Physically-Based Mesh Generation:

Automated Triangulation of Surfaces and Volumes via Bubble Packing

by

Kenji Shimada

S.M., Precision Machinery Engineering
University of Tokyo
March, 1985

B.S., Precision Machinery Engineering
University of Tokyo
March, 1983

Submitted to the Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy
in Mechanical Engineering

at the

Massachusetts Institute of Technology

May, 1993

Signature of Author _____
Department of Mechanical Engineering
May 14, 1993

Certified by _____
Professor David C. Gossard
Thesis Supervisor

Accepted by _____
Ain A. Sonin
Chairman, Department Committee

Physically-Based Mesh Generation:

Automated Triangulation of Surfaces and Volumes via Bubble Packing

by

Kenji Shimada

Submitted to the Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Mechanical Engineering.

Abstract

This thesis proposes a new physically-based approach to fully automated 2D/3D triangulation that satisfies the basic requirements of element shape regularity, precise node spacing control, and adaptive remeshing capability. The approach was inspired by shapes found in nature which have topology and geometry similar to ideal meshes. A good 2D example is the pattern assumed by soap bubbles floating in liquid, a repeating hexagonal pattern. Connecting the bubbles' center points produces a mesh of equilateral triangles. To create a graded mesh, a temperature distribution is applied, causing bubbles in the warm area to grow larger, and bubbles in the cold area to shrink. In the proposed method, mesh nodes are modeled as bubbles with repelling/attracting forces based on proximity, much like intermolecular van der Waals forces. A static force-balancing configuration of packed bubbles is then obtained by solving the equations of motion numerically, assuming a point mass at each bubble and the effect of viscous damping. Also implemented to maintain a necessary and sufficient number of bubbles for a given domain is adaptive population control, in which highly overlapping bubbles are automatically removed and lone bubbles in wide openings are divided into two. After all the bubble locations are decided, their center points are connected by Delaunay triangulation to give a complete mesh topology. The appeals of the approach are: the method is consistently applicable to 2D and 3D triangulation; no thin ill-shaped triangles or tetrahedra are generated; node spacing, or element size, is continuously controlled over the whole domain; and a mesh can be reconstructed adaptively with changes in geometry and node spacing distribution.

Thesis Advisor:	Professor David C. Gossard
Thesis Committee:	Professor Nicholas M. Patrikalakis
	Professor Alex P. Pentland

Acknowledgements

Many people and organizations gave me invaluable support in spirit or in action throughout the duration of my research. I would like to express my gratitude to them.

Professor David Gossard, my thesis advisor and doctoral committee chairman, provided me with the mental and technical guidance required to complete this thesis. The Computer Aided Design Laboratory, which Dr. Gossard founded and directs, made my graduate study at MIT an exceptional experience. I have appreciated his intensive experience in CAD research and constant effort to seek innovative research directions, which attracted many wonderful people to the lab.

Other thesis committee members, Professor Nick Patrikalakis and Professor Sandy Pentland, have also greatly contributed to the direction and quality of this work. I would like to thank Dr. Patrikalakis for providing a broad range of useful references and intensive feedback on my thesis writing. He has always inspired me with his strong work ethic and deep technical knowledge in the areas of geometric modeling and numerical computation. Dr. Pentland has provided many valuable technical ideas. His creative viewpoint, based on his rich experience in physically-based modeling and computer vision, improved my insight and understanding of the problem.

The time and opportunity to pursue this study were provided by IBM Japan Ltd. I would like to thank Dr. Norihisa Suzuki, the director of Tokyo Research Laboratory, and other members of IBM management for their help and encouragement. This research project was also partially funded by Ford Motor Co. Ltd. I would also like to thank RICOH COMPANY, LTD. for providing DESIGNBASE to our lab.

Professor Mark Jakiela and my CADLAB colleagues generated an environment of friendship and technical stimuli. Particularly, discussions with George Celniker, Hiroshi Sakurai, Ashok Kumar, Lian Fang, Barbara Balents, David Wallace, and Tamotsu Murakami were always insightful. I would also like to thank MIT friends, Boo-Ho Yang, Takashi Maekawa, Kenshiro Kato, and many others, for their help in surviving the difficult life of a doctoral candidate.

I would like to thank my parents, whose practice of freedom and love has helped me grow into what I am, and my former teachers, Professor Toshio Sata and Professor Fumihiko Kimura, who first introduced me to this exciting research area of CAD/CAE/CAM and have always remained supportive. Finally to Betti for her friendship and love.

Table of Contents

List of Figures

List of Tables

1

Introduction

Many problems of practical importance require meshing, the approximation of a given geometry into a set of simpler elements such as line segments, convex polygons and convex polyhedrons. Over the past few decades many such problems have emerged in various computer applications, along with a corresponding collection of solution techniques. These applications include engineering analysis, computer graphics, layered manufacturing, surface design, and shape reconstruction.

An exact mathematical representation is usually available, so why does one have to use an approximating mesh? There are at least two reasons:

- Numerical methods for analysis can solve only for discretised geometries.
- More robust geometric algorithms with less computational complexity are available for a meshed geometry.

As an example scenario, suppose you are an industrial designer who has just finished designing a smoothly curved automotive body panel using a CAD (computer aided design) system. The next step is to perform computer simulations that check the performance of the car. Given the shape and material property data, a class of commercial computer software is available to calculate the wind friction, structural

strength, crash strength and so on; it is not necessary to create a physical car model and perform physical experiments. However, the full geometry developed with CAD software cannot be used directly by most analysis software. The precise mathematical representation must be converted into a discretized shape representation, called a mesh.

The mesh input requirement comes from the underlying numerical methods used in analysis software, such as finite difference analysis, FEM (finite element analysis), and BEM (boundary element analysis). All of these techniques subdivide a shape of interest into elements. All relevant physical values inside the element are then interpolated using the values at the element's corners and appropriate interpolation functions. In this way, a continuous value distribution in a continuous region is represented by a finite number of variables, those at element corners. The analysis technique is then able to formulate a set of equations using the finite number of variables and to solve them numerically. Meshing, therefore, is an essential task when finite difference analysis, FEM, and BEM are employed.

Another motivation for generating meshed piecewise shapes is that more robust geometric algorithms with less computational complexity are available for piecewise linear shapes. Suppose an industrial designer wants to check the appearance of a car that has been designed using a CAD system. A state-of-the-art three dimensional graphic workstation can render a realistic image of the car, rotate it, and change the lighting conditions in real-time. This process performs many geometric calculations such as coordinate transformation, clipping, shading, and hidden surface removal in real-time. Such real-time calculations are usually made possible with specially designed hardware/firmware which operates on a polygonal mesh representation rather than the original curved surface representation.

In summary, a piecewise mesh representation is often necessary in many computer applications, because it is a simpler shape representation for which more efficient and robust mathematical tools and algorithms are available.

1.1 PROBLEM SCOPE

Traditional meshing problems are naturally classified into categories according to the dimension of the region to be meshed:

• One-dimensional (1D) meshing

The input region is a planar curve or a space curve. The curve is approximated by a set of line segments, the end nodes of which are located exactly on the original curve.

• Two-dimensional (2D) meshing

The input region is a closed region on a plane or a curved surface trimmed by boundary curves. The region is subdivided into a set of triangles or four-sided polygons, whose nodes are located exactly on the original plane or surface. Boundary nodes are located exactly on boundary curves.

• Three-dimensional (3D) meshing

The input region is a closed volumetric region in three dimensional space surrounded by boundary surfaces. The region is subdivided into a set of tetrahedra or quadrilateral elements. Boundary nodes are located exactly on boundary surfaces.

• Hybrid-dimensional meshing

The input region is a combination of 1D, 2D, and 3D regions.

Figure 1.1 contains a schematic summary of the above categories of meshing problems. Meshing problems in higher dimensional regions can be defined in a similar manner. Although the method proposed in Chapters 3 and 4 is easily extensible to higher dimensional regions, we have limited our problem scope to 3D because of this domain's obvious practical importance and utility.

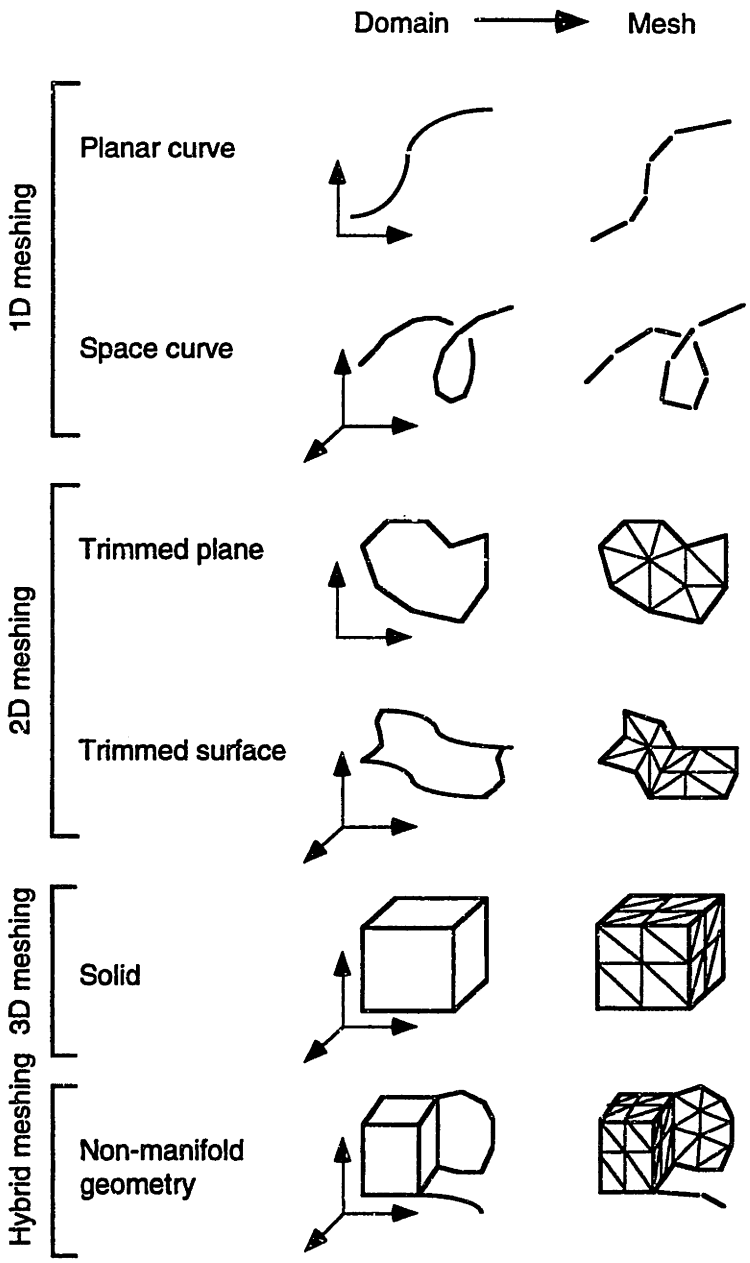


Figure 1.1 Problem scope classified by the dimension of region to be meshed. 1D meshing is further classified into planar curve and space curve, 2D meshing into plane and surface. Hybrid meshing includes domains from 1D to 3D.

1.2 REPRESENTATIVE USES OF MESHES

Although mesh generation has been studied most intensively in the research community of engineering analysis, other computer applications exist in which meshing plays a key role, such as computer graphics and freeform fabrication. The problem description and detailed requirements are slightly different application by application; we, however, want to define our meshing problem in as general a manner as possible and to pursue a consistent method applicable to all classes of problems.

The rest of this section reviews how a given mesh representation is utilized in different application areas. The goal of these examples is: (a) to underline the practical importance of mesh generation; (b) to serve as preparation for a general, formal definition of the meshing problem in Sections 1.3 and 1.4; and (c) to extract various common requirements of mesh generation in Section 1.5.

1.2.1 Engineering Analysis

In industry a great deal of design time is devoted to analysis, especially when physical experiments are performed on real components. In order to reduce the whole product development time, it is desirable to computerize analysis using numerical methods such as finite difference, FEM and BEM. Various kinds of commercial software are available for structural, fluid, and heat transfer analysis.

To be suitable for analysis software, the geometry of the shape that was created in the design phase must be transformed into a discretized model - a mesh - consisting of a collection of cells which must satisfy a number of geometric and topological conditions that are dictated by the method [Bathe 82]. The operation of transforming a geometric model, especially a 3D model, into a valid mesh is labor intensive, so a fully automated mesh generation scheme is desired.

Converting a CAD model to a mesh is performed in two steps (See Figure 1.2) :

(1) Simplification of the geometry

(2) Discretization of the geometry into a mesh

The first step is motivated by the need to reduce computational time and storage space. We delete portions of geometry which play no significant role in the analysis of the target physical phenomena. As depicted in Figure 1.2, there are two types of simplification:

- Simplifying by thinning

Pipe-like or beam-like geometries are often modeled as one dimensional curves. Shell-like geometries for which the thickness is small in comparison to the whole component size are approximated as two dimensional shells. Many mechanical sheet metal components and ship hulls are categorized into this type of geometry. (See Figure 1.2 (a), (b), and (c))

- Simplifying by detailed geometry feature removal

Insignificant geometric features, such as holes, protrusions, and grooves, are removed from the model. The criteria for determining which features are removable are not straightforward. For example, a groove of very small size can cause a fatal stress concentration in structural analysis, but can be negligible in a heat transfer analysis. This type of geometry simplification, therefore, is typically done by a highly skilled engineer. (See Figure 1.2 (d))

After the original geometry is simplified, it is discretized into a mesh. A curve is meshed into a series of one dimensional beam elements with nodes that lie on the curve. A shell is subdivided into two dimensional shell elements, either triangular or rectangular. Three dimensional shapes are meshed by a collection of tetrahedral or quadrilateral cells. In the most complicated case, a given solid geometry can be simplified as an union of 1D, 2D and 3D geometry, leading to a combination of hybrid-dimensional meshes as shown in Figure 1.2 (e).

In the discretization step, size and shape of mesh elements must be selected very carefully, because of the tradeoff between: (a) computational cost and required storage space; and (b) solution accuracy. Due to this fact, engineers commonly utilize finer meshes in regions critical to analysis results, and rougher meshes in regions of less importance. Starting with a uniform mesh, automated repetitive procedures that determine a better mesh size distribution during the course of analysis have been proposed, called adaptive remeshing.

All methods strive to have element sides as equal in length as possible. Thin, ill-shaped elements increase analysis error and make the numerical solution less stable/accurate.

