Mesh generation for voxel -based objects

Hanzhou Zhang
West Virginia University

Mesh Generation for Voxel-Based Objects

Hanzhou Zhang

Dissertation submitted to the

College of Engineering and Mineral Resources

at West Virginia University

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

in

Mechanical Engineering

Department of Mechanical and Aerospace Engineering

Morgantown, West Virginia

2005

ABSTRACT

Mesh Generation for Voxel-Based Objects

Hanzhou Zhang

A new physically-based approach to unstructured mesh generation via Monte-Carlo simulation is proposed. Geometrical objects to be meshed are represented by systems of interacting particles with a given interaction potential. A new way of distributing nodes in complex domains is proposed based on a concept of dynamic equilibrium ensemble, which represents a liquid state of matter. The algorithm is simple, numerically stable and produces uniform node distributions in domains of complex geometries and different dimensions. Well-shaped triangles or tetrahedra can be created by connecting a set of uniformly-spaced nodes. The proposed method has many advantages and potential applications.

The new method is applied to the problem of meshing of voxel-based objects. By customizing system potential energy function to reflect surface features, particles can be distributed into desired locations, such as sharp corners and edges. Feature-preserved surface mesh can then be constructed by connecting the node set.

A heuristic algorithm using an advancing front approach is proposed to generate triangulated surface meshes on voxel-based objects. The resultant surface meshes do not inherit the anisotropy of the underlying hexagonal grid. However, the important surface features, such as edges and corners may not be preserved in the mesh.

To overcome this problem, surface features such as edges, corners need to be detected. A new approach of edge capturing is proposed and demonstrated. The approach is based on a Laplace solver with incomplete Jacobi iterations, and as such is very simple and efficient. This edge capturing approach combined with the mesh generation methods above forms a simple and robust technique of unstructured mesh generation on voxel-based objects.

A graphical user interface (GUI) capable of complex geometric design and remote simulation control was implemented. The GUI was used in simulations of large fuel-cell stacks. It enables one to setup, run and monitor simulations remotely through secure shell (SSH2) connections. A voxel-based 3D geometrical modeling module is built along with the GUI. The flexibility of voxel-based geometry representation enables one to use this technique for both geometric design and visualization of volume data.

Acknowledgments

I would like to express my sincere thanks to my advisor Dr. Andrei V. Smirnov for his encouragement and unfailing support. His ideas and advices have been most valuable to me during these years. I also gratefully acknowledge the other members of my committee, Dr. Ismail Celik, Dr. Victor H. Mucino,Dr. Ibrahim Yavuz, and Dr. Frances L.Van Scoy, for their valuable suggestions and insights. I would like to thank Dr. Nilay Mukherjee, who served on my committee and had to leave because of his career change.

My appreciation also goes to my officemates Egemen Ogretim, Andrew C. Burt,Jun Li,Gusheng Hu,Cem Ersahin,Suryanarayana Raju Pakalapati and Christian E. Shaffer, whose friendship made my study here a great experience.

My deepest gratitude goes to my wife Lihua, for her inspiration and support. I thank our lovely son Gary for providing us so many laughs and fun times. He has made my life happy and fulfilling. I am also grateful to our parents for all their support.

This work was jointly sponsored by the U.S. Department of Energy, National Energy Technology Laboratory and West Virginia University.

Contents

List of Figures

Chapter 1

Introduction

Mesh generation is an important part of modeling physical processes and engineering structures by means of computer simulations. Meshes are used to create discrete representations of physical objects and environments, which are governed by set of laws expressed by partial differential equations (PDE). Mesh generation is a process of discretizing a physical domain into a set of smaller elements. In the simplest case these elements are represented by triangles or quadrilaterals for a two-dimensional domain and tetrahedra or hexahedra for a three-dimensional domain.

Mesh generation is a necessary pre-processing step for many numerical methods, such as boundary element methods (BEM), finite element methods (FEM) and finite volume methods (FVM), which are often used for numerical solution of PDEs. Due to the difficulty of solving differential equations for scientific or engineering problems analytically, especially when dealing with complex geometries, numerical methods are usually used to analyze and simulate physical phenomena such as heat transfer, fluid flow etc. By forming approximations over simple elements in a mesh, a system of linear equations is constructed and solved numerically by matrix inversion or in a iterative way.

A mesh can be structured or unstructured. A structured mesh generator attempts to align elements with boundaries of a geometric domain. And typically a structured mesh is all hexahedral, with interior nodes having an equal number of adjacent elements. On the other hand, an unstructured mesh allows any number of neighbors for a given element. Elements in an unstructured mesh can be triangular or hexahedral. While structured meshes are easier to implement and more efficient on simple geometries, unstructured meshes are more suitable for complex geometries, and convenient in mesh adaptivity. Unstructured mesh generation is the most adopted approach for complex physical simulations. Generally, it is more difficult to develop automatic algorithms for quadrilateral and hexahedral mesh generation. And thus, by far the most common form of unstructured mesh generation is triangle and tetrahedral meshing.

Although mesh generation has found applications in many areas, such as numerical analysis and simulation, computer graphics and visualization etc, the focus of this study is to find methods of generating good quality meshes for numerical methods. In this thesis we will exclusively consider unstructured triangular mesh generation, which triangulates a surface into triangles or a volume into tetrahedra.

The geometry of the domain has to be defined before it can be meshed. Some common methods used for describing geometries include constructive solid geometry (CSG), boundary representation (B-Rep), and piecewise linear complex (PLC). How a geometric entity is defined affects directly the meshing of this object. Most of geometries in practical scientific and engineering problems are modeled in CAD systems, and are typically based on points, curves and surfaces. And thus, most mesh generation methods are developed to deal with such.

The advancement in hardware, especially cheaper and larger memories, has brought attention to the technique of so-called volume graphics [32], which employs a volume buffer of voxels to represent 3D scenes. A voxel in volume graphics is the 3D analogy of a 2D pixel in raster graphics. It is the smallest distinguishable box-shaped unit of volume supplied with numerical values representing some properties of a real object or phenomenon. A collection of voxels forms a volumetric dataset. A particular voxel can be identified by the coordinates of its corners, or its center.

Traditionally volume graphics are obtained from measurements or simulations, such as biomedical images of computed tomography (CT) and magnetic resonance imaging (MRI), seismic measurements in geophysical explorations, and CFD simulation of fluid flow etc. Volume graphics is increasingly often applied to pure geometric modeling. The representation of geometric objects based on voxels has many advantages such as the ability to represent interiors and digital samples, easy implementation of boolean and block operations and the capability to model arbitrary complex geometries. And consequently the voxel-based modeling approach started gaining popularity [25, 71, 70, 4, 69, 31].

Although many meshing methods exist, few of them are designed to deal with voxel-based geometric objects. There has been very little work done on mesh generation directly from a voxel dataset. Most mesh generation algorithms need a surface representation of the geometry, which is not explicitly defined for voxel-based geometric objects.

The main objective in this study is to develop methodologies which can generate good quality meshes for voxel-based objects. To resolve the geometry with correct topology, the mesh also needs to be conformal to the boundaries with edges and sharp corners preserved.

In this study several findings are reported on new algorithms for mesh generation, which simplify the mesh generation procedure and at the same time improve the quality of meshes for both voxel based objects and general meshing of 2D and 3D domains. These findings are summarized in Chapter 6.

1.1 Mesh Quality Measures

Mesh quality is important for the solution of a numerical simulation. In order to evaluate mesh quality, many measures have been used, such as angle, skew, length ratio, and orientation etc. Knupp [35] uses an algebraic framework to form a mathematical theory of mesh quality metrics, which is based on the Jacobian and related matrices. The Jacobian matrix can be factored into geometrically meaningful parts to construct different quality measures. Another work by Shewchuk [55] explains the mathematical connections between mesh geometry, interpolation errors, and stiffness matrix conditioning for triangular mesh elements. Also several quality measures for evaluating triangular elements are presented.

In this study, the meshes generated will be isotropic triangular and tetrahedral meshes. Some simple quality measures are enough to judge the mesh quality. For triangles, we use aspect ratio, maximum and minimum angles as the quality measures. A triangle's aspect ratio is defined as the longest edge divided by the smallest height. For triangles, the smaller the aspect ratio, the better the quality. The best triangle is a equilateral triangle with an aspect ratio of $2/\sqrt{3}$, or 1.1547. Concerning angles, one good triangle mesh generator should try to maximize the minimal angle and minimize the maximum triangle. Generally speaking, for triangles in a good quality isotropic mesh, the minimum angle is usually greater than 20^o, the aspect ratio is usually less than 5.

For tetrahedra, we use radius-edge ratio and aspect ratio as the quality measures. A tetrahedron's radius-edge ratio is the radius of its circumsphere divided by its shortest edge length, and a tetrahedron's aspect ratio is its longest edge length divided by the diameter of its inscribed sphere. For tetrahedra, a small radius-edge ratio means a good tetrahedron except for a sliver, which is a tetrahedron whose vertices are almost coplanar and whose circumradius is not much larger than its shortest edge length. A sliver has large aspect ratio.

1.2 Previous Work on Triangular Mesh Generation

Most current triangular meshing techniques can be classified into three main categories: Delaunay-Voronoi based methods, advancing front methods, and quadtree / octree based methods [65]. First we are going to give a brief introduction to these three techniques, and then review the previous work on mesh generation from voxel dataset. For more extensive reviews we refer to the papers by Bern and Eppstein [6], Bern and Plassmann [7], Teng and Wong [64], and Owen [46].

1.2.1 Delaunay-Voronoi Based Meshing Methods

Given a set of points on the plane: $P = \{p_1, p_2, ..., p_n\}$, the Voronoi diagram of P is a partition of the plane into n polygonal regions, one for each point in P, with the property that every point of that region is closer to p than to any of the other points

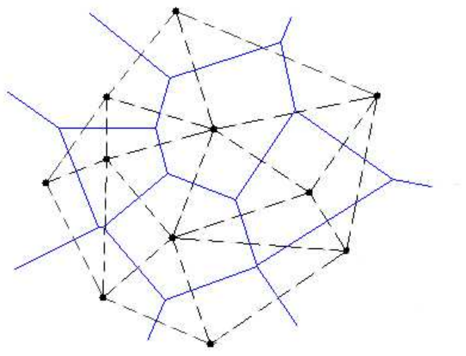


Figure 1.1: A Voronoi Diagram (solid line) and its dual - Delaunay Triangulation (dash line).

from P. The Delaunay triangulation is a dual diagram connecting any two points whose Voronoi regions intersect along a common line segment. An interesting property of the Delaunay triangulation is that no point in P will be inside the circumcircle of any triangle of the Delaunay set.These concepts generalize to higher dimensions.

The most popular meshing methods of triangular mesh generation are those utilizing the Delaunay criterion, which has the so-called *empty circle* (2D) or *empty sphere* (3D) property, that is, the circumcircle of any triangle or circumsphere of any tetrahedron does not have any mesh nodes inside.

The empty circle or sphere property of Delaunay triangulation leads to several other favorable characteristics. For example, in any dimension, the Delaunay triangulation minimizes the largest minimum-containment sphere, where the minimum-containment sphere of a simplex is the smallest sphere containing the simplex. In two dimensions, the Delaunay triangulation minimizes the largest circumcircle and moreover it maximizes the minimum angle in the mesh. And thus for a set of points in 2D, the Delaunay triangulation will generate the best quality mesh. Unfortunately the property of maximizing the minimum angle is lost in 3D and higher dimensions, and poor quality elements such as slivers can be formed by Delaunay triangulation.

Baker [5], George [26], and Weatherill [72] are among the first researchers utilizing the Delaunay criterion to develop meshing algorithms. The Delaunay criterion in itself is not a meshing algorithm. But it provides a criterion for how to connect points (or nodes) into a mesh. Any Delaunay meshing method is composed of two phases: placement of the mesh nodes and triangulation of the nodes. If the mesh nodes are well placed, the triangulation phase can be simple.

Constrained Delaunay Triangulation

Delaunay triangulation has a sound mathematical basis and desired optimization properties. However Delaunay triangulation is done over a set of points and the hull of the triangulation will always be convex. While the domains to be meshed are not usually convex, the Delaunay triangulation has to be forced to respect the boundaries of the domains. This new forced triangulation is called *Constrained Delaunay Triangulation.* The constrained Delaunay triangulation may not be strictly "Delaunay".

A free 2D constrained Delaunay triangulation program named *Triangle* was developed by Jonathon Shewchuk [53, 54]. It is based on a hybrid of Ruppert's Delaunay refinement algorithm [50] and Chew's Delaunay refinement algorithm [15]. The program is available at http://www-2.cs.cmu.edu/ quake/triangle.html. A 3D constrained Delaunay triangulation program - *Tetgen* developed by Si is available at http://tetgen.berlios.de. It implements the algorithms of Edelsbrunner and Shah [22], and Si and Gärtner [57].

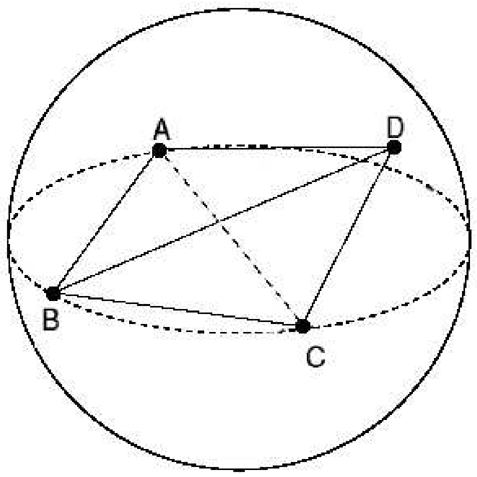


Figure 1.2: A sliver has small radius-edge ratio with almost coplanar vertices (A, B, C, D).

Sliver Elimination

The sliver is the only type of small volume tetrahedron whose radius-edge ratio does not grow with decreasing volume [23]. Figure 1.2 shows a sliver.

Well-shaped meshes can be generated by many Delaunay meshing methods in two dimensions. However in three dimensions, mesh generation becomes more difficult. Slivers may exist in Delaunay tetrahedralizations even when the node set is well-spaced [63]. And thus it is necessary to remove them in order to generate well-shaped meshes.

Eliminating slivers from 3D meshes is one of the difficult tasks in mesh generation. Some algorithms have been proposed for sliver elimination. Chew [16] proposed Delaunay refinement algorithm that inserts random points near the circumcenters instead of points at the circumcenter. This method can avoid generating new slivers but cannot guarentee to remove original slivers in the mesh. Cheng et al. [14] proposed weighted Delaunay triangulation and proved that no slivers will be generated in the interior of the domain. However slivers may present near the boundary of the domain. Motivated by the work of Chew [16] and Cheng et al. [14], Li [39] proposed two algorithms in his thesis: one is smoothing and cleaning-up based on moving mesh vertices; and the other one is based on adding Steiner points. While the former one has difficulties in treating boundaries, the later one generates non-uniform meshes and eliminates all original slivers without introducing new slivers in the final mesh. The tetrahedra near domain boundary are also well-shaped guranteed by a complete boundary treatment.

1.2.2 Advancing Front Meshing Methods

Advancing front meshing is another very popular family of triangular mesh generation techniques. In this approach, triangles or tetrahedra are created progressively inward from boundaries. An active front is maintained while creating new elements. Each time, an edge (for 2D) or triangle (for 3D) on the front is selected to form the next new triangle or tetrahedron with a newly generated node or an existing node. As the algorithm progresses, new edges/triangles are added to the front while closed old edges/triangles are removed from the front. Eventually the advancing front will become empty and the domain is filled with mesh elements.

The main contributors to the advancing front meshing techniques include Lo [42, 43], Peraire [47], and Löhner [44] etc.

1.2.3 Quadtree/Octree Based Meshing methods

Quadtree/Octree based meshing is less used comparing to the above mentioned two techniques. Some proposed quadtree/octree based mesh generation methods are [73, 3, 52, 33]. Generally speaking, in all such methods, a square or cube is first generated to cover the 2D or 3D domain. And then it is recursively subdivided into cells (quadrants or octants) of desired resolution. The corners of the inner cells are used as nodes in the mesh. For cells containing part of the domain boundaries, special consideration is used to deal with the boundary. Mesh size grading is controlled by varying the subdivision level of cells. In order to avoid dramatical change of element size, the maximum difference between the levels of adjacent cells is typically limited to one.

1.3 Previous Work on Meshing Voxel Dataset

1.3.1 Surface Meshing

Although many surface meshing methods exist today, few of them are designed to deal with voxel-based geometric objects.

The problem with surface meshing of a voxel-based object is that the surface of such an object is not explicitly defined. It is usually desirable to reconstruct the surface in the form of piecewise linear mesh from the voxel data.

Many algorithms were developed to extract iso-surfaces from volume data for the purpose of modeling and rendering, such as the marching cube algorithm [45] and its variants. The marching cube family of algorithms produce a uniform surface mesh with excessive triangles. These triangles could be in bad shapes and thus are not suitable for the purpose of numerical analysis. Also sharp features in the objects are not preserved. An extended marching cube algorithm [36] can detect and reconstruct sharp features by using an enhanced distance field representation and performing feature detection and sampling. An octree-based dual contouring algorithm [30] works by contouring a signed grid whose edges are tagged by hermite data (i.e exact intersection points and normals). The algorithm can generate adaptive iso-surfaces with good aspect ratios and preserve surface features, like sharp corners, without explicitly identifying and processing these features in the object. However exact intersection points and normals for edges are needed by this method.

1.3.2 Volume Meshing

There has been some work done on volume meshing from voxel dataset. The method proposed by Frey et al. [24] generates uniformly dense mesh by splitting every voxel into 5 or 6 tetrahedra. Hartmann and Kruggel [28] present an algorithm that allows fast and stable creation of very large 3D meshes from Magnetic Resonance Tomograms. Their algorithm is based on the idea of an image-based spatial decomposition of the problem domain yielding smaller subproblems that can be efficiently handled. Hale [27] meshes a 2D/3D image by combining the digital image with a lattice of atomic points whose coordinates are optimized to minimize a potential energy function of the combination. The mesh connected with the atoms tends to be aligned with image features, but is not guaranteed to be conformal to the domain boundary. Udeshi [68] generates a graded and adaptive tetrahedral mesh from voxel data. Mesh nodes are created from an octree, and connected into a tetrahedral mesh by constrained Delaunay tetrahedralization. Zhang and Bajaj et al. [79, 80] extract adaptive tetrahedral and hexahedral meshes directly from volumetric imaging data. The octree-based meshing approach is used by the authors.

1.4 Outline of the Thesis

In this thesis, a voxel-based geometric modeling system is built along with a GUI (graphical user interface), which was used in multi-physics simulations of fuel cells. The voxel-based objects used in our mesh generation studies are all modeled with this modeling system. In Chapter 2 we start to focus on mesh generation, in particular mesh generation for voxel-based objects. In this chapter we present an advancing front approach for surface meshing on voxel-based objects [58]. In chapter 3, we begin to develop a new mesh generation technique using particle simulation and Monte-Carlo methods [78, 76, 75, 74]. This approach uses energy minimization via Monte-Carlo simulation to optimize mesh node distribution, from which well-shaped meshes can be constructed. And in Chapter 4, the problem of surface meshing on voxel-based objects is revisited with this approach, which results in better mesh quality [77]. Chapter 5 describes the GUI and the voxel-based modeling system [60, 62, 61, 59]. The thesis is concluded with recommendation for future work.

Chapter 2

Surface Mesh Generation for Voxel-Based Objects with an Advancing Front Approach

This chapter starts to address the problem of constructing surface meshes for the voxel-based objects. A heuristic algorithm using an advancing front approach for generating a triangulated surface mesh is proposed. The resultant meshes do not inherit the anisotropy of the underlying hexagonal grid. Meshing strategies, such as surface curvature estimation and neighbor-detection techniques are described. Examples of applications are given.

2.1 Introduction

Mesh generation is a process of dividing a physical domain into a large number of small elements with relatively simple shapes. In general the surface of an object needs to be meshed before the application of a 3D mesh generator, which often fills in the interior of the object with tetrahedral or hexahedral elements. The most common unstructured surface meshing algorithms, which divide a surface into triangles, include Delaunay meshing methods [5], and advancing front methods [37].

While vector graphics dominate most CAD packages, 3D surfaces in many engineering and scientific applications are usually defined by means of parametric surfaces. Thus most surface meshing algorithms are designed to deal with such. There are essentially two types of surface meshing approaches: direct and indirect. With the indirect approach, the mesh is first generated in a 2D parametric space, and then mapped back to the actual surface in 3D space [12, 19, 66, 8]. In contrast, the direct approach generates the mesh directly over the original surface in 3D space [37, 21]. With this approach surface parameteriztion is not required.

Although many surface meshing methods exist, few of them are designed to deal with voxel-based geometric objects. In this study, a 3D surface meshing method using an advancing front approach is proposed to mesh the surface of voxel-based objects. Since no parametric surfaces exist for an voxel-based object, the algorithm proceeds directly with voxels, and in the end the surface is extracted as a triangulated mesh. Therefore, besides the purpose of generating quality mesh for numerical methods, the meshing method also serves another purpose: volume rendering, which is the objective of many surface construction algorithms, such as marching cubes algorithm [45].

2.2 Method

The main targets of the meshing method are voxel objects, which are created by a voxel-based modeling system, such as the one designed in this study (see Chapter

5) [62]. Voxels are arranged on a regular 3D grid. Each voxel has a type, which represents any property, such as a type of material, void space, etc. In a broader paradigm of physical modeling multiple physical properties can be further associated with each voxel type. A connected set of voxels of the same type represents a physical object. In the simplest case where there are only objects of one type, a voxel can have only two values: 0 and 1; with 1 representing a space occupied by an object and 0 representing the void space.

For each voxel, its neighbors can be reached by traversing the 3D grid. For a voxel at the boundary, at least one of its neighbor voxels is outside of the object, i.e. has a different type. Whether a voxel is at the boundary or not can be determined using this neighbor information. Since voxels are arranged in a simple hexagonal order voxel neighbors can be accessed by simply incrementing or decrementing any of its three spatial coordinate indexes.

