroy this document.
ACKNOWLEDGMENTS

I hardly have words with which to thank my parents, David and Teale Fackler. Let me simply say thank you for existence and uniqueness.

Gratitude must be shown to my undergraduate mathematics professors, Dr. David Coulliette, Dr. Duk Lee, and Dr. Ken Rietz for providing a strong initial gradient. I would never have considered something like this without their encouragement and training.

I would like to thank Dr. Steve Karman for his mentorship, for helping me see my first steps in this process, and for modeling the shape of enthusiasm in mesh generation research.

Dr. C. Bruce Hilbert deserves my gratitude for giving great advice, for enduring the bulk of my details-hashing, and for generally helping me see the forest in spite of my obsession with trees.

Thanks goes to Branch Technology, Inc., for allowing Dr. Hilbert time to continue iterating in his advisory role with myself along with his other students.

I thank Kim Sapp and Dr. Ethan Hereth for rendering me logistically functional and for helping me handle exceptions gracefully.

I would also like to express my thanks to my professors both in my undergraduate and my graduate training, for maintaining academic integrity and providing stability in the midst of unforeseen administrative oscillations.

Finally, gratitude belongs to the One born King of the Jews, Jesus. Though I have placed Him last here, there is a Day yet future when He will have preeminence in all things.
<table>
<thead>
<tr>
<th>TABLE OF CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT ................................................................. iv</td>
</tr>
<tr>
<td>DEDICATION ................................................................. v</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS ......................................................... vi</td>
</tr>
<tr>
<td>LIST OF FIGURES .......................................................... x</td>
</tr>
<tr>
<td>LIST OF TABLES ............................................................. xiv</td>
</tr>
<tr>
<td>LIST OF ALGORITHMS ...................................................... xv</td>
</tr>
<tr>
<td>LIST OF SYMBOLS .......................................................... xvi</td>
</tr>
<tr>
<td>CHAPTER</td>
</tr>
<tr>
<td>1. INTRODUCTION ............................................................. 1</td>
</tr>
<tr>
<td>1.1 Geometric Discretization Strategies ..................................... 1</td>
</tr>
<tr>
<td>1.2 Quality Measures ........................................................... 4</td>
</tr>
<tr>
<td>1.3 Chapter Previews ........................................................... 5</td>
</tr>
<tr>
<td>1.3.1 Chapter 2 ................................................................. 5</td>
</tr>
<tr>
<td>1.3.2 Chapter 3 ................................................................. 5</td>
</tr>
<tr>
<td>1.3.3 Chapter 4 ................................................................. 6</td>
</tr>
<tr>
<td>1.3.4 Chapter 5 ................................................................. 6</td>
</tr>
<tr>
<td>1.3.5 Chapter 6 ................................................................. 6</td>
</tr>
<tr>
<td>1.3.6 Chapter 7 ................................................................. 6</td>
</tr>
<tr>
<td>1.3.7 Appendices A, B, and C ............................................... 6</td>
</tr>
<tr>
<td>2. RELEVANT LITERATURE .................................................. 8</td>
</tr>
<tr>
<td>2.1 Bubbles ................................................................. 8</td>
</tr>
<tr>
<td>2.2 Molecular Dynamics Simulation .......................................... 12</td>
</tr>
<tr>
<td>2.3 Monte Carlo Potential Energy Minimization ................................ 14</td>
</tr>
<tr>
<td>2.4 Truss Equilibrium ......................................................... 17</td>
</tr>
</tbody>
</table>
2.5 Direct Simulation

3. METHOD OVERVIEW AND INTERFACE CONSIDERATIONS

3.1 Method Overview

3.1.1 Force Equilibrium Simulation

3.1.2 Population Adaptation

3.1.3 Domain Initialization

3.2 Interface Considerations

3.2.1 Geometry Processing

3.2.2 Tetrahedralization

3.2.3 Flow Solution

4. FORCE EQUILIBRIUM SIMULATION

4.1 Computing Forces

4.2 Local Time Stepping

4.3 Global Restrictive Time Step Ramp

4.4 Inter-Nodal Friction

4.5 New Locations and Spacings

4.6 Pair Force

5. POPULATION ADAPTATION

5.1 General Algorithm

5.2 Curve Adaptation Specifics

5.3 Surface Adaptation Specifics

5.4 Volume Adaptation Specifics

6. NUMERICAL TEST RESULTS

6.1 Sphere with Uniform Spacing

6.2 Ball In Box with Boundary Motion

6.3 Blow-Through CFD Adaptation From Non-Uniform Geometric Spacing

6.4 Ramp CFD Adaptation from Uniform Initial Spacing

6.5 Analytic Spacing Field Adaptation

6.6 ONERA M6 Wing

6.7 Store Separation

6.8 Timing Results

7. CONCLUSIONS AND FUTURE WORK

7.1 CFD Solution Adaptation
7.2 Viscous Layer Distributions ................................................................. 106
7.3 Distributed Memory Parallelism ......................................................... 106

REFERENCES ........................................................................................... 108

APPENDIX

A. SURFACE WALKING ........................................................................... 111
B. SURFACE (RE)TRIANGULATION ....................................................... 116
C. FLOW SOLUTION SPACING FIELD GENERATION ......................... 121

VITA ......................................................................................................... 124
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Bubble meshing pair force interpolant with $k = 1$ and $\sigma = 1$</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>Coulomb pair force definition with $C = 1$, $k = 2$, and $q_i = q_j = 1$</td>
<td>13</td>
</tr>
<tr>
<td>2.3</td>
<td>Modified Coulomb force definition with $C_R = C_A = 1$, $k = 3$, $m = 2$, and $q_i = q_j = 1$</td>
<td>14</td>
</tr>
<tr>
<td>2.4</td>
<td>Lennard-Jones pair potential (as pair force definition) with $a = 1$ and $\sigma = 1$</td>
<td>15</td>
</tr>
<tr>
<td>2.5</td>
<td>Linear spring force with $k = 1$ and $\sigma = 1$</td>
<td>17</td>
</tr>
<tr>
<td>3.1</td>
<td>Pair forces on node $i$</td>
<td>22</td>
</tr>
<tr>
<td>4.1</td>
<td>Vector from node $i$ to node $j$</td>
<td>30</td>
</tr>
<tr>
<td>4.2</td>
<td>Spacing values $q_i$ and $q_j$ and ideal spacing $\sigma_{ij}$ between node $i$ and node $j$</td>
<td>32</td>
</tr>
<tr>
<td>4.3</td>
<td>Pair forces between node $i$ and its local cloud neighbors</td>
<td>33</td>
</tr>
<tr>
<td>4.4</td>
<td>Net force at node $i$</td>
<td>34</td>
</tr>
<tr>
<td>4.5</td>
<td>Mirror point illustration</td>
<td>35</td>
</tr>
<tr>
<td>4.6</td>
<td>Node $i$’s minimum facing distance</td>
<td>38</td>
</tr>
<tr>
<td>4.7</td>
<td>Node $i$’s maximum translation vector</td>
<td>39</td>
</tr>
<tr>
<td>4.8</td>
<td>Normal force calculation</td>
<td>42</td>
</tr>
<tr>
<td>4.9</td>
<td>Modified spring pair force function</td>
<td>46</td>
</tr>
<tr>
<td>4.10</td>
<td>Pair pressure spring force function with $R = 0.05$, $k = 1$, and $\sigma = 1$</td>
<td>47</td>
</tr>
<tr>
<td>5.1</td>
<td>Example curve node overlap ratios</td>
<td>50</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>6.20</td>
<td>Ramp: Second CFD solution Mach number plot</td>
<td>79</td>
</tr>
<tr>
<td>6.21</td>
<td>Ramp stability statistics</td>
<td>79</td>
</tr>
<tr>
<td>6.22</td>
<td>Ramp: Surface adaptation and simulation after second CFD solution</td>
<td>80</td>
</tr>
<tr>
<td>6.23</td>
<td>Ramp: Tetrahedral mesh result after second CFD solution</td>
<td>81</td>
</tr>
<tr>
<td>6.24</td>
<td>Ramp: Third CFD solution Mach number plot and resulting spacing mesh (see Appendix C)</td>
<td>82</td>
</tr>
<tr>
<td>6.25</td>
<td>Ramp: Surface adaptation and simulation after third CFD solution</td>
<td>83</td>
</tr>
<tr>
<td>6.26</td>
<td>Two-dimensional version of analytic spacing function</td>
<td>84</td>
</tr>
<tr>
<td>6.27</td>
<td>Analytic spacing field resulting volume interior distribution</td>
<td>86</td>
</tr>
<tr>
<td>6.28</td>
<td>Analytic spacing field resulting surface distributions</td>
<td>87</td>
</tr>
<tr>
<td>6.29</td>
<td>Analytic spacing field resulting volume interior distribution</td>
<td>88</td>
</tr>
<tr>
<td>6.30</td>
<td>Analytic spacing stability statistics</td>
<td>89</td>
</tr>
<tr>
<td>6.31</td>
<td>ONERA M6 wing geometry top-down view</td>
<td>91</td>
</tr>
<tr>
<td>6.32</td>
<td>ONERA M6 wing geometry and seed points at base and tip</td>
<td>92</td>
</tr>
<tr>
<td>6.33</td>
<td>ONERA M6 wing base adaptation and simulation result</td>
<td>93</td>
</tr>
<tr>
<td>6.34</td>
<td>ONERA M6 wing tip adaptation and simulation result</td>
<td>94</td>
</tr>
<tr>
<td>6.35</td>
<td>ONERA M6 wing surface mesh</td>
<td>95</td>
</tr>
<tr>
<td>6.36</td>
<td>ONERA M6 volume distribution near wing surface</td>
<td>96</td>
</tr>
<tr>
<td>6.37</td>
<td>ONERA M6 wing tetrahedral mesh cut near wing surface</td>
<td>97</td>
</tr>
<tr>
<td>6.38</td>
<td>ONERA M6 stability statistics</td>
<td>97</td>
</tr>
<tr>
<td>6.39</td>
<td>Store separation stability statistics</td>
<td>98</td>
</tr>
<tr>
<td>6.40</td>
<td>Store separation initial adaptation and simulation result</td>
<td>99</td>
</tr>
</tbody>
</table>
LIST OF TABLES

6.1 Timing results for numerical test cases................................................................. 104
LIST OF ALGORITHMS

3.1 ControllerSequence .................................................. 21
3.2 FlowSolutionDriver .................................................. 28
4.1 ApplyFriction (Volume Interior) ................................. 41
4.2 ForceEquilibriumSimulation (Volume Interior) ............ 43
5.1 AdaptPopulation (General) .......................................... 52
5.2 CheckForDeletion (General) ........................................ 53
A.1 CurveWalk ............................................................... 112
A.2 SurfaceWalk ............................................................. 115
B.1 RetriangulateSurface ................................................ 119
B.2 InsertDomainNodes .................................................... 120
C.1 GenerateSpacingMesh ................................................ 123
LIST OF SYMBOLS

$\Delta t_i$  Node $i$’s local time step size

$\Delta x_i$  Distance node $i$ will travel in one simulation step

$\mu_k$  Inter-nodal kinetic friction coefficient

$\mu_s$  Inter-nodal static friction coefficient

$\psi$  Non-scaled pair force function

$\Psi_i$  Magnitude of non-scaled net force vector on node $i$

$\rho_i$  Overlap ratio of node $i$

$\sigma_{ij}$  Ideal spacing between node $i$ and node $j$, equal to $\frac{1}{2} \left(q_i + q_j\right)$

$c_i$  Drag coefficient for node $i$

$f$  Pair force function

$f_{ij}$  Magnitude of pair force between node $i$ and node $j$

$F_i$  Magnitude of net force on node $i$

$m_i$  Mass of node $i$
\( N_i \)  Number of neighbors in node \( i \)'s local cloud

\( q \)  Domain spacing field definition

\( q_i \)  Spacing value for node \( i \)

\( r_i^0 \)  Minimum facing distance for node \( i \): the distance to node \( i \)'s nearest neighbor in the hemispherical direction of its net force

\( r_{ij} \)  Distance from node \( i \) to node \( j \), magnitude of \( \mathbf{r}_{ij} \)

\( v_{i,0} \)  Node \( i \)'s initial velocity for a simulation step

\( W \)  Cutoff distance factor for local neighbor cloud

\( w_{ij} \)  Independent variable for non-scaled pair force function, \( w_{ij} = \frac{r_{ij}}{\sigma_{ij}} \)

\( \Delta t_{GM} \)  Global maximum time step size

\( \Delta x_i^{\text{max}} \)  Maximum distance node \( i \) is allowed to travel in one iteration

\( \delta (x) \)  The deletion flag on node \( x \)

\( \Delta x_i^{\text{max}} \)  Max translation vector for node \( i \) in one iteration

\( \Psi_i \)  Non-scaled net force vector on node \( i \)

\( f_{ij} \)  Pair force vector between node \( i \) and node \( j \)

\( F_i \)  Net force vector on node \( i \)

\( \mathbf{r}_{ij} \)  Vector from node \( i \) to node \( j \), \( \mathbf{r}_{ij} = \mathbf{x}_j - \mathbf{x}_i \)
\( x_i \)  Position of node \( i \)

\( \rho_{\text{add}} \)  Node addition threshold

\( \rho_{\text{del}} \)  Node deletion threshold

\( \rho_{\text{ideal}} \)  Ideal overlap ratio

TOL  Equilibrium tolerance
CHAPTER 1
INTRODUCTION

In computational continuum mechanics, there are multiple sources of error. Error is introduced from the discretization of the continuum physics model (the partial differential equations, for example), potentially from other algorithmic sources (such as early termination of an iterative process), and from numerical truncation inherent in floating-point arithmetic.