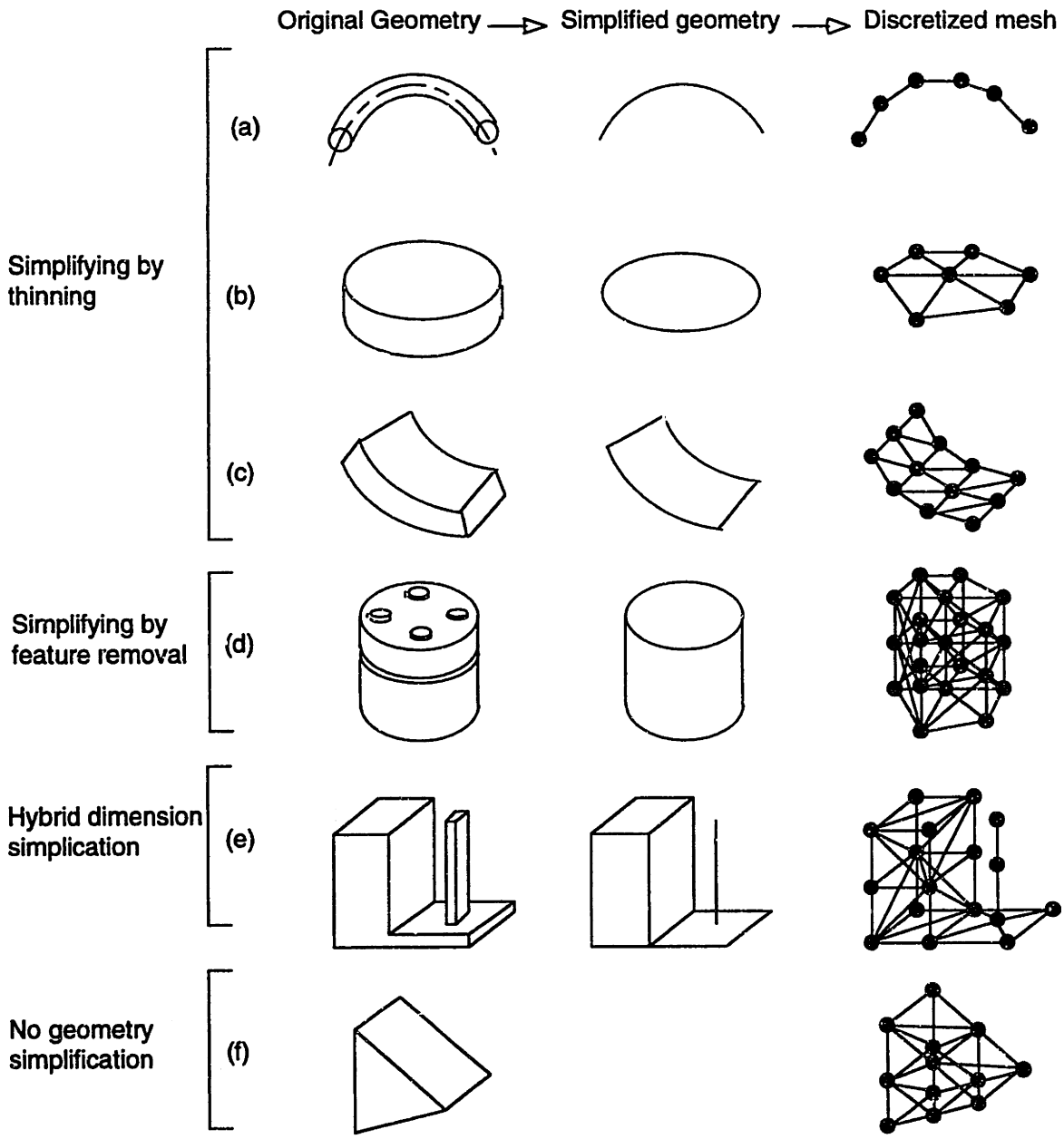


Figure 1.2 Two-step conversion of a CAD model to a mesh: (1) simplifying geometry, and (2) discretizing the geometry into a mesh. The former steps can be classified into two categories: (a)(b)(c) simplifying by thinning; and (d) simplifying by detailed feature removal.

1.2.2 Computer Graphics

The display of parametric curves and surfaces such as Bezier, Hermite polynomials and NURBS (non-uniform rational B-splines) is becoming increasingly important in CAD/CAM, computer animation and scientific visualization.

It is a common practice to convert a parametric surface to a polygonal mesh, a set of connected polygons. On a graphics workstation, polygons can be shaded individually using polygon shading methods, such as Gouraud shading and Phong shading, which make the polygonal mesh look like a smoothly curved surface as shown in Figure 1.3 [Foley 90]. The method used to make this possible is similar to that used for FEM; the intensity or the normal vector inside each polygon is approximated using values from the corner points of the polygons.

Gouraud shading, also called intensity interpolation shading, is performed in the following three steps:

(1) Normal vectors at vertices, or nodes, are calculated. They are either computed directly from the analytical description of the surface, or are approximated by averaged values of the normal vectors of adjacent facets. The latter method is employed when the original surface representation is not available.

(2) The intensity at each vertex is calculated using the vertex normals with some selected illumination model.

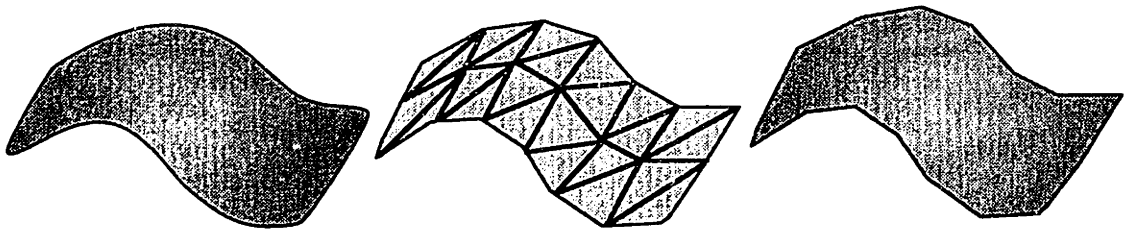
(3) The intensity values at arbitrary points in the polygons are calculated by linearly interpolating the intensities at vertices.

In Phong shading, normal vectors are used rather than intensity values. The normal vector in a polygon is found by linearly interpolating the normal vectors at corner vertices. This shading method takes more computational time than Gouraud shading, but the resultant image looks superior, especially when utilized with a specular reflection illumination model which yields more realistic highlights.

Because of these algorithms' simplicity and great performance, current 3D graphics workstations typically support one or both of these algorithms through a combination of hardware and firmware. For this reason, the conversion from parametric surfaces to polygonal meshes is key to most computer graphics applications.

In applying polygonal shading, element sizes must be controlled in order to avoid polygonal silhouettes that would be visually obvious -- generally surface portions with high curvature require finer elements for this reason.

One recent trend in graphics is "physically-based" modeling and animation, in which realistic shapes and motions are calculated using classic Newtonian physics. Sophisticated analysis methods such as FEM(finite element method) and BEM(boundary element method) are used to simulate realistic object behavior in computer animation. As discussed in Section 1.2.1, these methods require an analysis model consisting of discretized polygonal elements. To minimize computational time, critical portions of an object must be triangulated with finer triangles, and non-critical portions with larger triangles. Thin, ill-shaped triangles increase analysis error and make the numerical solution less stable/accurate.



(a) Original surface (b) Polygonal mesh (c) Polygonal shaded surface

Figure 1.3 Polygon shading by graphics workstations. Typically Gouraud shading and/or Phong shading algorithms are performed by a combination of hardware and firmware. Phong shading, which interpolates normal vectors, generates a more realistic image than Gouraud shading, which interpolates intensity, with a significant increase in computational time.

1.2.3 Freeform Fabrication

A new class of machines for rapid prototyping and manufacturing has been introduced since the first commercial laser stereolithograpy apparatus (SLA) was brought to market in 1988 [Rock 91]. This is a fairly new area where different terms to describe this new technology are utilized interchangeably, such as rapid prototyping, desktop manufacturing, layered manufacturing, additive fabrication, freeform fabrication, solid freeform fabrication, freeform manufacturing and so on.

In fact, commercially available machines for this purpose do not use the same means to generate solid material. The representative processes include:

• Stereolithography

This technology constructs prototype parts by selectively curing successive layers of liquid photopolymer using a UV laser. A part is held on an elevator, which descends one step size after each part layer is partially cured. By repeating this process layer by layer, adjacent layers are adhered and a solid part is produced.

• Three-dimensional printing

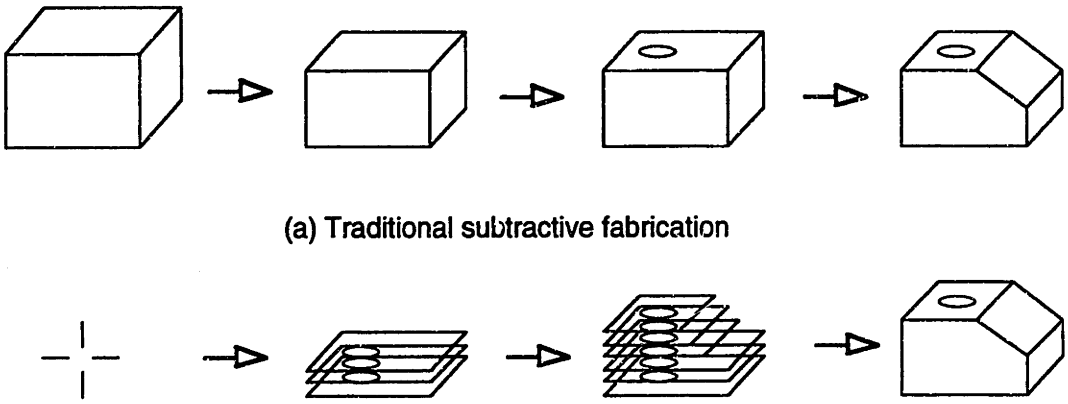
This technology has been developed at the Massachusetts Institute of Technology. Layers of ceramic powder are piled successively and a binder is partially applied to the powder using a mechanism similar to the one used in ink-jet printers.

• Selective laser sintering

This process has been developed at the University of Texas at Austin. It uses a laser to fuse various powder materials such as ABS, PVC, Polycarbonate, Nylon, and Investment casting wax. Successive layers of these materials are produced by fusing adjacent powder particles together via laser energy.

• Laminated object manufacturing

Adhesive coated sheets of material are piled up successively. In each layer, a laser is used to cut the cross sectional boundary shape. Because material outside of this boundary remains in place to support layers, an additional step to cut cross-hatches all over the outside material must be performed to remove excess materials after the first process is complete.



(a) Traditional subtractive fabrication

(b) New additive fabrication

Figure 1.4 Freeform fabrication technologies. Stereolithography, three-dimensional printing, selective laser sintering, and laminated object manufacturing make additive construction of various materials possible.

As the above descriptions suggest, many freeform fabrication processes are being developed at present. A common characteristic unites the different methods: all freeform fabrication machines construct a 3D shape by adding material layer by layer, as illustrated in Figure 1.4. This makes freeform fabrication quick and thus advantageous in rapid prototyping and manufacturing. In this additive type of fabrication, the time-consuming tasks involved in traditional subtractive fabrication are unnecessary, for example jig design, tool selection, and cutter path calculation.

Since these machines require sliced shape data to perform layer-by-layer additive operations, a new software-related issue has come up -- how can sliced shapes be more effectively calculated from a CAD representation? There are currently two ways to obtain the slice shapes: (a) cross-sections of a CAD model are directly calculated using sophisticated surface-surface intersection algorithms [Patrikalakis 93]; and (b) an approximate polygon model faceted with triangles is obtained and then sliced. In conventional practice, the second method is most often employed and a faceted triangular surface is utilized as the standard interfacing shape representation between the CAD system and the freeform fabrication machines, as shown in Figure 1.5.

The conversion from a CAD model to a faceted triangular surface is considered one form of mesh generation in which triangle sizes must be controlled carefully. Figure 1.6 shows how triangle size affects chord height, the error between the original curved

surface and the approximated triangular mesh. Obviously, smaller triangles cause less error -- a fabricated surface will then more accurately represent the original CAD surface.

To reduce the number of triangles while also maintaining chord height within a given range, it is necessary to control element size based on surface curvature. A relatively flat portion of a surface can be approximated with larger triangles and a highly curved portion with smaller triangles.

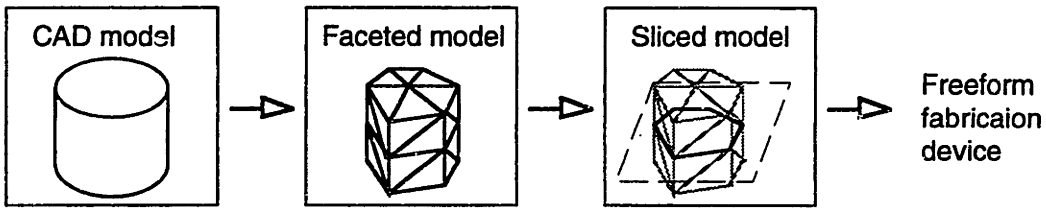


Figure 1.5 A faceted triangular surface as a common interfacing shape representation between a CAD system and freeform fabrication machines. With a triangulated surface, cross section calculation becomes much easier and more robust.