Unlike most isosurface extraction methods, which associate a set of faces with each voxel, we associate a single point with each voxel, located in the center of the voxel. The same approach is used in [2]. Figure 2.1 shows a voxel-based object rendered as points.

Since the boundary point set is usually very large, it is not practical to include all of them in the final mesh. Instead only a small set of boundary points will be enough to create a feature preversing surface mesh. To generate a surface mesh, we pick up points from surface boundary of the object as mesh nodes, that is any mesh node will be coincident with one of the boundary points.

Each object has an id-number, which is assigned to every voxel representing that object. The meshing routine receives the id-number of the object as an input param-

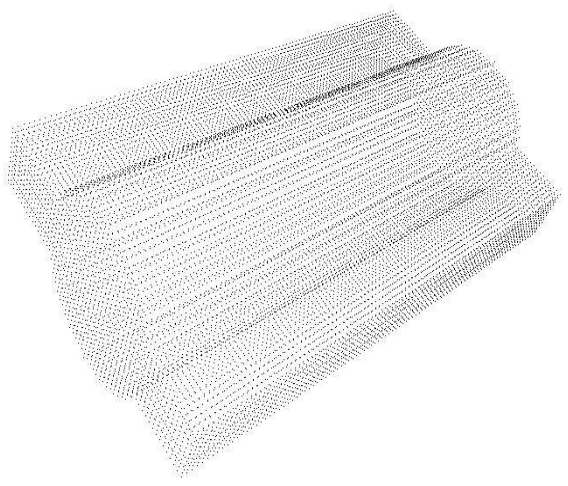


Figure 2.1: Voxels are treated as points.

eter and starts by scanning the voxel set for the first occurrence of this id. If such voxel was found the routine invokes the surface meshing algorithm, which creates the boundary mesh for that object. The routine continues then to search for voxels of the same type.

In this study, a *surface mesh* is represented as a linked list of triangles, where each triangle is represented by three vertexes, each of which belongs to a set of *boundary nodes*. A *boundary node* inherits the coordinates of the corresponding boundary voxel. The three vertexes of each triangle are oriented in a way such that the normal of the triangle area points outward the object volume.

When the mesh is being constructed the algorithm operates on nodes and edges, which can be of 2 types: *open* or *closed*. An open edge is an edge that belongs to only one surface triangle. An *open node* is a node, which is not completely surrounded by triangles. *Open edges* and *open nodes* are simply edges or nodes which are at the front of the newly constructed surface mesh, that is, the part which is currently under construction (Fig.2.2). We call the front-edges *open* because each of them belongs to only one triangle, whereas every edge inside the mesh belongs to two triangles. Likewise, every node at the front (*open node*) belongs to at least two open edges, whereas each node inside the mesh doesn't belong to any open edges. When the mesh is completely done all the nodes and edges should become closed and the front reduces to a zero-set. The mesh front is essentially a one-dimensional segmented line. Each open node has two pointers pointing to 2 neighboring open nodes, designated as *prev* and *next*, The direction from the *prev* node to the *next* node following the unmeshed area is in counter-clock wise direction with respect to the surface outward normal vector. The angle between node's two open edges on the unmeshed area side is called the *open angle*. Figure 2.2 shows an active node A, its *prev* node B and *next* node C, and its open angle. A closed node is also shown in this figure.

One step of the algorithm consists of closing a single node. For this purpose the algorithm selects a current node from the set of open nodes, which is called *this* node, and performs the *closure operation*. The selection of active nodes can be done in a preferential manner so as to provide for a more uniform angle distribution inside the surface triangles.

The very first node of the mesh selected on the boundary has its *prev* and *next* pointers point to itself, and its open angle is set to 2π[1]. When a node is closed, its open angle is 0. The set of open nodes is stored as a linked list. Each closure operation of the node may lead to the appearance of other open nodes, which are newly introduced nodes of the mesh. The meshing algorithm repeats the closure

[1]Strictly speaking, it can be less than 2π depending on the curvature of the surface at that point.

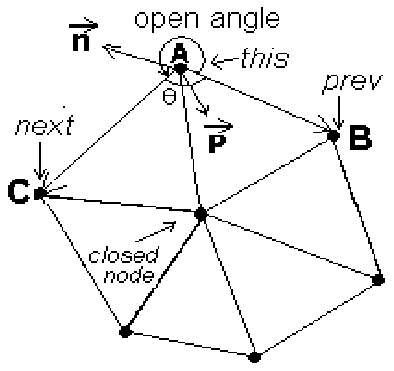


Figure 2.2: Node *this*,its *prev*, *next* nodes and open angle.

operation on all open nodes until the list becomes empty.

The meshing algorithm can be summarized as the following:

1. Construct the first triangle, and add its three nodes to the linked list;

2. Select the node from the list with the smallest open angle. Call it *this* node;

3. Recursively create surrounding triangles until *this* node is closed; In this procedure use either existing open nodes for new triangle vertexes or introduce new open nodes;

4. When *this* node is closed, remove it from the list, and consider if its neighbor nodes should be removed as well.

5. If there are open nodes in the list repeat from step 2.

The essential procedures performed by the algorithm are the following:

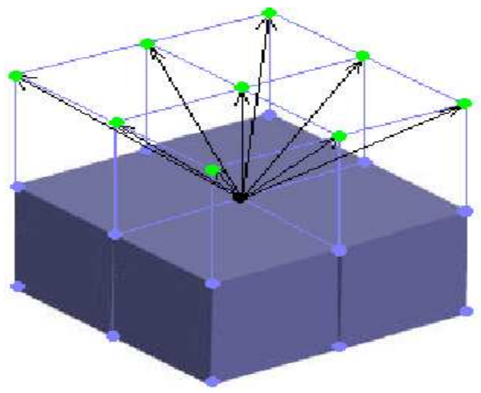


Figure 2.3: Estimating surface normal of a boundary point

2.2.1 Construction of a Surface Normal

Surface normal vector is needed to determine the node-closure direction. Given a rigid character of the surface in a voxel-based representation when looked at in the vicinity of only few grid-nodes, one needs to perform some averaging to get a fair estimate of the surface normal direction at each voxel. A quick way to do this estimate is to account all the neighboring voxels, which don't belong to the current object (outside voxels). For each such voxel the distance vector, $\vec{P}_i$, between *this* node and the neighbor node, i, is calculated. The sum of these vectors will approximately point in the surface normal direction. Thus, the normal vector $\vec{n}$ is estimated using the following equation (see also Figure 2.3):

$$\vec{n} = \frac{\sum_{i=1}^{k} \vec{P}_i}{|\sum_{i=1}^{k} \vec{P}_i|} \tag{2.1}$$

2.2.2 Open Angle Estimation

The open angle estimation procedure is illustrated in Figure 2.2. First the dot product of the two vectors $\vec{AB}$ and $\vec{AC}$ is computed to find the angle θ between them. Then the actual open angle could be this angle or its complimentary of 2π. To decide on this we need extra information, which comes from the normal vector $\vec{n}$ and the cross product $\vec{P}$ of vectors $\vec{AB}$ and $\vec{AC}$. If $\vec{n}$ and $\vec{P}$ are on the same side of the surface, i.e. their scalar product is positive, then the open angle is θ, otherwise the open angle is $2\pi - \theta$. Thus, we have:

$$\theta = \arccos\left(\frac{\vec{AB} \cdot \vec{AC}}{|\vec{AB}||\vec{AC}|}\right) \tag{2.2}$$

$$\vec{P} = \vec{AB} \times \vec{AC} \tag{2.3}$$

$$if(\vec{n} \cdot \vec{P} < 0) \Rightarrow \theta = 2\pi - \theta \tag{2.4}$$

2.2.3 Construction of a Triangle

To construct a new triangle for an open node, the following procedure is used (Fig. 2.4). We consider that at this point an open node was selected as *this* node:

1. Check nearby open nodes to *this* node, and see if there is one suitable for constructing a triangle, consisting of that node, *this* node and *prev* node, or consisting that node, *next* node and *this* node. A suitable node is a node which will lead to a triangle with good quality, e.g, from the respective of aspect ratio. If such node is found:

    - Then construct such triangle and call the newly found node the *old* node.

Check if *this*, *old* node, and *prev* (or *next*) nodes are now closed. Remove any closed node from the open node list.

2. Otherwise:

    - create a *new* open node by searching an appropriate boundary voxel, such that the distance to that voxel is consistent with the current mesh length-scale, and the angle between *this* $\rightarrow$ *prev* and *this* $\rightarrow$ *new* directions is around 60^o (see below).

3. Change *prev* and/or *next* pointers of any involved node.

As can be seen this algorithm takes care of the *meeting front* problem, when the open nodes from different parts of the front begin to converge on each other. That's why the existing nodes are tried first before creating *new* nodes in step 1 instead of generating new nodes to form a triangle.

2.2.4 Generating a New Node

A new node is to be generated from a boundary voxel, inheriting its coordinates. Two parameters are used in searching for a suitable boundary voxel. One is a control angle. We try to make it close to 60 degrees by dividing the open angle by an integer:

$$control\_angle = \frac{open\_angle}{round(open\_angle/60.0)} \tag{2.5}$$

Another parameter is a control distance, which is related to local mesh size control (see below).

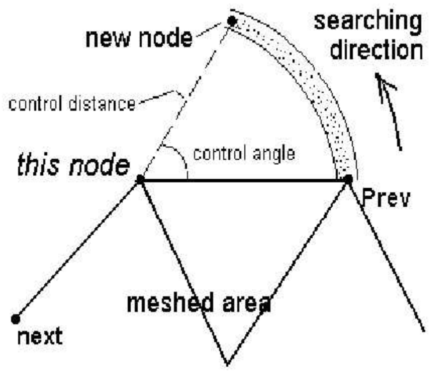


Figure 2.4: Constructing a new triangle

Figure 2.4 illustrates the searching process. Starting from the *prev* node, the searching process walks through the boundary voxels from neighbor to neighbor in counter clock wise direction keeping the distance from *this* node in a narrow strip, until a point with the right angle is found.

2.2.5 Mesh Size Control

The mesh size is selected on the basis of two control sizes: one is a global size, set by the user and serving as an upper bound of mesh-size; and another is local size, related to the local surface curvature. Highly curved surfaces should use triangles of smaller sizes. Therefore, to achieve the goal of mesh size control, surface curvature evaluation is necessary.

Least Square Quadric Surface Fitting

To find the local curvature one can first fit a surface patch with a set of local points around the point of interest, and then estimate curvature on the fitted surface. Many different surfaces can be used for this purpose. In the current implementation a quadric surface patch fit with least square method is used, as described as follows.

One approach to fit a set of 3D points is least-squares surface fitting with a quadric function:

$$f(x, y, z) = CV^T \tag{2.6}$$

where C is a vector of coefficients:

$$C = (c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9) \tag{2.7}$$

and

$$V = (x^2, y^2, z^2, xy, yz, xz, x, y, z, 1) \tag{2.8}$$

We need at least 10 points to find the ten unknown coefficients. Suppose the number of points N is greater than 10 points, generally they cannot be on the same quadric surface. To fit the best surface, we can attempt to choose vector C to minimize the error function:

$$E(C) = CMC^T \tag{2.9}$$

$$M = D^T D \tag{2.10}$$

where D is a $N \times 10$ matrix whose ith row is:

$$(x_i^2, y_i^2, z_i^2, x_i y_i, y_i z_i, x_i z_i, x_i, y_i, z_i, 1), i = 1, 2...N \tag{2.11}$$

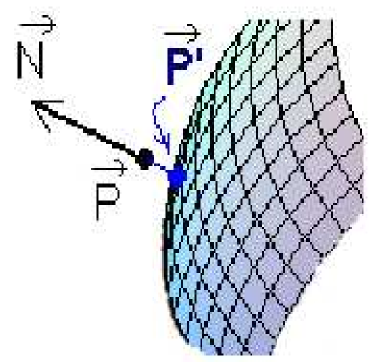


Figure 2.5: Locating the closest point.

When all coefficients are 0 the error function has the minimum value zero. Therefore there is a degree of freedom in this function. To get rid of it, we can choose C to be a unit-length vector as a constraint. Now this problem is an eigenvalue problem. The minimum error will be the smallest eigenvalue of M. And the eigenvector corresponding to the smallest eigenvalue is the coefficient vector C.

In the above construction, none of the sample points is guaranteed to be on the fitted surface.

Curvature evaluation

After the quadratic surface fit was found, the closest point on that surface patch from the point of interest is identified (since the fitted patch does not have to pass through that point) and the curvature at that point is used to approximate the curvature at the point of interest. The closest point is assumed to lie on the intersection between the boundary normal at the point of interest computed earlier and the fitted surface (Fig.2.5).

If $\vec{P}$ is the coordinate vector to the point of interest and $\vec{N}$ is the boundary normal vector at that point then the intersection point $\vec{P'}$ can be represented as: $\vec{P'} = \vec{P} + k\vec{N}$, where k can be obtained by substituting the coordinates of $\vec{P'}$ into the fitted surface equation. This will give a quadratic equation, which is easy to solve.

The curvature evaluation method used for 3D implicit quadric surface is shown below.

Let $F(x, y, z) = ax^2 + by^2 + cz^2 + exy + fyz + gxz + lx + my + nz + d = 0$ represent an implicit quadric surface in R3. F_x, F_y and F_z are the first order partial derivatives of F. F_{xx}, F_{xy}, ..., F_{zz} are the second order partial derivatives of F. The following equations are used to calculate the curvature of surface F(x, y, z) [20]:

$$|\Delta F| = \sqrt{F_x^2 + F_y^2 + F_z^2} \tag{2.12}$$

$$L = \frac{1}{F_z^2|\Delta F|} \begin{vmatrix} F_{xx} & F_{xz} & F_x \\ F_{zx} & F_{zz} & F_z \\ F_x & F_z & 0 \end{vmatrix} \tag{2.13}$$

$$M = \frac{1}{F_z^2|\Delta F|} \begin{vmatrix} F_{xy} & F_{yz} & F_y \\ F_{zx} & F_{zz} & F_z \\ F_x & F_z & 0 \end{vmatrix} \tag{2.14}$$

$$N = \frac{1}{F_z^2|\Delta F|} \begin{vmatrix} F_{yy} & F_{yz} & F_y \\ F_{zy} & F_{zz} & F_z \\ F_y & F_z & 0 \end{vmatrix} \tag{2.15}$$

$$E = 1 + \frac{F_x^2}{F_z^2} \tag{2.16}$$

$$H = \frac{F_x F_y}{F_z^2} \tag{2.17}$$

$$G = 1 + \frac{F_y^2}{F_z^2} \tag{2.18}$$

$$A = \begin{bmatrix} L & M \\ M & N \end{bmatrix} \tag{2.19}$$

$$B = \begin{bmatrix} E & H \\ H & G \end{bmatrix} \tag{2.20}$$

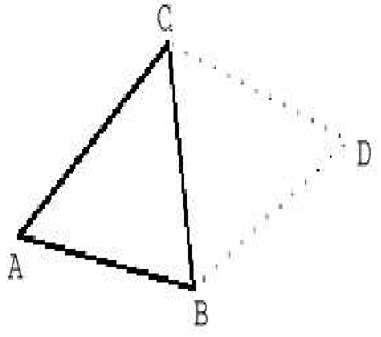
The eigenvalues of $B^{-1}A$ are the principle curvatures $k1$ and $k2$. The values of the Gaussian curvature K and the mean curvature H are then given by:

$$K = k1 \cdot k2 \tag{2.21}$$

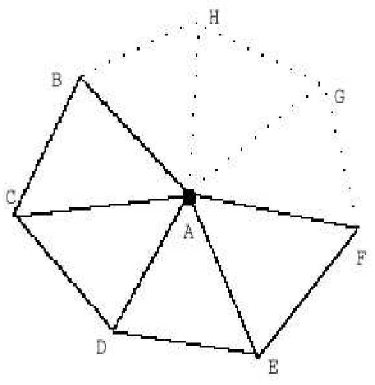
$$H = \frac{1}{2}(k1 + k2) \tag{2.22}$$

2.3 Discussion

The surface meshing method described here uses an advancing front technique, which is similar to traditional advancing front methods [37]. However in our method the front does not start from the boundary of a surface. Instead it can start from a random point. Mesh quality might be improved if the front starts from corners and edges of the object volume, but in that case corner detection and edge detection need to be addressed before the meshing process. Another difference in the proposed method is that it relies on the node closure routine introduced in the previous section. This routine constitutes a single step of the algorithm, which is repeated until no open nodes are left in the mesh. A conventional approach to front propagating uses *edge*



(a) Edge closure: new triangle CBD is to be constructed to close edge CB



(b) Node closure: new triangles AFG, AGH, and AHB are to be constructed to close node A

Figure 2.6: Edge vs node closure approaches to front propagation.

closure rather than node closure, in which for each open edge a new mesh triangle is constructed with this edge making one side of the triangle (Fig.2.6).

Comparing the two approaches one may note that the node closure algorithm is more complex in implementation than the edge closure one. However, the former has a potential for a better control of the angles, which may lead to a higher uniformity of the mesh. This is because the open angles for each node at the front are constantly computed and sub-divided into the appropriate number of sub-angles. This means that the node-closure algorithm will require less post-processing steps of mesh quality improvements.

The node-closure algorithm has also a wider range of *visibility* when it looks out for other front nodes as potential candidates to be joined with a current node. This is because the lookout radius for each node is about the size of the local edge-length of the mesh, whereas the lookout radius for the edge in the edge-closure case is usually smaller (Fig.2.6). It is still possible to employ the same strategy in the edge-closure case of using a larger lookout radius for both nodes of a given edge. However, this will lead to a considerable increase of the processing time, since each node will be processed as many times as there are edges connected to it, which may be on the order of 4 to 6 times more for surface meshes.

Even though each step of the node-closure algorithm takes more operations to accomplish, the overall efficiency may be comparable to the edge closure procedure, since several new mesh triangles can be generated during one step of the former, while the latter only leads to the creation of one new triangle at a time. This is a consequence of the fact that there are usually several times more edges than nodes in a typical mesh.

The advantage of using node-cloure operation as compared to an edge-closure operation used in a conventional advancing front method is that on the average it spans a larger area while examining a neighborhood of the node as compared to the edge. Indeed, to close an edge one needs to create a single node that will form a triangle with other two nodes on the edge. To close a node one needs to create more than one new node, therefore there is a wider range of possibilities and in fact an optimization can be used to select the new nodes. There is also a larger area examined in the process of selection of several nodes compared to just one. This all amounts to an improved distribution of nodes with respect to the selected mesh optimality criteria (aspect ratios, angles, etc.)

Speaking about the accuracy of surface representations, the mesh generated in the present approach provides the resolution of surface details down to 4 to 6 voxels in size. This is because the node closure operation requires a certain minimum separation of

$Exp.$	$Grid$	N	α_{max}	α_{min}	r_{max}
1	$70 \times 70 \times 70$	1158	114^o	24^o	3.35
2	$100 \times 100 \times 60$	1342	112^o	25^o	3.21
3	$40 \times 60 \times 100$	1880	113^o	20^o	3.08

Table 2.1: Characteristics of example meshes.

the open nodes to provide a more even angle distribution and to avoid inheriting the hexagonal unisotropy of the voxel grid. Although this property sets a limit on the resolution of surface details, such as sharp angles, it provides for the grid independence of the mesh.

2.4 Result

Figures 2.7, 2.8 and 2.9 show surface meshing examples of spheres, joint spheres and other composite objects. The voxel grid size, the number of tiangles, the maximum/minimum angle, and the maximum aspect ratio r are shown in Table 2.1. The global mesh length-scale was chosen to be 6.

The quality of triangles are affected by the resolution of the object and the selected global length scale. Since all coordinates of voxels are integer values, during the procedure of searching a new node, we cannot make the angle exactly equal to the control angle. The smaller the control distance (or the global length scale), the larger the error from the control angle. Therefore if we have high resolution voxel grid for the object, we can then choose larger length scale, and thus make the mesh quality better.

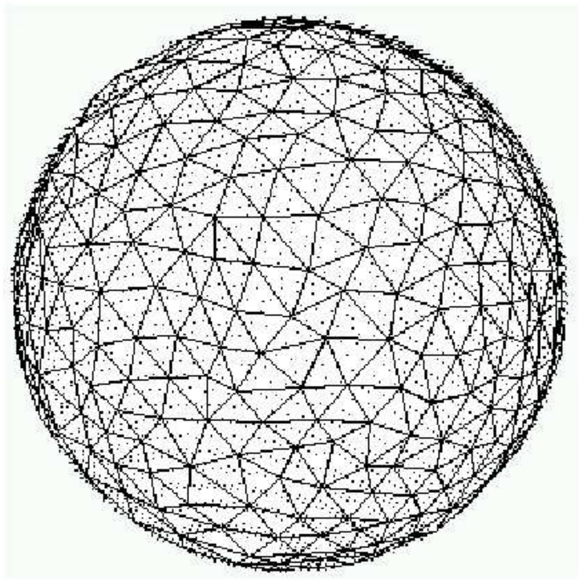


Figure 2.7: Example 1: Surface mesh on a sphere

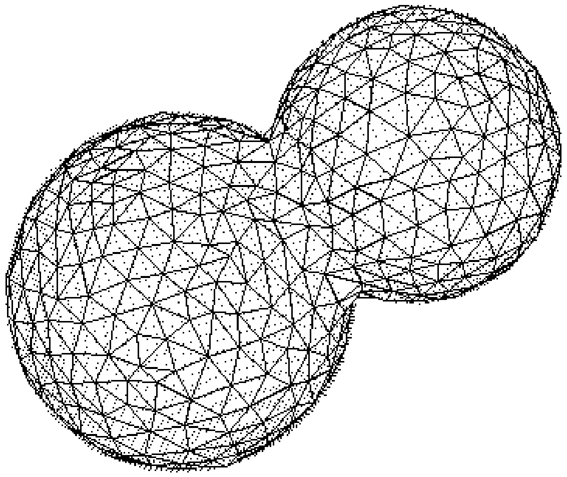


Figure 2.8: Example 2: Surface mesh on two overlaying spheres

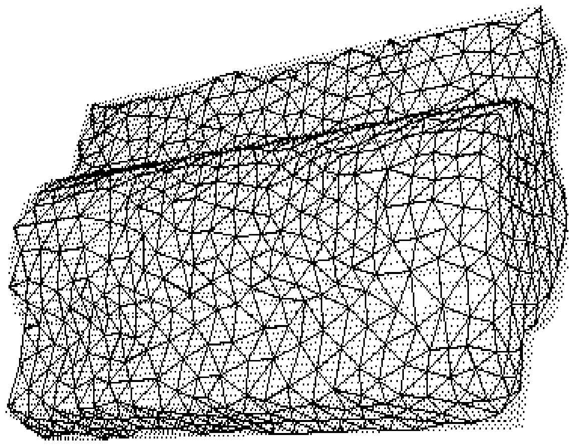


Figure 2.9: Example 3: Surface mesh on overlaying cylinder and cube

2.5 Conclusion

The proposed mesh generation scheme belongs to the family of advancing front methods, but exploits the idea of using nodes and their triangle closure as basic steps in building the surface mesh. The extension of this idea to 3D meshing is relatively straightforward. In this case the node has to be closed by the envelop of tetrahedral elements filling in the spherical region around the node. The construction of these elements can be done using a 2D algorithm described here for meshing the surface of the sphere, with subsequent connections of each newly generated node to the sphere-central node.