An additional source of error in a simulation is in the discretization of the problem domain, since the numerical methods cannot employ a continuous definition of the domain but instead must be solved at a finite number of points (optionally connected to form hyper-volumetric cells) which then form an approximation to the physical domain. The need to minimize this geometric discretization error is balanced by the constraints of time and computer memory. Therefore, much work has gone into strategies for providing fine resolution where necessary and coarser resolution in other regions of the domain. This the field known as grid (or mesh) generation.

1.1 Geometric Discretization Strategies

A structured grid is one for which the set of points are related to one another by the ordering of the data structure in which they are stored. For instance, a block of a three-dimensional structured mesh can be mapped to an axis-aligned cube such that there are $I$ points in the $x$-direction, $J$ points in the $y$-direction, and $K$ points in the $z$-direction, and the neighbors of the node $x_{i,j,k}$ can be
found simply by adding or subtracting any of the three indices $i$, $j$, or $k$. This type of grid fits naturally with the usual formulation of finite difference methods. General hexahedral volume cells are implied from this ordering either by using the nodes of the mesh as vertices or by defining the “faces” intersecting the midpoints between a node and its six neighbors. Thus, a finite volume method can also be applied on a structured grid. Notice that the structured grid is limited when it comes to discretizing complex geometric domains by the fact that a block must always have the same number of points along any $i$, $j$, or $k$ path. This constraint can be somewhat alleviated by composing a grid of multiple blocks that have different array sizes, but which must be fit together at their boundaries with matching face dimensions. Another approach is to construct multiple blocks (perhaps with one block discretizing each major feature of the domain geometry) which overlay one another in an arbitrary pattern and do not interact directly across faces. This is termed the “overset” grid generation technique. The overset blocks require a more complicated assembly process in which the computational field simulation data must be interpolated in the regions where the blocks overlap.

An unstructured grid is one in which the hyper-volumetric cells (that is, polygons in 2D, polyhedra in 3D) are explicitly defined because they cannot be extrapolated from the implied ordering of the nodes. Indeed there is no such ordering, and the relationships between nodes is obtained from the edges of the explicitly defined cells which connect the nodes. With an unstructured mesh, we may more naturally distinguish between a point set and the “connectivity” data, which together define the mesh. The same point set may be connected in different ways. The cells can be of arbitrary types, even from one cell to the next (for example, general polygons in 2D and polyhedra in 3D). The unstructured mesh is not constrained in the same way a structured mesh
is. There can be a more natural gradation in local cell size along any ray path in the mesh. This allows for tighter control of spacing in various regions of the domain and the freedom to coarsen the population where fine resolution is unnecessary.

Note that some computational field simulation methods, so-called meshless or mesh-free methods, do not require the connectivity at all, only the set of points. The point distributions generated by the proposed algorithms could be directly applied to such methods.

Given a set of points already distributed in the domain and a set of simplicial elements on the boundary of the domain, many have used the methods of Lawson [1] and of Bowyer [2] and Watson [3] to generate the connectivity as a complete set of internal simplices (triangles in 2D, tetrahedra in 3D). When no point distribution is available, usually the point set and connectivity are generated in tandem. Cartesian hierarchical refinement methods make use of fast geometric subdivision algorithms to produce cells with axis-aligned edges which are recursively subdivided to the desired local refinement [4, 5]. Extrusion methods insert points one layer at a time, marching out from the boundaries in paths initiated by the surface normal directions [6]. These methods can generate structured or unstructured meshes and are especially useful for producing high aspect ratio elements with wall spacing appropriate for resolving the viscous layer in computational fluid dynamics (CFD).

The standard techniques for generating point sets with corresponding simplicial connectivity are the Delaunay-based insertion methods and the advancing front methods [7], both of which insert points incrementally, updating the connectivity as necessary along the way.

The Delaunay-based algorithms usually create an initial superstructure mesh (overlapping the entire domain) into which the boundary faces are recovered. Subsequently, volume interior
nodes must be inserted sequentially, with the connectivity being updated with each insertion. A common strategy for interior node insertion is to iteratively insert points at the center of the circumscribing sphere of the element with the highest aspect ratio until some quality threshold is reached [8–11]. Of course, each candidate insertion must be tested to ensure it is inside the computational domain.

Advancing front methods insert points to form a layer (front) of elements at a time, marching away from each boundary, assuming that the ideal location of the next nodes can be computed from the boundary of the previous layer. When fronts collide (that is, when layers begin to overlap), elements in the collision region must be reconnected to avoid overlapping in the final product. In other words, the fronts must be joined [12–14].

Note that these various methods are often combined into a hybrid mesh, applying different techniques to different regions of the computational domain [4, 5].

1.2 Quality Measures

A plethora of metrics have been used to determine the quality of a given domain discretization in geometric terms. Although it is ultimately the quality of the solution for which the mesh is employed that determines the quality of the mesh, certain geometric properties are known to be desirable in certain application areas. Indeed, some finite volume CFD solvers place a requirement on, for example, the maximum internal angle of an element. For unstructured meshes, cell aspect ratio or weighted condition number are often used to measure the deviation of an element’s shape from that of an ideal element, and volume or edge-length ratios may be used to measure the size gradation from one cell to the next.
Talmor defined two metrics for a set of “well-spaced” points (WSP) on their own apart from the connectivity of the mesh [15]. Essentially, the spacing from a given node to its immediate neighbors should be bounded by some constant $\beta$ with respect to a spacing function for all nodes, and the size of all empty gaps in the domain should be bounded by some constant $\lambda$ with respect to the same spacing function. A spacing field for the purposes of this research is any function mapping a spatial position from the computational domain to a scalar value that defines the desired spacing in all directions at that point location.

The goal of the present research is to abstract the notion of spacing field function (the aspect of mesh generation that most requires human intuition), and automatically generate a set of points throughout the computational domain and its boundaries that are well-spaced, with respect to whatever arbitrary spacing field function is provided. The method is “physics-based” in that the nodes themselves are treated as physical entities, making use of applicable physics equations to produce the desired distribution.

1.3 Chapter Previews

1.3.1 Chapter 2

The literature relevant to physics-based point placement, especially as related to the current research, is reviewed. The primary work influencing this research is the “bubble” meshing work by Shimada and Gossard [16].

1.3.2 Chapter 3

A brief overview of the two proposed algorithms is presented. In addition, mention is made of the necessary supporting software, which the author’s current implementation depends on.
1.3.3 Chapter 4

The force equilibrium simulation algorithm is presented in detail, including force summing, local time stepping, global time step size restriction, the contribution of inter-nodal friction, and how node positions are updated. The differences among the specific algorithm implementations as applied to curves, surfaces, and volumes are noted. Finally, the inter-nodal pair force function definition used in the author’s implementation is presented.

1.3.4 Chapter 5

The algorithm for adapting the node population to an arbitrary spacing field is presented in detail, noting the differences necessary for applying the algorithm to curves, surfaces, and volume, individually.

1.3.5 Chapter 6

Numerical test case results are presented, demonstrating the ability of the proposed adaptation and simulation methods to match the desired spacing field. The method is also applied to moving boundaries and to CFD solution-based adaptation.

1.3.6 Chapter 7

The results of the method will be discussed and possibilities for future improvements will be suggested.

1.3.7 Appendices A, B, and C

Three aspects of completing the present research, which were themselves peripheral issues and yet required a significant amount of time and effort to accomplish, are presented. The first, a
surface walking algorithm, is a supporting geometric algorithm which the simulation and adaptation algorithms directly depend on. The latter two, a surface triangulation technique and a spacing field update based on flow solution data, were necessary in order to apply the proposed algorithms to adapting node distributions to better resolve CFD flow fields.
CHAPTER 2
RELEVANT LITERATURE

2.1 Bubbles

Shimada and Gossard [16, 17] presented a physics-based node placement method for mesh generation in which they hierarchically pack spheres in each dimensional entity (lines, surfaces, volume interior) and model the spheres as bubbles which interact with repulsive and attractive pair forces. They note, “the close packing of bubbles mimics a Voronoi diagram pattern, corresponding to well-shaped Delaunay triangles and tetrahedra” [16]. The node distribution is smoothed by numerically solving the differential equations of motion, written at the $i$th node as

$$m_i a_i = F_i - c_i \dot{v}_i$$  \hspace{1cm} (2.1)

or

$$m_i \frac{d^2 x_i}{dt^2} + c_i \frac{dx_i}{dt} = F_i$$ \hspace{1cm} (2.2)

where $x_i$ is the position of node $i$, $F_i$ is the sum of pair forces on node $i$

$$F_i = \sum_j f_{ij} (- \hat{r}_{ij})$$ \hspace{1cm} (2.3)
and \(-c_i v_i\) is the magnitude of the damping force based on node \(i\)'s current velocity. Damping is necessary to ensure that, solving these equations iteratively, the distribution will reach a stable configuration (equilibrium).

They devised a pair force function definition constructed as a cubic interpolant inspired by the van der Waals force. They constrain

\[
f = f(w_{ij})
\]

(2.4)

with

\[
w_{ij} = \frac{r_{ij}}{\sigma_{ij}}
\]

(2.5)

where \(r_{ij}\) is the distance between node \(i\) and node \(j\) (which are at the centers of the bubbles \(i\) and \(j\)), and \(\sigma_{ij}\) is the desired distance between the nodes

\[
\sigma_{ij} = \frac{q_i}{2} + \frac{q_j}{2}
\]

(2.6)

The distance \(\sigma_{ij}\) is computed as the sum of the radii of the \(i\)th and \(j\)th bubbles (that is, \(q_i\) is the diameter of the \(i\)th bubble), and is therefore the distance at which the two bubbles are just touching without overlap, or “kissing” [16].
To construct the interpolant \( f \), they enforce the following constraints:

\[
\begin{align*}
  f'(0) &= 0 \\
  f(1) &= 0 \\
  f'(1) &= -k \\
  f(1.5) &= 0
\end{align*}
\] (2.7)

where \( k \) is a linear spring constant (that is, near \( w = 1 \), the function should behave a linear spring force with spring constant \( k \)). With the last condition, they cut off any contribution to node \( i \)'s net force, \( F_i \), from neighbors \( j \) farther than \( 1.5\sigma_{ij} \) away from node \( i \). From these they derive

\[
f(w_{ij}) = \begin{cases} 
  k \left( 1.25w_{ij}^3 - 2.375w_{ij}^2 + 1.125 \right), & 0 \leq w_{ij} \leq 1.5 \\
  0, & 1.5 < w_{ij}
\end{cases}
\] (2.8)

This function behaves (similar to a spring) as repulsive \((f > 0)\) when \( r_{ij} < \sigma_{ij} \) and attractive \((f < 0)\) when \( r_{ij} > \sigma_{ij} \) up to the point that \( r_{ij} > 1.5\sigma_{ij} \). Figure 2.1 plots this pair force function definition, labeling it as \( f^{BM} \) to distinguish it from other pair force definitions presented in this dissertation. Note that equilibrium is reached when \( r_{ij} = \sigma_{ij} \), or when \( w_{ij} = 1 \).
Their method also applies adaptive bubble population control based on an “overlapping ratio” for each node. Population adaptation is necessary due to the fact that the repulsive and attractive forces can cause gaps in the domain and/or regions in which the bubbles are overlapped considerably beyond “kissing” distance. This “overlapping ratio” is said to represent the number of neighbors overlapped by a bubble with twice the diameter of the $i$th bubble

$$\rho_i = \frac{1}{q_i} \sum_j (2q_i + q_j - 2r_{ij})$$ (2.9)

Nie, Zhang, Liu, and Wang [18] followed up with a reproduction of this bubble meshing method and proved its “convergence” to a stable (equilibrium) configuration. Specifically, they proved that the average speed of the bubbles during the dynamic simulation tends to zero.
The bubble meshing method has had the most influence on this research since modeling the nodes as bubbles provides an easy means of intuitively understanding what should be happening in a node distribution. Let us now consider several other methods presented which are relevant to this topic.