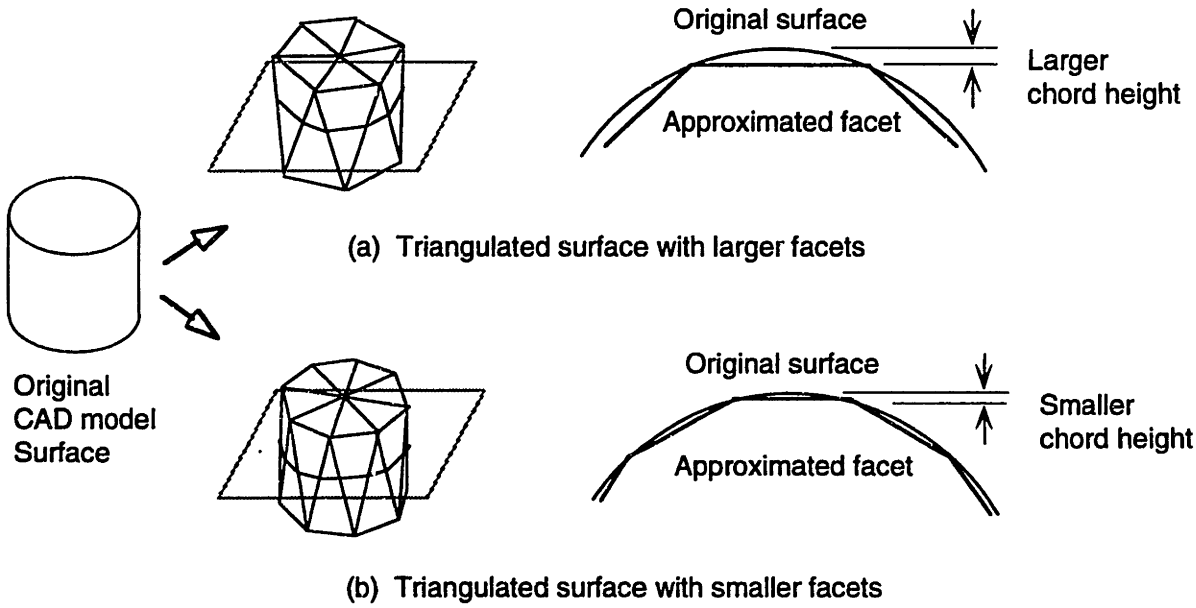


Figure 1.6 The effect of triangle sizes on approximation error. Triangle sizes affect chord height, the error between the original CAD surface and the approximated triangular mesh. The finer the triangles, the more faithfully the constructed shape will represent the original shape.

1.3 TERMINOLOGY AND NOTATION

1.3.1 Geometry Representation

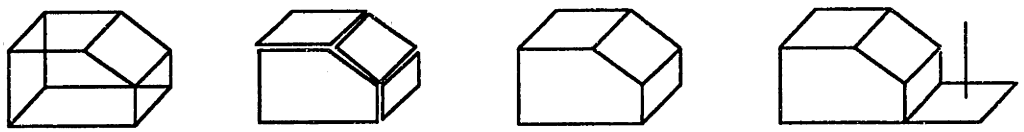
One dimensional space is denoted by R (or sometimes R^1) and, similarly, *two dimensional space* by R^2, and *three dimensional space* by R^3. An $m \times n$ matrix with element a_{ij} in row i and column j is written

$$A = [a_{ij}] \tag{1.3.1}$$

The n-vector consisting of the ith row of A is denoted by a_i and the m-vector which is the jth column of A by A_j. Although vectors are column vectors, when there is no danger of confusion, they are usually written

$$x = (x_1, \ldots, x_n) \tag{1.3.2}$$

Different categories of geometry exist, all of which are subsets of R^3. As shown in Figure 1.7, they include (a) the *wireframe model* consisting of one dimensional entities, or *curves*, (b) the *surface model* consisting of two dimensional entities, or *surfaces*, bounded by several closed loops of curves, (c) the *solid model*, a volume bounded by one or more closed shells of curves and surfaces, and (d) the *non-manifold model*, a combination of wireframe, surface and solid models. We assume that input geometry for mesh generation is one of these models.



(a) Wireframe model | (b) Surface model | (b) Solid model | (d) Non-manifold model

Figure 1.7 The scope of geometry to be meshed. (a) Wireframe model consisting of one dimensional entities, curves. (b) Surface model consisting of two dimensional entities, surfaces, bounded by curves. (c) Solid model as a volume bounded by curves and surfaces. (d) Non-manifold model as a union of wireframe, surface, and solid models.

Since the non-manifold model is the most general form, let us study an example of a non-manifold data structure, proposed in [Masuda 89, Masuda 93, Kawabe 88]. In the definition, "non-manifold geometry" means a Euclidean cell complex, which intuitively is a set of wireframe, surface, and solid models. In a Euclidean cell complex model, a 0-cell is a vertex, a 1-cell an edge, a 2-cell a face, and a 3-cell a volume. As shown in Figure 1.6, topological entities in the model include:

- Vertex A point in three dimensional Euclidean space.
- Edge A bounded, open curve.
- Loop A cyclic list of Edges that form a closed loop.
- Face A bounded, open surface. Holes may exist in a face.
- Shell A set of Faces that form a closed shell.
- Volume A bounded, open three dimensional Euclidean space. Through-holes and cavities may exist.
- Complex The highest level of geometry representation.

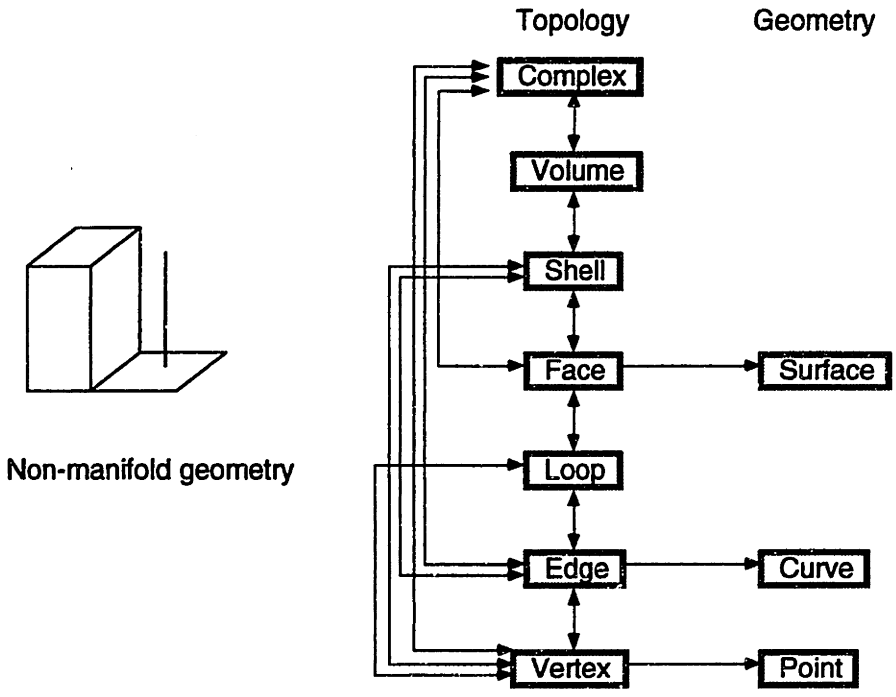


Figure 1.8 An example of a non-manifold geometry data structure. Different types of geometric models, wireframes, surfaces, solids are stored in a consistent data structure. (Proposed in [Masuda 89, Masuda 93, Kawabe 88].)

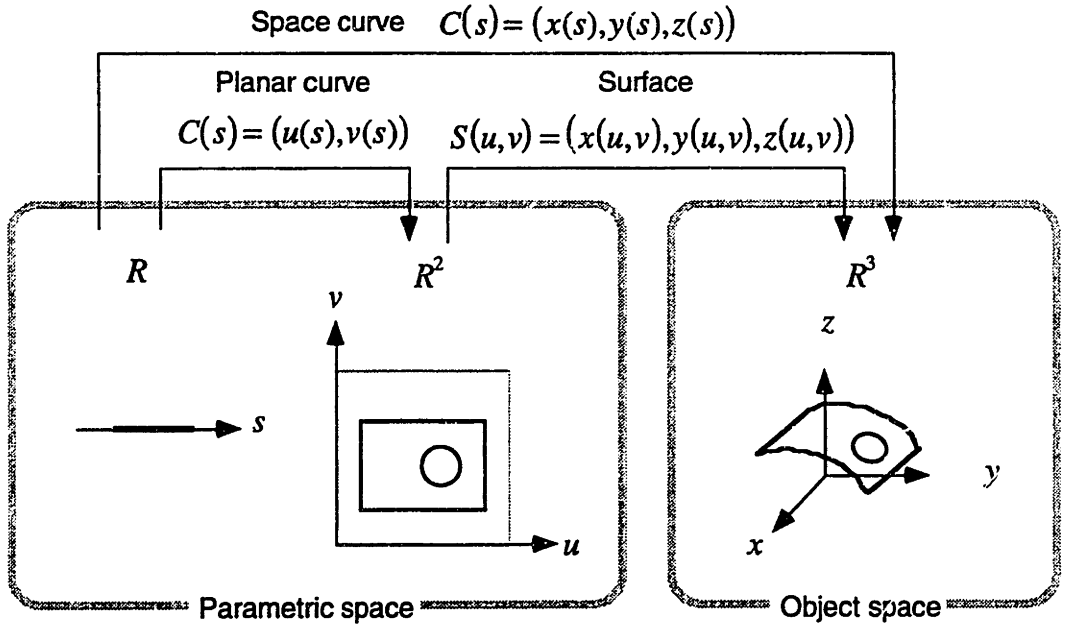


Figure 1.9 Geometric entities defined as a mapping. A surface is defined as a mapping $S(u,v)=(x(u,v),y(u,v),z(u,v))$, from parametric space R^2 to object space R^3. A planar curve, or trimming curve, is defined as a mapping $C(s)=(u(s),v(s))$, from parametric space R to R^2. A space curve is defined as a mapping $C(s)=(x(s),y(s),z(s))$ from parametric space R to object spaceR^3.

As in most solid modeling systems with boundary representation, geometric information is stored under three categories of geometric entities: points, curves, and surfaces. All geometry is defined in R^3 *object space*. A point is a coordinate triple in the object space, denoted by

$$v: (x,y,z) \in R^3 \tag{1.3.3}$$

A *curve* is defined over a bounded R^1 space called the *parametric space*, which is then transformed into object space as shown in Figure 1.9. In other words, a curve geometry is given as a mapping from parametric space to object space

$$C(s)=(x(s),y(s),z(s)) \tag{1.3.4}$$

where s represents a parameter value in parametric space.

Similarly, a surface is defined over a rectangular region in R^2, *parametric space*, which is then transformed into object space.

$$S(u,v)=(x(u,v),y(u,v),z(u,v)) \tag{1.3.5}$$

The surface can be trimmed by restricting the rectangular region to a subset called the *trimmed region*, and its boundary curves are called *trimming curves*, denoted by

$$C(s) = (u(s), v(s)) \tag{1.3.6}$$

Although trimming curves are planar curves in two dimensional parametric space, they can be interpreted as a mapping from parametric space s to object space,

$$C(s) = \big(x(u(s), v(s)), y(u(s), v(s)), z(u(s), v(s))\big) \tag{1.3.7}$$

Actual curve and surface representation can be of any form as long as it is continuous.

1.3.2 Mesh Representation

A *set* of elements s_1, s_2, s_3, ... is written explicitly as

$$S = \{s_1, s_2, s_3, \ldots\} \tag{1.3.8}$$

and a set defined to contain all elements x for which a condition P is true is defined by

$$S = \{x : P(x)\} \tag{1.3.9}$$

As shown in Figure 1.10, *mesh elements*, or topological entities, contained in the definition of meshes are:

- A *vertex*, or *node*, denoted by v is defined as a point in Euclidean space. Only this element carries geometric information. A set of vertices is denoted by V.

$$v = (x, y, z) \in R^3$$
$$V = \{v_1, \ldots, v_n\}, n \geq 1 \tag{1.3.10}$$

- An *edge*, or *arc*, denoted by e, is represented as an unordered pair of end vertices, v_1 and v_2. A set of edges is denoted by E.

$$e = [v_1, v_2] \tag{1.3.11}$$
$$E = \{e_1, \ldots, e_k\}, k \geq 1$$

- A *triangle*, denoted by p, is represented as an unordered list of three bounding edges, e_1, e_2 and e_3. A set of edges is denoted by P.

$$p=[e_1,e_2,e_3] \tag{1.3.12}$$

$$P=\{p_1,\ldots,p_l\},l\geq 1$$

- A *tetrahedron*, denoted by t, is defined as a unordered list of four bounding triangles, p_1, p_2, p_3, and p_4. A set of edges is denoted by T.

$$t=[p_1,p_2,p_3,p_4] \tag{1.3.13}$$

$$T=\{t_1,\ldots,t_m\},m\geq 1$$

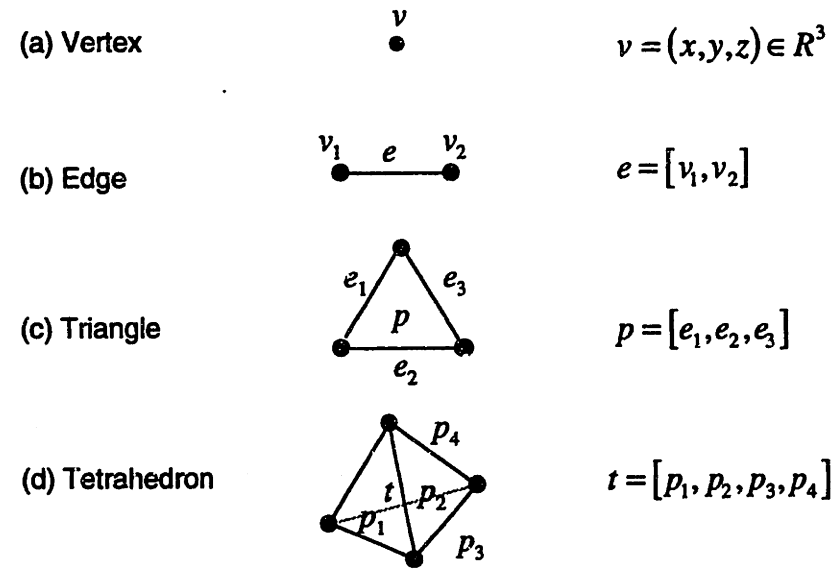


Figure 1.10 Definition of mesh elements. (a) a Vertex, denoted by v is defined as a point in Euclidean space. Only this element carries geometric information, $(x,y,z)\in R^3$. (b) an Edge, denoted by e, is represented as an unordered pair of end vertices, v_1 and v_2. (c) a Triangle, denoted by p, is represented as an unordered list of three bounding edges, e_1, e_2 and e_3. (d) a Tetrahedron, denoted by t, is defined as an unordered list of four bounding triangles, p_1, p_2, p_3, and p_4.

Figure 1.11 depicts a variety of mesh definitions. According to the types of mesh elements represented, four kinds of meshes are defined in the following manner.

• *1D mesh*

A 1D mesh is a pair $M_c = (V, E)$, which contains at least two vertices and an edge. This type of mesh is generated as the output when meshing a wireframe model.

• *2D mesh*

A 2D mesh is a pair $M_s = (V, P)$, which contains at least three vertices, three edges, and one triangle. This type of mesh is generated as the output when meshing is performed on a surface model or the boundary of solid model.

• *3D mesh*

A 3D mesh is a pair $M_v = (V, T)$, which contains at least one tetrahedron, four triangles, six edges, and four vertices. This type of mesh is generated as the output when meshing is performed on a solid model.