The method suggested here works best on smooth surfaces of objects, since it has a minimum resolution limit of several voxel sizes. Thus the method may not preserve the sharp corners and edges. To correct this situation, corner detection and edge detection are needed. We will address these problems in the later chapters.

Chapter 3

Mesh Generation with Monte Carlo Simulation

In this chapter, a new approach of node placement for unstructured mesh generation is proposed. It is based on the Monte-Carlo method to place nodes for triangular or tetrahedral meshes. Surface or volume geometries to be meshed are treated as atomic systems, and mesh nodes are considered as interacting particles. By minimizing system potential energy with Monte Carlo simulation, particles are placed into a near-optimal configuration. Well-shaped triangles or tetrahedra can then be created after connecting the nodes by constrained Delaunay triangulation or tetrahedrization. The algorithm is capable of controlling mesh element sizes. The method works in an almost identical way for 2D and 3D meshing. The algorithm is simple and easy to be implemented. To the best of our knowledge this is the first application of Monte-Carlo method for mesh generation.

3.1 Introduction

Typically a mesh generation method consists of node placement and node connection sub-routines. Node placement, i.e. generating nodes at various locations inside the geometry is a key step in any meshing algorithm. Well distributed nodes will make node connection process easier and result in good meshes. Different strategies for node placement have been reported in various methods, and certainly each will have a different effect on the final mesh.

While some methods place nodes incrementally and connect a new node into the existing mesh right after it is generated, other meshing methods distribute all nodes throughout the domain first and then connect the nodes into mesh.

In Delaunay meshing nodes are typically inserted incrementally into the existing mesh. As each new node is inserted, the Delaunay criterion is maintained by reconnecting nodes. Weatherill and Hassan in their tetrahedral meshing method [72] propose to place new nodes at tetrahedrons' centroids. Chew [17] and Ruppert [49] propose to place new nodes at centers of element circumcircles or circumspheres. With this strategy, the quality of triangles is guranteed with a minimum bound on any angle in the mesh.

For advancing front triangular meshing methods, nodes are also inserted incrementally. Typical steps of these methods consist of selecting an edge or a triangle from the front to form a new triangle or tetrahedron by joining it with either an existing node on the front, or a newly created node. In the tetrahedral meshing scheme by Lo [43], a set of nodes are generated beforehand inside the volume on a series of parallel planes. When forming a tetrahedron, a node from this set is chosen based on some optimization criteria. In some other tetrahedral meshing methods [47, 29], new nodes are generated simultaneously when creating tetrahedral elements. Peraire et al. [47] choose to place a new node on a front triangle's normal vector which is passing through its centroid. The distance from the new node to the centroid is chosen such that the average length from the new node to the triangle's three vertices is unity in a normalized space. Jin and Tanner [29] suggest a similar place for a new node on the triangle's normal vector which is passing its centroid. Besides they also introduce many backup nodes which are located on 7 normal vectors. The authors try to select the best one from existing neighbor nodes and the new nodes when forming the next new tetrahedron.

In contrast to the approach of incrementally inserting nodes one by one into an existing mesh, some mesh generation methods place all the nodes first before connecting them into a mesh. Nodes are first distributed into optimal locations to cover the whole domain, and then connected into mesh. In the bubble meshing method [56], Shimada and Gossard place nodes at centers of a set of packed spheres or bubbles. An inter-bubble force function is defined between adjacent bubbles and a force balancing configuration of the nodes is found by performing dynamic simulation. After node placement, the method connect the nodes into well-shaped mesh elements by constrained Delaunay triangulation or tetrahedrization. In DistMesh [48], Persson and Strang treat a simplex mesh as a truss structure and mesh points as nodes of the truss. A linear force-displacement function is defined for each pair of nodes connected by a bar (or edge). Nodes are gradually moved to optimal locations by iteratively solving for force equilibrium. At each iteration, the node set is re-triangulated using Delaunay triangulation algorithm in order to decide the edges. In the end, a high quality triangular mesh is obtained.

In our study, as in the bubble meshing method, a methodology of particle simulation is used for node placement. Nodes are considered as interacting particles in a force field. However, instead of solving for equilibrium by a deterministic approach, we use statistical sampling based Monte Carlo method, to minimize system potential energy, and thus find a near equilibrium configuration of nodes. With the optimized configuration, nodes can then be connected into well-shaped elements by constrained Delaunay triangulation algorithm.

Although Monte Carlo simulation is used in many fields, to our best knowledge it was never used for mesh generation. However, by treating domains as energy systems, it is natural to apply Monte Carlo simulation to obtain minimum-energy structures. Monte Carlo simulation was used by Lim, Nebus and Assad [40] to find extremal states of the logarithmic N-body problem on a sphere. Although they did not aim to address a mesh generation problem, the resulted polygons on the spherical surface are actually representing well-shaped surface mesh. A similar approach is used in our study: after the energy function is defined for particle interactions, nodes are placed in optimized locations by energy minimization using Monte Carlo simulation.

3.2 Method

3.2.1 The Energy System

A standard meshing procedure for 2D or 3D geometries produces a set of elements consisting of connected nodes. In the current method the nodes are viewed as a collection of atomic particles interacting with each other by a prescribed pairwise potential energy function. Each realization of positions of all the particles results in a certain value for the total potential energy of the system. In a stable configuration of particles, the system should have minimal potential energy. Therefore, if the pair potential function is known, it is possible to find a minimum-energy configuration of particles by energy minimization.

This optimization approach to mesh generation starts with the definition of the potential energy function describing interaction forces between the particles. Generally, the potential energy of an atomic system may come from an external force field, pair interactions, three-body interactions etc. Usually the most important part of the potential is the pair potential, which depends only on the distance between two particles [1]. Considering the pair potential only, the system potential energy (φ) can be represented as:

$$\varphi = \sum_{i} \sum_{j>i} \phi(r_{ij}) \tag{3.1}$$

where r_{ij} is the distance between a pair of particles i and j, $\phi(r_{ij})$ is the pair potential.

In our study, we choose the Lennard-Jones (LJ) 12-6 potential as the pair potential:

$$\phi(r) = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \tag{3.2}$$

Figure 3.1 shows the Lennard-Jones pair potential with $\sigma = 10$ and $\varepsilon = 150$. Parameter σ will determine the separation of particles in an equilibrium situation, which happens when the particles occupy the positions corresponding to the minimum of energy. For a pair of particles the equilibrium separation is equal to $\sigma_0 = 2^{1/6}\sigma \approx 1.122\sigma$, which corresponds to the minimum of the curve in Fig.3.1. When the distance

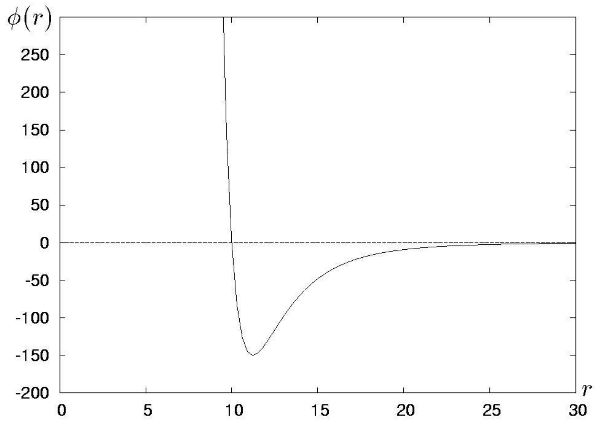


Figure 3.1: The Lennard-Jones pair potential.

is less than σ_0, the pair potential rises steeply as r decreases, modeling the repulsion between the pair of particles when they are too close to each other. When the distance is larger than σ_0, the pair potential has an attractive tail which also leads to energy increase relative to its minimum. The potential is approaching a constant value as the separation between the particles increases. This value is set to be zero, since it corresponds to the absence of any interactions when two particles are infinitely far from each other. For practical purposes the particles can be considered out of interaction range when their separation is greater than 2σ. Parameter ε determines the depth of the negative well of the potential.

With this energy function, in an idealized situation, all mesh nodes or particles will be separated at distances corresponding to the minimum of the superposition of all pair-potentials of the surrounding particles. Thus, the total system energy will also be at the minimum. It should be noted, that the superposition of many potential functions can make particle separation at equilibrium to be less than σ_0. However, because of the steep rise of the LJ potential at the distances less than σ_0 this effect is very small. In addition to that it is be further reduced by the introduction of the cutoff distance (2σ in our case).

The situation of stable equilibrium for an ensemble of particles occurs at the minimum of system energy with respect to particle positions. This minimization can be accomplished using molecular dynamics simulations or the Monte Carlo (MC) method. However, the resultant particle distribution can be highly uneven if the total number of particles was not guessed correctly. For this reason we introduce an empirical particle insertion/deletion criterion based on total system energy level and use it in combination with the Monte Carlo methods to determine both the total number of particles and their positions.

3.2.2 Metropolis Monte Carlo Approach to Node Placement

One of the Monte Carlo methods used in our study is the Metropolis Monte Carlo method (MMC) [1]. The MMC method uses random moves to explore the search space, and accepts the moves depending on the energy states of the system. One advantage of this method is its simplicity and straightforward implementation for both 2D and 3D domains.

In MMC method, a trial move is accepted or rejected based on the following rule:

$$\begin{aligned} &\texttt{accept} \quad if\ e^{-\beta\Delta\varphi} > R \\ &\texttt{reject} \quad if\ e^{-\beta\Delta\varphi} \leq R \end{aligned} \tag{3.3}$$

where $R \in (0,1)$ is a random number generated at each trial move, $\Delta\varphi$ is the

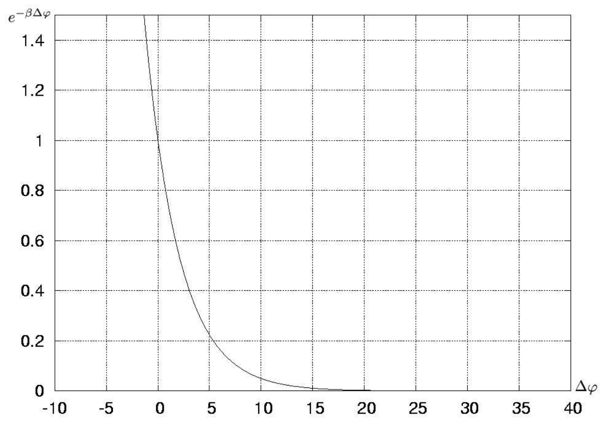


Figure 3.2: The function $e^{-\beta\Delta\varphi}$ vs $\Delta\varphi$ with $\beta = 0.3$.

energy change of the system after the trial move, and β is actually $1/kT$, where k is Boltzmann constant and T is temperature.

The origin of the function $e^{-\beta\Delta\varphi}$ is the Boltzmann Distribution law, in which β depends on the system temperature. Since our system is not a real atomic system, the temperature scale is irrelevant. However, we need to consider the order of energy change $\Delta\varphi$ when choosing the value of β. It should be selected such that the probabilistically accepted energy increase range is reasonable for the system.

Figure 3.2 shows the change of $e^{-\beta\Delta\varphi}$ related to $\Delta\varphi$ when $\beta = 0.3$. As illustrated in the figure, when $\Delta\varphi < 0$, the value of $e^{-\beta\Delta\varphi}$ is always greater than R (i.e. $e^{-\beta\Delta\varphi} > 1.0 > R$), and the move will be always accepted. If $\Delta\varphi > 0$, the move is probabilistically accepted when the energy increase is small (e.g. $\Delta\varphi < 20$). The move is essentially rejected when the energy increase is large (e.g. $\Delta\varphi > 20$), since in this case $e^{-\beta\Delta\varphi}$ is very close to 0. Overall the system's potential energy will decrease when increasing the number of random moves of the nodes.

During the simulation, the same procedure is repeated over and over again until

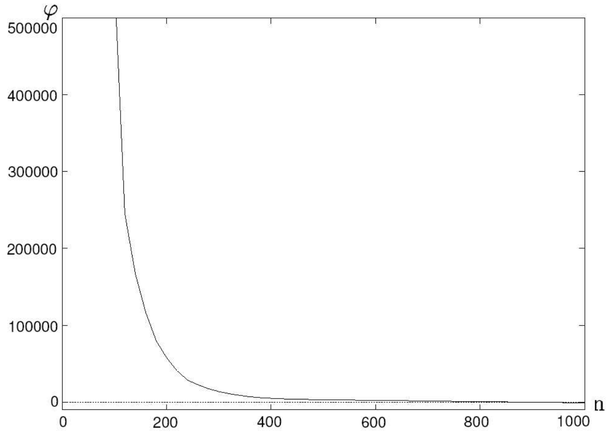


Figure 3.3: The system energy declines when increasing the number of sweeps.

an equilibrium is found or a chosen number of sweeps is done. A sweep is a series of N trials, where N is the number of nodes in the system. Since the Monte Carlo method is a statistical method, it is impossible for it to find a configuration of exact energy minimum with a finite number of sweeps. However, at the start of the procedure the system energy declines rapidly, and within hundreds of sweeps reaches a near equilibrium. Figure 3.3 shows a typical energy decline with the number of sweeps. For mesh generation, a configuration of nodes near the equilibrium is good enough. Usually, after 500 sweeps the nodes can be placed in very good locations. After the node placement, one can connect the nodes into triangles or tetrahedrons by constrained Delaunay triangulation or tetrahedrization.

3.2.3 Node Population Control

When starting the Monte Carlo simulation, one usually doesn't know how many nodes need to be distributed in the system in order to get a good mesh. Too few nodes will result in a stable but uneven node distribution with node clusters in some regions and voids in the others. For a potential function defined by (3.2), this state will be characterized by total system energy being negative, and will correspond to a solid lattice of particles partially occupying the domain. If there are too many nodes in the domain, the distances between the nodes may become smaller than the stable equilibrium distance and the total system energy may become positive and large. This will be a state of unstable or dynamic equilibrium and will correspond to a liquid.

As we mentioned before, for LJ potential on 2D lattices the stable equilibrium will occur at average node separations between σ and σ_0. However, by trial and error it was found that it is beneficial to create a slightly denser system by packing particles at an average distance of σ. By "liquifying" the system slightly in this way we essentially prevent the occurrence of void regions. In addition to this it gives us a way to automatically control the number of nodes, using system energy level. Since all node separations of more than σ contribute with the negative potential and all those less than σ give positive contributions (Fig.3.1), we can use a simple criterion for mesh-size control: nodes should be inserted when the equilibrium energy of the system is negative, and removed when it is positive. And we stop when the system equilibrium energy level is zero or slightly above zero.

3.2.4 Grand Canonical Monte Carlo Approach to Node Placement

One disadvantage of the Metropolis Monte Carlo method is that the number of nodes during the simulation is fixed. Since we usually don't know in advance how many nodes are needed, sometimes it is more convenient to use the Grand Canonical Monte

Carlo (GCMC) method, which is able to dynamically change the number of particles by imposing a fixed system potential. During the simulation, the boundary nodes are kept fixed, while the interior nodes are inserted/removed/displaced to satisfy a chosen zero energy level, and in the mean time particles are distributed in optimal locations.

One important feature of GCMC is that the potential is imposed during the simulation, while the number of particles fluctuates. It is a good fit for the node placement problem since we don't know in advance how many nodes are needed to cover the domain but we know the desired system potential is zero.

There are three types of trial moves in a GCMC simulation. They are:

1. Displacement of particles. A particle is randomly chosen to move with a random displacement. The displacement is accepted or rejected based on the same displacement rule as in the Metropolis Monte Carlo method:

$$\begin{aligned} &\texttt{accept} \quad if\; e^{-\beta\Delta\varphi} > R \\ &\texttt{reject} \quad if\; e^{-\beta\Delta\varphi} \leq R \end{aligned} \tag{3.4}$$

2. Insertion of particles. A particle is inserted into the system at a random position. The insertion is accepted or rejected based on the following criteria:

$$\begin{aligned} &\texttt{accept} \quad if\; \frac{zV}{N+1}e^{-\beta\Delta\varphi} > R \\ &\texttt{reject} \quad if\; \frac{zV}{N+1}e^{-\beta\Delta\varphi} \leq R \end{aligned} \tag{3.5}$$

3. Removal of particles. A particle is randomly chosen to be removed. The removal is accepted or rejected based on the following criteria:

$$\begin{aligned} &\texttt{accept} \quad if\ \frac{N}{zV}e^{-\beta\Delta\varphi} > R \\ &\texttt{reject} \quad if\ \frac{N}{zV}e^{-\beta\Delta\varphi} \leq R \end{aligned} \tag{3.6}$$

In the above equations, R is a randomly generated number in $(0, 1)$ at each trial move; $\Delta\varphi$ is the energy change of the system after the trial move; N is the number of particles in the system before the trial move; V is the volume of the system; $\beta = 1/kT$, where k is Boltzmann constant and T is temperature of the system; $z = e^{\beta\mu}/\Lambda^3$, where μ is the fixed system potential and should be zero for our problem, and Λ is called thermal de Broglie wavelength.

3.2.5 Implementation

Mesh element size control. The mesh element size control is achieved by the parameter σ in the pair potential function (3.2). If we need to obtain a uniform mesh of equally sized elements, we set σ to be constant. To produce a non-uniform mesh σ is set according to an appropriate node spacing function, which is a function of coordinates. For each node, the node spacing function defines the desired distance from this node to the other nodes. Since each node may have different desired distance, when calculating pair potential for a pair of particles, we set σ as the average of the two desired distances.

Displacement of particles. During the simulation, at each trial the Monte Carlo method randomly select a node to move with a random displacement. Suppose Cartesian coordinates (x, y, z) are used for the mesh nodes, then a trial move of a node i can be expressed as:

$$\begin{aligned} x'_i &= x_i + \Delta * (R_x - 0.5) \\ y'_i &= y_i + \Delta * (R_y - 0.5) \\ z'_i &= z_i + \Delta * (R_z - 0.5) \end{aligned} \tag{3.7}$$

where R_x, R_y, and R_z are random numbers between 0.0 and 1.0.

The displacement of x_i, y_i, and z_i will be from $-\Delta/2$ to $\Delta/2$. At each trial move, we check first if the new location is in the bounding space of the domain to be meshed, and reject the move if it is not.

The size of each moving step and the number of moves attempted are two important factors that control how well the Monte Carlo simulation will perform. If the size is too large, the change of the energy between two steps may become significant, and virtually all of the attempts will be rejected. In this case the algorithm may get stuck at a particular place in the search space. On the other hand, if the size is too small, a large number of moves may be needed to explore the whole search space. In this case the simulation will become slow. Fortunately, this size can be dynamically adjusted according to the acceptance ratio during the simulation process.

In our study, we increase the size 5 percent after each sweep, if the acceptance ratio is greater than 0.4; or decrease the size 5 percent, if the acceptance ratio is less than 0.2; or don't change the size otherwise. Since the adjusting process takes time, it is still important to choose a reasonable initial moving size according to the search space and the energy function. For a domain with varying node spacing, extra care needs to be taken since one step size which is small enough in one region could be too large in another region. What we can do is to set different maximum displacement steps for different regions according to their required local node spacings.

For a node displacement on a surface, only two coordinates are to be displaced during each move. If the surface is a flat surface on the $x-y$ plain, then coordinates to be displaced are x and y ; if the surface is a parametric surface in the 3D space, such as a nurbs surface which has an underlying $u-v$ representation, then the coordinates to be displaced are u and v.

Computation of system energy change. The system energy change is checked after a trial move to decide whether the move is accepted or not. When calculating the energy change between two states, it is not necessary to calculate the total system energy, only the change associated with the moving node needs to be calculated.

The energy change after displacing node i from state n to state $n+1$ is calculated as:

$$\Delta\varphi = \sum_{j=1;j\neq i}^{N} \phi(r_{ij}^{n+1}) - \sum_{j=1;j\neq i}^{N} \phi(r_{ij}^{n}) \tag{3.8}$$

The energy change after inserting node $N+1$ into the system is calculated as:

$$\Delta\varphi = \sum_{j=1}^{N} \phi(r_{(N+1)j}) \tag{3.9}$$

The energy change after removing node i from the system is calculated as:

$$\Delta\varphi = -\sum_{j=1;j\neq i}^{N} \phi(r_{ij}) \tag{3.10}$$

Further improvement can be done, considering that the nodes which are interacting with the moving node should be within certain short distance from the latter. For the energy system with the above energy function (3.2) a particle can be considered to be only interacting with its neighbor particles which are within a sphere of radius less than 2σ, since the Lennard-Jones pair potential gets close to zero when $r > 2\sigma$. By doing this, computing time can be drastically reduced.