2.2 Molecular Dynamics Simulation

A simulation method similar to the bubble meshing method was proposed by Zheleznyakova and Surzhikov [19], with the main difference being their choice of pair force definition coming from molecular dynamics. They treat the nodes as particles and apply Coulomb’s law as the inter-particle pair force. Every node is given positive charge, and the force acting between a pair of nodes is given by

\[ f_{ij} = C \frac{q_i q_j}{r_{ij}^k} \]  

(2.10)

where \( k \geq 2 \) and \( C \) is a constant. This function is never negative, but asymptotically approaches zero as \( r_{ij} \) increases. It therefore only ever applies a repulsive force between two nodes, even when they are farther away from each other than the ideal distance \( \sigma_{ij} \).

Note how \( q_i q_j \) relates to \( \sigma_{ij} \)

\[ q_i q_j = 2\sigma_{ij}^2 - \frac{1}{2} \left( q_j^2 + q_i^2 \right) \]  

(2.11)

as twice the square of the average minus the average of the squares. And when \( q_i = q_j \), they are equivalent. Thus, in Figure 2.2, for illustrative consistency, we keep the independent variable \( w \) by taking \( q_i = q_j = 1 \).
In order to effect proper clustering near boundaries, they modify the force definition when an interior node is interacting with a boundary node. That is, for an interior node $i$, and a wall node $j_w$, the pair force is defined as

$$f_{ij_w} = C_R \frac{q_i q_{j_w}}{r_{ij_w}^k} - C_A \frac{q_i q_{j_w}}{r_{ij_w}^m}$$  \hspace{1cm} (2.12)$$

with $m < k$, such that an interior node is attracted to the boundary when it is within the cutoff distance. This modified force function is shown in Figure 2.3. As in the bubble meshing method, a drag force is applied to each node, except that an exponent $p$ is applied

$$f_i^{\text{drag}} = -K v_i^p$$  \hspace{1cm} (2.13)$$

with $p \geq 2$ and $K$ a constant. The equations of motion, similar to (2.2), are numerically integrated to simulate the behavior of the nodes as particles at each time interval.
Figure 2.3  Modified Coulomb force definition with $C_R = C_A = 1$, $k = 3$, $m = 2$, and $q_i = q_j = 1$

### 2.3 Monte Carlo Potential Energy Minimization

A somewhat different approach was taken by Zhang and Smirnov [20], who modeled the nodes as particles and used a Monte Carlo simulation in their physics-based node placement scheme to minimize the system’s total potential energy

$$U = \sum_i \sum_{j>i} \phi_{ij}$$  \hspace{1cm} (2.14)

where $\phi_{ij}$ is computed as the Lennard-Jones pair potential between nodes $i$ and $j$

$$\phi_{ij} = \phi \left(\sigma_{ij}, r_{ij}\right) 4a \left[ \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6} \right]$$  \hspace{1cm} (2.15)
or, using our independent variable \( w_{ij} \) (since they also use \( \sigma_{ij} = \frac{1}{2} (q_i + q_j) \)),

\[
\phi_{ij} = \phi (w_{ij}) = 4a \left[ \left( \frac{1}{w_{ij}} \right)^{12} - \left( \frac{1}{w_{ij}} \right)^{6} \right] \tag{2.16}
\]

which can be rewritten as

\[
\phi_{ij} = 4a \left( \frac{1}{w_{ij}} \right)^{6} \left[ \left( \frac{1}{w_{ij}} \right)^{6} - 1 \right] \tag{2.17}
\]

This function is illustrated in Figure 2.4 (labeled as \( f^{LJ} \)) on the same scale as the preceding pair force functions.

![Figure 2.4](image)

Figure 2.4  Lennard-Jones pair potential (as pair force definition) with \( a = 1 \) and \( \sigma = 1 \)

Note that this function is a pair potential in the Monte Carlo method, and the corresponding pair force is

\[
f_{ij} = -\frac{d\phi_{ij}}{dr_{ij}} = 4a \left[ \frac{12\sigma_{ij}^{12}}{r_{ij}^{13}} - \frac{6\sigma_{ij}^{6}}{r_{ij}^{7}} \right] \tag{2.18}
\]
which simplifies to

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

(2.19)

This function does not cross zero at $w_{ij} = 1$. Rather, as noted by the authors, equilibrium is reached at a slightly larger spacing of

$$\sigma_{0,ij} = 2^{\left(\frac{1}{6}\right)} \sigma_{ij} \approx 1.1225 \sigma_{ij}$$

(2.20)

In their Monte Carlo simulation, a node is moved in a random direction and the change in the node’s potential energy sum is tested to determine if the move made an improvement. The test, using the Boltzmann distribution law, applies the following logic

$$\begin{align*}
\text{accept} & \quad \text{if } \exp(-\beta \Delta \phi_i) > R \\
\text{reject} & \quad \text{if } \exp(-\beta \Delta \phi_i) \leq R
\end{align*}$$

(2.21)

where

$$\Delta \phi_i = \sum_j \phi_{ij}^{n+1} - \sum_j \phi_{ij}^n$$

is the change in node $i$’s potential energy from state $n$ to state $n + 1$, $R \in (0, 1)$ is a random number generated at each trial move, and $\beta = \frac{1}{kT}$, in which $k$ is the Boltzmann constant and $T$ is temperature. The temperature $T$ is not true temperature, but is instead a parameter that is adjusted so that the acceptable range of energy increase is reasonable for the system.

The authors also implement a means of adaptive node population control based on the total system potential energy. When $U < 0$, there are gaps in the domain and nodes are added. When $U \gg 0$, the packing is too dense and nodes are removed.
2.4 Truss Equilibrium

Also of note is the force equilibrium simulation method of Persson and Strang [21], in which the triangulation is treated as a truss system. A simple linear, strictly repulsive spring force is used as the pair force function, defined here with our nomenclature

\[
 f_{ij} = \begin{cases} 
 k (1 - w_{ij}), & 0 \leq w_{ij} \leq 1 \\
 0, & 1 < r_{ij}
\end{cases}
\]

where \( k \) is the spring constant. This function is illustrated in Figure 2.5.

![Figure 2.5 Linear spring force with \( k = 1 \) and \( \sigma = 1 \)](image)

Nodes are moved directly according to the net force on them, and after each motion iteration, the nodes are re-triangulated at their new positions in order for the triangles to maintain the Delaunay
criterion. This means that at each iteration there is potentially a different set of truss edges from which forces are computed.

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

### 2.5 Direct Simulation

The author presented a method in two dimensions for force equilibrium simulation on node sets using the Lennard-Jones pair potential as the pair force [23]. This method simplifies the equations of motion (as shown in Chapter 4) such that numerical integration is not necessary, but instead the time step is restricted to ensure stability of node movement.

Karman and Wyman [24] apply a similar method in three dimensions to volume interior node distributions, and show that the overall quality of the tetrahedral meshes produced from the resulting node distributions is better than that obtained by traditional tetrahedral mesh generation techniques used in Pointwise [25]. They use the pre-triangulated surface mesh, as well as source points (and lines), to define local node spacing throughout the domain. They also involve population adaptation based on an overlap ratio that is different from the one used here and introduced by Shimada [16]. Rather, they compute a value that represents a net normalized sphere area covered by the projection of neighboring node’s spheres onto the sphere of the node in question.

\[
\rho_i = \sum_j \left[ \frac{\pi q_i^2}{4} \frac{q_j}{q_j} \right] \left( \frac{q_j}{2r_{ij}} \right) \tag{2.22}
\]
which can be simplified to

\[ \rho_i = \frac{1}{2q_i} \sum_j \left[ \frac{q_j^2}{r_{ij}} \right] \]  

(2.23)
CHAPTER 3

METHOD OVERVIEW AND INTERFACE CONSIDERATIONS

Here the main aspects of the force equilibrium simulation and the population adaptation algorithms are discussed as well as some utilities and 3rd-party tools used to aid in the implementation and validation of these algorithms.

3.1 Method Overview

The proposed method is split into two distinct parts which work together to distribute nodes in the domain: 1) a force equilibrium simulation used to smooth the distribution of nodes, and 2) a population adaptation algorithm used to properly match an arbitrary background spacing field. These two topics are each further split into three algorithms, one for curve interior nodes (excluding the endpoints), one for surface interior nodes (excluding the bounding curve nodes), and one for volume interior nodes (excluding all surface nodes). It should go without saying that curve nodes are surface nodes. Thus the modifying word “interior” is used of each dimensional entity. Note that, at each dimensionality, the algorithm is driven by the boundaries: a curve interior by its endpoints, a surface interior by its bounding curves, and the volume interior by the bounding surfaces (and curves).

The algorithms are structured together such that one adaptation sweep is applied, first to curves, then to surfaces, then to the volume, followed by a simulation sweep, first on curves, then on
surfaces, then in the volume (see Algorithm 3.1). Due to the inherent parallelism in that the curves are not dependent on each other and likewise the surfaces are not dependent on each other, the simulation and adaptation of the curves, and, in turn, the surfaces, is easily made multi-threaded.

### Algorithm 3.1: ControllerSequence

```plaintext
for <number-of-adapt-sweeps> do
  Adapt population in all curve interiors
  Adapt population in all surface interiors
  Adapt population in volume interior (see Algorithm 5.1)
  Smooth all curve interior nodes
  Smooth all surface interior nodes
  Smooth volume interior (see Algorithm 4.2)
end for
```

Note that, while it is not technically required by the algorithms, in the author’s current implementation a curve must be defined with endpoints, though it may have its beginning and end at the same point location. A surface, on the other hand may be defined without any bounding curves (a sphere, for example).

Note that the proposed method throughout depends heavily on a geometric searching algorithm for locating node neighbors. For this an implementation of the alternating digital tree (ADT) algorithm of Bonet and Peraire [26] was used. This tree structured is versatile in that it can store items based on bounding box in addition to storing based on a point location. Therefore, it is also used to store and query entities in the geometry definition, such as facets and segments.
3.1.1 Force Equilibrium Simulation

Given an initial distribution of points, a force equilibrium simulation is used to seek a stabilized “equilibrium” distribution of points. Figure 3.1 illustrates (in 2D) the process of computing and summing the pair forces on a node. A local point cloud is gathered within a capture radius based on a user-specified factor of the current node’s spacing value. This is illustrated by the large light blue circle. The semi-transparent shaded circle around each point represent that point’s desired spacing. Therefore, when two such circles overlap, there is a repulsive pair force, and when there is a gap between two circles, there is an attractive force. The equilibrium position is achieved when the circles are just touching without overlap. The net force on a node is the sum of the pair forces computed with each neighboring node captured in its local point cloud.

![Figure 3.1 Pair forces on node i](image)

Figure 3.1 Pair forces on node $i$
The physics-based equilibrium approach has been shown (see Chapter 2) to produce smooth point distributions which in turn can be tessellated to high quality meshes. The simulation used in the current work is based on the bubble meshing and molecular dynamics approach. In both of those works, a node’s force is summed from pair forces with its neighbors and numerical integration is used to solve the equations of motion, which includes a drag force dependent on the node’s current velocity. The physics of particle dynamics and bubble dynamics are borrowed and are of concern in the scope of this research only so far as they are useful for generating point distributions for computational meshes. Therefore, the point motion simulation used by [16] and [19] is significantly simplified. Starting from the generic equation of motion written at the $i$th node

$$m_i \frac{d^2 x_i}{dt^2} + c_i \frac{dx_i}{dt} = F_i$$  \hspace{1cm} (3.1)$$

where

$$F_i = \sum_{j=1}^{N_i} f_{ij}$$  \hspace{1cm} (3.2)$$

is the net force on node $i$, the following simplifications are made:

- Take $m_i = 1, \forall i$

- Remove damping

- Rewrite (3.1) as a scalar equation (in the direction of $F_i$)

$$\frac{d^2 x_i}{dt^2} = F_i$$  \hspace{1cm} (3.3)$$

23
• Assume $F_i$ is constant for the current time step and solve (3.3) to get

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

(3.4)

• Reset velocity to zero at each time step ($v_{i,0} = 0$)

• Allow for distinct local time step size

$$\Delta x_i = \frac{1}{2} \Delta t_i^2 F_i$$

(3.5)

Because of these simplifications, there is no automatic physical mechanism to stabilize the motion of the nodes. That is, without damping, the nodes would continue to “jitter” about indefinitely. Therefore, to force the distribution to “converge” to a stable configuration, the time step size is limited and ultimately ramped down to restrict node movement. In addition, instead of allowing the simulation to continue until some “convergence” criteria are met, the number of iterations (over which the global maximum time step ramp is applied) for each execution of the simulation is specified by the user before the run.