• *Hybrid mesh*

A hybrid mesh is a list $M = (V, E, P, T)$, which contains a combination of 1D, 2D, and 3D meshes. Meshing a non-manifold model generates this type of mesh. This is the most general mesh representation since it can represent 1D, 2D, and 3D meshes.

To represent the above mesh topology and geometry, one can define a simple data structure as shown in Figure 1.12 (a), or a more complete data structure as shown in Figure 1.12 (b). The latter data structure requires more storage space, however it holds pointers to the original geometric entities in the geometric modeller, so that mesh elements which represent the original boundary can be found easily.

A *valid* mesh consists of properly connected elements, e.g., in a 1D mesh, edges are connected only at the end vertices. The same concept applies to 2D, 3D, and hybrid meshes. So, in a valid 2D mesh, two adjacent triangles can be connected only on a common edge. A valid 3D mesh consists of tetrahedra, adjacent sides of which share one triangle. Self-intersection is not allowed. Figure 1.13 shows some examples of *invalid* meshes.

Node spacing, which determines the *size* of a mesh, is denoted by $d(x, y, z)$ or $d(u, v)$. Node spacing designates the distance between two connected nodes, i.e. an edge length. The desired node spacing is given by a user based on an application's requirements. A

uniform value can be defined over the geometry so that the resultant mesh comes out in a nearly uniform size everywhere. In most applications, however, a varying node spacing is desired -- in a mesh for engineering analysis, a finer mesh is required in regions where the solution value(s) are rapidly changing; in computer graphics applications, a surface region of high curvature should be approximated by smaller triangles.

In order for a mesh to approximate the given geometry, some mesh elements must be placed exactly on the boundary of the original geometry. Such mesh elements are called *boundary mesh elements.*

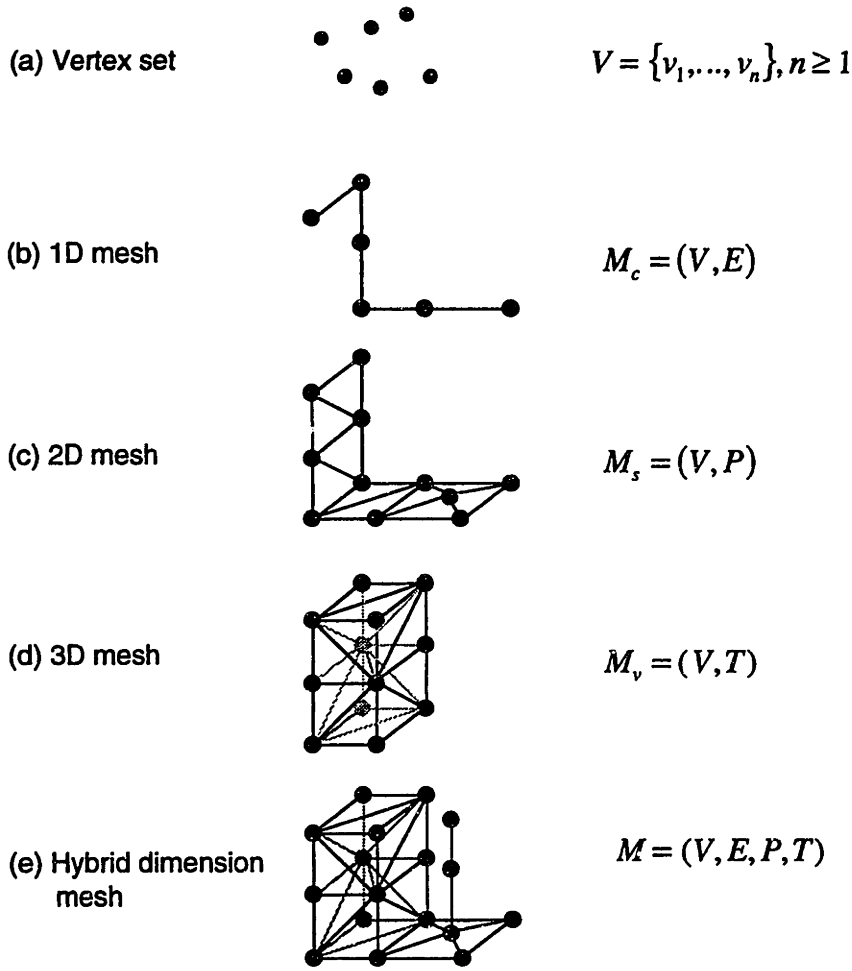
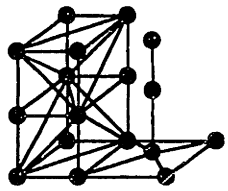
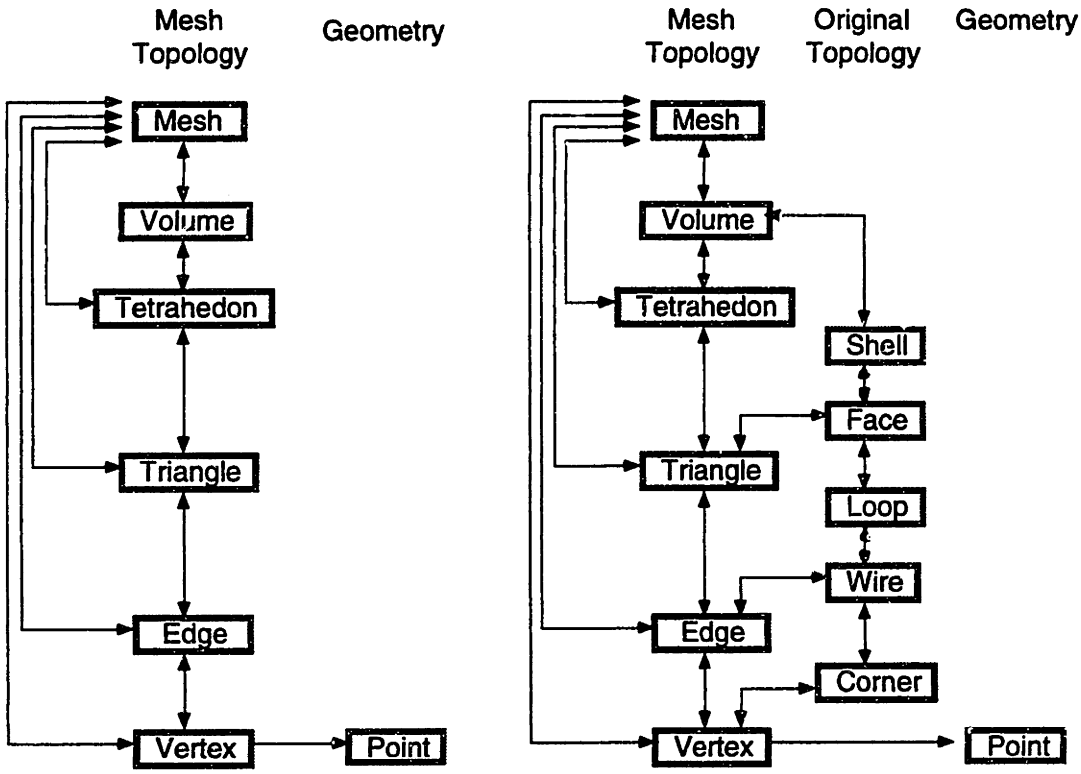


Figure 1.11 Definition of a mesh. (a) a Vertex set is a set of vertices. All other meshes are defined by the vertex set and topology, or connectivity , on it. (b) a 1D mesh, denoted by M_c, is a set of edges. (c) a 2D mesh, denoted by M_s, is a set of triangles. (d) a 3D mesh, denoted by M_v, is a set of tetrahedra. (e) a Hybrid dimension mesh, denoted by M, is a combination of 1D, 2D, and 3D meshes. M_c, M_s, and M_v are in the subset of M.



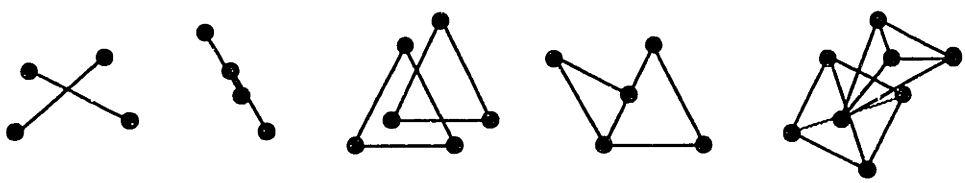
Hybrid dimension mesh



(a) A simple data structure for a hybrid-dimensional mesh

(b) A complete data structure for a hybrid-dimensional mesh

Figure 1.12 Data structures for a hybrid-dimensional mesh. (a) An example of a simple hybrid-dimensional mesh without original topology information. (b) An example of the complete data structure for a hybrid-dimensional mesh with the original topology information



(a) Invalid 1D mesh (b) Invalid 2D mesh (c) Invalid 3D mesh

Figure 1.13 Examples of invalid meshes

1.4 PROBLEM STATEMENT

In mesh generation, a given geometry, either 1D, 2D, 3D, or non-manifold, is subdivided into a valid mesh, i.e., a properly connected set of elements, edges, triangles, and tetrahedra. We assume that a geometry is explicitly represented using a non-manifold B-rep model consisting of vertices, edges and surfaces. All such geometry is denoted by G, a set of vertices, edges, surfaces and volumes.

Again, it is essential to control the size of mesh elements according to a node spacing function. This is given by a user or an application program as a function of position in either parametric space or object space, denoted by $d(u,v)$ and $d(x,y,z)$ respectively.

The class of meshing problems we are trying to solve is described as follows.

Given:
- a geometry, a subspace of object space, $G \in R^3$
  (vertices, curve(s), surface(s), volume(s), or non-manifold geometry)
- a node spacing function, $d(x,y,z)$ or $d(u,v)$

Generate:
- a valid mesh M

Depending on the dimension of the given geometry, several types of mesh generation problems are identified as previously shown in Figure 1.1; they are (a) 1D meshing, (b) 2D meshing, (c) 3D meshing, and (d) hybrid-dimensional meshing. Figure 1.14 depicts the typical input and output in 2D meshing.

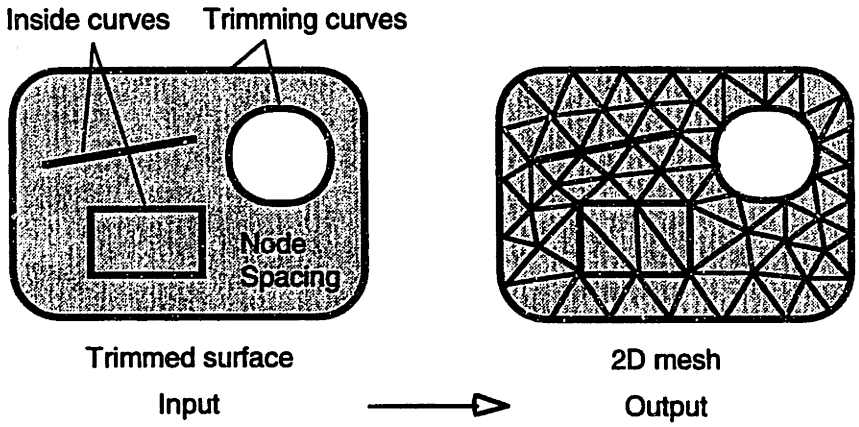


Figure 1.14 An example of a 2D meshing problem. It takes as input a 2D geometry and inside curves on which nodes are placed and generates as output a triangular mesh that satisfies a given node spacing.

1.5 MESHING REQUIREMENT

Mesh generation is necessary for a wide variety of applications, with requirements that vary widely by application. In this Section, some essential requirements are summarized, so that we can devise a general-purpose algorithm in later chapters.

• *Meshed domain*

It is desired to have a consistent algorithm applicable to all of the problem domains. As stated earlier in Section 1.3, there are three basic domains that a meshing algorithm has to cope with: curves, surfaces and volumes. Sometimes, a combination of these three domains, i.e., a non-manifold domain, has to be meshed.

Another rather exceptional meshed domain is a set of discrete data points. In the application area called *shape reconstruction,* a polygonal surface (2D mesh) is fit to a given set of points. A surface is generated to fit a set of scattered 3D data points. This is a common requirement in many applications including machine vision and medical image processing.

• *Element type*

In 1D meshing, the edge is the only type of mesh element. In 2D and 3D, however, there are two types of mesh elements, *triangular* and *quadrilateral.* In 3D, triangular elements are tetrahedra. Quadrilateral 2D elements are four-sided convex polygons, and in 3D, six-sided convex polyhedra.

Generally, in engineering analysis, quadrilateral elements are advantageous in terms of solution accuracy and time for convergence. Therefore, one should use quadrilateral elements whenever a meshed domain shape is very simple and node spacing requirements are not strict. In meshing a general shape bounded by curved elements with arbitrarily defined node spacing, however, it is much easier for an automatic algorithm to generate triangular elements.

For this reason, we pursue algorithms for triangular mesh elements in Chapters 3, 4, and 5. Some methods for automatic conversion from a triangular mesh to a quadrilateral mesh are discussed in Section 2.2.2.

• *Element regularity.*

In general mesh elements should not be thin or badly distorted, e.g., in 2D meshing we want triangles to be as equilateral as possible. This requirement is especially important in engineering analysis applications; elements that are too thin cause more analysis error and make convergence slower. The deviation from equilateral shape is called *geometrical irregularity.*

Another related measurement is *topological irregularity.* Frey addressed 2D topological irregularity intensively in [Frey 91]. In 2D triangulation, a *regular* node has six adjacent nodes, but typically some nodes have more or less than six neighboring nodes. The paper addresses this problem with a relaxation method developed to obtain as regular a triangular lattice as possible by exchanging edges among nodes.

We can extend Frey's idea to 3D and define a regular 3D node. Suppose we are packing tetrahedra with edges of equal length in 3D space, so that there are no gaps among them. A node in such a configuration has 12 adjacent nodes, called topologicaly regular nodes.

There are at least two causes of geometric and topological irregularity, (a) boundary constraints, and (b) varying node spacing. Regular triangles or tetrahedra with uniform sizes can be paved to fill spaces with certain boundary shapes. However, given an arbitrary shape, this is not always possible and some irregular nodes tend to be introduced. The same thing happens when element sizes are varied to meet a given node spacing requirement.

In short, some irregularity is inevitable in a general meshing problem. Our goal, therefore, is to minimize the irregularity while satisfying given boundary constraints and node spacing requirements.