To confine the pair interactions within a small local range, we use a quadtree (for 2D) or octree (for 3D) structure. This tree structure is constructed to cover the whole domain to be meshed. Based on the geometric domain and the node spacing function, the quatree or octree is divided into many leaves whose sizes are in $(2\sigma, 4\sigma)$. During the simulation, particles are indexed into the leaves of the tree structure, and thus for any particle its neighbor particles within the cutoff distance (2σ) can be found quickly by traversing through this tree structure. An even faster method would be to use a fully threaded tree [33], however we opted for a simpler implementation at this stage.

Since the interaction for each particle is in a short range, the time complexity of the computation at each iteration in the simulation is $O(N)$.

Initial configuration of nodes. The simplest way to get an initial set of nodes is to randomly generate a set of nodes. That is a specified number of nodes are inserted into random locations in the domain to be meshed. And they will be distributed into optimal location after the simulation.

However in order to speed up the simulation, a good initial configuration of nodes would be helpful. To get a set of good configured initial particles, we can take advantage of the above mentioned quadtree (for 2D) or octree (for 3D) structure. Since the size of a leaf of the tree structure is related to the desired local node spacing when the leaf is inside the domain, we can make a fairly good guess for the number of nodes inside each leaf, accordingly put a certain number of nodes into each leaf which is inside the domain, and make them well separated before the MC simulation starts.

Choosing parameters

Although we are considering the geometric domain as an energy system full of particles, it is not a real energy system. On one hand, we lose the advantages of using available parameters from real energy systems; On the other hand, we may have more freedom to setup the parameters. For example, when choosing β which is actually $1/kT$, we don't need to worry about the temperature. But still it is important to choose a suitable value such that the probabilistically acceptable energy increase range is reasonable for the system. The bigger the β, the smaller the probabilistically acceptable energy increase range.

3.2.6 Summary of the Meshing Procedure

If using the Metropolis Monte Carlo method, the meshing procedure can be summaried as follows:

1. Define node spacing function, energy function, choose values for various parameters.

2. Configure an initial node placement inside the domain using a quadtree (for 2D) or octree (for 3D) structure. Treat the inside nodes as moving particles. Take the boundary nodes such as end vertices of edges or boundary mesh nodes as fixed particles.

3. Use Monte Carlo simulation to minimize system energy, in the mean time produce a near equilibrium distribution of particles.

4. Check the minimal system energy. Increase the number of nodes if it's negative, or decrease the number of nodes if it's too large.

5. Repeat from step 3 until the final system energy is close to zero. The node placement is done.

6. Connect the nodes into a mesh by constrained Delaunay triangulation or tetrahedrization.

If using the Grand Canonical Monte Carlo method, then the whole procedure can be simplified since the number of nodes will be dynamically adjusted. The meshing procedure is summarized as follows:

1. Define the energy function, the node spacing function, and values for various parameters.

2. Configure an initial node placement inside the domain using a quadtree (for 2D) or octree (for 3D) structure. Treat the inside nodes as moving particles. Take the boundary nodes such as end vertices of edges or boundary mesh nodes as fixed particles.

3. Start the Monte Carlo simulation with GCMC method to displace/insert/remove particles until an near equilibrium is found or a chosen number of sweeps is done. And in the mean time, nodes are distributed into well spaced configuration.

4. Connect the nodes into a mesh by constrained Delaunay triangulation or tetrahedrization.

When it comes to the overall time complexity of the algorithm, it can be noted that, since the time complexity for each iteration of the Monte Carlo simulation is $O(N)$ and the number of iterations k does not increase with the number of nodes N, the computational time for the node placement stage is $kO(N)$. The time complexity of the node connection stage will depend on the constrained Delaunay triangulation algorithm used, which can be as good as $O(Nlog(N))$. Therefore, the time complexity for the whole meshing method can be as good as $kO(N) + O(Nlog(N)) = O(Nlog(N))$.

A Java applet demonstration of the algorithm for 1D, 2D and 3D cases can be run at http://mulphys.mae.wvu.edu/mesh/mc.

In next section, we will show some results for meshing both 2D and 3D domains. The algorithm works in an almost identical way for both 2D and 3D cases.

3.3 Results

To illustrate the working of the method we considered several 2D and 3D examples, with the MMC method and GCMC method respectively.

The input geometry is to be described by a *.poly* file or represented by a signed distance function.

For a 2D geometry, a *.poly* file can be used to describe a Planar Straight Line Graph (PSLG) containing a list of vertices and segments, and some other information such as holes. This is a format used in Shewchuk's triangle [53] program. For a 3D geometry, a .poly file can be used to represent a piecewise linear complex (PLC) as well as some additional information. It consists of a list of points, a list of facets, and some other information such as holes. This format is used in Si's tetgen program (see http://tetgen.berlios.de) . When using a .poly file, the input vertices are considered as particles in the energy system, but they are fixed and don't move during Monte Carlo simulation.

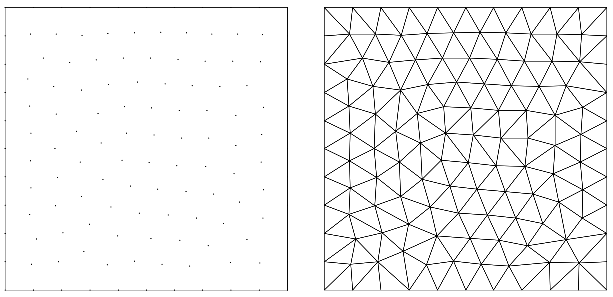
A signed distance function implicitly describes the geometry by defining the distance from any node to its nearest boundary of the domain. A node has a negative distance when it is inside the boundary, positive distance when it is outside, or zero when it is on the boundary of the domain. If the boundary has been discretized, we can add the boundary discretiztion nodes as fixed particles for the GCMC simulation.

Actually the input geometry can be anything as long as we have a way to keep the particles inside the geometry when performing random moves. For geometries implicitly described by signed distance functions, inside geometry testing is trivial.

3.3.1 2D Surface Mesh Examples with the MMC Method

When using the MMC method, the number of particles is fixed. In order to get a final energy slightly over zero, we might need to repeat the simulation several times by trying different number of nodes.

Figure 3.4 shows node placement inside a 200×200 square by the MMC after 500 sweeps, and the corresponding constrained Delaunay mesh. The value of ε chosen for


(a) Node placement in a square

(b) Constrained Delaunay triangulation of the nodes

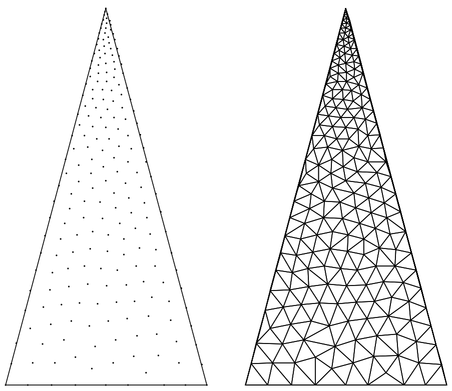
Figure 3.4: Node placement in a square and the corresponding constrained Delaunay mesh

the energy function is 100. The mesh size is to be uniform over the square by setting $\sigma = 20$.

The square is described in a *.poly* file with 40 segments of size 20, where we can consider the 4 edges have already been meshed. The 40 nodes on the 4 edges are considered as fixed particles in the system. The number of nodes we randomly generated for the inside area is 90. After 500 sweeps, the system energy drops from $1.0e^{19}$ to about 3000.

It is interesting to note that one can actually mesh the square without meshing the edges first. During the Monte Carlo simulation, some particles inside will be pushed very close to the boundary edges. We can make these particles become the boundary nodes by moving them to their projection points on the closest edges. In this way the edges are meshed at the same time as the surface.

Figure 3.5 shows an example of this kind of situation. The triangle edges are represented by 3 vertices and not meshed before the Monte Carlo simulation. After


(a) Node placement in a triangle

(b) Constrained Delaunay triangulation of the nodes

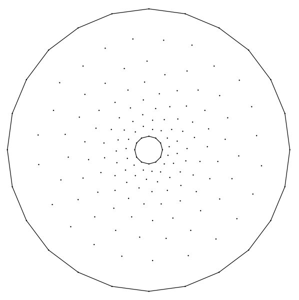
Figure 3.5: Node placement in a triangle and the corresponding constrained Delaunay mesh

500 sweeps, some nodes are pushed onto or very close to the edges. A good mesh can be obtained if these close nodes are moved to their projection points onto the edges. The mesh shown in the figure has some bad triangles without an extra moving procedure applied before connecting the nodes into mesh.

The triangle in Figure 3.5 is a isosceles triangle whose three angles are 30 degree, 75 degree and 75 degree. The vertical height of the triangle is 100. The mesh element size changes linearly with the height by setting the node spacing function as:

$$s(r) = -tan15^{\circ}(h - 100)/4 + 1.0 \tag{3.11}$$

Figure 3.6 shows the node placement and the corresponding constrained Dealaunay mesh in a hollow disc of radius 100 after 500 sweeps. The hole is a circle of


(a) Node placement in a hollow disc

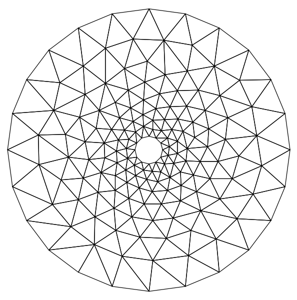

(b) Constrained Delaunay triangulation of the nodes

Figure 3.6: Node placement in a hollow disc and the corresponding constrained Delaunay mesh

radius 10. Since we use PSLG to describe the circles, the circles are represented by segments, i.e. they can be considered as a meshed boundaries. The inside circle is meshed with 12 equal segments of size 5.18, and the outside circle is meshed with 24 segments of size 26.11. A node spacing function (eqn. 3.12) is used to change the mesh size linearly from 5.18 to 26.11 with the distance r from the center of the disc. The value of ε chosen for the energy function is still 100. The 36 input vertices are considered as fixed particles, the 130 inside particles are randomly generated initially. After 500 sweeps, the system energy drops from $1.0e^{15}$ to about 1500.

$$s(r) = 0.2325r + 2.86 \tag{3.12}$$

The quality of a triangle can be measured with its aspect ratio, which is defined as the longest edge divided by the smallest height. The smaller the aspect ratio, the better the quality. The best triangle is an equilateral triangle with an aspect ratio of $2/\sqrt{3}$, or 1.1547.

The meshes shown in Figures 3.4 and 3.6 have fairly good mesh qualities. For the former one, the smallest angle is 38 degree, the largest angle is 102 degree, and the largest aspect ratio is 2.46. For the latter one, the smallest angle is 35 degree, the largest angle is 104 degree, and the largest aspect ratio is 2.57.

The mesh shown in Figure 3.5 has some bad triangles. Since no edge meshing was done in that case, preceding the surface meshing, many nodes close to the boundary edges are not exactly on the edges. To get a good mesh, we need to move these nodes to the boundaries, or mesh the edges before the surface node placement.

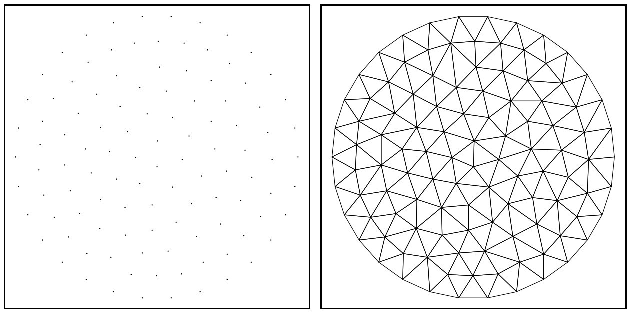
3.3.2 2D Surface Mesh Examples with the GCMC Method

To avoid guessing the right number of nodes and repeating the simulation, the GCMC method can be used instead of the MMC method.

Figure 3.7 shows a node placement inside a circle of radius 100 with the GCMC method after 500 sweeps, and the corresponding Delaunay mesh. The node spacing is desired to be uniform over the domain by setting $\sigma = 20$. The mesh has fairly good quality. The largest aspect ratio of triangles is 2.2; the smallest angle in the mesh is 35.6 degree, and the largest angle is 95.2 degree.

The boundary of the circle has been discretized before the MC simulation. The 30 equally spaced nodes on the boundary are given as fixed particles for the simulation. After 500 sweeps, the GCMC method adjusts the number of the inside particles to 80. And the final system potential is slightly over zero (1590).

Figure 3.8 shows a node placement and the corresponding constrained Dealaunay mesh in a hollow square after 500 sweeps. The square is of size 200*200, and the hole is a circle of radius 20 in the center. The boundaries of the circle and the square



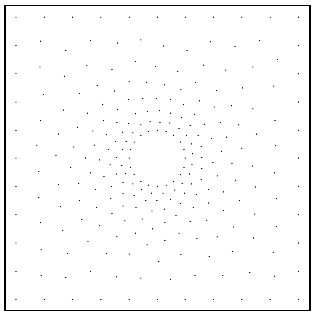
(a) Node placement in a circle

(b) Delaunay triangulation of the nodes

Figure 3.7: Node placement in a circle and the corresponding Delaunay mesh

are discretized before the MC simulation. The circle boundary is discretized by 20 equally separated nodes, and the 4 square edges are discretized by 40 equally spaced nodes. The 60 boundary nodes are given as fixed particles. A node spacing function $s = min(r/5.0 + 2.0, 20.0)$ is used to change the spacing with the distance r from the center. After 500 sweeps, the GCMC method adjusts the number of inside nodes to 193. Again the mesh is fairly good. The largest aspect ratio of the triangles is 2.6. The smallest angle in the mesh is 30.8 degree, and the largest angle is 104.3 degree. And the final system potential is slightly over zero (2896).

As mentioned above, one can actually distribute nodes in the domain without discretizing the edges first. During the Monte Carlo simulation, some particles will be pushed very close to the boundary edges. By projecting these particles onto the boundary, the edges can be discretized at the same time as the surface. Figure 3.9 shows such an example with the GCMC method. This geometry is borrowed from [48]. It is represented by the following distance function:


(a) Node placement in a hollow square

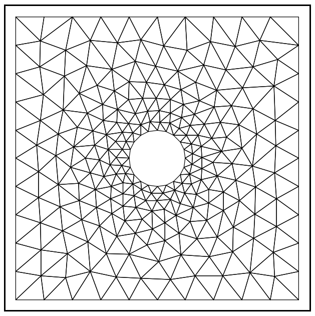

(b) Constrained Delaunay triangulation of the nodes

Figure 3.8: Node placement in a hollow square and the corresponding constrained Delaunay mesh

$$\begin{aligned} d_1 &= \sqrt{x^2+y^2}-1 \\ d_2 &= \sqrt{(x+0.4)^2+y^2}-0.55 \\ d &= \max(d_1,-d_2,-y) \end{aligned} \tag{3.13}$$

Also the node spacing function is from the corresponding mesh size function in [48] with a slight change.

$$\begin{aligned} s_1 &= 0.15-0.2d_1(x,y) \\ s_2 &= 0.06+0.2d_2(x,y) \\ s_3 &= (d_2(x,y)-d_1(x,y))/3 \\ s &= \min(s_1,s_2,s_3)+0.05 \end{aligned} \tag{3.14}$$

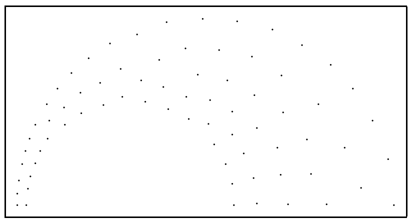


Figure 3.9: Node placement without boundary discretization first

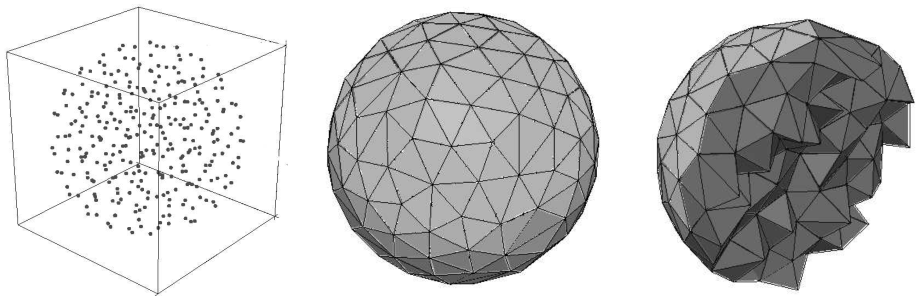
Besides the distance function, the 4 end vertices are set as fixed particles for the MC simulation. After 1000 sweeps, the total number of particles becomes 74 and the nodes are distributed into a fairly good configuration. From the picture we can see some nodes are pushed onto or very close to the boundary edges. A good Delaunay mesh can be obtained if these close nodes are moved to their closest boundary points. However in order to speed up simulation and improve mesh quality, it is better to discretize boundaries first and take the boundary nodes as fixed particles in the MC simulation.

3.3.3 3D Volume Mesh Examples with the MMC Method

The algorithm for 3D meshing is almost identical to the one for 2D, and thus, in contrast to other meshing techniques, moving from 2D to 3D meshing does not increase the complexity of the method.

Figure 3.10 shows the node placement in a sphere after 1000 sweeps with the MMC method, and the corresponding constrained Delaunay tetrahedrization. The number of nodes is 300. No surface meshing was done in this example.

Figure 3.11 shows the node placement in a cube, and the corresponding constrained Delaunay mesh. The cube has a sphere hole in the center. The size of the



(a) Node placement in a sphere

(b) Constrained Delaunay tetrahedrization of the node configuration

(c) A cross section of the mesh

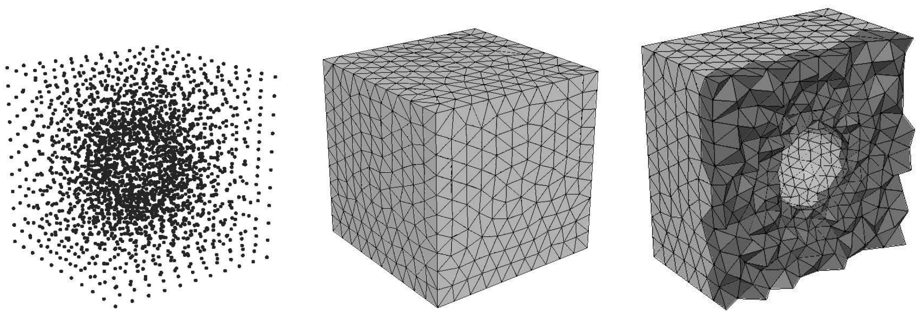
Figure 3.10: Node placement in a sphere and the corresponding constrained Delaunay tetrahedrization

cube is $200 \times 200 \times 200$, and the radius of the sphere is 40.

The surface meshing is done first for the sphere surface and the 6 square surfaces of the cube before the volume meshing. The meshing for a square was shown in the above 2D examples. The meshing for the sphere surface is using the same method although it is a 3D surface meshing. The initial random points are generated on the spherical surface, and they can only move on this surface during the MC simulation.

The mesh element size for the sphere surface mesh is about 10, and the mesh element size for the cube surfaces is about 20. The number of mesh nodes of the sphere surface is 200, and the number of the mesh nodes for the cube surfaces is 655.

After the surface meshing, we consider the surface mesh nodes as fixed particles in the energy system, and insert random nodes into the volume. After 500 sweeps, we get the node placement as shown in Figure 3.11. The number of nodes distributed inside the volume is 1700. After the node placement with MC simulation, we connect


(a) Node placement in a hollow hollow cube

(b) Constrained Delaunay tetrahedrization of the node configuration

(c) A cross section of the mesh

Figure 3.11: Node placement in a hollow cube and the corresponding constrained Delaunay tetrahedrization

the nodes into constrained Delaunay mesh. The total mesh nodes of the volume mesh is 2555.

The mesh element size is changing with the distance r from the center by setting the spacing function as:

$$s(r) = min(10.0/3.0 + r/6.0, 20) \tag{3.15}$$

3.3.4 3D Volume Mesh Examples with the GCMC Method

Figure 3.12 shows the node placement in a cube with the GCMC method after 500 sweeps and the corresponding constrained Delaunay tetrahedrization. The cube is of size $200 \times 200 \times 200$. The node spacing is desired to be uniform over the domain by setting $\sigma = 20$. The six boundary surfaces are discretized (by Monte Carlo simulation also) before the node placement inside the volume. The 656 surface nodes are fixed

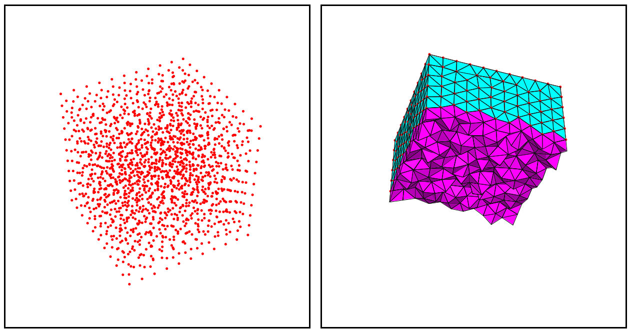

(a) Node placement in a cube (b) A cross section of the tetrahedral mesh

Figure 3.12: Node placement in a cube and the corresponding constrained Delaunay tetrahedrization

for the MC simulation. The initial inside particles are generated with the help of a quadtree structure. After 500 sweeps, the total number of particles for the cube becomes 1663, and the final system potential is slightly over zero.

All of the above tetrahedral meshes have very good radius-edge ratio. the largest radius-edge ratio is less than 2.0, and for most tetrahedra it is less than 1.0. However some tetrahedra have very large aspect ratio. This is because some slivers are generated. Slivers can be generated with Dealaunay methods even the nodes are well spaced [63]. Sliver elimination can be achieved by using the algorithms introduced in Chapter 1.