Finally, to allow more natural convergence, inter-nodal friction is contributed from neighbors which are in range to decrease the node’s force magnitude. The details of the force equilibrium simulation algorithm are presented in Chapter 4.
3.1.2 Population Adaptation

The “overlapping ratio” proposed by Shimada [16,17] is used here in a modified form (using $\sigma_{ij}$ in place of both $q_i$ and $q_j$)

$$\rho_i = \sum_j \left( 3 - \frac{2r_{ij}}{\sigma_{ij}} \right)$$  \hspace{1cm} (3.6)

as a measure of how well a node is packed within its neighbors. A node with perfectly distributed neighbors (that is, when each term in the sum in (3.6) are one) will have an overlap ratio of 2 for a curve node, 6 for a surface node, and 12 for a volume node. If a node’s overlap ratio is higher than the ideal value, the packing is too dense. Likewise, if it is lower, the packing is too sparse.

The overlap ratio can be interpreted as a measure of how many immediate neighbors a node has. Therefore it tells us not only whether a node’s packing is dense or sparse, but it actually gives us an estimate of the number of nodes that need to be added or deleted to fix the packing in the immediate region surrounding that node. This is used in an adaptation algorithm to correct the population concentration throughout the computational domain. The details of this as applied to curve, surface, and volume populations are provided in Chapter 5.

3.1.3 Domain Initialization

Several initialization techniques were investigated early on in this research. See the author’s master’s thesis [23] for examples in two dimensions. None of those will be discussed here, however, because once the adaptation scheme above was implemented, it became clear that it produced a better initial node distribution, and was no more expensive computationally. When a new node is inserted by the adaptation algorithm, it is added to the end of the queue, and it is therefore used, in its turn, to spawn more nodes around itself, and so on. Thus, from a single seed node, the
entire domain is populated, appropriately matching the given spacing field. As shown in Algorithm 3.1, the adaptation is done first in the controller sequence. Initialization, therefore, needs only to consist of ensuring that there is at least one seed point on each surface. In the author’s current implementation, the surfaces are defined by the distinct surface patches of the geometry, and so this amounts to copying the curve endpoints from the geometry. Those curve end points are seed the adaptation on curves, and the resulting curve nodes seed the adaptation of the surfaces they bound. In the case of a geometric surface which has no bounding curves (such as a sphere), the initialization copies one geometric surface point as seed node for that surface.

3.2 Interface Considerations

3.2.1 Geometry Processing

At many points throughout the proposed algorithms it is necessary to know where a node is in relation to the provided geometry definition. To make this straightforward, the author implemented a set of classes to provide an easy interface with a discrete faceted geometry definition. Most of the internal code for initially processing the geometry, performing inside-outside tests, and finding closest points on the surface was borrowed and refactored from Dr. Steve Karman’s geometry library. Some functionality was added which makes use of VTK [27, 28]. The most interesting contribution of the author to this geometric utility library is an algorithm for moving a distance along a triangulated surface. See Appendix A for the details of this “surface walking” algorithm as referenced in Chapters 4 and 5. In addition, the author’s implementation of the ADT [26] is used for geometric intersection testing with geometry facets and segments.
The geometries for all the numerical test cases in this research are defined as faceted surfaces and were generated using Pointwise [25].

3.2.2 Tetrahedralization

In order to obtain CFD solutions from a finite volume flow solver, the nodes distributed by the current method are connected into a linear tetrahedral mesh using the Tetmesh library written by C. Bruce Hilbert [29]. However, the Tetmesh library requires as part of its input a set of bounding triangles. An algorithm was therefore designed (peripheral to the current research) to generate a surface triangulation as an adaptation starting from the associated input geometry surface. The surface nodes are added incrementally to the geometry surface mesh, and subsequently, the original geometry nodes are removed, leaving a triangulation of only the surface nodes distributed by the proposed method. See Appendix B for the details.

3.2.3 Flow Solution

To thoroughly validate the proposed population adaptation scheme, the points are tessellated into a tetrahedral mesh which is subsequently fed to the basic finite volume Euler equation solver of SU2 [30, 31]. Following this, the spacing field for the nodes must be updated to appropriately adapt to the gradients of the solution. This is accomplished by applying a new spacing value to the nodes of the foregoing tetrahedral mesh and from that mesh interpolating spacing values to nodes in the proposed method as they are inserted or moved within the domain. See Appendix C for the details of how these spacing values are generated.
Note that when running flow solutions, the controller sequence above (see Algorithm 3.1) is called multiple times as shown in Algorithm 3.1. See Algorithm B.1 from Appendix B for the surface triangulation and Algorithm C.1 from Appendix C for the spacing mesh generation.

**Algorithm 3.2: FlowSolutionDriver**

1. for `<number-of-flow-solves>` do
2. call `ControllerSequence` (Algorithm 3.1)
3. Generate surface triangulations
4. run Tetmesh to generate tetrahedral mesh
5. run SU2 to obtain flow solution
6. Generate spacing values on tetrahedral mesh
7. Update node spacings
8. end for
CHAPTER 4
FORCE EQUILIBRIUM SIMULATION

In this chapter, the details of the force equilibrium method are provided. The main line of thinking follows the algorithm as applied to volume interior nodes (see Algorithm 4.2). Notes will be made for how the algorithm differs for surface interior nodes and curve interior nodes. The explanatory figures are in two dimensions (representative of surface interior node simulation) because that makes visualization easier and the extension of these operations to three dimensions is uncharacteristically trivial.

4.1 Computing Forces

At the beginning of each smoothing iteration, a net force must be computed on each node. This is done by summing pair forces between each node and its local cloud neighbors. This local cloud is defined by the nodes found within a certain distance from that node. This set of nodes may change from iteration to iteration since the nodes are moving. Therefore an ADT [26] instance must be reconstructed each iteration. This we will call the node tree. A similar approach is used for surface interior nodes, except that pair forces are contributed only by other surface nodes. For curve interior nodes, a tree is not used because the forces on a curve interior node is contributed to only by the immediate neighbors before and after that node on the curve (note that curve endpoints remain fixed).
The radius of the node’s local cloud is defined as the node’s current spacing value, \( q_i \), multiplied by a user specified cutoff distance factor, \( W \), taken to be 1.5 in all of our numerical test cases. In the figures that follow in this section, the cloud neighborhood is represented by the large light blue semi-transparent circle, using again \( W = 1.5 \).

For each neighboring node \( j \) in node \( i \)’s local cloud the vector, \( \mathbf{r}_{ij} \), from node \( i \) to node \( j \) is computed (see Figure 4.1).

Figure 4.1  Vector from node \( i \) to node \( j \)
From node \( i \)'s spacing value, \( q_i \), and node \( j \)'s spacing value, \( q_j \), the desired spacing between the two nodes is computed as the average

\[
\sigma_{ij} = \frac{1}{2} (q_i + q_j)
\]  

(4.1)

The spacing \( q_i \) is visualized as the diameter of the “bubble” centered at node \( i \). Therefore, the ideal pair spacing \( \sigma_{ij} \) is the sum of the radii of the pair of bubbles, such that if the skins of the bubbles were just touching, \( \sigma_{ij} \) is how far apart the centers would be.

Multiple definitions for the pair force are possible. In general, it is desirable for the nodes to repel one another if the distance \( r_{ij} \) is less than \( \sigma_{ij} \) and attract one another if it is greater. In Figure 4.2, since the distance from node \( i \) to node \( j \) is greater than \( \sigma_{ij} \), an attractive force will pull node \( i \) toward node \( j \).
The pair force between node $i$ and node $j$ is computed as a scalar value applied along the negative of the unit vector from node $i$ to node $j$:

$$f_{ij} = f_{ij} \left(-\frac{r_{ij}}{r_{ij}}\right)$$  \hfill (4.2)

where

$$f_{ij} = f(q_i, q_j, r_{ij})$$  \hfill (4.3)

Figure 4.3 illustrates all the neighbor pair forces acting on node $i$. (Note that local cloud nodes are numbered locally in this figure and vector lengths are not meaningful in an absolute sense, only
illustrating relative attraction and repulsion.) This figure illustrates the repulsive and attractive forces based on distance relative to desired distance. When node $i$’s bubble overlaps a neighboring bubble, a repulsive force on node $i$ results. When there is a gap between node $i$’s bubble and a neighboring bubble, node $i$ is attracted to that node. For example, $f_{i4}$ was computed as repulsive and thus points in the direction away from node 4, whereas $f_{i7}$ was results as an attractive force due to the gap between node $i$ and node 7.

![Figure 4.3 Pair forces between node $i$ and its local cloud neighbors](image)

The net force on node $i$ from its neighbors is the sum...
\[ \mathbf{F}_i = \sum_{j=1}^{N_i} \mathbf{f}_{ij} \]  

(4.4)

Figure 4.4 illustrates the resultant net force vector on node \(i\). (Again, the length of the vector is illustrative only.) If the maximum net force magnitude on a node within a collection (curve, surface, or volume) is less than the equilibrium tolerance, \(TOL\), then the simulation stops before reaching the specified number of simulation steps.

Figure 4.4 Net force at node \(i\)

When a node is near a boundary (that is, boundary nodes or geometry facets are intersected by the node’s local cloud neighborhood), an additional influence contributes (per boundary surface
the node is near) to the node’s net force. A phantom “mirror” point is created at a position on the surface from which the surface normal points to the node in question. This mirror point is given a spacing value equal to one fourth that of the current node, and its pair force is added to the node’s net force, $F_i$. This prevents an interior volume node from crossing the boundary while also allowing nodes to settle into positions appropriately close to the surface between surface nodes. An example of this is shown in Figure 4.5, in which the relative size of the “bubbles” has been reduced for clarity. Surface nodes are light blue and volume nodes are dark blue. One volume node near the surface, highlighted in green, is affected by a mirror point (in red). The triangles shown are from the geometry surfaces.

Figure 4.5  Mirror point illustration

35
4.2 Local Time Stepping

Recall from Equation (3.5) rewritten here for convenience

\[ \Delta x_i = \frac{1}{2} \Delta t^2 F_i \]  

(4.5)

That is, we allow for the time step on each node to be distinct.

Here we point out that the pair force definition is a function, \( \psi \), of the ratio of the current distance, \( r_{ij} \), between two nodes to the desired distance, \( \sigma_{ij} \), between them, and therefore, must be scaled by the local spacing. That is, to expand (4.3)

\[ f_{ij} = f(q_i, q_j, r_{ij}) = q_i \psi(w_{ij}) \]  

(4.6)

where

\[ w_{ij} = \frac{r_{ij}}{\sigma_{ij}} \]  

(4.7)

The value of \( \psi(w_{ij}) \) is not scaled by the local spacing. With this fact we can write (4.4) as

\[ F_i = \sum_{j=1}^{N_i} f_{ij} = \sum_{j=1}^{N_i} \left[ q_i \psi(w_{ij}) \left( -\frac{r_{ij}}{r_{ij}} \right) \right] = q_i \sum_{j=1}^{N_i} \left[ \psi(w_{ij}) \left( -\frac{r_{ij}}{r_{ij}} \right) \right] \]  

(4.8)

If we label the sum on the right hand side

\[ \Psi_i = \sum_{j=1}^{N_i} \left[ \psi(w_{ij}) \left( -\frac{r_{ij}}{r_{ij}} \right) \right] \]  

(4.9)
and its magnitude $\Psi_i$, we can write (4.5) as

$$\Delta x_i = \frac{1}{2} q_i \Delta t^2 \Psi_i$$

(4.10)

Therefore, we set a maximum distance node $i$ is allowed to travel $\Delta x_i^{\text{max}}$ and solve (4.5) for $\Delta t^2_i$

$$\Delta t^2_i = \frac{2}{q_i \Psi_i} \Delta x_i^{\text{max}}$$

(4.11)

This maximum distance is determined by computing the distance, $r^0_i$, to node $i$’s nearest neighboring node in the hemispherical direction of $F_i$

$$r^0_i = \min_j \left\{ r_{ij} : \frac{r_{ij}}{r_{ij} \cdot F_i} > 0 \right\}$$

(4.12)

From our previous illustrative example, we see in Figure 4.6 that node 5 is the nearest of the facing neighbors 1, 5, 6, and 7.
Figure 4.6  Node $i$’s minimum facing distance