• *Node spacing control*

The density of the nodes should be controllable over the meshed region based on some application-driven criteria. For example, in structural FEM, the stress or strain jump indicates analysis error due to gross discretization, so the node spacing in such an area should be decreased. In surface meshing for computer graphics or freeform fabrication, node spacing decides the accuracy of shape approximation. Therefore, it is desirable to have a denser mesh in regions of high curvature, and a coarser mesh in low curvature regions. Our meshing scheme must allow the specification of a continuously-changing, arbitrary node spacing function over the region.

• *Node constraint capability*

Geometric constraints on some node locations must be satisfied. The most obvious of these are boundary constraints, i.e. all the perimeter nodes must remain on the original boundary. In addition to this basic requirement, it is often necessary to allow the specification that some nodes be located precisely on specific edges and surfaces inside the boundary.

• *Adaptive remeshing*

In some applications, continuous remeshing is required, such as when the geometry and/or node spacing are changing incrementally. A good algorithm would quickly modify a previous mesh to create the subsequent one, rather than reconstructing the mesh from scratch.

One example of an application where this capability could be used is large deformation engineering analysis, in which an object is significantly deformed as time passes. In such an analysis, the original mesh is distorted badly and becomes invalid. Also, in general, analysis problems that solve for a time-transient response require adaptive remeshing when the desired node spacing is changed during the course of simulation.

If the changes in geometry and node spacing are local, then a local remeshing capability, in which only a part of a region is remeshed, is desired. An adaptive remeshing capability can thus help to reduce the computation time significantly.

• *Computational efficiency*

As in most algorithmic problems, computational complexity is one of major concerns in mesh generation. The complexity of a meshing problem is primarily measured by the number of nodes or mesh elements - they are usually of the same order. Denoting the primary number by N, typical indices of algorithmic complexity include; (a) 1, or *constant*, (b) $\log(N)$, or *logarithmic*, (c) N, or *linear*, (d) $N\log(N)$, for which we do not have a good adjective, (e) N^2, or *quadratic*, (f) N^3, or *cubic*, and (g) 2^N, or *exponential.* A solution's computational complexity depends both on the nature of the problem itself and the way the algorithm is devised.

If the solution is unique, for example in a simple sorting problem, one can say an algorithm with less complexity is better. In some problems, including our meshing

problem, however, there exist multiple feasible solutions, in fact an infinite number of solutions, and thus we also need to consider the "quality" of the solution.

Usually there is a tradeoff between speed and quality. Of course, it would be perfect if we could satisfy both speed and quality considerations at the same time, but in reality, we have to chose between them based on application requirements. A general algorithm should allow the user to adjust between speed and quality.

Another emerging concern is the "parallelability" of the algorithm. When cost effective parallel machines become widely available, algorithms suitable for parallel processors will draw more attention.

• *Degree of automation*

As available computational power increases drastically, human intervention costs more and more, and fully-automated methods are desired. A good example is mesh generation for engineering analysis. In early days, companies did not mind the large cost of manual mesh generation, because computation was so expensive that analysis itself cost a lot. Now thanks to drastically increased computer performance, a large scale analysis problem which used to take a couple of days or weeks on a mainframe computer can be solved in several hours or even minutes on a desktop workstation. In this situation, manual meshing or any form of human intervention becomes the bottleneck of the whole process in terms of time and cost. Fully-automated meshing algorithms are thus quickly becoming a business necessity.

Table 1.1 summarizes the various meshing requirements in each major application domain. As reviewed above, the general mesh generation problem is very demanding and it is not easy to devise an algorithm that satisfies all the requirements at the same time. For this reason, various algorithms have been crafted thus far on a case by case basis to satisfy some of these requirements exclusively.

Table 1.1 Meshing requirements.

	Engineering Analysis	Computer Graphics	Layered Manufacturing
Meshed Domain			
curve	√√	√√	√√
surface	√√	√√	√√
volume	√√		
non-manifold geometry	√√		
discrete data points		√√	
Element Type			
triangular	√	√	√
quadrilateral	√	√	
Element Regularity			
topological	√√	√	√
geometrical	√√	√	√
Node Spacing Control			
discrete	√√	√√	√√
continuous	√	√	√
Node Constraint			
boundary constraint	√√	√√	√√
internal constraint	√√		
Adaptive Remeshing			
adaptive to geometry	√√	√√ †	
adaptive to node spacing	√√	√√ †	
local remeshing	√	(√)	
Computational Efficiency			
less complexity	√	√	√
parallel processor	(√)	(√)	(√)
Degree of Automation	√√	√√	√√

† : √√, essential when physically-based model is employed.

√√	essential	√	good to have
(√)	optional	?	unclear
blank	unnecessary		

1.6 SUMMARY

Mesh generation is an essential problem of practical importance in many computer related applications such as engineering analysis, computer graphics, and freeform fabrication. This is because an approximating mesh is often required: due to limitations in our mathematical tools, capabilities of hardware and firmware; and as a standard data format.

Meshing algorithms take as input a 1D/2D/3D/non-manifold geometry and a node spacing function, and generate as output a valid mesh consisting of edges, triangles, and/or tetrahedra. The problem is very demanding with various requirements, such as element regularity, node spacing, and node placement constraints.

Our goal is to develop a consistent and unified mesh generation algorithm which satisfies all these requirements, rather than crafting domain-specific or application-specific methods on a case by case basis.

The following chapters are organized as follows. In Chapter 2, various algorithms proposed thus far are classified by the nature of their algorithm. Then some limitations of the previous methods are addressed. Chapter 3 introduces a new physically-based approach, called *bubble mesh generation*, and summarizes the technical issues involved. Technical details of the approach are described in Chapter 4. Chapter 5 shows results that depict meshes generated for different applications. Finally, Chapter 6 gives conclusions and recommendations for future work.

2 Previous Work

This section attempts to provide a survey of previous mesh generation methods. An immense literature on mesh generation has been written over the past couple of decades. The main difficulty in categorizing the various methods is that some do not seem to fit into any group, while others could be placed in two or more groups. This is because most of the proposed methods are composed of several different techniques, which themselves can be thought of as independent categories of mesh generation. In addition, there are some "post-processes" for improving an initial mesh. It is, therefore, important to classify the existing approaches with all the common post-processes excluded from the meshing procedures.

Section 2.1 reviews previous methods and delineates six major sub-procedures proposed in these methods: (a) node placement and connection, (b) coarse domain decomposition, (c) mesh template mapping; (d) element-level domain decomposition; (e) grid-based spatial subdivision; and (f) faceting parametric surfaces in parametric space. Typically a complete meshing process is performed by a combination of these sub-procedures. Section 2.2 reviews some common post processing techniques of mesh improvement, applicable to an initially generated mesh; these post-processes includes: (a) mesh smoothing and mesh relaxation, (b) element type conversion, and (c) polygonal surface remeshing. Section 2.3 summarizes the limitations of the previous methods, leading to the technical goals to be accomplished in the thesis.

2.1 CLASSIFICATION OF MESH GENERATION METHODS

There are several reviews of mesh generation methods [Thacker 80a, Shephard 88, Ho-Le 88, Sapidis 89]. Ho-Le, in his comprehensive survey paper [Ho-Le 88], gives a systematic classification based on the temporal order in which nodes and elements are created. The resultant classification is well-accepted and referred to by many other researchers. One problem, as he also mentions in the paper, is that placement of individual methods into categories is not easy -- some methods do not seem to fit into any class, while others could be put in two or more classes. Actually, many proposed methods consist of several techniques representing different categories in the classification, so that the final placement into categories is necessarily somewhat arbitrary.

In this Section, therefore, key sub-processes, commonly used in existing meshing methods, are summarized and reviewed. The sub-processes includes: (a) node placement and connection; (b) coarse domain decomposition; (c) mesh template mapping; (d) element-level domain decomposition; (e) grid-based spatial subdivision; and (f) faceting parametric surfaces in parametric space. These methods have been pursued in the FEM/BEM research community with the exception of the last one, which was developed mainly by the computer graphics community. One complete meshing scheme is characterized by a combination of these sub-processes, performed sequentially or merged into a single process.

Mesh improvement methods not included in those sub-processes are taken up separately in Section 2.2, as they can be applied to meshes produced by any combination of sub-processes above.

2.1.1 Node Placement and Connection

In this process, a mesh is constructed by two stages: (1) node placement, and (2) node connection. In the first stage, starting with boundary vertices, additional nodes are first placed on the domain boundary and then in the interior of the domain, so that the distances between nodes follows a given node spacing criteria. These nodes are then connected to construct a complete mesh topology in the second stage.

Node placement and connection can serve as a complete meshing process. The process has become popular recently due to its simplicity and the availability of a robust mathematical algorithm for node connection, called Delaunay triangulation.

Node Placement

During node placement, an appropriate number of nodes are inserted in a well-distributed configuration, so that a mesh with small distortion and precise gradation is created in the later stage of node connection. With a complicated domain geometry and variable node spacing, this is not an easy task.

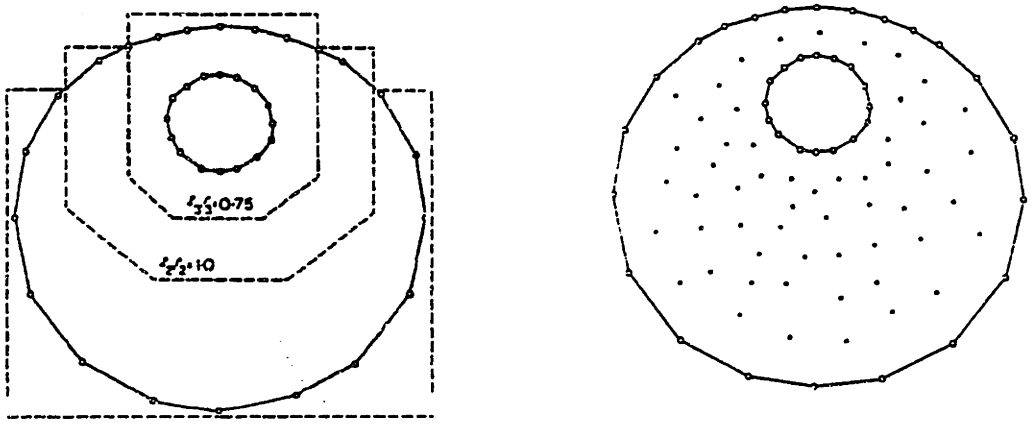
Several algorithms using random node placement followed by validity checks have been proposed [Fukuda 72, Cavendish 74, Cavendish 85, Mascardini 83].

In [Cavendish 74], a 2D domain is interactively subdivided into a set of *zones*, then an automatic node distribution procedure is performed in each zone. A uniform square grid, the size of which is determined by a node spacing function, is superimposed over the zone, and one interior node is randomly inserted per grid cell. If this node falls inside the domain, and is sufficiently distant from the boundary and other nodes, it is accepted. Otherwise, the node is deleted and another node is inserted. This process is repeated a certain number of times, and if none of them is acceptable then no node is inserted in the cell. [Cavendish 85] presents a 3D meshing technique based on similar random node insertions. A solid is intersected by a set of parallel cutting planes, thus decomposing the 3D node placement problem into a set of 2D problems. Within each plane, the user has control over local node densities using the same technique as in [Cavendish 74].

In these random node placement approaches, node spacing is controlled zone by zone in a discrete manner -- zones with gradually changing node spacing are interactively specified for a node spacing smooth distribution. Figure 2.1 (adapted from [Cavendish 85]) shows such an example of zones, each with a specified node spacing distribution and the generated set of nodes.

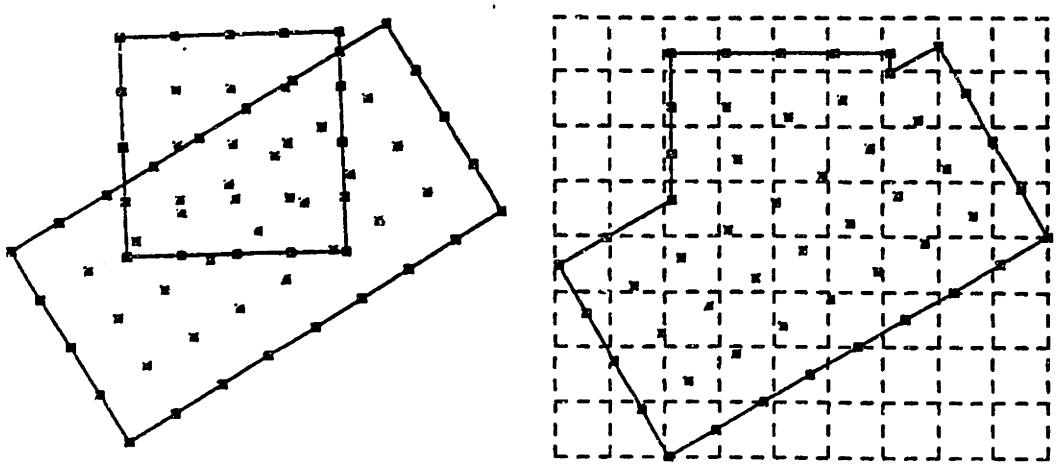
Lee proposed a CSG-based node placement method [LeeYT 84, LeeYT 83] for 2D domains. Regular node distribution patterns are prescribed for all CSG primitives, which are then combined by Boolean set operations, such as unions, intersections and differences, into a single mesh. Figure 2.2 (adapted from [LeeYT 84]) depicts a union operation of two rectangular domains. To create well-shaped elements, some nodes in the overlapping regions need to be redistributed. This is performed at two levels: subtrees and the individual points. At the tree level, actions are taken on nodes on the

basis of whether they are in, on, or outside the objects. At the individual point level, the node modification is performed within the locality of a node with respect to its neighbors.



(a) Definition of node density (b) Inserted nodes

Figure 2.1 Zone by zone node placement. (Adapted from [Cavendish 85])



(a) Before combination (b) After combination

Figure 2.2 Combining point sets under the union operator. (Adapted from [LeeYT 84])

In [Lo 89, Lo 85], parallel, horizontal cutting lines with uniform spacing are superimposed on a 2D domain, and line segments inside the domain are identified. For each line segment, nodes are placed at an interval approximately equal to the spacing between the horizontal cutting lines. Each candidate node point thus obtained is tested to determine whether the point and a set of two adjacent nodes on a boundary form a well-shaped triangle, i.e. a virtually equilateral triangle. A *quality of triangle* rating, which takes the maximum value of 1 for equilateral triangles, is defined, and only candidate nodes which give larger values than a user-given threshold are accepted.