In this study, only the Lennard-Jones 12-6 potential is used to construct the energy function. And it is related to distance or node spacing only. Introducing more parameters, such as dihedral angle (which induces bending energy), into the energy function might be helpful to address this problem. More work need to done to investigate this possibility.

$Mesh$	$Angle_{min}$	$Angle_{max}$	r_{max}
by the Delaunay refinement method	35.6^o	108.3^o	2.77
by our method	35.6^o	95.2^o	2.20

Table 3.1: Mesh quality comparison with the Delaunay refinement method.

3.4 Discussion

There are many advantages for the proposed meshing method, such as:

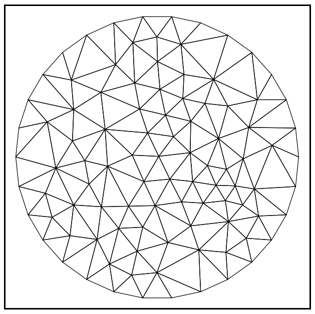
1. It generates higher quality mesh than the Delaunay refinement method.

   As mentioned in Chapter 1, one of the most popular meshing method is the Delaunay refinement method [50, 54]. And the *Triangle* program by Shewchuk is 2D mesh generator using the Delaunay refinement method. Here we compare the meshes produced by our method and *Triangle*.

   We use *Triangle* to mesh the same geometry shown in Figure 3.7. The input to the program is the same 30 boundary discretization nodes. And a minimal angle of 35.6 is used as its refinement criteria. Figure 3.13 shows both meshes produced by our method and the Delaunay refinement method. Figure 3.14 compares the aspect ratio histograms of both meshes. We can see from the figures that our mesh has better quality. Table 3.1 compares the quality measurements. Both meshes have the same minimal angle. But the mesh produced by our method has much smaller maximal angle and maximal aspect ratio.

   It is understandable that our mesh has better quality because the node locations are globally optimized in our method.

   The mesh quality of our method is also comparable with some commercial code. Figure 3.15 and Table 3.2 show a comparison with a mesh produced by


(a) Mesh generated by the Delaunay refinement method

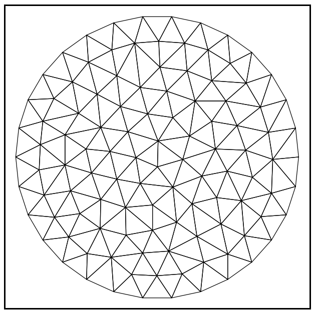
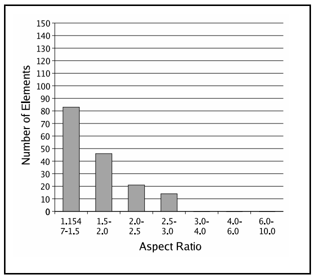
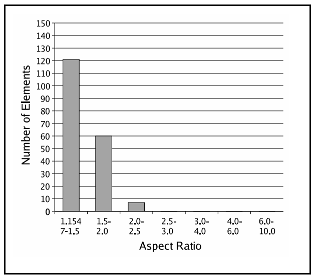

(b) Mesh generated by our method

Figure 3.13: Mesh comparison with the Delaunay refinement method.

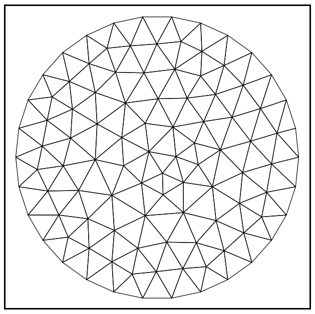


(a) Histogram of the mesh by the Delaunay refinement method



(b) Histogram of the mesh by our method

Figure 3.14: Aspect ratio histogram comparison with the Delaunay refinement method.


(a) Mesh generated by Gambit

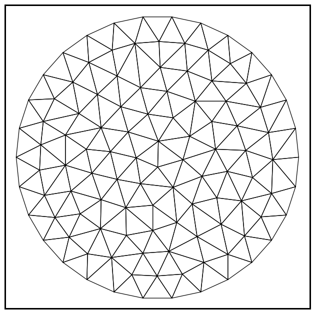

(b) Mesh generated by our method

Figure 3.15: Mesh comparison with mesh by Gambit.

$Mesh$	$Angle_{min}$	$Angle_{max}$	r_{max}
by Gambit	34.0^o	91.8^o	2.08
by our method	35.6^o	95.2^o	2.20

Table 3.2: Mesh quality comparison with Gambit.

Gambit, a mesh generator of Fluent (*http://www.fluent.com/*). Although the mesh generated by Gambit has slightly better qualities in terms of angles and aspect ratio, this mesh is less uniform than our mesh. Also the mesh generated by Gambit is not a Delaunay mesh. The algorithm used for the mesh generation is Gambit's tri/pave scheme.

Another mesh comparison is shown in Figure 3.16 and Table 3.3. This time although the sphere surface mesh produced by our method is good, the mesh produced by Gambit is even better. Again the tri/pave scheme is used for Gambit to generate the mesh.

2. It is simple and easy to implement, and works identical for 1D, 2D and 3D problems.

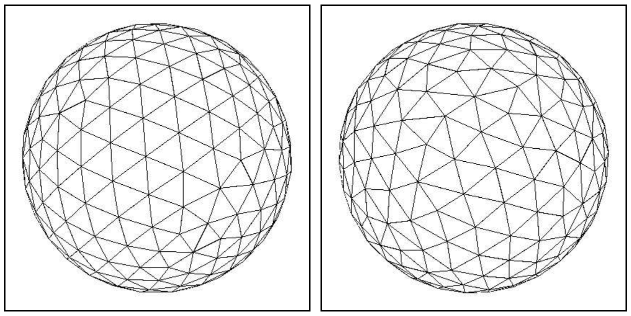

(a) Mesh generated by Gambit (b) Mesh generated by our method

Figure 3.16: Sphere surface mesh comparison with mesh by Gambit.

$Mesh$	$Angle_{min}$	$Angle_{max}$	r_{max}
by Gambit	44.3^o	78.5^o	1.65
by our method	36.7^o	101.5^o	2.46

Table 3.3: Sphere surface mesh quality comparison with Gambit.

One advantage of the Monte Carlo method is that it is unaffected by the dimensionality of the problem. For any dimension, the rules for random moves of particles and acception/rejection of trial moves are the same. Unlike other meshing methods, moving from one dimensionality to another will not increase the complexity of the method.

3. The algorithm enables one to produce a surface mesh without meshing the edges first, and volume mesh without meshing the boundary surfaces first.

   As demonstrated in the examples above, during the MC simulation, some nodes can be pushed very close to boundaries and thus they can be used to mesh boundaries. However, because of its stochastic nature, it is impossible for MC simulation to push nodes to exactly coincide with the boundaries. In order to get a high quality mesh, an extra step is needed to project these nodes onto the boundaries after the MC simulation.

In some applications separate edge, surface, and volume meshing would be preferred. In this case one can mesh a domain in the order of edge meshing, surface meshing and volume meshing, with each higher-dimensional procedure retrieving the boundary information from the low-dimensional one. Thus, the edge meshing procedure will use its end vertices as fixed boundary points, the surface meshing procedure will obtain its boundary nodes from the edge meshing procedure, and the volume meshing procedure will obtain its boundary nodes from the surface meshing procedure. Because the algorithm works in almost identical manner for 1, 2, and 3 dimensions, this approach can be easily implemented with one basic routine.

4. The method is capable to control mesh element size. By customizing the node spacing function, the resultant mesh can be uniform, or non-uniform with changing nodal density.

5. The method is flexible. By customizing energy functions, the method may be able to generate different meshes, such as anisotropic meshes.

   In molecular mechanics, due to the effects of different interaction energies, such as the bonding energy, bending energy, and stretching energy etc, many different molecular structures exist. From this prospective, one may be able to mimic the energy functions found in molecular mechanics for certain crystal structure and consequently generate similar shape mesh elements.

6. The method can be used directly to generate nodes for mesh free methods [75].

Since no mesh required for mesh free methods, there is no need to connect nodes.

7. The MC simulation approach can be used for optimization of an existing mesh.

   For an existing mesh, we can use the available element connectivities to define particle interactions. For example, one node can be considered interacting with nodes only connected directly with it. With the MC simulation, nodes can be moved to optimal locations while keeping the element connectivities.

Our method uses a particle simulation approach which is similar to that is used in the bubble mesh scheme[56]. As mentioned in the bubble mesh scheme, because of the non-linear characteristics of the force (or energy in our approach) and the strict geometric constraints imposed on each degree of freedom to keep a particle on the domain to be meshed, a direct solution to the static force balance, using a multi-dimensional root-finding method such as the Newton-Raphson method, is not efficient. Instead, a particle simulation approach is more appropriate.

Due to their similarity, it is worthy to compare our method with the bubble mesh scheme.

1. The bubble mesh scheme defines a inter-bubble force between a pair of particles, while our method defines energy for the system. The energy approach is more flexible.

2. While our method uses a stochastic approach, the bubble mesh scheme uses a deterministic approach, which is molecular dynamics (MD) simulation. A set of differential equations of motion is integrated through time and solved iteratively. A careful design for a combination of physical parameters is needed in order to

avoid instabilities and slow convergence. Consequently, this approach appears to be excessively complicated for the problem at hand.

3. During each time step of the bubble mesh scheme, the particles can move out of the domain to be meshed. In order to confine particles to curves and surfaces, the bubble mesh scheme has to correct the movement of particles in each time step. While this correction is possible for parametric curves or surfaces, it might be difficult for some other geometries, such as the surface of a voxel-based object. In our method, a particle is always confined to the domain to be meshed during a random trial move, there is no need to use an extra step to correct particle movement.

4. Comparing with the bubble mesh scheme, another advantage with the proposed MC approach is its simplicity of controlling node population. In the bubble mesh scheme, an additional procedure is used to examine local bubble (node) population, and add or remove bubbles (nodes) according to the node distances. With our approach, the number of particles (nodes) can be dynamically adjusted by adopting GCMC with a fixed system potential, which is desired to be zero.

5. Since for each particle, it interacts directly only with neighbor particles within a small range. The time complexity of the node placement with Monte Carlo simulation is linear to the number of nodes N, that is $O(N)$. The time complexity for the whole proposed meshing method will depend on the node connecting stage. When using a constrained Delaunay triangulation algorithm, the best time complexity can be $O(Nlog(N))$. While our method might need more iterations to get an equilibrium configuration of particles, it has the same best

time complexity $O(Nlog(N))$ as the bubble mesh scheme.

Although theoretically the time complexity for our meshing method can be as good as others, our method can be less efficient when the number of nodes is small. The reason is that we need many iterations for the Monte Carlo simulation to reach a near equilibrium, and the time for the node placement stage can outweigh the node connection stage. To improve the efficiency of the method, some improvements can be done:

1. Adopt more efficient Monte Carlo methods.

   The MMC method is a simple, inefficient MC method, many more efficient MC methods are possible to be adopted.

2. Use the Monte Carlo simulation in combination with the advancing front technique.

   In this study, the node placement is done for all the nodes before the nodes are connected into a mesh. However the node placement can also be used in combination with the advancing front approach for mesh generation. Instead of distributing all nodes inside the surface/volume at once, we can choose a small energy system around one edge/triangle on the front, and optimize the location for only one node each time with the MC simulation. In this way, the time complexity is the same as the advancing front method. Some other possibilities include partitioning a big domain into several smaller domains, and applying the MC simulation in each domain separately.

3. Use Monte-Carlo approach in combination with molecular dynamics simulations.

In this case the direction of particle displacement will be determined from the direction of the cumulative force computed in MD loop and thus, will reduce the trial-and-error effort of MC algorithm. by a factor of 3. This is because the direction of particle move is determined by 2 degrees of freedom, which will be fixed, and only the magnitude of the displacement will be selected at random. This approach is currently pursued as a further continuation of this study, and the java-based demonstrations are available at
http://mulphys.mae.wvu.edu/mesh/mc

4. Use parallel meshing.

   This method is easy to be parallelized. A domain can be partitioned into several sub-domains. Each sub-domain will be meshed with the same approach by keeping boundary points fixed.

An important finding of this study is the realization that dynamic node allocations and removal can be achieved by using the *liquid state ensemble of particles at energy levels above that of static equilibrium*. In other words, modeling the state of liquid instead of solid provides the possibility for a much more neat alignment of nodes and uniform filling of complex geometrical objects. This is a natural consequence of the behavior of fluids to fill out the available spaces, which so far was not exploited in the area of mesh generation.

3.5 Conclusion

A new approach of mesh generation with Monte Carlo simulation is proposed and demonstrated. Domains to be discretized are considered as energy systems of interacting particles. With Monte Carlo simulation, particles, i.e. nodes, are distributed in optimal locations. Well-shaped mesh can then be constructed by connecting the nodes. The algorithm is simple and easy to be implemented for both 2D and 3D domains.

The new concept of *liquid* state node ensemble was used to generate dynamic node distributions. In contrast to solid-state static equilibriums used before the liquid-state dynamic equilibrium enables one to fill out all interiors inside complex boundaries and achieve the desired node distribution.

The proposed approach is very flexible in addressing different mesh generation criteria by simple customizing of the energy function, the node spacing function etc.

One reason that Monte-Carlo based mesh generation was not used so far lies in the seemingly low efficiency of the algorithm related to the random nature of Monte Carlo statistical sampling procedure. This low efficiency is especially noticeable when the number of nodes is small and the ratio of node-placement to node connectivity efforts is relatively large. However, as large mesh sizes become more common this ratio is decreasing and the method becomes more attractive. In addition to this the efficiency of the method can be improved considerably using various techniques discussed in the previous section.

Next chapter we are going to revisit the surface meshing for the voxel-based objects using this approach.

Chapter 4

Surface Mesh Generation for Voxel-Based Objects via Monte Carlo Simulation

In Chapter 2, we proposed an advancing front meshing method for the surface mesh generation for voxel-based objects. As seen from the results, one obvious disadvantage of that method is that the surface features are not preserved in the final meshes.

In this chapter, the problem of triangular surface meshing for voxel-based objects is revisited by using the energy minimization approach proposed in the previous chapter. A voxel-based object is treated as an energy system and surface mesh nodes are treated as interacting particles. By customizing system potential energy function to reflect surface features, particles can be distributed into desired locations. Feature-preserved surface mesh can then be constructed by connecting the node set. The Metropolis Monte-Carlo method is used as the energy minimization tool. In this study, a surface edge capture method using a simple Laplace solver with incomplete

Jacobi iterations is also proposed and demonstrated.

4.1 The Methodology

As described in Chapter 2, the voxels of a meshing target are considered as points organized in a 3D grid. Each surface mesh node will be coincident with one of the boundary points. Since the boundary point set is usually very large, it is not practical to include all of them in the final mesh. Instead only a small set of boundary points will be enough to create a feature preserving surface mesh.

To find such set of boundary points or the locations of the mesh nodes, the mesh nodes are introduced as interacting atomic particles with a characteristic interaction potential. A certain number of particles can first be generated from random boundary voxel locations. These particles are then set to interact with each other, and move inside the set of boundary voxels. In a stable configuration, the system will have a minimal energy. Therefore, if the potential energy function of the system is known, then it is possible to find a minimum-energy configuration of particles by energy minimization.

4.1.1 Energy Function

Defining a suitable system energy function is the most important and difficult part of the algorithm. We need an energy function such that when the system energy is at its minimal, the particles will tend to occupy locations at important surface feature points such as the edges, sharp corners etc. At the same time the surface mesh connected through these particles or nodes should be conformal to the original surface within certain given error. The next section presents several effective customizations for the energy function.

4.1.2 Monte-Carlo Method

The version of the Monte-Carlo method used in our study to minimize the system energy is the Metropolis Monte-Carlo method (MMC), which is described in the previous chapter.

The MMC method uses random moves to explore the search space, and accepts or rejects the moves depending on the energy states of the system. In the simulation, each time a randomly selected particle will move from a boundary voxel to a randomly selected neighbor boundary voxel. For each voxel in the 3D grid, it has 26 neighbor voxels. Therefore there are 26 choices for a particle's trial move in order to move it to a neighbor voxel. However some of the choices will lead the particle away from the boundary. In order to keep the particles on the boundary surface, the trial moves to voxels which are not on the boundary are always rejected.

In the beginning of the simulation, a specified number of particles are randomly generated at the locations of the boundary voxels. The MMC method is then used to move the particles around the boundary surface.

In the MC simulation, the same procedure is repeated over and over again until an equilibrium is found or a chosen number of sweeps is done. A sweep is a series of N trials, where N is the number of particles in the system.

Since the Monte Carlo method is a statistical method, it is impossible for it to reach an exact equilibrium within a finite number of sweeps. However, the system can reach a near equilibrium within hundred of sweeps. In our implementation, a fairly good node distribution can be obtained within 500 sweeps.

When calculating the energy change between two states, it is not necessary to calculate the total system energy, only the change associated with the moving particle needs to be calculated. And for a typical energy system, a particle doesn't interact with all the particles in the system, instead it interacts only with its close particles within a certain cutoff distance. Therefore, we can only consider neighbor particles when calculating the energy change. By doing this, computing time can be drastically reduced, especially when the number of particles in the system is large. For this purpose, an octree covering the whole voxel grid is divided into many leaves whose sizes are about the cutoff distance. During a simulation, each particle is indexed into a leaf of the tree structure, and neighbor particles can be found quickly through the tree structure.

Another advantage of this locality feature of the algorithm is its easy parallelization, which may become an issue when the model is too large to fit onto a single workstation.

4.1.3 Connecting Node Set into a Surface Mesh

One way of connecting a node set into a surface mesh is to use Delaunay tetrahedrization. That is after the node distribution with the MC simulation, we can first compute a 3D Delaunay tetrahedrization from the node set to get a volume mesh. This will create a convex hull of the node set. For a convex object, the boundary mesh of this volume mesh will be the surface mesh of the original object. But for an object with concave surface features, some part of the volume mesh needs to be removed in order to extract the original surface. This can be done by checking the Hausdorff distance from each triangle to the boundary point set of voxel-based object.

Hausdorff distance is defined as the maximum of the minimum distances from one set of points to another set of points. The following equation defines the Hausdorff distance h from the point set E to the point set S :

$$h(E,S) = \max_{e \in E} \min_{s \in S} \parallel e - s \parallel \tag{4.1}$$

For a surface mesh all the edges should be conformal to the surface within a certain error. To get the surface mesh of a concave object, we need to check the Hausdorff distance from each triangle to the boundary of the object. If the distance is within a certain allowed range, the triangle is considered as part of the surface mesh, otherwise it is removed. In the end, only triangles conformal to the original surface will remain. And this will constitute the final surface mesh. The procedure is like sculpting.

Another way of connecting nodes is to use the advancing front approach. Since nodes are already distributed into optimal locations, it will be easy to connect the nodes into mesh one by one. Each time when forming a new triangle, try to select an optimal node and make the new triangle conformal to the boundary surface of the object.

4.2 Resolving Surface Features

4.2.1 Energy Function for Simple Convex Objects

For simple convex objects such as sphere, cylinder and rectangular prism, a simple energy function as the following is enough for a good surface meshing.

$Object$	$Angle_{max}$	$Angle_{min}$	$AspectRatio_{max}$
Rect. prism	112.6^o	24.1^o	3.00
Cylinder	114.8^o	27.6^o	3.13
Sphere	110.3^o	27.0^o	2.97

Table 4.1: Characteristics of example meshes.

$$\varphi = \sum_i \sum_{j>i} \phi(r_{ij})$$
$$\phi(r_{ij}) = \left(\frac{c}{r_{ij}}\right)^2 \tag{4.2}$$

where r_{ij} is the distance between the particles i and j, and c is a constant. To make the change of $\phi(r_{ij})$ be sensitive to r_{ij}, one can choose a big c, such as greater than the cutoff distance for particle interaction. The above pair potential decreases with the increase of the distance between the particles.

Figure 4.1 shows the triangular meshes connected through nodes (particles) distributed with the Monte Carlo method on the surfaces of a rectangular prism, a sphere and a cylinder after 500 sweeps. The number of nodes on each surface is 300.

The meshes are in fairly good quality. The maximum angle, minimum angle and the maximum aspect ratio are listed in table 4.1.

We can see from the pictures that the important surface features such as corners and edges are preserved. In fact, the most important points such as corners can usually be located no matter how many particles are put into the system. In an extreme example, we can reconstruct the surface of the rectangular prism with only eight particles, which are distributed by Monte Carlo simulation right at the eight

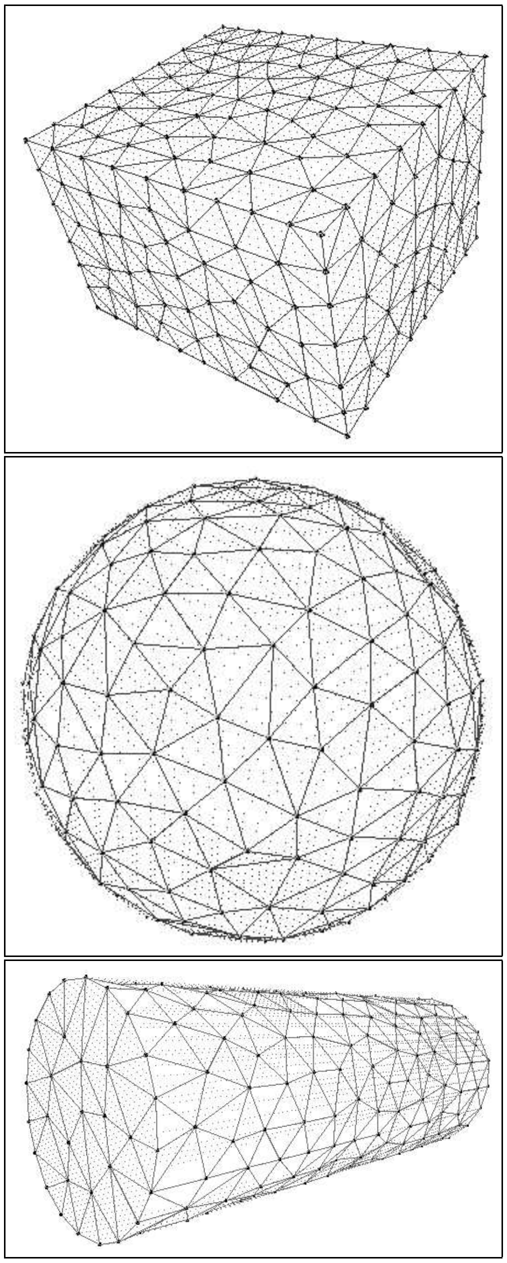


Figure 4.1: Surface meshing for simple convex objects.

corners of the object.