We compute the maximum allowed travel distance as a fraction of this minimum facing distance, $\frac{2}{3}$ in the author’s current implementation ($\frac{1}{3}$ for curve nodes).

$$\Delta x_{i}^{\text{max}} = \frac{2}{5} r_{i}^{0}$$  \hspace{1cm} (4.13)

The vector

$$\Delta x_{i}^{\text{max}} = \Delta x_{i}^{\text{max}} \frac{F_{i}}{F_{i}}$$ \hspace{1cm} (4.14)

is illustrated in Figure 4.7.
which makes our candidate time step calculation

\[ \Delta t_i^2 = \frac{4r_i^0}{5q_i \Psi_i} \]  

(4.15)

4.3 Global Restrictive Time Step Ramp

With the formulation up to this point, each node being allowed to move its \( \Delta x_i^{\text{max}} \), the simulation would never reach stability, because the nodes would jitter about endlessly. In terms of a much simpler equilibrium problem, it is like modeling a single marble in a bowl as maintaining constant acceleration (increasing velocity) even after passing the bottom of the bowl. One way of preventing this is to apply a global maximum time step size to try to inch the marble toward
minimum potential energy without overstepping it. This is not easy to do. If time step is too large, the problem persists, and if it is too small, the nodes won’t be free enough to smooth out properly or they will take a very long time to do so. Therefore we have employed a simple linear ramp, from high to low, within a range of time step sizes based on the non-scaled pair force definition, such that the simulation begins with the nodes moving relatively freely and, by the end of the user-specified number of simulation steps, the motion of the nodes is almost prohibited. With this, node $i$’s time step size is updated with

$$
\Delta t_i^2 = \min \{ \Delta t_i^2, \Delta t_{GM} \}
$$

(4.16)

### 4.4 Inter-Nodal Friction

Another measure taken to encourage the distribution to stabilize more quickly (and more naturally) is to apply friction from neighboring nodes. Once the candidate time step $\Delta t_i$ has been calculated, as shown in Section 4.2, based on the sum of pair forces on node $i$, the nodes whose “bubble skins” are overlapping with that of node $i$ contribute to a net normal force. Each such neighboring node’s contribution is taken as the magnitude of the pair force vector with the component along the direction of $F_i$ removed

$$
F_{i,\text{normal}} = \sum_{j=1}^{N_i} \|\mathbf{f}_{ij} - \text{proj}_{\mathbf{F}_i} \mathbf{f}_{ij}\| \quad (4.17)
$$

The user-specified values for static and kinetic friction coefficients are then used to reduce node $i$’s net force, either to zero if $\mu_s F_{i,\text{normal}}$ is large enough to cancel out $F_i$, or by $\mu_k F_{i,\text{normal}}$. This logic is made explicit in Algorithm 4.1. Figure 4.8 illustrates this process. The individual complement
vectors, \( \mathbf{f}_{ij,n} = \mathbf{f}_{ij} - \text{proj}_{\mathbf{F}_i} \mathbf{f}_{ij} \), are placed on the neighboring nodes from which they are computed to avoid clutter. Note that friction is not computed in the simulation for curve nodes).

\[ \text{Algorithm 4.1: ApplyFriction (Volume Interior)} \]

\begin{algorithm}
\begin{algorithmic}
\State \textbf{Input: } \( i \)
\State \textbf{begin}
\State \hspace{1em} \( F_{i,\text{normal}} = 0 \)
\State \hspace{1em} \textbf{for } \( j \) \textbf{ from } 1 \textbf{ to } N_i \textbf{ do}
\State \hspace{2em} \textbf{if node } \( j \) \textbf{ overlaps with node } \( i \) \textbf{ then}
\State \hspace{3em} \( F_{i,\text{normal}} = F_{i,\text{normal}} + \| \mathbf{f}_{ij} - \text{proj}_{\mathbf{F}_i} \mathbf{f}_{ij} \| \)
\State \hspace{2em} \textbf{end if}
\State \hspace{1em} \textbf{end for}
\State \hspace{1em} \textbf{if } \( F_i - \mu_s F_{i,\text{normal}} \leq 0 \) \textbf{ then}
\State \hspace{2em} \( F_i = 0 \)
\State \hspace{1em} \textbf{else}
\State \hspace{2em} \( F_i = F_i - \mu_k F_{i,\text{normal}} \)
\State \hspace{1em} \textbf{end if}
\State \textbf{end}
\end{algorithmic}
\end{algorithm}
4.5 New Locations and Spacings

Once every node’s force and time step size have been calculated, the location of each node is updated as

\[ x_i = x_i + \Delta x_i \frac{F_i}{F_j} \]

where \( \Delta x_i \) is computed according to (4.5). Surface and curve nodes travel the distance \( \Delta x_i \) as an arc length along their respective surface or curve. See Appendix A for the details of how this is done.
At a new location, a node’s spacing value must be updated from the domain spacing field since it is a function of location in the domain,

\[ q_i = q(x_i) \quad (4.18) \]

Algorithm 4.2 outlines the steps detailed in this chapter in pseudo-code.

**Algorithm 4.2: ForceEquilibriumSimulation (Volume Interior)**

```plaintext
for <number-of-simulation-steps> do
    Store node tree
    for i from 1 to <number-of-interior-nodes> do
        Gather local cloud in neighborhood of node i
        for j from 1 to N_i do
            Contribute pair force from node j to node i
        end for
        if node i is near at least one boundary then
            Contribute normal forces from mirror nodes
        end if
        Compute candidate time step
        call ApplyFriction(i)
    end for
    if max_i \{F_i\} < TOL then
        Break out of simulation loop
    end if
    Apply global maximum time step to all nodes
    Update node locations
    Update node spacings
end for
```
4.6 Pair Force

The non-scaled pair force function could conceivably be defined arbitrarily. Indeed, in the author’s implementation, one may specify at run-time which function object to use to evaluate the pair force. Generally speaking, for the purposes of this method, it is desirable to generate a repulsive force if nodes are too close and an attractive force if the nodes are not quite close enough. This is seen in the cubic polygon of Shimada [16] and the Lennard-Jones pair potential exploited by Zhang [20] and used in the author’s master’s thesis [23]. For the numerical test cases in this research, we have devised a modified spring force

\[
f^{MS} = M \begin{cases} 
1 - \alpha w, & 0 \leq w < 1 - R \\
0, & 1 - R \leq w \leq 1 + R \\
1 - \alpha (w - 2R), & 1 + R < w \leq T \\
\beta (w - W), & T < w \leq W \\
0, & W < w 
\end{cases}
\tag{4.19}
\]

with the two slopes defined by

\[
\alpha = \frac{1}{1 - R} \tag{4.20}
\]

and

\[
\beta = \frac{1 - \alpha (T - 2R)}{T - W} \tag{4.21}
\]

This function has several constant parameters \( M, W, T, \) and \( R \) which allow for the shape to be adjusted. These parameters are defined as follows.
$M$ Scaling factor

$R$ Radius of tolerance around $w = 1$ allowing for small overlap (or gap)

$T$ Transition point at which the slope turns positive: $T \in (1 + R, W)$

$W$ The cutoff distance factor is used to specify where the function reaches zero

Figure 4.9 plots this function with the parameters as used in all experimental cases:

\[
M = 1 \\
R = 0.05 \\
T = 1.15 \\
W = 1.5
\]
This function is preferable to the Lennard-Jones function used by the author previously for several reasons. First, it is much cheaper to compute. Second, the unit scale makes values more meaningful and easier to work with. Finally, the weaker slope and the tolerance radius, \( R \), around \( w = 1 \) accord with more realistic expectations for stability, whereas the steep slope of the Lennard-Jones function passing through \( w = 1 \) make it nearly impossible for a pair of nodes to satisfy equilibrium. This pair force function along with the inter-nodal friction presented in Section 4.4 constitute a shift from a microscopic model of the nodes to a macroscopic model. Intuitively, the nodes should interact and behave in the way bubbles do, not like microscopic particles.

One further note about force functions is needed here. In order for neighbor nodes that are just touching node \( i \) to have a friction contribution, this piece-wise modified spring force cannot be
used, since it contributes a zero value in that configuration. A different force function, called the pair pressure force, is used when contributing to $F_{i,\text{normal}}$ as discussed in Section 4.4. This function (shown in Figure 4.10) is a linear spring force modified so as to apply a positive force even with $w_{ij}$ slightly greater than one. Using $R$ to designate how far past $w = 1$ to shift the function, we have

$$f_S = \begin{cases} 
k (1 + R - w), & 0 \leq x \leq 1 + R \\
0, & 1 + R < x 
\end{cases}$$

(4.22)

Figure 4.10  Pair pressure spring force function with $R = 0.05$, $k = 1$, and $\sigma = 1$
5.1 General Algorithm

As seen in Algorithm 3.1, curves, surfaces, and the volume interior are adapted separately and in that order. The algorithms applied to each of these distinct types of node collections are equivalent in the general steps they take and differ only in the details of how those steps are executed.

A delete flag is applied to each node as the algorithm progresses. This flag may have one of three possible values

\[
\delta(x) = \begin{cases} 
1, & \text{x will be deleted} \\
0, & \text{x currently will not be deleted} \\
-1, & \text{x may not be deleted this sweep}
\end{cases} \quad (5.1)
\]

We make use of two queues: a set of nodes to visit for adaptation, A, and a set of nodes to be perturbed, P. Both of these start as the empty set, \(\emptyset\). All boundary nodes are added to A first and automatically given a delete flag of -1 since the boundary distribution is taken as immutable during the adaptation of the current collection. Note that for curves, the boundary is the curve end point nodes, which may or may not correspond to geometry critical points. For surfaces, the boundary
includes all curve nodes. And for the volume, the boundary includes all surface and curve nodes. After boundary nodes are added, interior nodes which exist at entrance to the algorithm are added to the adaptation queue, $A$, and their delete flags (re)set to 0. As noted in the overview of this algorithm in Chapter 3, there must be at least one seed node belonging to the collection at the beginning of this algorithm, because the algorithm is driven by the nodes in the adaptation queue, $A$.

A node $x$ is popped from the front of $A$ and its overlap ratio $\rho$ is computed from

$$\rho_i = \sum_j \left( 3 - \frac{2r_{ij}}{\sigma_{ij}} \right) = \sum_j (3 - 2w_{ij})$$

For surface and volume adaptation this requires gathering a local neighbor cloud using the ADT [26], but for curve adaptation, just the immediate curve neighbors of $x$ are used. When computing the overlap ratio of a boundary node, an additional pair overlap of 1 is added in the place of each missing neighbor outside the collection. For curve adaptation, there is one phantom neighbor, for surface adaptation there are two phantom neighbors, and for volume adaptation there are four phantom neighbors.

The pair overlap ratio is a linear function of the ratio $w_{ij}$. If the two nodes are perfectly spaced, the pair overlap ratio is one. Figure 5.1 shows some examples for curve nodes. For nodes on the left, each pair overlap ratio is one, but for nodes on the right, where $w_{ij} = \frac{1}{2}$, each pair overlap ratio is two. Note that the end points get one extra phantom pair overlap ratio.
Likewise in Figure 5.2, examples are shown for surface nodes. The total overlap ratio shown is for the central node in each sub-figure. In Sub-Figure C, the nodes to the left and right contribute a pair overlap ratio of 1.5 each since the ratio $w_{ij}$ is $\frac{3}{4}$.

If the overlap ratio exceeds the deletion threshold, $\rho_{\text{del}}$, we must delete one or more nodes. The logic for this is provided in Algorithm 5.2. If the overlap ratio is lower than the addition threshold, $\rho_{\text{add}}$, new nodes must be spawned around node $x$. The immediate neighbors of $x$ are given a delete flag of -1 since deleting them would contradict the need to insert new neighbors. These new nodes are added to the back of the adaptation queue, $A$, and they and their immediate
neighbors are also given a delete flag of -1. If the overlap ratio is between the addition and deletion thresholds, nothing is done.

Note that the adaptation queue, $A$, grows when new nodes are spawned such that those new nodes are visited in turn to potentially spawn more nodes. Therefore, as noted in Chapter 3, this algorithm may be used to fill the entire geometric entity with nodes provided at least one seed node.

When the adaptation queue, $A$, is finally exhausted, each node collected in the perturbation queue, $P$, is visited. The node $x_i$ is perturbed by a random vector in the closed ball of radius $\gamma q_i$, where $\gamma$ is a factor between 0 and 1 (for the numerical test cases, $\gamma = 0.2$ was used). Finally, all nodes with delete flag of 1 are deleted from the collection.