Node placement can be also determined based on a spatial grid [Schroeder 90].

Node Connection

There are many algorithms for connecting nodes into a complete mesh. They are divided into two categories: (a) Delaunay triangulation-based approaches and (b) non-Delaunay triangulation-based approaches. The former category has become popular in recent years for both 2D and 3D meshing problems, and many algorithms based on this idea have been proposed [Cavendish 74, Cavendish 85, Lo 89, Lo 85, Schroeder 90, Field 86, Frey 87, George 91, Joe 91c, Joe 86b, Joe 86a, Joe 91a, Joe 91b, Schroeder 88, Shimada 92, Field 91b, Ruppert 92, Ruppert 93, Meshkat 91]. Efficient algorithms for Delaunay triangulation have been intensively reviewed and studied in number of textbooks [Preparata 85, Edelsbrunner 87] and papers [Bowyer 81, FangT-P 92, Sloan 87, Lawson 86, Sloan 84, Field 91a, Watson 81, Sibson 78, Green 78, LeeDT 84, LeeDT 86, Aurenhammer 91, Bowyer 81]. A number of non-Delaunay node connection methods have been also proposed by some authors [LeeYT 83, Frederick 70, Lewis 77, Nguyen-Van-Phai 82, Van-Phai 92].

Examples of Delaunay triangulation, Dirichlet tessellation, and circumcircles for 13 points on a plane are adapted from [Cavendish 85] and shown in Figure 2.3. Let us summarize the basic ideas of Delaunay triangulation and Dirichlet tessellation based on the description on pages 333-334 of the same paper. Consider N distinct points p_i, $1 \le i \le N$, in the plane R^2, and define the sets V_i, $1 \le i \le N$, as

$$V_i = \left\{ x : \|x - p_i\| < \|x - p_j\| \text{ for all } j \neq i \right\} \tag{2.1.1}$$

where $\|\cdot\|$ denotes Euclidean distance in R^2. The set V_i, called a Voronoi polygon, represents a region of the plane whose points are closer to point p_i than to any other point. The collection of Voronoi polygons is called the Voronoi diagram or Dirichlet tessellation. The boundaries of the Voronoi polygon are portions of the perpendicular

bisectors of the lines jointing point p_i to point p_j, when V_i and V_j are contiguous. In most cases, a vertex on a Voronoi polygon is shared by two other neighboring polygons so that we can construct a triangle by connecting the three points in three adjacent polygons. The set of such triangles is called the Delaunay triangulation, the geometric dual of Dirichlet tessellation. This construct can be shown to be a triangulation of the convex hull of the points. Another Delaunay criterion is that a circumscribing circle of a Delaunay triangle, called a circumcircle, does not contain other points inside as shown in Figure 2.3 (b).

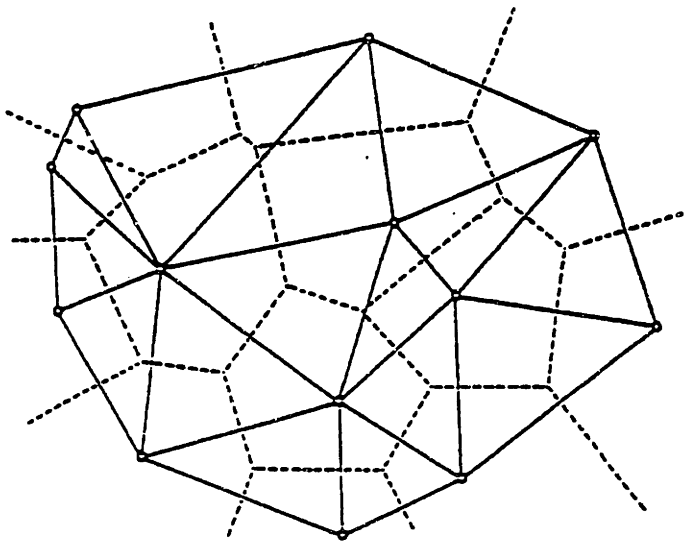
Delaunay triangulation is considered suitable for finite element analysis, as this triangulation maximizes the sum of the smallest angles of the triangles. It creates triangles as equilateral as possible for the given set of points; thus ill-conditioning and thin triangles are avoided whenever possible [Sibson 78]. The same property holds in three dimensional Delaunay tetrahedra; the triangular faces of the tetrahedra are as equilateral as possible.

One problem to be noted here is that the union of Delaunay triangles is a superset of the domain, so that some extraneous triangles must be deleted. It is also necessary to insure that two adjacent points on boundaries are connected. Triangulation with such constraints is sometimes called a constrained Delaunay triangulation, DDT (Domain Delaunay Triangulaion), or a generalized Delaunay triangulation. It is shown in [LeeDT 86] that the constrained Delaunay triangulation also has the maximum of minimum angles property. Actual algorithms for DDT are described in [Sapidis 91a, Sapidis 91b, Sapidis 92].

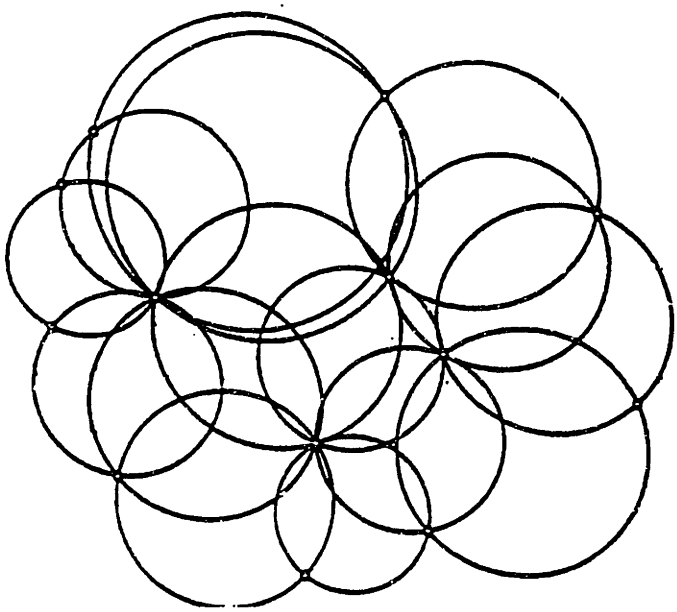
Thus far we have reviewed methods proposed for node placement and node connection. Although most approaches place nodes at one time and then connect them once, some methods utilize two stages of node placement and connection in an iterative manner[Frey 87, Ruppert 92, Ruppert 93].

In [Frey 87], a set of nodes is first inserted along the boundaries of a 2D domain. A triangular mesh is then created by Delaunay triangulation. Usually many of these triangles are too large and ill-shaped. In such a triangle, a new node is introduced and Delaunay triangulation is updated. Nodes are thus iteratively added to the domain until the prescribed node spacing is satisfied in the domain. In these approaches, the two stages of node placement and connection are repeated every time a new node is introduced.

The most critical problem in node placement and connection methods is where to place nodes so that they are connected to yield well-shaped triangles or tetrahedra. This is especially difficult when an arbitrary varying node spacing is specified over the domain to be meshed. None of the proposed methods can satisfy continuously changing node spacing requirements without user intervention.



(a) Dirichlet tessellation (dashed lines) and Delaunay triangulation (solid lines)



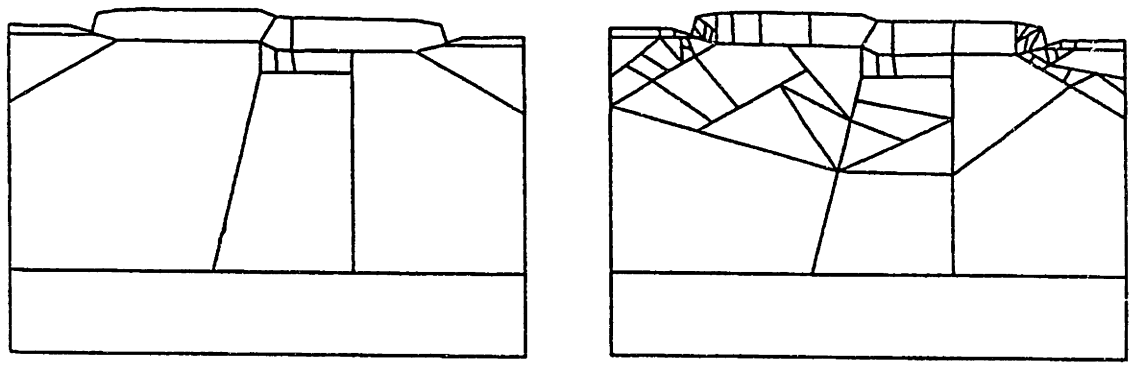
(b) Circumcircles

Figure 2.3 Dirichlet tessellation, Delaunay triangulation, and circumcircles for the same point set. (Adapted from [Cavendish 85])

2.1.2 Coarse Domain Decomposition

In this process, the domain to be meshed is decomposed into a set of coarse sub-domains. This process must be followed by other techniques such as: template mapping; node placement and connection; and element-level domain decomposition -- these sub-domains are very gross and are thus not appropriate in many applications.

In methods proposed in [Joe 91a, Bykat 83], the original domain is decomposed into simple convex sub-domains, in which nodes are next inserted for further decomposition to the element level. Figure 2.4, adapted from [Joe 91a], illustrates an example of such domain decomposition in 2D. The first stage, shown in Figure 2.4(a), decomposes the domain into convex polygons such that small angles are avoided. The second stage in Figure 2.4(b) then further subdivides the convex polygons into smaller convex parts automatically, the sizes of which are controlled by a given node spacing. In the third stage, not shown in the figure, mesh nodes are inserted on a quasi-uniform grid in each convex polygon. Finally nodes are connected by Delaunay triangulation in each convex polygon. It is reported that the approach works similarly for 3D tetrahedrization [Joe 91c].



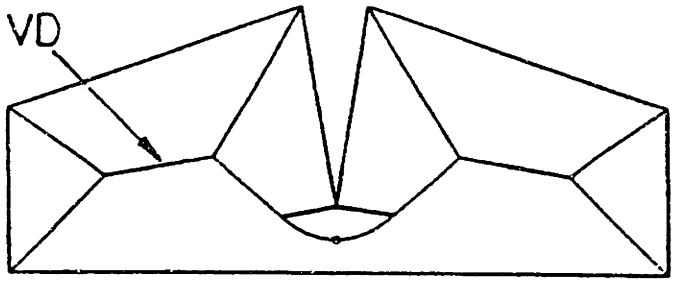
(a) Original domain (b) Convex polygons

Figure 2.4 Decomposition of a 2D domain into convex sub-regions. (Adapted from [Joe 91a])

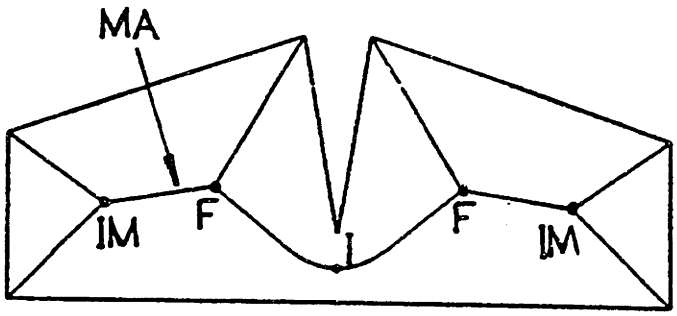
Another rather recently studied approach to coarse domain decomposition uses a medial axis transformation (MAT) [Gursoy 89, Gursoy 91, Gursoy 92a, Gursoy 92b, Srinivasan 90, Armstrong 91]. In a 2D domain, a medial axis is a subset of the Voronoi diagram of the object edges and vertices -- the Voronoi diagram defines sets of points equidistant from two elements, i.e., edges or vertices, of the 2D object boundary. (See Figure 2.5 (a) for Voronoi diagram and (b) for medial axis.) More detailed discussions on these issues can be found in [Gursoy 89, Srinivasan 90, Blum 78, Blum 73, Patrikalakis 90].

Figure 2.5 portrays Gursoy and Patrikalakis's 2D mesh generation method, adapted from [Gursoy 91]. In the method, a 2D domain is defined by outer boundaries and inner boundaries, each represented as a series of boundary elements, i.e., curve segments and vertices. Free-form boundary curves, such as Bezier and B-spline curves, are approximated within a prescribed tolerance by line segments and circular arc segments. Artificial branches of the medial axis newly introduced by the polyline approximation are selectively deleted wherever angles between two adjacent medial-axis-branches are smaller than a given threshold. The resultant medial axis is calculated analytically by offsetting boundaries directed towards the interior of a region. This process generates a medial axis consisting of straight line segments, parabolas, ellipses, and hyperbolas, as shown in Figure 2.5 (b). This completes the first stage of coarse subdomain decomposition. In the next stage, shown in Figure 2.5 (c), these subdomains are further decomposed into simpler parts by introducing Voronoi edges and cutting edges at initial branch points. If necessary, these simple parts can be further subdivided into finer triangles using a simple procedure, as shown in Figure 2.5 (d).

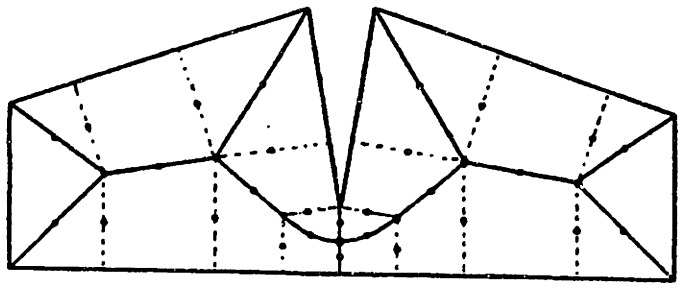
Another type of algorithm, based on boundary representation, is proposed in [Wordenweber 81, Wordenweber 84, Woo 84]. Wordenweber introduced three operations to extract tetrahedral elements from a 3D rectilinear domain: OP_1 removes a tetrahedron from a convex corner by introducing a single cutting plane; OP_2 extracts a tetrahedron from a convex edge with two cutting planes; and OP_3 creates a new vertex inside the domain and removes a tetrahedron with three cutting planes which intersect at the new vertex. It is shown that these operators suffice for simple 3D manifolds which do not include internal cavities or through-holes. For a general 3D manifold, two more operations are required: OP_j connects an internal cavity to the outer surface of a polyhedron; and OP_p cuts a toroid at a triangular cross-section. These operators for the 2D/3D domain are illustrated in Figure 2.6. Resultant coarse sub-domains are then refined into finer tetrahedral elements.