Figure 4.2 shows surface meshes of the rectangular prism constructed with 8, 20, and 500 particles. This implies that multi-resolution representations can be constructed for the same object, and a volumetric dataset can be simplified with the important feature points preserved.

However for concave objects, the energy function (4.2) is no longer good. With this function, the particles won't be able to locate edges or corners of surface cavities. To address this problem, we need to detect edges and corners, and reflect these features in the system energy function.

4.2.2 Capturing Edges with Incomplete Laplace Iterator

Consider a 3D scalar field such as temperature, electric potential etc, given on a grid of voxels and governed by the Laplace equation:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = 0 \tag{4.3}$$

with a Dirichlet boundary condition $u = 0$. And the initial values are 1 for all voxels occupied by the object and 0 for all voxels representing the void space. This equation can be discretized using central differencing, and then solved numerically with simple Jacobi iterations:

$$u_{i,j,k}^{n+1} = \sum_{q=\{-1,1\}} \frac{u_{i+q,j,k}^{n} + u_{i,j+q,k}^{n} + u_{i,j,k+q}^{n}}{6} \tag{4.4}$$

Please note we are not aiming to obtain the equilibrium solution. The purpose is to produce two iso-surfaces, which can be used to identify the edges. Hence, after a

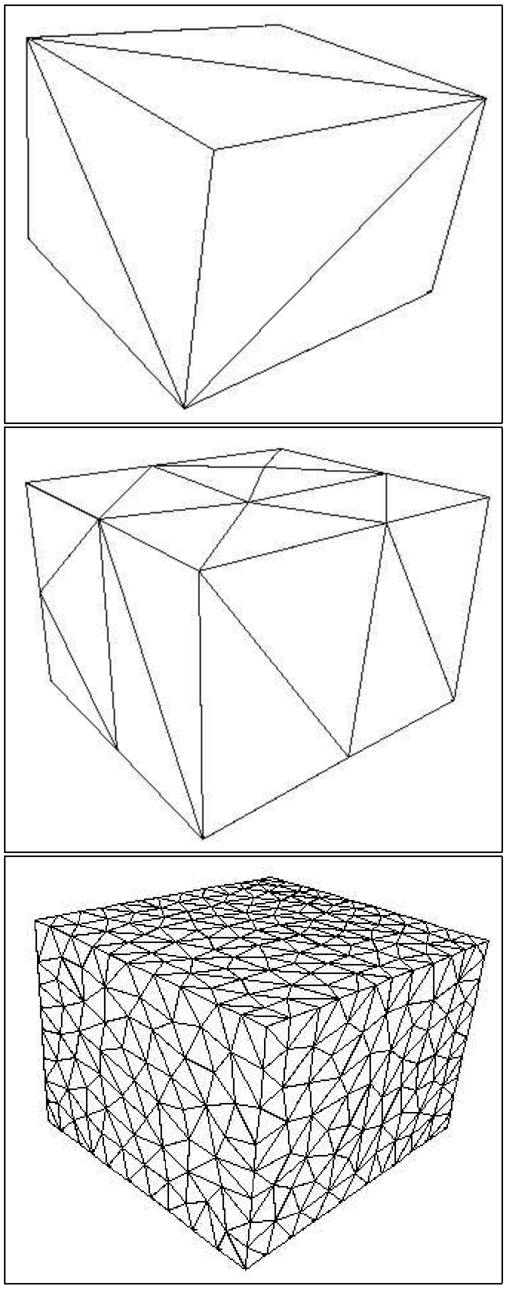


Figure 4.2: Multi-resolution meshes of the same object.

couple of Jacobi iterations, the "temperature" u for voxels on the object surface will change from 1 to some lower values. Based on the maximum (u_{max}) and minimum (u_{min}) values on the surface, we pick two isotemperature surfaces whose temperatures are $u_{max}+\epsilon_1$ and $u_{min}-\epsilon_2$, where ϵ_1 and ϵ_2 are some small positive numbers chosen to satisfy $u_{max} < u_{max}+\epsilon_1 < 1.0$ and $0.0 < u_{min}-\epsilon_2 < u_{min}$. The isotemperature surface with the bigger temperature will be inside of the object , and the isotemperature surface with the smaller temperature will be outside of the object. The shapes of the isotemperature surfaces are similar to the shape of the voxel-based object but tend to smooth out the edges and corners.

For a convex edge, the shortest distance h_1 from a point on the edge to the outside isotemperature surface is smaller than that from a point on a flat place; while the shortest distance h_2 from a point on the edge to the inside isotemperature surface is larger than that from a point on a flat place. For a concave edge, the opposite is true, i.e. shortest distance h_1 from a point on the edge to the outside isotemperature surface is larger than that from a point on a flat place; while the shortest distance h_2 from a point on the edge to the inside isotemperature surface is smaller than that from a point on a flat place. As can be seen the asymmetry between h_1 an h_2 will naturally peak around edges, corners and other surface derivative singularities. This is a natural consequences of Laplace operator producing a smooth solution, which will be in contrast with any irregularities of the original surface. Based on this observation, we can customize the energy function, so that the edge locations are favored for particle distribution:

$$\begin{aligned} \varphi &= \sum_i \sum_{j>i} \phi(r_{ij}) + \sum_i \phi_E \\ \phi(r_{ij}) &= \left(\frac{c_1}{r_{ij}}\right)^2 \\ \phi_E &= -c_2 \frac{(h_1+h_2)^2}{h_1 h_2} \end{aligned} \tag{4.5}$$

The term ϕ_E is a negative potential caused by edge effects. For a point on an edge, this term is smaller than that a point on a flat place. To minimize system energy, the particles will tend to move to locations on edges.

Figure 4.3 shows particle distribution on the surfaces of an object, which is a cylinder overlaying a rectangular prism, and an U-shape object. The first object has 4 concave edges which are the intersections of the cylinder and the rectangular prism. The second object has 2 concave edges. As we can see, beside the convex edges and convex corners, concave edges are also nicely detected by the particles. However the two end points of a concave edge are not captured. The reason is that these corners represent saddle points on the surface, which have both concave and convex features, and this confuses the Laplace detection scheme. However, the problem can be solved using another technique, as discussed below.

4.2.3 Capturing corners with corner detection

Although convex corners are well captured with the above energy function, saddle points can still be missed. We can address this problem by first detecting corners and then customizing the energy function so that corners are favorite places for particles

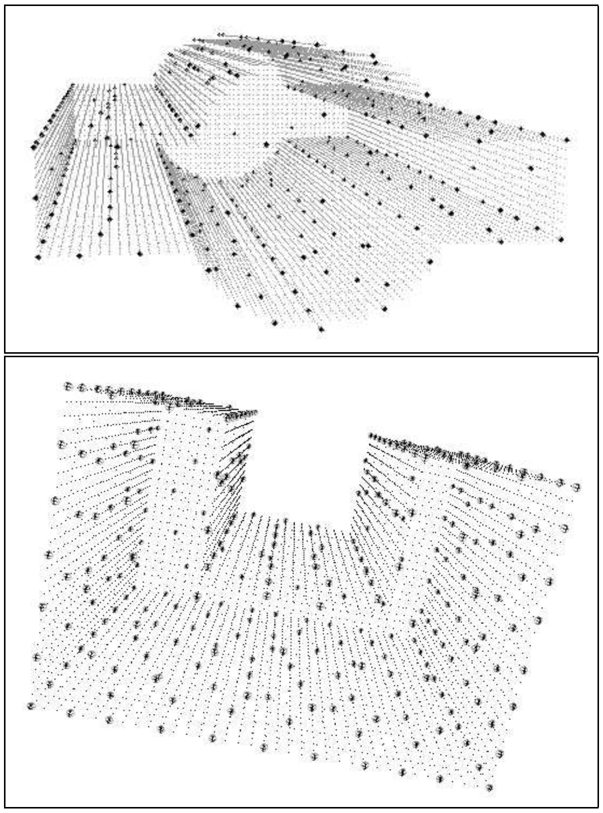


Figure 4.3: Node distributions on concave object surfaces with edges located.

to move to.

A corner detection algorithm in [67] can be used for this purpose. In this case a matrix C is constructed for each voxel point with the gradients I_x, I_y and I_z, which are computed with the central differencing scheme.

$$C = \begin{vmatrix} \sum I_x^2 & \sum I_x I_y & \sum I_x I_z \\ \sum I_y I_x & \sum I_y^2 & \sum I_y I_z \\ \sum I_z I_x & \sum I_z I_y & \sum I_z^2 \end{vmatrix} \tag{4.6}$$

where the summations are taken over a $3 \times 3 \times 3$ neighborhood of the voxel. To

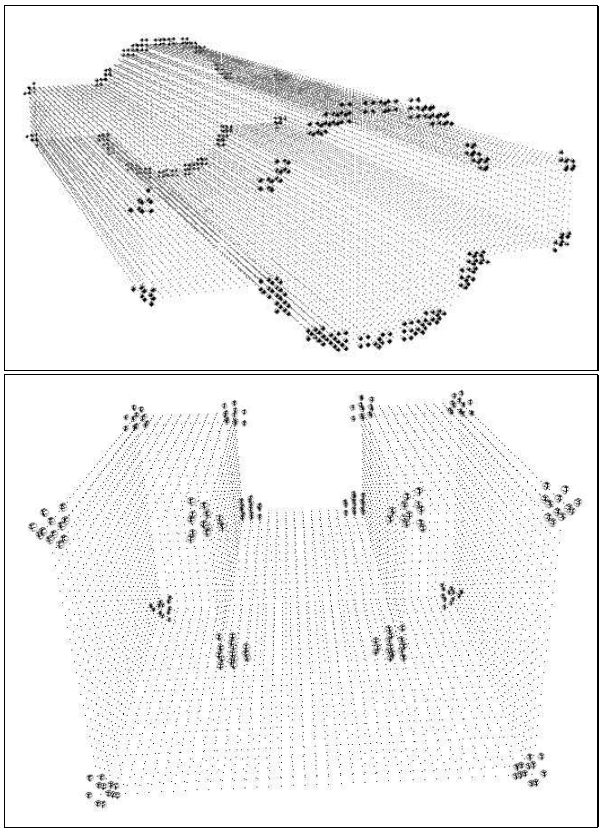


Figure 4.4: Corner voxels are detected with a corner detection algorithm.

detect corners, we first solve for the eigenvalues λ_1, λ_2 and λ_3 of the symmetric matrix C. Assume λ_3 is the smallest. If λ_3 is greater than a threshold, than the voxel is considered as a corner.

This procedure can lead to more than one voxel around a corner being marked. 4.4 shows the corner detection results with a threshold of 0.4. All real corner locations are detected. Because of the discretized voxel-based modeling approach, some unwanted places are also marked as corners.

Now, to make corners favorable in node distribution, we should customize the energy function such that the system potential will decrease when a node is moving closer to a corner. As shown in the following, an negative potential term ϕ_C is added to achieve this effect.

$$\varphi = \sum_i \sum_{j>i} \phi(r_{ij}) + \sum_i \phi_E + \sum_i \phi_C$$
$$\phi_C = -c_3 \sum_k (\frac{\lambda_3}{h_C})_k \tag{4.7}$$

where the term ϕ_C represents a negative potential caused by corner voxels. For each particle i we search in a certain cutoff distance for detected corner voxels and thus find out the distance $(h_C)_k$ from the particle to a corner voxel k. Factor λ_3 represents the corner strength of a corner voxel. The greater the λ_3, the more attraction force from that voxel, and as a result of this, particles will tend to move to corner locations to achieve energy minimization.

Figure 4.5 shows the resulting surface mesh of concave objects after adding the corner potential term. As shown on the pictures, the concave corners are not exactly located, but very close with only some small error. Also the surface meshes are in fairly good quality. The smallest angle is 23.2^o, the largest angle is 116.6^o, and the largest aspect ratio is 3.42.

4.3 Discussion

The results of this study show that the choice of the energy function is important. The energy functions proposed here produce fairly good results for a large class of shapes, including convex and concave objects. For more complex objects such as

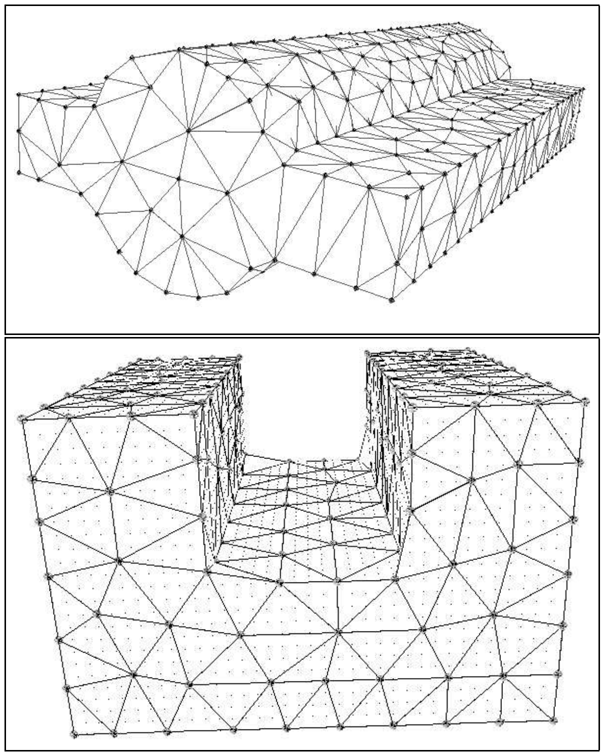


Figure 4.5: Surface meshing for concave objects.

those with arbitrary varying surface curvatures, more testing may need to be done.

Generally more nodes need to be located in the regions of high curvature than in those with lower curvature. The methods proposed above will naturally create such node distributions in most cases. However, one may eventually introduce a mesh size parameter, which can be used to control the node distributions and be adaptive to the surface curvature. One way to introduce the mesh size parameter σ into the energy function is to use Lennard-Jones 12-6 potential as the pair potential:

$$\phi(r) = 4\varepsilon \left[(\frac{\sigma}{r})^{12} - (\frac{\sigma}{r})^{6} \right] \tag{4.8}$$

In order to make mesh size be adaptive to the surface curvature, curvature needs to be evaluated at very place. This can be done by applying the least square fit to a local set of points on a surface patch and then evaluating curvature of the surface patch (see Chapter 2). However, this can make the Monte Carlo approach less efficient.

Another choice is to embed into the energy function the Hausdorff distance from the edge of a pair of particles to the surface of the voxel-based object. Only particle pairs whose Hausdorff distances are within a certain small range are allowed to interact with each other. By including Hausdorff distance into the pair potential, we actually take into account of surface curvature indirectly. For places with higher curvature, particles will tend to be more concentrated since they have to be closer before there are interactions.

4.4 Conclusion

The meshing approach via Monte Carlo simulation proposed in Chapter 3 is applied to surface meshing of voxel-based objects. Its simplicity and versatility is demonstrated in this study. The generated surface meshes preserve important surface features such as edges and sharp corners as present in the volumetric model. With suitable energy function, it is possible to generate meshes which are adaptive to surface curvatures.

This new meshing approach is flexible, and can be used for reduction of volumetric datasets and constructing of mulit-resolution represenions of the underlying objects.

The same meshing approach via Monte Carlo simulation can be used in an identical way for volume meshing. For voxel-based objects, it is trivial to judge whether a particle is inside the volume or not, therefore it is easy to keep particles inside the volume while displacing particles. By taking surface mesh nodes as fixed particles, and placing nodes inside the volume with Monte Carlo simulation, a set of well spaced nodes can be distributed. After that, volume mesh can be generated by connecting the set of nodes.

A new approach of capturing surface edges is proposed and demonstrated. The approach is based on a Laplace solver with incomplete Jacobi iterations, and as such is very simple and efficient.

Chapter 5

User Interface for Fuel-Cell Simulation and Voxel-Based Modeling

This chapter describes a graphical user interface (GUI) which is developed for a fuel-cell simulator program and used in simulations of large fuel-cell stacks. Built with Java technology, the GUI enables one to setup, run and monitor simulations remotely through secure shell (SSH2) connections. The GUI can be either run as a normal local application or invoked from a web-browser using Java webstart technology.

A voxel-based 3D geometrical modeling module is built along with the GUI. The voxel-based objects used in previous chapters are all created by this modeling module. The geometric design module is implemented using 3D voxel sculpting methodology, which is prototyped after 2D pixel graphics systems. The developed approach was primarily aimed at the design of complex multi-component engineering systems. However, the flexibility of voxel-based geometry representation enables one to use this technique for both 3D geometric design and visualization of volume data. Examples of both applications are presented, with the focus on fuel-cell stack simulations.

5.1 Introduction

Distributed memory computer platforms, such as Beowulf clusters are increasingly used for complex scientific simulations of physical processes and engineering systems. Fuel cells offer a way of using the capabilities of distributed processing for efficient simulation of single fuel cells and fuel cell stacks. The modularity of fuel-cell stacks can be exploited on computer clusters by running the simulation of each fuel cell on a separate processor. In earlier work the results of simulations of fuel cell stacks using continuum solvers and distributed simulation techniques was reported in [11]. In this study we focus on the issues of efficient simulation control on remote clusters and modeling of a single fuel cell as a multi-component system.

Until recently it was common to consider two basic geometries for fuel-cells: tubular and planar. Currently we witness a proliferation of various designs aimed at increased efficiency and power density. But even in the domain of simple planar designs there are multitudes of configurations of different components, such as anode, cathode, electrolyte, air/fuel channels, interconnect, separator plates, seals, current collectors, etc. Each component is typically represented by it's own physical model. Many geometrical designs are employed, resulting in co-flow, counter-flow and cross-flow configurations [51]. Consequently, there are two issues that arise in the design of these complex multi-component, multi-physics systems: geometric design and physical modeling. This chapter gives a brief outline of the basic principles of general physical modeling used in our fuel-cell simulations, but is primarily concerned with geometric design of a single fuel cell. In particular, for typical fuel cells configurations we found that the design can be simplified by adopting a relatively straightforward method of *voxel sculpting* [13, 71, 25].

Another aspect of simulating fuel cell stacks concerns simulation control on a remote cluster. The simulation solver has to be specifically implemented for execution on a distributed memory system, using domain decomposition techniques and message-passing interfaces (MPI, PVM). After such solver has been implemented, to perform a simulation one has to go through the stages of setup, execution, data processing and visualization. All the stages face challenges associated with the distributed nature of computations, especially when geometrically complex 3D systems are involved. The task becomes extra difficult when the cluster has to be accessed through the Internet from a remote workstation. In this case the user of the cluster would greatly benefit from an accessibility to a graphical user interface (GUI), which could provide for remote control of the simulation. In this study we developed such a GUI, and used it in simulations of fuel-cell stacks on Beowulf clusters. The GUI performed functions of (1) simulation setup, including complex 3D geometric design, (2) monitoring and runtime control of the simulation, and (3) distributed data sampling and visualization.

5.2 Method

5.2.1 Client - Server Model

In order to effectively monitor and control the execution of a parallel application running on a remote cluster we use a client-server model (Fig.5.1). In this approach

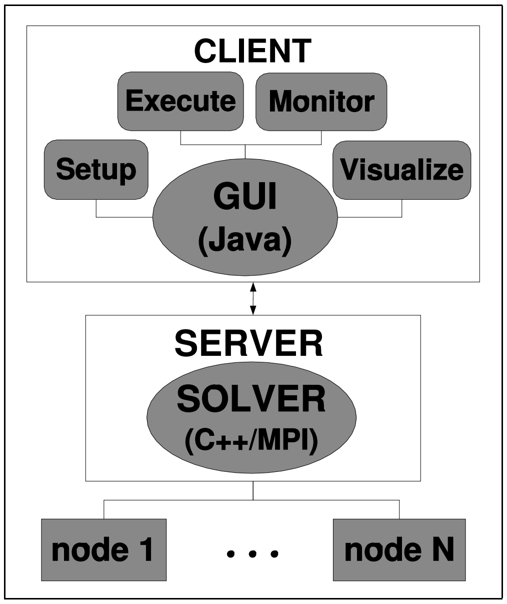


Figure 5.1: Simulation setup with a remote GUI control based on a client-server model.

a client is run on a local workstation to connect a server process running on a remote cluster. The client process enables the user to setup and remotely control the simulation, as well as retrieve and visualize data samples.

Through the server process, the client can control the execution of a parallel solver running on the cluster. In the simplest version the information can be exchanged through files. Files containing user requests are created by the client and periodically sent to the server, and server side information is periodically written into files and read by the client.

5.2.2 Modeling Framework

Client and server exchange information on parameters, variables and domains, which represent generic data types used by most continuum and discrete dynamics solvers.

Each parameter stores a single value attributed to the given model or to the simulation as a whole. Examples of parameters are total current through the system, ambient temperature, number of processors, data sampling interval, etc.

Variables are represented by a set of multi-dimensional values (scalars, vectors, etc.) with each element of the set attributed to one element of the domain. For example, distributions of temperature, current, chemical species, etc.