The overlap ratio can be interpreted as the number of overlapping neighbors a node has. It therefore can be used to indicate how many nodes need to be deleted. Therefore in Algorithm 5.2, it is not assumed that only one node will be deleted when the overlap ratio exceeds $\rho_{del}$. Instead, an estimate of how many nodes need to be deleted locally is calculated as

$$c = \text{round} \left( \frac{\rho - \rho_{\text{ideal}}}{\rho_{\text{del}} - \rho_{\text{ideal}}} \right)$$

(5.3)

If $c$ is one, the sensible choice is to delete the central node, $x$. In this case, the delete flags of the immediate neighbors of $x$ are set to -1, since we have determined they do not need to be deleted. If $c > 1$, however, it makes more sense to delete $c$ of $x$’s neighbors and keep $x$ itself.
Algorithm 5.1: AdaptPopulation (General)

begin

$A = \emptyset$

$P = \emptyset$

for all boundary nodes, $x_b$ do

$A \leftarrow x_b$

$\delta (x_b) = -1$

end for

for all interior nodes, $x_i$ do

$A \leftarrow x_i$

$\delta (x_i) = 0$

end for

while $A \neq \emptyset$ do

$x = \text{pop}(A)$

Compute overlap ratio $\rho = \rho (x)$

if $\rho > \rho_{del}$ then

call CheckForDeletion($x, \rho, P$)

else if $\rho < \rho_{add}$ then

for all $x$'s neighbors, $x_{nbr}$ do

$\delta (x_{nbr}) = -1$

end for

Spawn new nodes around $x$, $\{x_{new}\}$

$A \leftarrow \{x_{new}\}$

for all new nodes and their neighbors, $x_n$ do

$\delta (x_n) = -1$

end for

end if

end while

while $P \neq \emptyset$ do

$x_i = \text{pop}(P)$

$u = \text{rand} \{-1, 1\}$

$\theta = \text{rand} \{0, 2\pi\}$

$r = \text{rand} \{0, \gamma\}$

$x_i = x_i + q_i r \left[ \sqrt{1-u^2} \cos \theta, \sqrt{1-u^2} \sin \theta, u \right]$

end while

Delete nodes with $\delta (x) \equiv 1$

end
Algorithm 5.2: CheckForDeletion (General)

Input: $x$, $\rho$, $P$

begin
if $\delta(x) \leq 0$ then
  $c = \text{round} \left( \frac{\rho - \rho_{\text{ideal}}}{\rho_{\text{del}} - \rho_{\text{ideal}}} \right)$
  if $c \equiv 1$ then
    $\delta(x) = 1$
    for immediate neighbors of $x$, $\{x_{nbr}\}$ do
      $\delta(x_{nbr}) = -1$
    $P \leftarrow x_{nbr}$
  end for
else
  while $c > 0$ and $x$ has neighbors left do
    $x_{nbr} = \text{next nearest neighbor of } x$
    if $\delta(x_{nbr}) \equiv 0$ then
      $\delta(x_{nbr}) = 1$
      $c = c - 1$
    end if
  end while
  $P \leftarrow x$
  $\delta(x) = -1$
end if
end
5.2 Curve Adaptation Specifics

For curve adaptation, the random perturbation step in Algorithm 5.1 is omitted, as are any references to the list \( P \) of nodes to be perturbed. Perturbing the nodes is used in volume and surface adaptation to avoid equilibrated hulls when a point is deleted. Also, the selfish deletion portion of Algorithm 5.2 may include the entire curve, not just the immediate neighbors of the current node.

When spawning new nodes from a curve node, \( x_i \), the distance to the node’s immediate neighbors is checked, and at most two nodes are inserted as needed, one before and/or one after \( x_i \). These insertions are done a distance of \( q_i \) away from \( x_i \) along the curve using the curve walking algorithm of Appendix A. This corresponds with the ideal overlap ratio for curve nodes of \( \rho_{\text{ideal}} = 2 \). Addition and deletion thresholds used for curve nodes in the numerical test cases are \( \rho_{\text{add}} = 1.75 \) and \( \rho_{\text{del}} = 3 \), respectively.

5.3 Surface Adaptation Specifics

The ideal overlap ratio for a surface node \( x \) is \( \rho_{\text{ideal}} = 6 \), and (with approximately uniform local spacing), the neighboring nodes will be arranged in a regular hexagonal pattern surrounding \( x \). From this comes the motivation for the method of spawning new surface nodes. We form a reference regular hexagon, which is subsequently transformed: scaled with \( x \)’s spacing value, centered at \( x \)’s location, and rotated to the orientation of the local surface normal. This means that, if \( x \) has no neighbors and is not near a bounding curve, then six new nodes will be spawned in a regular hexagonal pattern around \( x \) (the vertices of the transformed reference hexagon). In general, however, \( x \) does have neighbors. The reference hexagon is rotated once to line up the first point with the first neighbor encountered. Existing neighbors then eliminate points from the reference
distribution. Additionally, if \( x \) is near or on a bounding curve, points from the reference distribution which fall outside the surface are removed. Finally any reference points leftover are inserted as \( \{ x_{\text{new}} \} \).

Note that, to insert a new point spawned from \( x \) on the surface, we cannot simply insert the vertex from the transformed reference hexagon, since the surface is not necessarily planar. Therefore, the distance and direction from \( x \) to the reference vertex are used in the surface walking algorithm of Appendix A to insert the new node. The addition and deletion thresholds used for surface nodes in the numerical test cases are \( \rho_{\text{add}} = 5 \) and \( \rho_{\text{del}} = 8 \), respectively.

This surface node insertion process is illustrated by an exaggerated example in Figure 5.3. First a single node (in blue) exists as a seed node on the surface. The reference hexagonal distribution is oriented to the surface normal at that point, and since there are no neighbors, all six vertex points (in red) are added to the surface. In the second row, new nodes are spawned from one of the nodes just inserted (in green). Once again the reference distribution is oriented at that point. This time, three of the vertices are eliminated due to the presence of neighboring nodes (\( \rho = 3 \)), and the remaining three nodes (in red) are inserted on the surface.
5.4 Volume Adaptation Specifics

The approach taken to spawn new interior volume nodes is similar to that for surface nodes, but remember that the ideal overlap ratio for a volume node is $\rho_{\text{ideal}} = 12$. Therefore the reference distribution used is a regular icosahedron. Once again this reference distribution is translated to
be centered at \( x \)'s location. It is scaled by \( 0.975q \) where \( q \) is \( x \)'s spacing value. This is to ensure that the vertices have a distance to each other of about \( q \) as that would not be the case with a radius of \( q \). Since we now have three rotational degrees of freedom, the reference distribution is rotated three times, once to line up a vertex with each of the first three immediate neighbors encountered in \( x \)'s local cloud. Each of \( x \)'s immediate neighbors is used to eliminate a point from the reference distribution, and if \( x \) is near or on a bounding surface, points from the reference distribution which lie outside the computational domain are removed. Finally, the remaining reference points are inserted as \( \{ x_{\text{new}} \} \). The addition and deletion thresholds used for volume nodes in the numerical test cases are \( \rho_{\text{add}} = 11 \) and \( \rho_{\text{del}} = 16 \), respectively.
CHAPTER 6
NUMERICAL TEST RESULTS

In this chapter, several cases are shown to demonstrate the results obtained from the algorithms presented. Node distribution plots were created using VisIt [32]. Figures labeled “adaptation” refer to results of the population adaptation algorithm prior to smoothing. Likewise, figures labeled “simulation” refer to results of the force equilibrium simulation smoothing the distribution.

Results will consider the use of several different spacing field functions:

• uniform spacing

• a geometry-based spacing field (updated at run-time with boundary motion)

• an analytic spacing field

• run-time modification of the spacing field to adapt to CFD solution data

The geometry-based spacing field is used in the initialization for the CFD cases. For surface and curve nodes, the average local geometry spacing is used. For interior volume nodes, an inverse-distance-squared weighted average is used to combine spacing values from multiple surfaces

\[
q = \frac{\sum_{i=0}^{N_s} \left( \frac{q_s}{d_s^2} \right)}{\sum_{i=0}^{N_s} \left( \frac{1}{d_s^2} \right)}
\]

(6.1)
6.1 Sphere with Uniform Spacing

A sphere geometry is used (defined by eight surfaces), and a uniform spacing value (slightly larger than the average geometric spacing) throughout the domain. The geometry and initial seed nodes (copied from the geometry curve end points) are shown in Figure 6.1. In Figure 6.2, the results of the adaptation and simulation algorithms are shown for curves, surfaces, and the volume interior for half the sphere. Figure 6.3 shows the history of the maximum and average force magnitudes for the volume interior simulation.

Figure 6.1 Sphere geometry and initial seed nodes
(a) Curve adaptation.  
(b) Curve simulation.  

c) Surface adaptation.  
(d) Surface simulation.  

(e) Volume adaptation.  
(f) Volume simulation.  

Figure 6.2 Sphere case distribution results
Histograms for the well-spaced points (WSP) metrics are shown in Figure 6.4 for the volume interior nodes. The $\beta$ metric is the normalized distance to each node’s nearest neighbor, and the $\lambda$ metric is the normalized diameter of the largest gap connected to each node. A gap connected to a node is defined as an open ball region which has that node (and possibly others) on its boundary but contains no nodes. The outlier with $\lambda = 2$ means that somewhere in the volume distribution there is a gap large enough to insert one node.
Figure 6.4  WSP metrics for sphere case uniform-spacing volume interior distribution
6.2 Ball In Box with Boundary Motion

In this case, a sphere within a cube, the geometry-based spacing field is used. Figure 6.6 shows a partial cut of the geometry with the initial seed nodes followed by the subsequent adaptation from these nodes. The geometry-based spacing field as computed in (6.1) changes as the sphere boundary moves downward in the box. The sphere is moved 100 times in the downward direction by the same amount. The resultant node distribution at selected steps in this motion series are shown in Figure 6.7, and the stability history for the entire run is shown in Figure 6.5.

![Figure 6.5 Ball in box stability statistics](image-url)
Figure 6.6  Ball in box initial adaptation result

(a) Before adaptation.

(b) After adaptation.
Figure 6.7 Ball in box boundary motion results
6.3 Blow-Through CFD Adaptation From Non-Uniform Geometric Spacing

This case is initially adapted to the geometry-based spacing field using a cube geometry with dramatically varied local facet spacing as shown in Figure 6.8. Figure 6.9 shows the initial adaptation and simulation results using the geometric spacing, and in Figure 6.10, the distribution of “well-spaced points” (WSP) metrics is shown as computed for every interior volume node in this initial point distribution.

![Non-uniform geometry](image)

(a) Full  
(b) Zoomed

Figure 6.8 Non-uniform geometry
Figure 6.9  Non-uniform initial distribution
Figure 6.10  WSP metrics for non-uniform initial distribution
This initial distribution was iteratively coarsened by adapting to a “blow-through” CFD solution. That is, with Mach number gradients of zero everywhere in the domain, the spacing adjusted spacing field (see Appendix C) was incrementally increased until it reached the specified maximum value everywhere. The results at each step of this process are shown in Figure 6.12, and the stability history of the entire run is shown in Figure 6.11.

Figure 6.11 Blow-through stability statistics
Figure 6.12 Blow-through coarsening steps
6.4 Ramp CFD Adaptation from Uniform Initial Spacing

This case begins with (almost) uniform geometric spacing on a ramp geometry as shown in Figure 6.13. The corner points of the geometry are used as fixed seed nodes. The initial distribution based on the geometry spacing is shown in Figure 6.14, and the histograms of the WSP metrics for the interior volume nodes of this initial distribution are shown in Figure 6.15. Note that the relative gap size $\lambda$ skews more closely to unity.