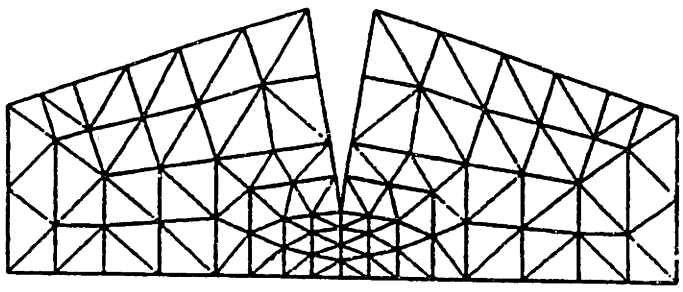
(a) Voronoi diagram



(b) Medial axis

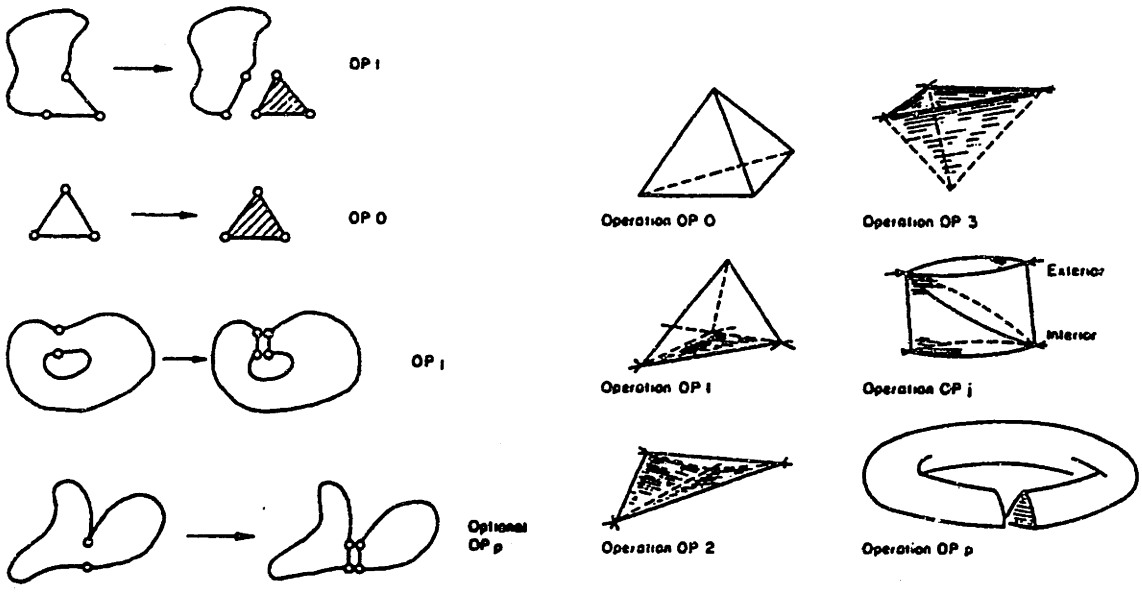


(c) Subdomains



(d) Fine triangle mesh

Figure 2.5 Coarse domain decomposition with MAT. (Adapted from [Gursoy 91])



(a) 2D operations (b) 3D operations

Figure 2.6 Operations for triangulation. (Adapted from [Wordenweber 84])

2.1.3 Mesh Template Mapping

When used for 2D meshing and surface meshing, the mesh template mapping technique maps a prescribed simple mesh template, such as square grid and a unit triangle, into a given four- or three-sided patch using a blending function. A similar blending function can be utilized for 3D meshing, in which a unit cube or tetrahedron is mapped to a four- or six-sided volume in 3D space. This approach has been studied and employed for the past two decades [Cavendish 77, Cavendish 84, Cohen 80, Cook 74, Crawford 87, Gordon 71, Gordon 73, Harber 81, Harber 82, Imafuku 80, Wellford 88, Zienkiewicz 71, Stephenson 89, Blacker 88].

The mapping technique has been one of the most popular approaches in commercial software, usually employed after a complicated domain is subdivided into a set of simple subdomains manually. Some reasons for this method's popularity are: (a) the method has the longest history; (b) the method is computationally very effective once coarse domain decomposition is performed by some other method; and (c) template mapping can be

consistently applied to 2D regions, 3D surfaces, and 3D volumes. However, some serious problems exist: (a) it is not easy to control node spacing distributed over the domain, as a mapped mesh is uniform in most cases; and (b) the method by itself is not complete, and thus cannot be fully automatic, unless an efficient and robust automatic coarse domain decomposition is incorporated.

2.1.4 Element-Level Domain Decomposition

Element-level domain decomposition refers to the process of subdividing a domain to the element-level by one of two methods: (a) iterative element extraction [Lo 89, Peraire 88, Lohner 88, Sadek 80, Tracy 77, Bykat 76, Lindholm 83, Blacker 91]; and (b) recursive domain splitting to the element level[Bykat 83, Sluiter 82, Schoofs 79]. In the former method, mesh elements are extracted iteratively one by one starting at the boundary -- thus the original domain is "eaten" inward from the outside. Because most of the approaches in this category modify and advance the boundary inward step-by-step, they are called "advancing front methods". The other element-level domain decomposition method recursively splits the domain into halves until all the subregions are appropriate mesh elements.

Iterative Element Extraction

In [Tracy 77], an iterative procedure is described which extracts one or two triangles from the boundary of a 2D domain. The algorithm first removes all vertices whose angle is less than 90 degrees by extracting the corner triangle. Then corners with angles between 90 and 180 degrees are cut off by two triangles that share a newly introduced cutting edge at the corner. The above process is repeated until the whole domain is triangulated. Similar algorithms that cut off triangles or four-sided polygons from the boundary are presented in [Sadek 80, Lindholm 83].

In [Blacker 91], element extraction is performed more systematically to create a quadrilateral mesh for a 2D domain. A meshing front that starts from the exterior boundary is advanced inward and counterclockwise, while a meshing front that starts from an interior boundary progresses outward and clockwise. The whole algorithm is rather complicated, involving a number of different operations which must be tightly controlled to ensure mesh validity and quality. First, the beginning and ending node of the next sequence or row of elements to be added is found (row choice). Then, each

paving boundary is checked to see it has more than six nodes; otherwise a prescribed mesh template is mapped to fill the interior (closure check). If the paving boundary has more than six nodes, the boundary is advanced (row generation). Then the mesh is smoothed (smoothing), and seamed, i.e. small interior angles are closed by connecting opposing elements (seam). If a self-intersection occurs, it is adjusted and connected to form new paving boundaries (intersection).

The above process is repeated until a complete row is created. Once the row is done, the sizes of elements in the row are checked to avoid overly small and overly large elements. Ill-sized elements are adjusted by the introduction of irregular elements, called tucks or wedges (row adjustment).

The processes of intersection and seaming are then applied again to the modified mesh. The whole process is repeated until only loops with six or fewer nodes are left. An example of such a process is illustrated in Figure 2.7, adapted from [Blacker 91]. The method works quite well for a uniform node spacing. A graded mesh can be generated by changing the node spacing on boundaries -- interior element size is not explicitly specified, but is 'interpolated' from boundary node spacing.

Recursive Domain Split to Element

Recursive subdivision of a 2D domain is presented in [Bykat 83]. Before this method is applied, the original domain is subdivided into a set of convex subdomains using a method described in [Bykat 76] (a coarse domain decomposition). Nodes are then inserted on the boundary of a convex subdomain, so that it is divided into two parts by introducing a split line. After new nodes are inserted on the split line according to node spacing requirements, the splitting process is recursively applied to the two halves until they become triangular elements.

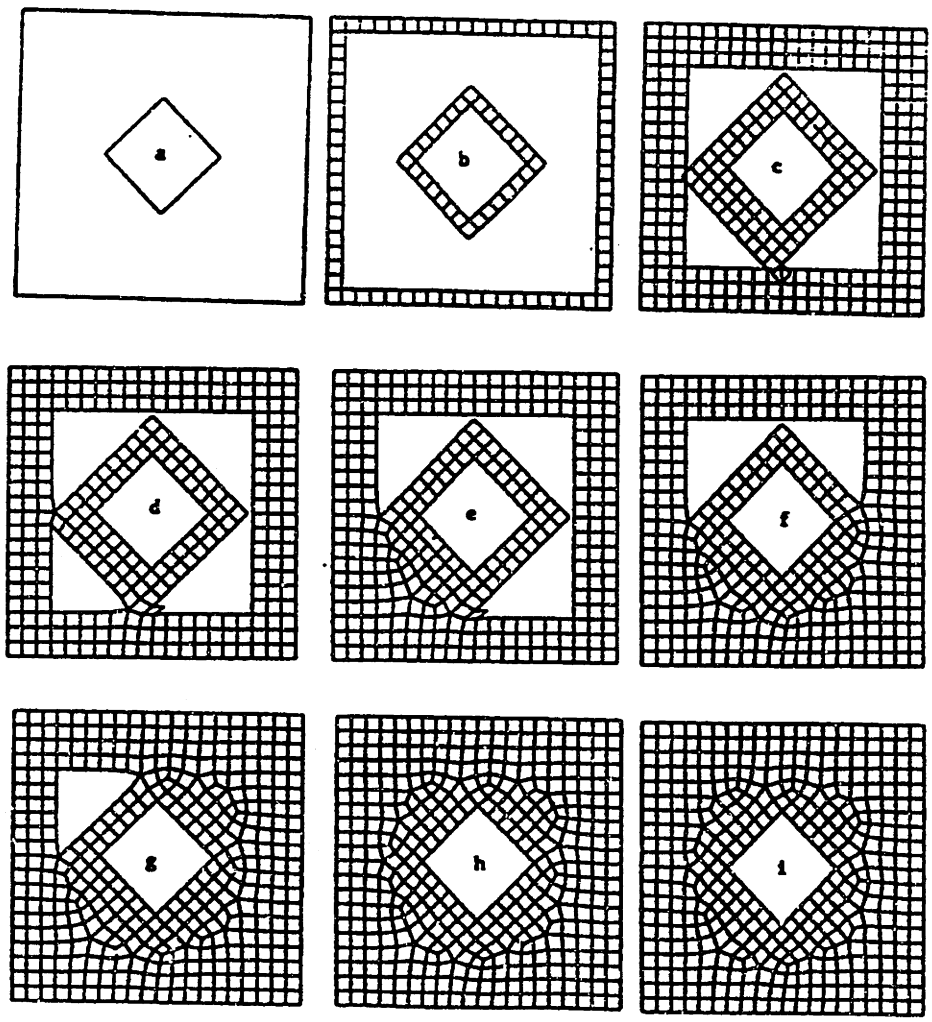


Figure 2.7 Paving for quadrilateral 2D mesh generation. (Adapted from [Blacker 91])

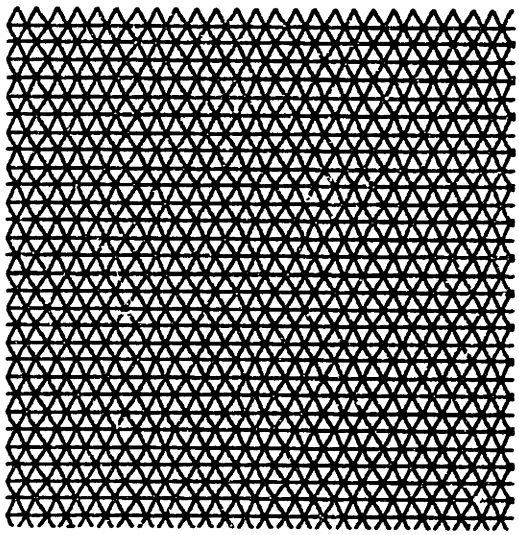
2.1.5 Grid-Based Spatial Subdivision

Grid-based spatial subdivision methods superimpose a uniform/hierarchical, 2D/3D, triangular/quadrilateral grid onto the domain to be meshed. Such methods are typically followed by a two-stage procedure: (1) classification of grid elements into three types, i.e., inside/outside/on-boundary; and (2) adjustment of on-boundary elements to be consistent with the original boundary. Methods of grid-based spatial subdivision are classified in two groups for further review: (a) uniform grid based spatial subdivision [Thacker 80b, Shimada 91, Heighway 82, Kikuchi 86]; and (b) hierarchical grid based spatial subdivision [Schroeder 90, Schroeder 88, Shephard 91, Shephard 86, Yerry 84, Yerry 83, Perucchio 89, Shephard 91].

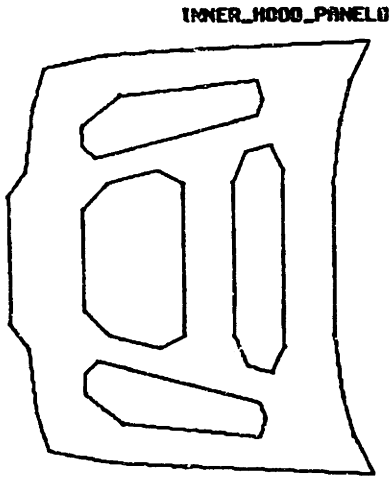
Uniform Grid Based Spatial Subdivision

This is probably the most simple method of mesh generation -- a uniformly-sized grid is superimposed onto a domain and elements in the vicinity of the boundary are moved and adjusted so that nodes are placed on the exact boundary. A simple implementation of the algorithm for a 2D domain is shown in Figure 2.8 [Shimada 91]. The regular triangular grid in Figure 2.8 (a) is superimposed onto a 2D domain shown in (b). Then all the grid nodes are classified into four types: inside, outside, inside-next-to-boundary, and outside-next-to-boundary, as shown in Figure 2.8 (c). In the final stage, all outside nodes are removed, and some of the inside- and outside-next-to-boundary nodes are selectively pulled to the boundary, yielding the mesh in Figure 2.8 (d). This selective boundary adjustment requires case-by-case handling in order to classify the positional relationships between boundary and inside-/outside-/next-to-boundary nodes.

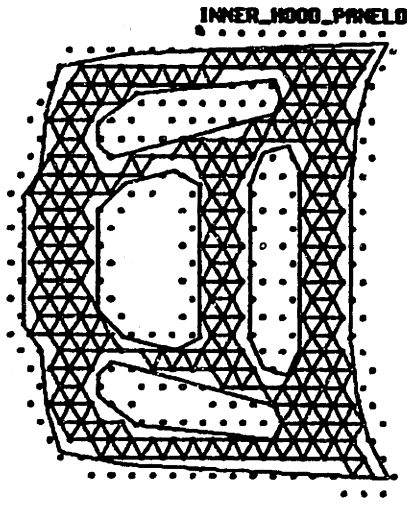
Obvious limitations of such simple algorithms are: that element size is not controllable; and that very thin elements can be created around boundaries. The former limitation is rectified by introducing a hierarchical spatial subdivision scheme, described next.