Domain is a connected region of space assigned to a specific physical model. For the purpose of numerical integration each domain is discretized by decomposing it into smaller and geometrically simple regions (elements), where physical laws are considered to be homogeneous and isotropic. The group of connected elements represents a grid, which can be of a structured type (global connectivity information) or unstructured type (local connectivity), which is also called *mesh.* Thus, a domain consists of a mesh, a set of variables and the solution procedure. The introduction of the domain data class provides the basis for muti-physics simulations, where different physical models can be assigned to the different regions of space.

5.2.3 Graphical User Interface

The graphical user interface (GUI) is implemented using Java Swing with the funtionalities of a secure shell (SSH2) client. The purpose of the local client process is to initiate the following actions through the user interface:

1. Problem setup on a local workstation.

2. Transfer of the data and source-code files to the remote cluster.

3. Building of the application executables and input files on the cluster.

4. Submitting the remote application for execution.

5. Monitoring of the remote run.

6. Sampling of data from cluster nodes.

7. Terminating the run.

8. Collecting the data.

9. Visualizing

After the physical model has been setup the client can initiate the transfer of necessary files to the cluster and schedule the simulation for execution. Once started, the simulation can be monitored by periodic data sampling from the cluster nodes and displaying them in numerical or graphical format. The data sampling strategy is set from the considerations of bandwidth and problem size. One dimensional (vector) data can be displayed as 2D plots.

Simulation parameters represent the input data of the problem, which are not affected by the simulation, such as initial/boundary conditions, number of processors, the duration of the run etc. Almost all the simulation parameters can be changed dynamically during program execution. This enables one to change simulation conditions in real-time.

Table 5.1 displays the list of some parameters used to control the sampling sizes and frequencies as well as several physical parameters of a fuel-cell model. Some fields in the parameter table can be set by the user, and others are fixed. Each parameter is identified by several properties. *Scope* determines if the parameter represents a variable, defined on the nodes of computational mesh, such as temperature or concentration, or a single value valid for all the simulation, such as total current or the ambient temperature. Parameters which belong to parameter-scope are not modified by the solver, and can be changed by the user during the simulation. Parameters of the variable-scope are subdivided into *control variables* and *variables*. Variables are modified by the solver during the run, and thus can only be set as initial parameters of the simulation, whereas the control variables can be set by the user during the run. The type of the parameter identifies it's numerical representation as an integer or a real number. The parameter *dimension* identifies it as a scalar (0), a vector (1), or a general n-rank tensor (n). The *value* and *monitor* fields are set by the user, where the latter indicates if the parameter's values will be monitored during the run.

It should be noted that the flexibility of setting up the control parameters enables one not only to start/stop the execution but also to change model parameters during the simulation, i.e. ambient temperature, total current, etc.

In addition to providing visualization capabilities, remote data monitoring, and control of the simulation, the interface essentially hides from the user the intricacies of the underlying operating system running on the cluster. Some of the interface menu functions can in fact be developer-defined. Thus, it is possible for the code developer to assign different Unix-type commands for the user to execute on the cluster without requiring proficiency with Unix. These commands can be changed or implemented without the need to recompile the interface executable itself.

Name	Scope	Type	Dim	Value	Monitor
NP	parameter	int	0	10	false
MonitorPlane	parameter	int	0	10	false
TotCurrent	parameter	real	0	600.0	false
TemperatureAmb	parameter	real	0	1250.0	false
StopTime	parameter	real	0	9000.0	false
PrintCntStep	parameter	int	0	1e9	false
PrintTimeStep	parameter	real	0	100.0	false
CathodeInletVel	parameter	real	0	1.214	false
AnodeInletVel	parameter	real	0	0.407	false
CathodeInletT	parameter	real	0	1073.0	false
AnodeInletT	parameter	real	0	1073.0	false
TimeStep	controlvar	real	0	0.0	false
TemperaturePEN	variable	real	0	1200.0	true
TemperatureAir	variable	real	0	1200.0	true
TemperatureFuel	variable	real	0	1200.0	true
TemperatureSep	variable	real	0	1200.0	true
TemperatureTop	variable	real	0	1200.0	true
CurrentDensity	variable	real	0	0.0	true

Table 5.1: Simulation control parameters for a fuel-cell application

5.2.4 Voxel-Based Geometric Modeling

While vector graphics still dominates engineering CAD applications, voxel-based volume graphics are getting more attention with the advancement in hardware, especially cheaper and larger memories. Vector graphics describe geometrical shapes using mathematical expressions. This approach is simple, flexible, and good enough at representing simple, well defined shapes. However, it has some drawbacks, such as the inability to represent well 3D objects of complex geometry. On the other hand, volume graphics, which uses pixel-like volume elements - *voxels*, has the capability to easily define arbitrary shapes, such as biological and geographical structures.

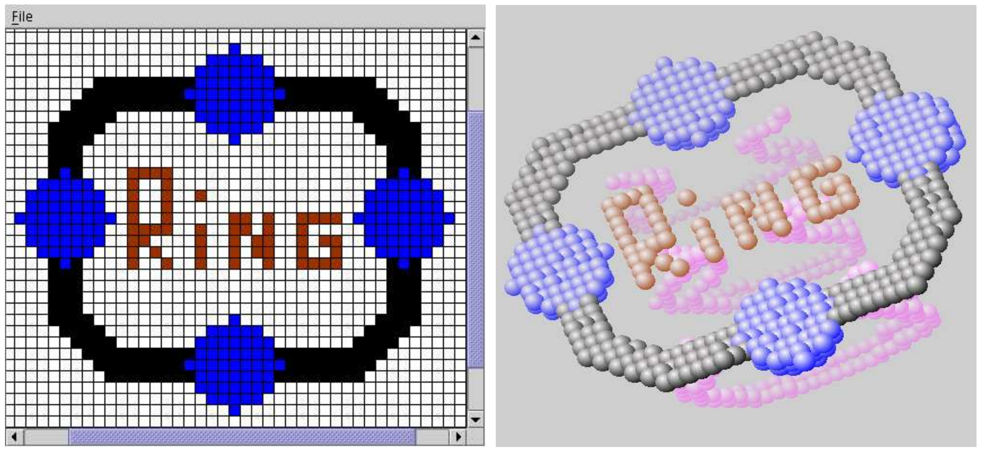
Built along with the GUI is a voxel-based 3D modeling and rendering model. The modeling methodology is a 3D counterpart of 2D pixel graphics. Complex 3D shapes can be rendered in different modes, such as points, wireframes and surfaces. The package is written in Java language and used in simulations of fuel cell stacks on Beowulf clusters.

One disadvantage to voxel-based approach is seemingly inefficient usage of space, which has to be uniformly filled by the voxels. However, the very uniformity of voxel distribution opens the possibility to use efficient compression algorithms. With today's cheaper and larger memories, it is possible to use the simple and robust voxle graphics techniques for engineering design and scientific applications that involve dynamic 3D geometries and complex scene transformations. In applications to fuel cells design the approach offers a simple technique for geometric design of these multi-component systems.

3D Drawing

The volume graphics drawing methodology used is a direct extension of 2D graphics packages, such as Xpaint. While a 2D drawing function operates on pixels in a plane, a 3D drawing function needs to operate on voxels in a three dimensional space. A simple technique to extend 2D pixel-drawing to 3D is to apply an extrusion operation in the third direction. This extension is relatively straightforward, and can be easily implemented on top of existing 2D drawing routines.

In essence, each 2D drawing tool can have its 3D counterpart, with a third spatial dimension added to the tool. For example, the drawing pen can be extended into 3D as a ball with a certain radius selected by users. And in simple cases, many objects such as cubes and cylinders, can be modeled by simply extruding images



(a) 2D drawing plane (b) 3D representation

Figure 5.2: Voxel sculpting of 3D shapes.

drawn on a 2D plane into the third dimension, analogously to the operation done in conventional CAD applications. This approach is adequate for the purposes of designing many engineering systems, and was used in simulations of fuel cell stacks on Beowulf clusters.

A more general approach such as *voxel-sculpting* [25, 71] can be pursued with introduction of 3D sculpting tools. Figure 5.2 shows the example of voxel-based sculpting of arbitrary 3D shapes.

To navigate in the drawing cube, it is important to have the capability of identifying the position and orientation. In a simplified case 3D Cartesian coordinates are enough for this purpose. And the drawing planes can be in 3 different orientations with respect to Cartesian coordinates.

Introducing 3D controls can be as simple as changing the position and orientation of the drawing plane. However, to alleviate the frustration of dealing with hundreds of drawing planes in case of high-resolution 3D scenes, more advanced 3D drawing tools and motions controls should be introduced. Since there are three parameters required for tool-positioning operation, it can be done, using only mouse controls. For example, one can use two mouse-position coordinates to set the direction vector, and mouse-wheel to move the tool along that direction. More sophisticated 3D navigation tools can also be developed using prototype controls of a flight-simulator application.

3D Surface Rendering

3D visualization of the drawn scene is the key to successful drawing capabilities. An almost trivial feature in 2D graphics, visualization and surface rendering become the major efforts in 3D. For most purposes of engineering design a simple wireframe rendering mode is usually enough. This can be accomplished in a number of ways, and in a manner consistent with the resolution of the image, i.e, the ratio of the image size to the grid-cell size. The advantages of wireframe rendering are that it is relatively simple to handle algorithmically, and sufficiently fast to work well even without accelerated graphics. It also provides one with the depth-perspective (Fig.5.3).

Three different wireframe rendering modes were implemented in the current system. The first mode is based on a grid-type wireframe, where each voxel is represented as a cube with 12 edges. Only the surface edges of the object are actually rendered (Fig.5.3(a)). This wireframe type offers the most detailed rendering mode. However, it introduces unnecessary edginess into the surface carried over from the three dimensional voxel-grid. In addition to this the amount of detail is excessive for many high-resolution scenes.

The second wireframe type is based on a relatively versatile and fast method of constructing cutting-plane contours. The number of planes, their orientations and separations can be set by the user, thus, adjusting the rendering to high vs. low resolution scenes (Fig.5.3(b),5.3(c)).

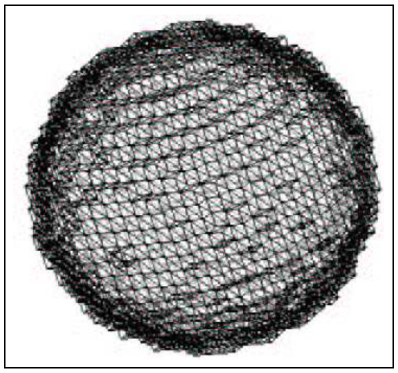
The third rendering mode is based on surface mesh representation. In this representation the surface mesh preserves only the characteristic features of the surface and skips the non-essential information on the 3 directions of the underlying voxel-grid. There are different ways to extract such mesh representation from the voxel data. For example, the marching cube algorithm [45] can be used for this purpose.

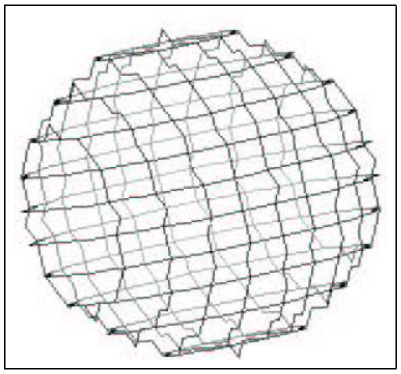
In this thesis, two different approaches for surface meshing from voxel data have been proposed in the previous chapters. It should be noted that such polygon surface mesh representation is more compact than the original voxel-representation, which is important when the graphical information should be transferred over slow networks.

5.3 Simulation of fuel cell stacks

The methodology of integrated simulation setup and control based on voxel-graphics and Java-technology was applied to simulations of fuel cell stacks on Beowulf clusters. In this case the geometric design of a fuel cell is done on a local workstation by means of voxel-based graphics tools implemented in Java. The geometric information and the setup parameters are then transferred to the cluster. After the simulation is started it can be monitored from on the workstation by periodically retrieving data samples and displaying them in graphics format.

Figure 5.2 shows the example of voxel-based sculpting of arbitrary 3D shapes. Application of this voxel-based sculpting to a cross-flow fuel-cell geometry is shown


(a) Grid-wireframe


(b) 2-directional contour-wireframe

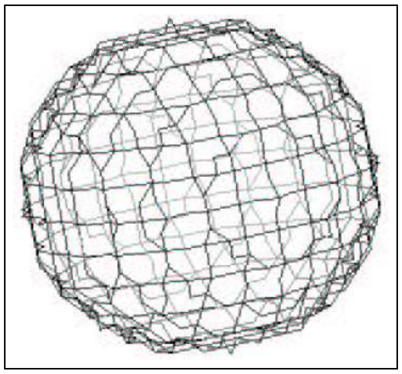
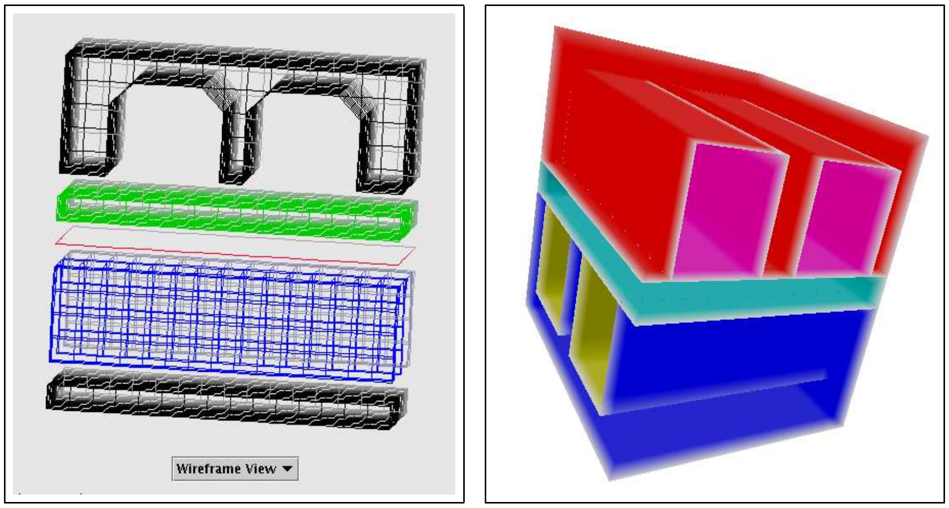

(c) 3-directional contour-wireframe

Figure 5.3: Different wireframe rendering modes.

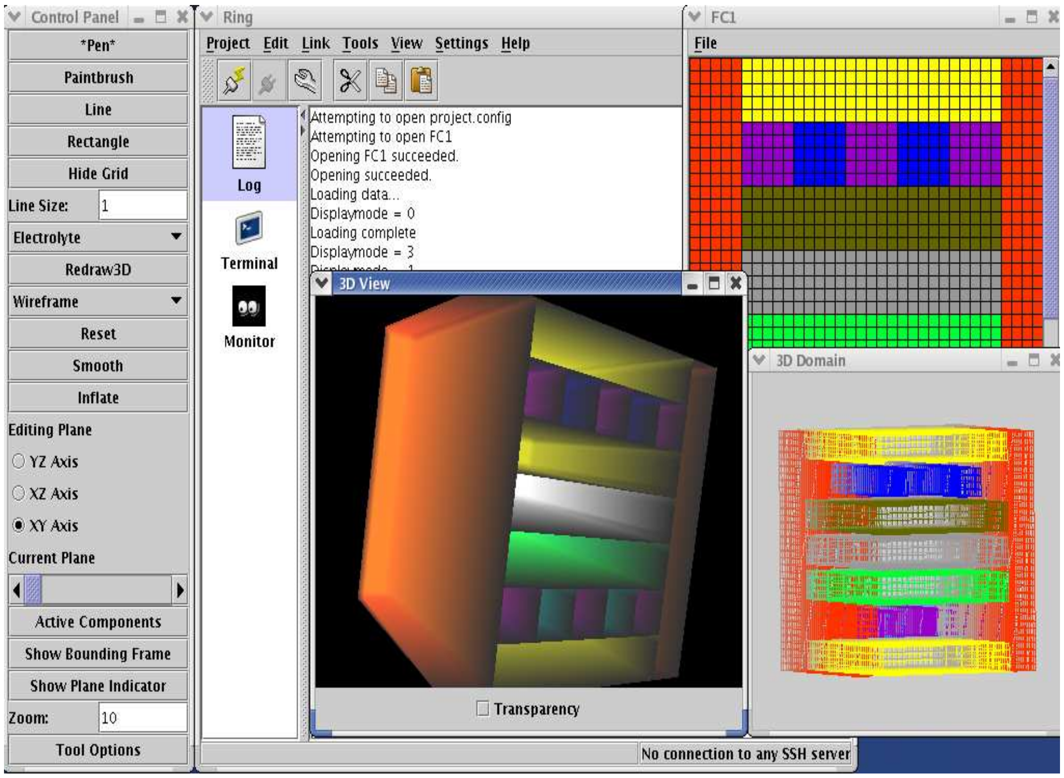


(a) Wireframe representation (b) Surface representation

Figure 5.4: Geometric design of fuel cells: surface representations.

in Fig.5.4. The geometry can be displayed either in wireframe representation or using surface rendering.

A screen-shot of the GUI is shown in Fig.5.5 where the main components, such as the main panel, the control panel, the 3D view panel, the 2D drawing pane, and the 3D wireframe view are displayed. Figure 5.6 shows the screenshot of the monitoring window where the transient temperature retrieved from the remote cluster nodes are displayed in a graphical format. Considering small time-steps that are required for electrical and chemical sub-models of the solver, such simulation may take large computer resources in terms of time and memory. Thus it is important to realize a flexible system of simulation control which enables one to adjust parameters during the run. Thus, in Fig.5.6 thermal responses to changes in total current are observed. It should also be noted that large fuel cell stacks may exhibit unexpected temperature

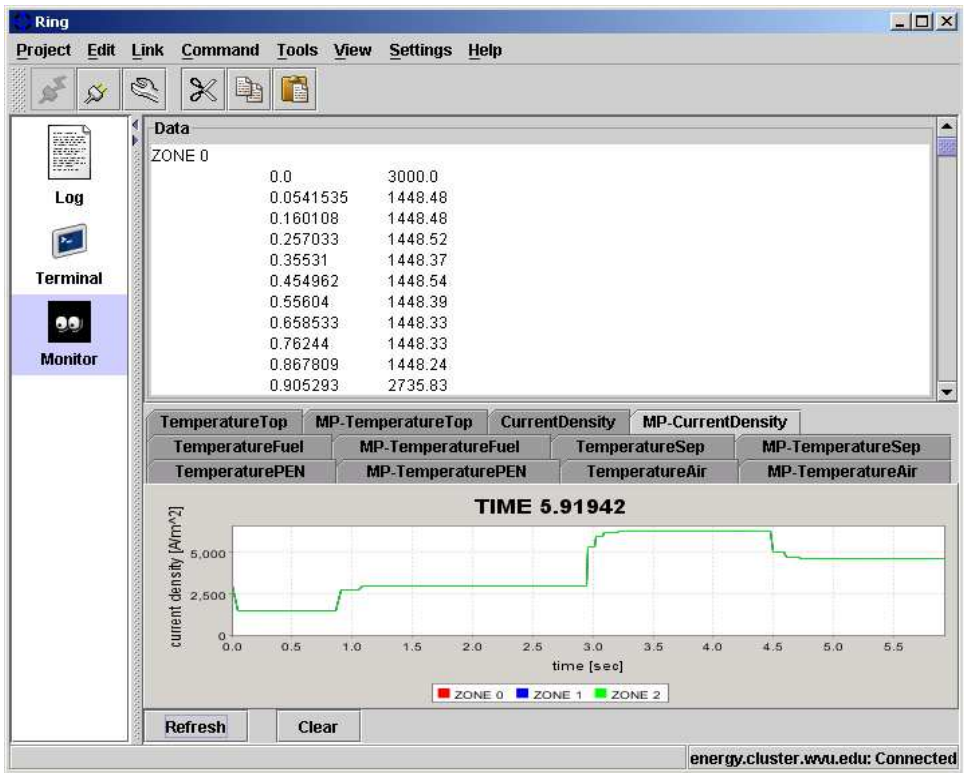


Line Size: 1
Electrolyte
Wireframe
YZ Axis
XZ Axis
[✓ XY Axis]
Zoom: 10

```
Attempting to open project.config
Attempting to open FC1
Opening FC1 succeeded.
Opening succeeded.
Loading data...
Displaymode = 0
Loading complete
Displaymode = 3
```

No connection to any SSH server

Figure 5.5: GUI control and monitoring windows: Control panel (left), 3D view panel (Middle bottom), 2D drawing pane (Right top), 3D wireframe view (Right bottom), Main panel, (in the back)

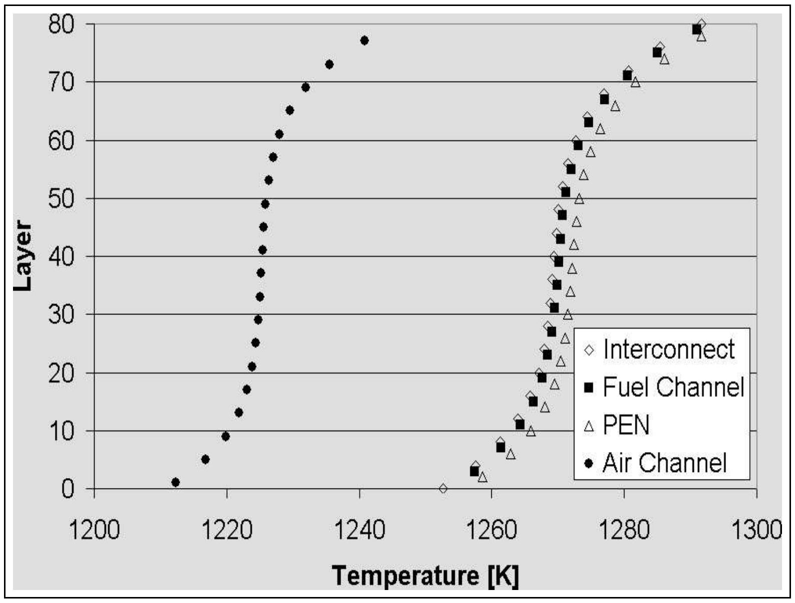


```
ZONE 0
            0.0          3000.0
            0.0541535    1448.48
            0.160108     1448.48
            0.257033     1448.52
            0.35531      1448.37
            0.454962     1448.54
            0.55604      1448.39
            0.658533     1448.33
            0.76244      1448.33
            0.867809     1448.24
            0.905293     2735.83
```

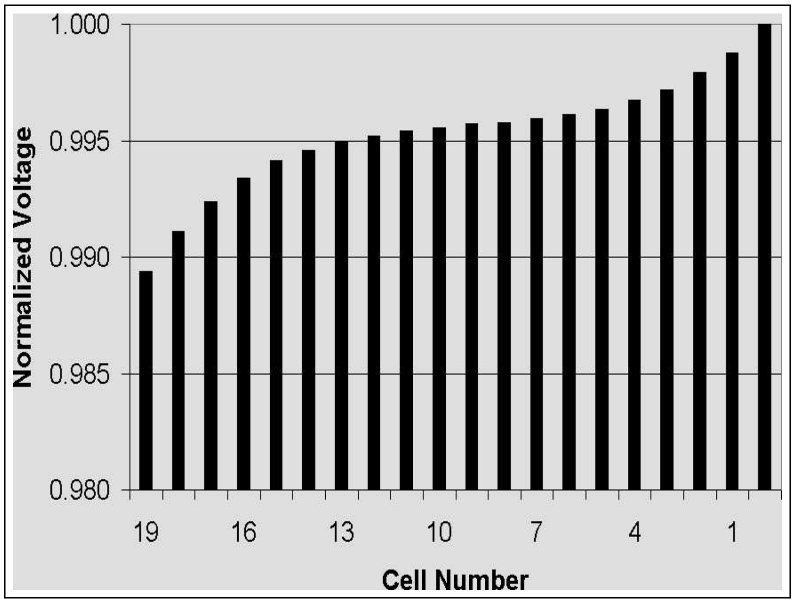
Figure 5.6: Remote monitoring of transient temperatures.

and voltage variations, depending on the performance of separate cells. The developed system can be effectively used to simulate various stack operation scenarios, where the failure of one of several cells may affect the overall stack performance.