![Figure 6.13 Ramp geometry with initial seed nodes](image)

After triangulating the surface nodes (see Appendix B) and subsequently generating a tetrahedral mesh in the volume, a CFD solution is obtained on said mesh using SU2 [31]. The gradient of Mach number from this solution is computed and the spacing field (previously uniform) is
adjusted to match the intensity of those gradients (see Appendix C for the details). The population is adapted to this new spacing field, and the entire process is repeated. The flow solution and adaptation results of performing this cycle three times are shown in Figures 6.16, 6.17, 6.18, 6.20, 6.22, 6.24, and 6.25. The tetrahedral meshes generated from the node distributions are shown in Figures 6.16, 6.19, and 6.23, showing both the front surface mesh and a crinkle cut midway through the volume. The stability statistics for the entire run are shown in Figure 6.21.
Figure 6.14  Ramp initial adaptation and simulation result
Figure 6.15  WSP metrics for ramp case initial distribution
Figure 6.16  Ramp: Tetrahedral mesh of initial distribution and first CFD solution Mach number plot.
Figure 6.17  Ramp: Surface adaptation and simulation after first CFD solution
Figure 6.18  Ramp: Volume adaptation and simulation after first CFD solution
Figure 6.19  Ramp: Tetrahedral mesh result after first CFD solution
Figure 6.20  Ramp: Second CFD solution Mach number plot

Figure 6.21  Ramp stability statistics
Figure 6.22  Ramp: Surface adaptation and simulation after second CFD solution
Figure 6.23  Ramp: Tetrahedral mesh result after second CFD solution
Figure 6.24  Ramp: Third CFD solution Mach number plot and resulting spacing mesh (see Appendix C)
Figure 6.25  Ramp: Surface adaptation and simulation after third CFD solution
6.5 Analytic Spacing Field Adaptation

A cube geometry is used here with $x, y, z \in [0, 1]$, but the distribution is adapted to an analytic spacing field defined as a spherical wave function

$$q = a_0 + a_1 \left( 1 + \sin \left( a_2 \sqrt{x^2 + y^2 + z^2} \right) \right) \quad (6.2)$$

where $a_0 = 0.025$, $a_1 = 0.05$, and $a_2 = 10.0$. The two-dimensional version of this function, $a_0 + a_1 \left( 1 + \sin \left( a_2 \sqrt{x^2 + y^2} \right) \right)$ is plotted in Figure 6.26 for $x, y \in [-1, 1]$. The upper quadrant of this plot, then, is the region corresponding to the domain in this test case.

Figure 6.26 Two-dimensional version of analytic spacing function
Results are shown for two different surfaces in Figure 6.28 and for the volume interior in Figure 6.29. The stability history for the run is shown in Figure 6.30. A perspective view is shown in Figure 6.27 below.
Figure 6.27 Analytic spacing field resulting volume interior distribution
Figure 6.28 Analytic spacing field resulting surface distributions
Figure 6.29  Analytic spacing field resulting volume interior distribution
Figure 6.30  Analytic spacing stability statistics
6.6 ONERA M6 Wing

The methods presented in this work were used on a more “real-world” geometry, namely NASA’s ONERA M6 wing geometry [33], shown with seed points in a top-down view in Figure 6.31 and in a perspective view at the wing base and tip in Figure 6.32. Distribution visualizations are provided for the wing base and tip in Figures 6.33 and 6.34, respectively. Figure 6.36 shows the volume distribution around the wing cut halfway down the wing, and Figure 6.38 provides the stability history of the run. Notice that the spacing field is somewhat chaotic on the surface of the wing and this is radiated into the volume by the geometry-based spacing field as can be seen in Figure 6.36. A tetrahedral mesh was generated from the resulting node distribution. The wing surface mesh is shown in Figure 6.35, and a crinkle cut of the volume tetrahedra near the wing surface is shown in Figure 6.37.
Figure 6.31  ONERA M6 wing geometry top-down view
Figure 6.32  ONERA M6 wing geometry and seed points at base and tip
Figure 6.33 ONERA M6 wing base adaptation and simulation result
Figure 6.34   ONERA M6 wing tip adaptation and simulation result
(a) Surface mesh at wing base.

(b) Surface mesh at wing tip.

Figure 6.35  ONERA M6 wing surface mesh
Figure 6.36  ONERA M6 volume distribution near wing surface
Figure 6.37  ONERA M6 wing tetrahedral mesh cut near wing surface

Figure 6.38  ONERA M6 stability statistics
6.7 Store Separation

This case is similar to the ball in a box case of Section 6.2. The geometry-based spacing field function is used based on wing, pylon, and store geometry. The figures here are cut and zoomed in to show the space between the pylon and the store as the store is moved downward over 100 equally-sized motion steps. Figure 6.40 shows the initial population adaptation which populates the domain. Figure 6.39 shows the stability history of the entire run, and in Figures 6.41, 6.42, and 6.43 show the resulting distribution after the force equilibrium simulation at selected steps in the motion series. In the figures below, the more brightly colored spheres are surface nodes, and the dark blue spheres are the volume interior nodes.

Figure 6.39 Store separation stability statistics
Figure 6.40  Store separation initial adaptation and simulation result
Figure 6.41  Store separation boundary motion results
Figure 6.42  Store separation boundary motion results
Figure 6.43  Store separation boundary motion results
6.8 Timing Results

Some run times from the above test cases are provided in Table 6.1. All of these cases were run on a machine with 16 2.10GHz cores. Note that a few of the cases with a comparable number of nodes have significantly different run times (for example the analytic function case compared with the second ramp case run). This variation can be attributed mainly to the cost of the different spacing functions used. The spacing function used in the ramp case searches for the closest tetrahedron in the background spacing mesh and interpolates the spacing values to the test point. The analytic function, on the other hand, is just a mathematical function evaluation without any geometric searching, and is therefore much faster.
Table 6.1  Timing results for numerical test cases

<table>
<thead>
<tr>
<th>Case</th>
<th>No. of Nodes</th>
<th>Adaptation (s)</th>
<th>Adaptation (mm:ss)</th>
<th>Simulation (s)</th>
<th>Simulation (mm:ss)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sphere</td>
<td>4132</td>
<td>0.33</td>
<td>0:00</td>
<td>5.81</td>
<td>0:06</td>
</tr>
<tr>
<td>Blow Through Box</td>
<td>2717</td>
<td>0.46</td>
<td>0:00</td>
<td>1.64</td>
<td>0:02</td>
</tr>
<tr>
<td></td>
<td>404</td>
<td>0.05</td>
<td>0:00</td>
<td>1.23</td>
<td>0:01</td>
</tr>
<tr>
<td></td>
<td>116</td>
<td>0.02</td>
<td>0:00</td>
<td>1.06</td>
<td>0:01</td>
</tr>
<tr>
<td>Analytic</td>
<td>39177</td>
<td>0.02</td>
<td>0:00</td>
<td>7.7</td>
<td>0:08</td>
</tr>
<tr>
<td></td>
<td>13926</td>
<td>3.79</td>
<td>0:04</td>
<td>11.8</td>
<td>0:12</td>
</tr>
<tr>
<td></td>
<td>36667</td>
<td>3.27</td>
<td>0:03</td>
<td>15.13</td>
<td>0:15</td>
</tr>
<tr>
<td></td>
<td>538920</td>
<td>38.14</td>
<td>0:38</td>
<td>145</td>
<td>2:25</td>
</tr>
<tr>
<td>ONERA M6 Wing</td>
<td>2054477</td>
<td>1816</td>
<td>30:16</td>
<td>3709</td>
<td>61:49</td>
</tr>
<tr>
<td>Store Separation (avg)</td>
<td>333076</td>
<td>366</td>
<td>6:06</td>
<td>489</td>
<td>8:09</td>
</tr>
</tbody>
</table>
CHAPTER 7

CONCLUSIONS AND FUTURE WORK

The results of the previous chapter demonstrate that the population adaptation scheme and the force equilibrium simulation presented herein together successfully generate well-spaced, relatively smooth point distributions matching any given continuous spacing field function. The global time step restriction and the application of friction from overlapping neighboring nodes provides for the method to finish in a feasible time frame, but at the cost of smoothness. When smoother distributions are desired, the friction can be decreased, the time step size increased. But this would require that the simulation be run for more time steps. There is therefore a trade-off between smoothness and the program run time. Another option is to perform further smoothing on the resulting tetrahedral mesh, as was done by Karman and Wyman [24] using Pointwise [25].

Several future work goals proposed in the author’s master’s thesis [23] were accomplished in this work, foremost among them being the population adaptation algorithm (which incidentally proved to be a better initialization method as well). Several important features that still require investigation are noted here.

7.1 CFD Solution Adaptation

Note that the adjusted spacing field shown in Figure 6.24 is not nearly as tightly focused as the Mach number solution in the same figure. However, notice that the subsequent adaptation
in Figure 6.25 still matched that spacing field, unsatisfactory though it was. The algorithm for adjusting the spacing field obviously needs further improvements. This is important because if the spacing field could be adjusted more accurately, the adaptation scheme presented here would potentially require fewer flow solution runs to acquire the desired spatial resolution than traditional edge splitting mesh adaptation methods.

### 7.2 Viscous Layer Distributions

The method presented in this research produces locally isotropic meshes, which are not in themselves useful for real-world CFD analysis. In its current form, however, it could be useful, for example, in the stitching region between a viscous layer extruded mesh and a hierarchical Cartesian far field mesh. On the other hand, one area of potential future investigation would be to provide a built-in means of producing viscous layer node distributions. An idea for this is to grow filaments of nodes extending from each surface node. These filaments would behave similarly to boundary curves, except that the shape of the curve would be allowed to change as the filament is grown and as it interacts with neighboring filaments.

### 7.3 Distributed Memory Parallelism

A note on parallelism is needed here. Parts of the author’s current implementation use shared memory parallelism. When curves or surfaces are being adapted or smoothed, one curve does not depend on another and one surface does not depend on another. Therefore the algorithms can be run on these entities on multiple threads. Likewise the interior volume simulation is easily multi-threaded, since computing forces on one node does not affect the forces on any other
node. However, if these algorithms were ever applied in a large-scale production environment, an appropriate means of distributed memory parallelism would have to be investigated.
REFERENCES


[32] "VisIt." Lawrence Livermore National Laboratory, wci.llnl.gov/simulation/computer-codes/visit.

[33] "ONERA M6 Wing." http://www.grc.nasa.gov/WWW/wind/valid/m6wing/m6wing.html.
APPENDIX A

SURFACE WALKING
Given a position on a segmented curve $C$ or triangulated surface $S$, we compute a relative position, provided a distance $d$ and an initial direction $\hat{v}$, by walking along the surface the distance $d$ and updating the direction $\hat{v}$ as the surface normal changes. For walking along a curve this is relatively straightforward (see Algorithm A.1 below). The segments in the curve are ordered. We choose a version of a “next” operator and a “dir” (direction) operator based on the orientation of $\hat{v}$ with respect to the curve ordering (forward or reverse). Then, from one segment to the “next”, if the combined segment lengths do not exceed the distance $d$, we move on to the next segment. Otherwise we return the point along the segment at which the combined length is exactly $d$. Note the last parameter of the function is a segment $e$ which is assumed to contain the point $x$.

**Algorithm A.1: CurveWalk**

Input: $x$, $d$, $\hat{v}$, $C$, $e$

Output: $x_{\text{new}}$, $e$

```
begin
    Choose “next” and “dir” operator from $\hat{v}$
    $x_{\text{new}} = x$
    $\Delta x = 0$
    while $\Delta x < d$ and $e$ is valid do
        if $\Delta x + \text{length}(e) > d$ then
            $x_{\text{new}} = x_{\text{new}} + (d - \Delta x) \text{dir}(e)$
            $\Delta x = d$
        else
            $x_{\text{new}} = x_{\text{new}} + \text{length}(e) \text{dir}(e)$
            $\Delta x = \Delta x + \text{length}(e)$
        end if
        $e = \text{next}(e)$
    end while
    return $\{x_{\text{new}}, e\}$
end
```
Walking along a triangulated surface requires a little more math (see Algorithm A.2 below). Once again we begin with an element, a triangle, which is assumed to contain the point \( \mathbf{x} \). The point \( \mathbf{x}_{\text{new}} \) starts at \( \mathbf{x} \) and will travel from one triangle to a neighboring triangle iteratively until it has traveled the distance \( d \) or reached the edge of the surface \( S \). At each iteration in the loop, the direction vector \( \mathbf{v} \) is projected onto the plane of the current triangle, \( e \). Then, the side of \( e \) which aligns most closely with \( \mathbf{v} \) is determined and labeled \( s_{\text{max}} \). The “next” side of \( e \) from \( s_{\text{max}} \) in the direction of \( \mathbf{v} \) is labeled \( s_{\text{next}} \). This is presumed to be the side of \( e \) which \( \mathbf{x}_{\text{new}} \) wishes to cross, traveling in the \( \mathbf{v} \) direction to the neighboring triangle on the opposite side of \( s_{\text{next}} \). This is what will happen if the test segment \( s_{\text{test}} \) intersects the segment \( s_{\text{next}} \).

The test segment is formed from \( \mathbf{x}_{\text{new}} \) along \( \mathbf{v} \) some distance \( l_{\text{test}} \) greater than the size of the triangle. If this intersection test fails, our presumption proved false, and we test for the intersection of \( s_{\text{test}} \) with \( s_{\text{max}} \). If these segments intersect, then \( s_{\text{next}} \) is set to \( s_{\text{max}} \), and that is the side of \( e \) that \( \mathbf{x}_{\text{new}} \) will cross. If this intersection also fails, this represents an error with respect to geometric numerical tolerance. If the segments \( s_{\text{test}} \) and \( s_{\text{max}} \) are determined to be parallel, the node shared by \( s_{\text{max}} \) and \( s_{\text{next}} \) is chosen as the intersection point. In the author’s implementation, the no-intersection error is handled more thoroughly by testing if the point \( \mathbf{x}_{\text{new}} \) is actually outside the triangle \( e \) and contained in one of \( e \)’s neighbors. But for this presentation of the algorithm, we assume \( \mathbf{x}_{\text{new}} \) is always contained in \( e \).