This system of remote setup and monitoring was successfully applied to the simulation of large stacks of up to 40 solid oxide fuel cells [11, 9, 10]. Figure 5.7 shows sample distributions of temperature and voltage within a 20-cell stack. This simulation was done under uniform stack conditions with respect to fuel and oxidizer supply. However, non-uniform variations of temperature and voltage can clearly be observed for the bottom and top group of cells in the stack.



(a) Temperature



(b) Voltage

Figure 5.7: Distributions of physical parameters within the stack.

5.4 Conclusions

An approach to 3D graphics based on voxel-representation was successfully applied to the setup of typical fuel-cell geometries. The approach offers considerable simplicity and flexibility. It also enables one to combine geometric design and data visualization in a single framework.

Making use of Java-technology and a client-server model provides tools for the design of web-based user interfaces for remote control and monitoring of scientific and engineering simulations on Beowulf clusters.

Because of inherent modularity of fuel-cell stacks these systems can be effectively simulated on distributed memory platforms, such as workstation clusters. These simulations can benefit from remote interfaces with graphical capabilities, such as the one developed in this study. An extra advantage of the interface is the flexible control of the simulation, which provides the possibility of playing out different operation scenarios.

Based on the results of this study we conclude that voxel-graphics is a promising technique for applications in grid and cluster computing, related to 3D geometric design and data visualization.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

1. A new approach of unstructured mesh generation based on Monte-Carlo simulation of a particle ensemble in dynamic equilibrium is proposed and demonstrated. In contrast to the earlier approaches the method uses the concept of a liquid state of matter to produce uniform node distributions inside complex domains. The correct number of nodes is automatically selected to satisfy the condition of positive system energy. This approach works for general complex geometries, and in particular, it was applied in this study to meshing of voxel-based objects. Besides the advantages we discussed in Chapter 3, it has many potential applications, such as:

   (a) **Mesh optimization for quality improvement.**

   For an existing mesh, we can use the available element connectivities to define particle interactions. For example, one node can be considered interacting with nodes only connected directly with it. With the MC simulation, nodes can be moved to optimal locations while keeping the element connectivities.

(b) **Moving mesh generation.**

During the Monte Carlo simulation, particles (or nodes) are moved towards more optimal locations after each iteration. For a moving domain, we can always start from the previous node configuration, and get the next configuration with a few additional iterations.

To improve efficiency, particles which are already well distributed and far away from the moving interface can be fixed. And in the region near the moving interface, particles need to be constantly displaced, inserted or removed.

(c) **Anisotropic mesh generation.**

The shape of mesh elements is directly related to the energy function. By customizing energy functions, we are able to control the shape of mesh elements.

The same approach can be used as mesh optimization method for quality improvement of an existing mesh. For an existing mesh, since we know the node connectivity already, particle interaction can be restricted among interconnected nodes. Because of this, the efficiency would be much better than when it is used for mesh generation.

(d) **Node generation for mesh free methods[75].**

In meshless methods [41], problem domains are represented by a set of scattered nodes instead of meshes or elements. And thus comparing to mesh based numerical methods such as FEM, FVM, meshless methods can eliminate the need for mesh generation. Instead, Node generation is needed for the preprocessing of meshless methods. However, the node generation problem has not been paid enough attention to, and there are very few dedicated node generation methods[34, 18, 38]. Ironically, most people obtain node sets for meshless methods from meshes which are generated using mesh generators. With our approach, nodes can be generated directly for mesh free methods. Since no mesh required for mesh free methods, there is no need to connect the nodes after node placement.

2. A new heuristic algorithm to generate triangulated surface meshes on voxel-based objects is proposed and implemented. This approach uses an advancing front approach and enables to generate voxel-grid independent surface representations.

3. A new approach of capturing important surface features, such as edges, corners and other irregularities was proposed and demonstrated. The approach is based on a Laplace solver with incomplete Jacobi iterations and is simple and time-efficient.

4. A graphical user interface (GUI) capable of complex geometric design and remote simulation control was implemented. The GUI was used in simulations of large fuel-cell stacks.

6.2 Future Work

It is appropriate to conduct more study on the potential applications of the new meshing approach, such as its use for mesh optimization, on energy function customization for generation of desired mesh shapes, generation of anisotropic and moving meshes.

More study needs to be done to increase the mesh generation efficiency. One improvement can be achieved by using a combined molecular-dynamics - Monte-Carlo approach, where the direction of node displacement will no longer be determined by a random choice but rather by a direction of the cumulative force acting on the particle. This study is being currently pursued as a continuation of this work[1].

Other possible solutions include using smarter Monte Carlo methods, as well as simulation in small energy systems with the advancing front technique, using parallel computing and grid computing technologies.

[1]http://mulphys.mae.wvu.edu/mesh/mc

Bibliography

[1] M. P. Allen and D. J. Tildesley. *Computer Simulation of Liquids*. Clarendon Press, Oxford, 1987.

[2] Sergei Azernikov, Alex Miropolsky, and Anath Fischer. Surface reconstruction of freeform objects based on multiresolution volumetric method. *Journal of Computing and Information Science in Engineering*, 3:334–338, December 2003.

[3] Peggy L. Baehmann, Scott L. Wittchen, and Mark S. Shephard. Robust geometrically-based,automatic two-dimensional mesh generation. *International Journal for Numerical Methods in Engineering*, 24:10439–1078, 1987.

[4] J. A. Bærentzen and N. J. Christensen. Interactive modelling of shapes using the level-set method. *International Journal of Shape Modelling*, 8(2):79–97, Dec 2002.

[5] Timothy J. Baker. Automatic mesh generation for complex three-dimensional regions using a constrained delaunay triangulation. *Engineering with Computers*, 5:161–175, 1989.

[6] Marshall Bern and David Eppstein. Mesh generation and optimal triangulation. Technical Report P92-00047, Xerox PARC, 1992.

[7] Marshall Bern and Paul Plassmann. Mesh generation. In Jörg Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*. Elsevier Science, 1997.

[8] Houman Borouchaki, Patrick Laug, and Paul-Louis George. Parametric surface meshing using a combined advancing-front generalized delaunay approach. *International Journal for Numerical Methods in Engineering*, 49:233–259, 2000.

[9] A.C. Burt, I.B. Celik, R.S. Gemmen, and A.V. Smirnov. Cell to cell performance variations within a stack. In *Eighth International Symposium on Solid Oxide Fuel Cells (SOFC VIII)*, pages 217–223, Paris, France, 2003.

[10] A.C. Burt, I.B. Celik, R.S. Gemmen, and A.V. Smirnov. Influence of radiative heat transfer on variation of cell voltage within a stack. In *First International Conference on Fuel Cell Science, Engineering, and Technology*, pages 1487–1500, Rochester, NY, 2003.

[11] A.C. Burt, I.B. Celik, R.S. Gemmen, and A.V. Smirnov. A numerical study of cell to cell variations in a SOFC stack. *Journal of Power Sources*, 126:76–87, 2004.

[12] Hao Chen and Jonathan Bishop. Delaunay triangulation for curved surfaces. In *Proceedings of the 6th Interlational Meshing Roundtable*, pages 115–127, 1997.

[13] Hui Chen and Hanqiu Sun. Real-time haptic sculpting in virtual volume space. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 81–88. ACM Press, 2002.

[14] S. W. Cheng, T. K. Dey, H. Edelsbrunner, M. A. Facello, and S. H. Teng. Silver exudation. In *Proceedings of the 15th Symposium on Computational Geometry*, pages 1–13, 1999.

[15] L. Paul Chew. Guaranteed-quality mesh generation for curved surfaces. In *Proceedings of the Ninth Symposium on Computational Geometry*, volume 73, pages 274–280, 1993.

[16] L. Paul Chew. Guaranteed-quality delaunay meshing in 3d (short version). In *Proceedings of the 13th Symposium on Computational Geometry*, pages 391–393, 1997.

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

[18] Y.J. Choi and S.J.Kim. Node generation scheme for the meshfree method by voronoi diagram and weighted bubble packing. In *Fifth U.S. National Congress on Computational Mechanics*, 1999.

[19] J. C. Cuilliere. An adaptive method for the automatic triangulation of 3d parametric surfaces. *Computer-Aided Design*, 30(2):139–149, 1998.

[20] Hartmann E. Blending an implicit with a parametric surface. *Computer Aided Geometric Design*, 12(9):825–835, 1995.

[21] Hartmann E. A marching method for the triangulation of surfaces. *The Visual Computer*, 14:95–108, 1998.

[22] H. Edelsbrunner and N. R. Shah. Incremental topological flipping works for regular triangulations. *Algorithmica*, 15:223–241, 1996.

[23] Herbert Edelsbrunner. *Geometry and Topology for Mesh Generation.* Cambridge University Press, 2001.

[24] Pascal Frey, Benoit Sarter, and Michel Gautherie. Fully automatic mesh generation for 3-d domains based upon voxel sets. *International Journal for Numerical Methods in Engineering*, 37:2735–2753, 1994.

[25] Tinsley A. Galyean and John F. Hughes. Sculpting: an interactive volumetric modeling technique. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 267–274. ACM Press, 1991.

[26] P.L. George, F. Hecht, and E. Saltel. Automatic mesh generator with specified boundary. *Computer Methods in Applied Mechanics and Engineering*, 92:269–288, 1991.

[27] Dave Hale. Atomic images - a method for meshing digital images. In *Proceedings of the 10th Interlational Meshing Roundtable*, pages 185–196, Newport Beach, CA, Oct. 2001.

[28] U. Hartmann and F. Kruggel. A fast algorithm for generating large tetrahedral 3d finite element meshes from magnetic resonance tomograms. In *Proceedings of the IEEE Workshop on Biomedical Image Analysis*, pages 184–192, Santa Barbara, CA, June 1998.

[29] H. Jin and R. I. Tanner. Generation of unstructured tetrahedral meshes by advancing front technique. *International Journal for Numerical Methods in Engineering*, 36:1805–1823, 1993.

[30] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. In *SIGGRAPH 2002, Computer Graphics Proceedings*, pages 339–346. ACM Press / ACM SIGGRAPH, 2002.

[31] K. Kase, Y. Teshima, S. Usami, H. Ohmori, C. Teodosiu, and A. Makinouchi. Volume cad. In I. Fujishiro, K. Mueller, and A. Kaufman, editors, *Volume Graphics 2003 Eurographics / IEEE TCVG Workshop Proceedings*, pages 145 – 150, Tokyo, Japan, 2003.

[32] Arie Kaufman, Daniel Cohen, and Roni Yagel. Volume graphics. *Computer*, 26(7):51–64, 1993.

[33] A.M. Khokhlov. Fully threaded tree algorithms for adaptive mesh refinement fluid dynamic simulations. *Journal of Computational Physics*, 143:519–543, 1998.

[34] O. Klaas and M.S. Shephard. An octree based partition of unity method for three dimensional problems. In *Fifth U.S. National Congress on Computational Mechanics*, 1999.

[35] Patrick M. Knupp. Algebraic mesh quality metrics. *SIAM Journal of Scientific Computing*, 23(1):193–218, 2001.

[36] Leif P. Kobbelt, Mario Botsch, Ulrich Schwanecke, and Hans-Peter Seidel. Feature sensitive surface extraction from volume data. In Eugene Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 57–66. ACM Press / ACM SIGGRAPH, 2001.

[37] T.S. Lau and S.H. Lo. Finite element mesh generation over analytical surfaces. *Computers and Structures*, 59(2):301–309, 1996.

[38] X. Li, S. Teng, and A. Ungor. Point placement for meshless methods using sphere packing and advancing front methods. In *ICES00:International Conference on Computational Engineering Science*, 2000.

[39] Xiangyang Li. *Sliver-Free Three Dimensional Delaunay Mesh Generation*. PhD thesis, Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign, 2000.

[40] Chjan C. Lim, Joseph Nebus, and Syed M. Assad. Monte-carlo and polyhedron-based simulations i: Extremal states of the logarithmic n-body problem on a sphere. *Discrete and Continuous Dynamical Systems - Series B*, 3(3):313–341, 2003.

[41] Gui-Rong Liu. *Mesh Free Methods: Moving Beyond the Finite Element Method*. CRC Press, 2003.

[42] S. H. Lo. Volume discretization into tetrahedra-i. verification and orientation of boundary surfaces. *Computers and Structures*, 39(5):493–500, 1991.

[43] S. H. Lo. Volume discretization into tetrahedra-ii. 3d triangulation by advancing front approach. *Computers and Structures*, 39(5):501–511, 1991.

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

[45] W.E. Lorensen and H.E. Cline. ”marching cubes: a high resolution 3d surface reconstruction algorithm. In *Computer Graphics (Proceedings of SIGGRAPH ’87)*, volume 21:4, pages 163–169, 1987.

[46] Steven J. Owen. A survey of unstructured mesh generation technology. In *Proceedings of the 7th Intl. Meshing Roundtable*, pages 239–267, 1998.

[47] J. Peraire, J. Peiró, and K. Morgan. Adaptive remeshing for three-dimensional compressible flow computations. *J. of Computational Physics*, 103:269–285, 1992.

[48] Per-Olof Persson and Gilbert Strang. A simple mesh generator in matlab. *SIAM Review*, 46(2):329–345, 2000.

[49] Jim Ruppert. A new and simple algorithm for quality 2-dimensional mesh generation. Technical Report UCB/CSD 92/694, University of California at Berkely, Berkely California, 1992.

[50] Jim Ruppert. A delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. Algorithms*, 18(3):548–585, 1995.

[51] EG&G Services, Inc. Parsons, and Science Applications International Corporation. Fuel cell handbook. Fifth edition. Technical Report DOE/NETL-2000/1110, U.S. Department of Energy, Office of Fossil Energy, Federal Energy Technology Center, 2000.

[52] Mark S. Shephard and Marcel K. Georges. Three-dimensional mesh generation by finite octree technique. *International Journal for Numerical Methods in Engineering*, 32:709–749, 1991.

[53] Jonathan R. Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In Ming C. Lin and Dinesh Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of

Lecture Notes in Computer Science, pages 203–222. Springer-Verlag, May 1996. From the First ACM Workshop on Applied Computational Geometry.

[54] Jonathan R. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry: Theory and Applications*, 22:21–74, 2002.

[55] Jonathan R. Shewchuk. What is a good linear element? interpolation, conditioning, and quality measures. In *Proceedings of the 11th Interlational Meshing Roundtable*, pages 115–126, Ithaca, NY, Sept. 2002.

[56] Kenji Shimada and David C. Gossard. Bubble mesh: automated triangular meshing of non-manifold geometry by sphere packing. In *SMA '95: Proceedings of the Third Symposium on Solid Modeling and Applications*, pages 409–419, 1995.

[57] H. Si and K. Gärtner. An algorithm for three-dimensional constrained delaunay tetrahedralizations. In *Proceeding of the Fourth International Conference on Engineering Computational Technology*, Lisbon, Portugal, September 2004.

[58] A. V. Smirnov and H. Zhang. Surface mesh generation on voxel-based objects. In *Proceedings of the 13th Interlational Meshing Roundtable*, pages 291–298, Williamsburg, VA, Sept. 2004.

[59] A.V. Smirnov, A. Burt, H. Zhang, and I. Celik. Component-based modeling of multi-physics systems. In *Proceedings of the 16th IASTED International Conference on MODELLING AND SIMULATION*, volume MS-05, pages 295–300, Cancun, Mexico, May 2005.

[60] A.V. Smirnov, H. Zhang, A. Burt, and I. Celik. Fuelcell simulator interface. *Journal of Power Sources*, 138:187–193, 2004.

[61] A.V. Smirnov, H. Zhang, A. Burt, and I. Celik. Remote interface for geometric design and simulation control. In H.R. Arabina, editor, *The 2004 International Conference on Imaging Science, Systems, and Technology: CISST'04*, pages 241–247, Las Vegas, NV, 2004. CSREA Press.

[62] A.V. Smirnov, H. Zhang, and B. Sowers. Voxel-based volume graphics system for multi-physics modeling. In *The 8th World Multi-Conference on Systemics, Cybernetics and Informatics*, volume 5 of *Computer Science and Engineering*, pages 144–149, Orlando, FL, 2004. International Institute of Informatics and Systemics.

[63] D. Talmor. *Well-Spaced Points for Numerical Methods*. PhD thesis, Carnegie Mellon University, 1997.

[64] C. W. Teng, S. H.and Wong. Unstructured mesh generation: Theory, practice, and perspectives. *International Journal of Computational Geometry and Applications*, 10(3):227–266, 2000.

[65] Joe F. Thompson, Bharat Soni, and Nigel P. Weatherrill. *Handbook of Grid Generation*. CRC Press, 1998.

[66] Joseph R. Tristano, Steven J. Owen, and Scott A. Canann. Advancing front surface mesh generation in parametric space using a riemannian surface definition. In *Proceedings of the 7th Interlational Meshing Roundtable*, 1998.

[67] E. Trucco and A. Verri. *Introductory Techniques for 3D Computer Vision.* Prentice-Hall, 1998.

[68] T. Udeshi. Tetrahedral mesh generation from segmented voxel data. In *Proceedings of the 12th International Meshing Roundtable*, pages 425–436, Santa Fe, NM, Sept. 2003.

[69] T. Udeshi and E. Parker. Memulator: A fast and accurate geometric modeling, visualization and mesh generation tool for 3d mems design and simulation. In *Technical Proceedings of the 2003 Nanotechnology Conference and Trade Show*, volume 2, pages 480 – 483, 2003.

[70] V.Chandru, S.Manohar, and C.E.Prakash. Voxel-based modeling for layered manufacturing. *IEEE Transactions on the computer Graphics and Applications*, pages 42–48, Nov 1995.

[71] Sidney W. Wang and Arie E. Kaufman. Volume sculpting. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 151–ff. ACM Press, 1995.

[72] N. P. Weatherill and O. Hassan. Efficient three-dimensional delaunay triangulation with automatic point creation and imposed boundary constraints. *International Journal for Numerical Methods in Engineering*, 37:2005–2039, 1994.

[73] M.A. Yerry and M.S. Shephard. Three-dimensional mesh generation by modified octree technique. *International Journal for Numerical Methods in Engineering*, 20:1965–1990, 1984.

[74] H. Zhang and A. V. Smirnov. Node distribution for numerical methods with monte carlo simulation. *International Journal of Modelling and Simulation*, 2005. Submitted.

[75] H. Zhang and A. V. Smirnov. Node generation for meshless methods by monte carlo simulation. In *Eighth U.S. National Congress on Computational Mechanics*, Austin, Texas, July 2005. To appear.

[76] H. Zhang and A. V. Smirnov. Node placement for triangular mesh generation by monte carlo simulation. *International Journal for Numerical Methods in Engineering*, 2005. In press.

[77] H. Zhang and A. V. Smirnov. Surface mesh generation for voxel-based objects by energy minimization. In *International Conference on Imaging Science, Systems, and Technology: Computer Graphics (CISST'05)*, pages 55–61, Las Vegas,Nevada, June 2005. CSREA Press.

[78] Hanzhou Zhang and Andrei V. Smirnov. Node distribution for numerical methods with monte carlo simulation. In *Proceedings of the 16th IASTED International Conference on MODELLING AND SIMULATION*, volume MS-05, pages 363–368, Cancun, Mexico, May 2005.

[79] Y. Zhang and C. Bajaj. Adaptive and quality quadrilateral/hexahedral meshing from volumetric imaging data. In *Proceedings of the 13th Interlational Meshing Roundtable*, pages 365–376, Williamsburg, VA, Sept. 2004.

[80] Y. Zhang, C. Bajaj, and B-S. Sohn. Adaptive and quality 3d meshing from imaging data. In *Proceedings of the 8th ACM Symposium on Solid Modeling and Applications*, pages 286–291, Seattle, WA, June 2003.