Finally, with logic similar to that for curve walking, if traveling to the boundary of \( e \) at \( \mathbf{x}_s \) exceeds the travel distance \( d \), then the point within \( e \) is found which satisfies that distance. Otherwise, \( \mathbf{x}_{\text{new}} \) and \( e \) are updated and the loop continues.
Note that if the loop exits due to $e$ being invalid, this means that the point $x_{\text{new}}$ traveled up to the edge of the surface. This information can be useful in different contexts when applying this algorithm.
Algorithm A.2: SurfaceWalk

Input: \( x, d, \hat{v}, S, e \)
Output: \( x_{\text{new}}, e \)

begin

\( x_{\text{new}} = x \)
\( \Delta x = 0 \)

while \( \Delta x < d \) and \( e \) is valid do

\( \hat{n} = \) surface normal of triangle \( e \)
\( v = \hat{v} - (\hat{v} \cdot \hat{n}) \hat{n} \)
\( \hat{v} = \frac{v}{\|v\|} \)

\( s_{\text{max}} = \arg \max_{s \in e} \{\|\text{dir} (s) \cdot \hat{v}\|\} \)

if \( \text{dir} (s_{\text{max}}) \cdot \hat{v} < 0 \) then

\( s_{\text{next}} = \text{prev}_e (s_{\text{max}}) \)

else

\( s_{\text{next}} = \text{next}_e (s_{\text{max}}) \)

end if

\( s_{\text{test}} = \{x_{\text{new}}, x_{\text{new}} + l_{\text{test}} \hat{v}\} \)

if intersection \( (s_{\text{test}}, s_{\text{next}}) \neq \emptyset \) then

\( x_s = \text{intersection} (s_{\text{test}}, s_{\text{next}}) \)

else if intersection \( (s_{\text{test}}, s_{\text{max}}) \neq \emptyset \) then

\( x_s = \text{intersection} (s_{\text{test}}, s_{\text{max}}) \)

\( s_{\text{next}} = s_{\text{max}} \)

else if \( s_{\text{test}} \parallel s_{\text{max}} \) then

\( x_s = \) node shared by \( s_{\text{max}} \) and \( s_{\text{next}} \)

else

\( \text{error} \) (no intersection)

end if

if \( \Delta x + \text{distance} (x_{\text{new}}, x_s) > d \) then

\( x_{\text{new}} = x_{\text{new}} + (d - \Delta x) \hat{v} \)
\( \Delta x = d \)

else

\( x_{\text{new}} = x_s \)
\( \Delta x = \Delta x + \text{distance} (x_{\text{new}}, x_s) \)

end if

\( e = \text{neighbor}_e (s_{\text{next}}) \)

end while

return \( \{x_{\text{new}}, e\} \)

end
APPENDIX B

SURFACE (RE)TRIANGULATION
Given a triangulation $T_G$ of a set of points $V_G$ on a surface $S$, a triangulation $T_D$ of another set of points $V_D$, which also lie on $S$, may be constructed in the following manner (see Algorithm B.1). An intermediate triangulation, $T$, which will involve nodes from both $V_G$ and $V_D$, is initially equivalent to $T_G$. For clarity we refer to the nodes of $V_G$ as geometry nodes and the nodes of $V_D$ as domain nodes.

Some remarks on notation are necessary before we begin. The set of nodes forming the triangle $e$ is denoted $N(e)$ and may be sub-scripted with an index $i \in \mathbb{Z}_3$ When a side $s$ of the triangle $e$ is used as a sub-script it refers to the node at the beginning of side $s$. There are three distinct (but related) uses of the calligraphic letter $T$. The unary operator $T(x)$ represents the set of all the triangles in $T$ which share the node $x$. The binary operator $T(e,s)$ is the set of at most one triangle which neighbors the triangle $e$ opposite its side $s$. The unary operator $T(P)$ represents the triangulation of the general polygon $P$. On that note the unary operator $H(T(x))$ is the transformation of the set of triangles attached to node $x$ into a single polygon with the node $x$ and all edges directly connected to $x$ removed. Finally, $B(e)$ is the set of edges of the triangle $e$.

Each domain node is added, in turn, to the triangulation $T$ by one of three possible means. First, if the domain node is close to a geometry node, it simply replaces the geometry node in the connectivity, removing all references to that particular geometry node in the triangulation $T$. Second, if $x_D$ is on a bounding curve of $S$, the triangle, $e$, containing $x_D$ is removed and split into two new triangles sharing a new edge between $x_D$ and the node opposite the boundary edge of $e$. Third, if the domain node $x_D$ is not on a bounding curve of $S$, the triangle $e$ is removed and replaced with three new triangles, each of which connects the two end points of one of $e$’s sides with $x_D$. For
the latter two cases, the edges of the newly created triangles are all tested and flipped if necessary to maximize quality (see Lawson’s method [1]).

Once all the nodes from $V_D$ are added to the triangulation, the geometry nodes that were not replaced by domain nodes in the previous step must be removed. This is done for each such node, $x_G$, by combining the triangles attached to $x_G$ into one polygon with $x_G$ and the edges directly connected to it removed. Those triangles are removed from $T$, and the polygon, $P$, is triangulated. This set of new triangles is added to $T$ to replace the ones attached to $x_G$ so that there is no gap.

Once all geometry nodes have been removed, the algorithm follows up with a number of sweeps in which the edges of all triangles in $T$ are tested and flipped if necessary to get as close to attaining the Delaunay criterion as possible (the triangulation is constrained by bounding segments).

The author makes no claim of originality for this algorithm. (In fact, the CheckEdge method referenced in the algorithm was refactored from code lifted from $vtkDelaunay2D.cxx$, and the polygon triangulation operator, $T(P)$ is in fact a call to $vtkPolygon::Triangulate$. [27, 28]) This is simply the author’s best attempt after failing to find an open-source library that would triangulate a set of surface points. Any such software that could be used for that required that the surface be transformable to the $x − y$ plane. And this is not always possible.
Algorithm B.1: RetriangulateSurface

Input: $T_G, V_G, V_D$
Output: $T_D$

1 begin
2 $T = T_G$
3 call InsertDomainNodes($T, V_D$)
4 for $x_G \in V_G$ with $T(x_G) \neq \emptyset$ do
5 Form polygon $\mathcal{P} = \mathcal{H}(T(x_G))$
6 $T = T \setminus T(x_G)$
7 Triangulate polygon, $T(\mathcal{P})$
8 $T = T \cup T(\mathcal{P})$
9 end for
10 for $e \in T$ do
11 for $s \in \mathcal{B}(e)$ with $T(e, s) \neq \emptyset$ do
12 call CheckEdge($s$)
13 end for
14 end for
15 return $T_D = T$
16 end
Algorithm B.2: InsertDomainNodes

Input: $T, V_D$

begin
for $x_D \in V_D$ do

    $x^c_G = \arg \min_{x_G \in V_G} d(x_D, x_G)$

    if $d(x_D, x^c_G) < TOL_{rep}$ and $T(x^c_G) \neq \emptyset$ then

        for $e \in T(x^c_G)$ do

            $N(e) = (N(e) \setminus \{x^c_G\}) \cup \{x_D\}$

            $T(x^c_G) = T(x^c_G) \setminus \{e\}$

            $T(x_D) = T(x_D) \cup \{e\}$

        end for

    end if

else

    Find triangle $e \in T$ closest to $x_D$

    if $x_D$ is on a side $s$ of $e$ and $T(e, s) \equiv \emptyset$ then

        Create two new triangles: $e_1, e_2$

        $N(e_1) = \{x_D, N(e)_{s+1}, N(e)_{s+2}\}$

        $N(e_2) = \{x_D, N(e)_{s+2}, N(e)_{s+3}\}$

        $T = T \cup \{e_1, e_2\}$

    end if

else

    Create three new triangles: $e_1, e_2, e_3$

    $N(e_1) = \{x_D, N(e)_0, N(e)_1\}$

    $N(e_2) = \{x_D, N(e)_1, N(e)_2\}$

    $N(e_3) = \{x_D, N(e)_2, N(e)_0\}$

    $T = T \cup \{e_1, e_2, e_3\}$

end if

$T = T \setminus \{e\}$

for $e_{new} \in T(x_D)$ do

    for $s \in B(e_{new})$ with $T(e_{new}, s) \neq \emptyset$ do

        call CheckEdge(s)

    end for

end if

end for

end
APPENDIX C

FLOW SOLUTION SPACING FIELD GENERATION
After running SU2 [30, 31] on a tetrahedralization $T$ of the domain nodes (see Chapter 3), the flow solution Mach number is read from the SU2 restart file for each node in the mesh. Any variable (or combination of variables) might have been used, but Mach number is the variable chosen for applying the method proposed in this research to flow solution adaptation. The gradient of Mach number is computed for each node, and the maximum and minimum gradient magnitudes are found, $G_{\min}$ and $G_{\max}$. Note that the tetrahedral mesh is the connectivity generated on the current set of domain nodes. Thus the current spacing value is used as the starting point for the spacing mesh. A tolerable range of gradient magnitude is specified by $[G_0, G_1]$, and a maximum and minimum allowable spacing is specified as $[q_{\min}, q_{\max}]$. Additionally, the maximum and minimum factors by which a local spacing value may be increased or decreased are defined by $\{R_0, R_1\}$, where $R_1 \in (0, 1)$ and $R_0 > 1$.

If the magnitude of the gradient at a node is within the tolerable range $[G_0, G_1]$, no change is made. For $\|\nabla M_i\| \in [0, G_0)$, the spacing is increased on a scale of how much smaller than $G_0$ the value is. On the other hand, for $\|\nabla M_i\| \in (G_1, G_{\text{cutoff}}]$, the spacing is decreased on a scale of how much larger than $G_1$ the value is. The cutoff value $G_{\text{cutoff}}$ is defined as some relatively large number simply to limit the size of the interval used to update the spacing values. The spacing value $q_i$ may not be decreased to less than $q_{\min}$ or increased to more than $q_{\max}$.

After updating the spacing value at every node in the tetrahedral mesh, $T$, the gradation of the spacing values across the domain must be smoothed, since we allow for $R_0$ and $R_1$ to be somewhat ambitious. This is done by limiting the increase in spacing across any single tetrahedron $e$ in $T$ to a factor $\nu$, which is set to something around 2.0. That is, the spacing field may not increase by more than double across an element in the mesh $T$. 

122
Algorithm C.1: GenerateSpacingMesh

Input: \( T, G_0, G_1 \)

begin

Read Mach number from SU2 output

Compute gradients

\[ G_{\text{min}} = \min_i \{\|\nabla M_i\|\} \]

\[ G_{\text{max}} = \min \left\{ \max_i \{\|\nabla M_i\|\}, G_{\text{cutoff}} \right\} \]

\[ \{D_0, D_1\} = \left\{ R_0 - 1, \frac{1}{2} - R_1 \right\} \]

Set current spacing values on mesh nodes

for each node \( x_i \) do

if \( \|\nabla M_i\| < G_0 \) then

\[ q_i = q_i \left( 1 + D_0 \frac{G_0 - \|\nabla M_i\|}{G_0 - G_{\text{min}}} \right) \]

\[ q_i = \max \{q_i, q_{\text{max}}\} \]

else if \( \|\nabla M\| > G_1 \) then

\[ q_i = q_i \left( \frac{1}{2} - D_1 \frac{\|\nabla M_i\| - G_1}{G_{\text{max}} - G_1} \right) \]

\[ q_i = \min \{q_i, q_{\text{min}}\} \]

end if

end for

smoothing = true

while smoothing do

smoothing = false

for \( e \in T \) do

\[ q_{\text{limit}} = \nu \min_{i \in N(e)} \{q_i\} \]

for \( i \in N(e) \) do

if \( q_i > q_{\text{limit}} \) then

\[ q_i = q_{\text{limit}} \]

smoothing = true

end if

end for

end for

end while

end
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, receiving the Commonwealth diploma in May, 2004. He graduated with honors from Asbury College (now Asbury University) of Wilmore, Kentucky, in May of 2008 with a Bachelor of Arts degree in Mathematics. In December of 2013, Philip received a Master of Science degree in Computational Engineering from the University of Tennessee at Chattanooga.