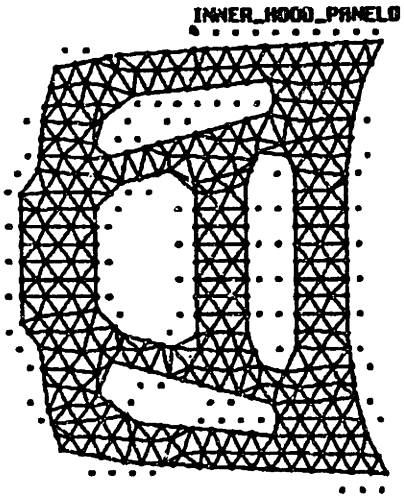
(a) Uniform triangular grid



(b) 2D domain to be meshed



(c) Classification of grid nodes



(d) Boundary adjustment

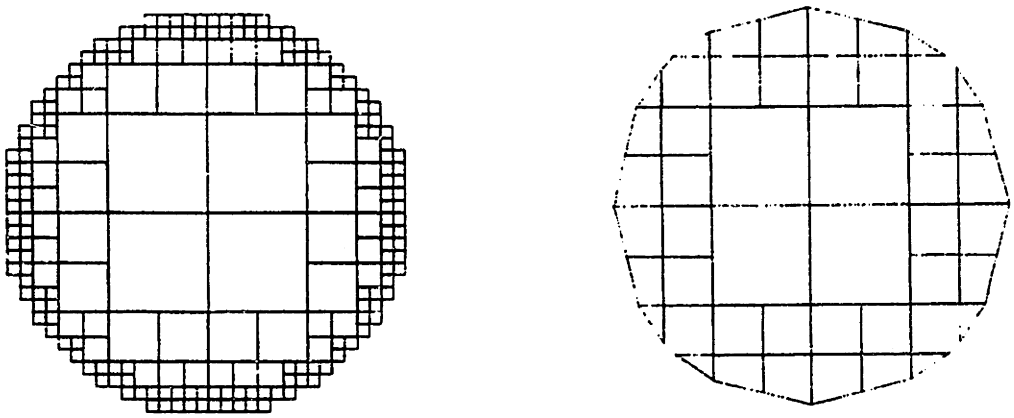
Figure 2.8 Uniform grid based spatial subdivision.

Hierarchical Grid Based Spatial Subdivision

This approach employs a modified quadtree and octree encoding for two-dimensional and three-dimensional domains, respectively. Here, a two dimensional case is explained based on Shephard and Yerry's work [Shephard 91, Yerry 83] . In the original definition of quadtree encoding for a 2D domain, a square encompassing the entire domain is created and subdivided into four quadrants. A classification is made for each quadrant to see if it is inside the domain (full), outside the object (empty), or overlapping the boundary (partial). Only partial quadrants are further subdivided into four next-level quadrants, which are then again classified into the three types mentioned above. This process is recursively repeated until an approximated boundary of square elements satisfies a given tolerance. Figure 2.9 (a) shows a quadtree decomposition of a circle.

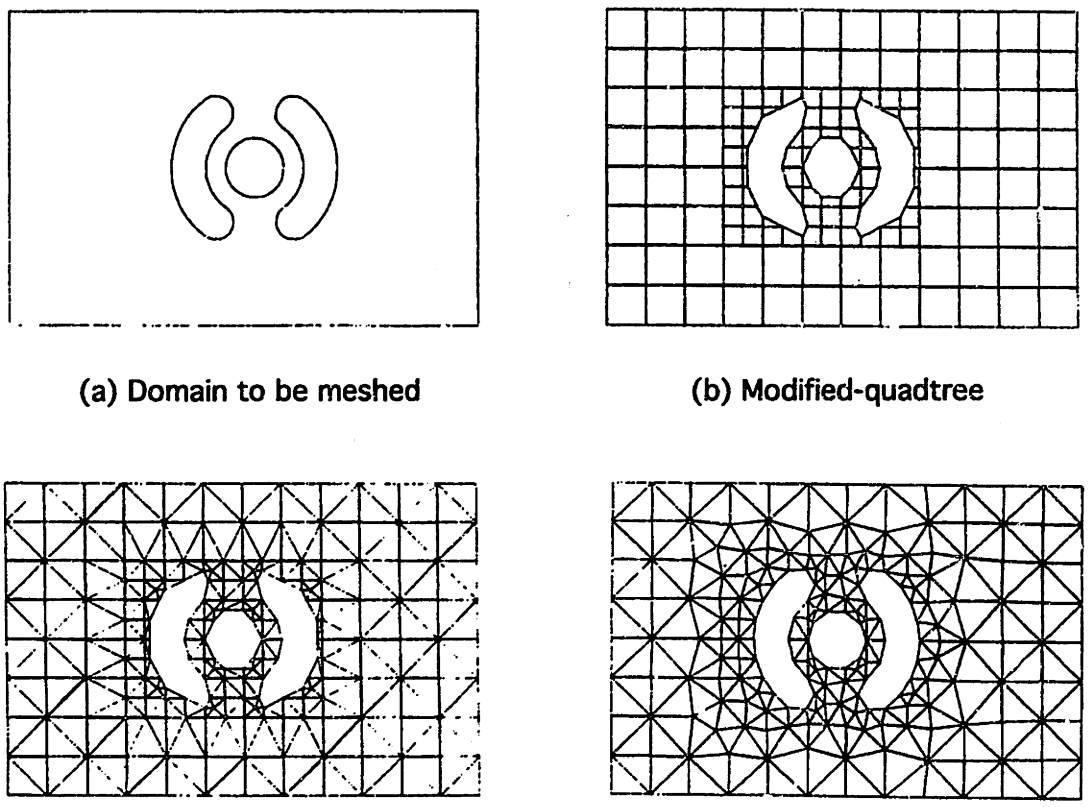
In Shephard and Yerry's modified quadtree methods, there are three major modifications specially devised for mesh generation. Firstly, the level of subdivision, which determines element size, is selected based on a desired node spacing, rather than a possible maximum size as in the original quadtree. Secondly, the difference in element size between adjacent squares is limited to a specified number, usually one. This is to avoid skinny triangles created by connecting nodes in adjacent squares of significantly different element sizes. Finally, the most essential modification is that the quadrant on the boundary may have cut corners. With these cut squares on the boundary, undesirable 90-degree re-entrant corners are eliminated, and thus all unnecessarily small squares are removed. Figure 2.9(b) shows a modified-quadtree decomposition of a circle.

In Figure 2.10, an example of a complete 2D meshing process using modified-quadtree is illustrated. The modified-quadtree in Figure 2.10(c) is further subdivided into a triangular mesh by introducing more nodes inside cut squares. The final two steps are: (1) pulling boundary nodes exactly onto the domain boundary; and (2) smoothing the locations of the internal node points. The smoothing is essential, as this type of grid-based approach creates badly-distorted elements around boundaries -- yet boundary nodes placed too close are not completely fixed, so that tiny or thin elements remain unsmoothed.



(a) Quadtree for a circle (b) Modified-quadtree for a circle

Figure 2.9 Quadtree and modified-quadtree for a circle. (Adapted from [Shephard 91].)



(a) Domain to be meshed (b) Modified-quadtree

(c) Initial mesh (d) Boundary adjustment and smoothing

Figure 2.10 2D mesh generation based on modified-quadtree. (Adapted from [Shephard 91].)

2.1.6 Faceting Parametric Surfaces in Parametric Space

Methods in this category are specially designed for faceting parametric surfaces in parametric space and are mainly pursued in the computer graphics community. The major goals of these methods are speed and compactness of data, or small number of facets -- the element shape regularity is usually not the focus in surface rendering.

A uniform faceting of trimmed surfaces is described in [Rockwood 89]. As the method is designed for real-time surface rendering, it requires rather short computational time. The mesh quality, however, is not a goal; mesh size is not controlled, and many thin triangles are created around the surface boundary. [Dolenc 90] presents a similar algorithm for trimmed surfaces. Recent work [Sheng 92] considers surface triangulation requirements for rapid prototyping and offers an algorithm to triangulate a trimmed surface with an error bounding condition. The method is rather quick because it is performed completely in parametric space. It, however, creates thin triangle elements, which would be inappropriate for other applications like FEM/BEM analysis.

2.2 POST-PROCESSES FOR MESH IMPROVEMENT

2.2.1 Mesh Smoothing and Mesh Relaxation

In general, some mesh elements produced by combinations of the algorithms described in Section 2.1 will be ill-shaped or too skinny. The goal of smoothing is to obtain more regular elements while keeping the same nodal connectivity. When finite element analysis is done using triangular elements, a set of virtually equilateral triangles gives a more accurate solution and quicker convergence. A thin triangle with an angle close to 180 degrees must be eliminated, and it is desirable to avoid obtuse angles as much as possible [Babuska 76, Deljouie-Rakhshandeh 90].

All smoothing does is to reposition nodes by averaging adjacent node positions with weighting factors. Laplacian smoothing, the simplest and most widely-used technique, moves each interior node to the centroid of its connected neighbors in an iterative fashion. Although this scheme is easy to implement and in most cases improves the appearance of a mesh by reducing small angles, it has a couple of significant limitations which must be considered for practical uses.

First, in some situations such as those illustrated in Figure 2.11, a centroid of connected nodes may be moved to the exterior of the domain, destroying the mesh topology validity. This never happens for a convex domain; in a general non-convex case, however, the validity of the mesh must be checked after every iteration.

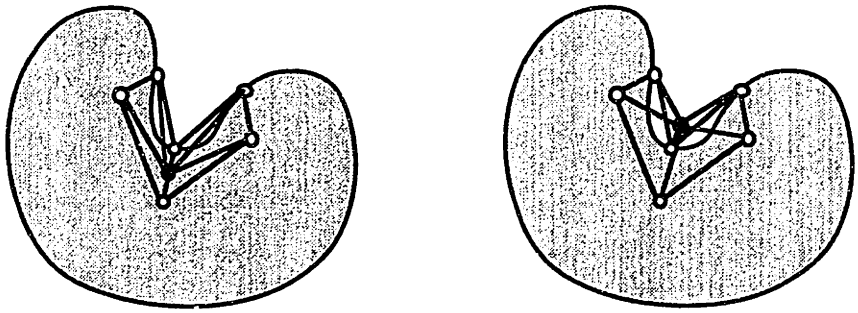
The second problem is that topological irregularity, i.e., a node that has too few or too many neighbors, is not fixed by Laplacian smoothing. In two dimensions, the topologically regular triangular mesh is a hexagonal arrangement where each node has six neighbors. If a node has only four neighbors, the best we can do is to have four 90-degree angles; usually at least one of the angles is obtuse. Solutions to overcome this problem were recently studied by Field [Field 88] and Frey [Frey 91]. In Field's Laplace-Delaunay smoothing, nodes are reconnected from time-to-time in the iterations of Laplacian smoothing. Frey proposed a mesh relaxation method which reduced the number of irregular nodes by swapping edges.

The third problem is that Laplacian smoothing is useful only for internal nodes. If two adjacent boundary nodes are placed too close to each other, which often happens in grid-based spatial subdivision methods, a triangle containing these nodes tends to become skinnier. Thus, Laplacian smoothing does not always improve a mesh.

In [Shepherd 91], a rather complicated two-step smoothing scheme is employed for improving a mesh that has high irregularity around its boundary, a typical bi-product of octree-based methods. The first step is constrained Laplacian smoothing applied only on the nodes on the boundary. Then adjacent to the boundary, a node is moved discretely along the line from its current position to the target location, the centroid of neighboring nodes. The second step of the smoothing considers only nodes belonging to elements whose shape measure is less than a threshold. The optimal location of such a node is determined on a line between the original point and the target point, such that the new location maximizes the worst shape among all the adjacent elements. The second step is repeated for a fixed number of times or until no elements remain with shape measures below the threshold.

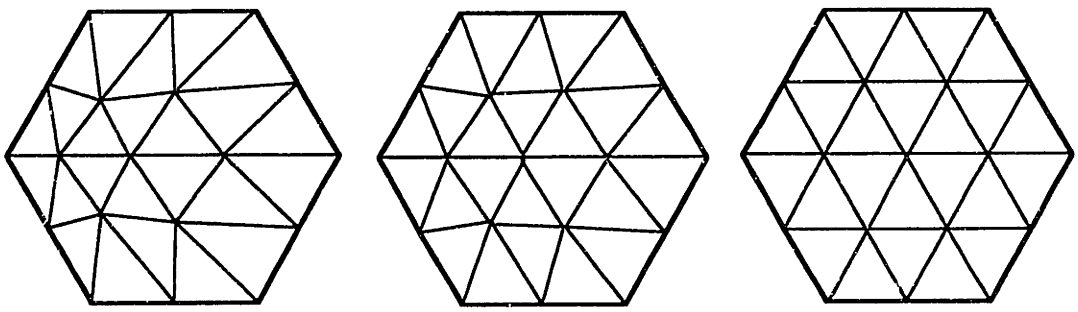
Up to this point, we have reviewed three limitations in Laplacian smoothing and some remedies for them. There is another problem not considered in any of these smoothing methods, the problem best illustrated in Figure 2.12. In the figure, a topologically regular 2D mesh with a perfect hexagonal node arrangement, shown in (a), is smoothed by Laplacian smoothing as shown in (b). As the smoothing is repeated, the mesh asymptotically approaches the equilateral mesh as shown in (c). This final mesh is the best one only when a uniform node spacing is desired. In other words, if the original node locations in Figure 2.12 (a) are generated in such a way that they satisfy a given

node spacing distribution, the Laplacian smoothing will destroy this controlled node spacing by making it more uniform. In a situation where precise node spacing control is required, therefore, it is desirable to employ a mesh generation scheme that can generate a good quality mesh without performing smoothing.



(a) Original mesh (b) After the Laplacian smoothing

Figure 2.11 A case where Laplacian smoothing does not work. The middle node is moved to exterior of the boundary, causing an invalid mesh.



(a) Original mesh (b) Laplacian smoothing (c)More Laplacian smoothing

Figure 2.12 A case where Laplacian smoothing destroys an original node spacing distribution.

2.2.2 Element Type Conversion

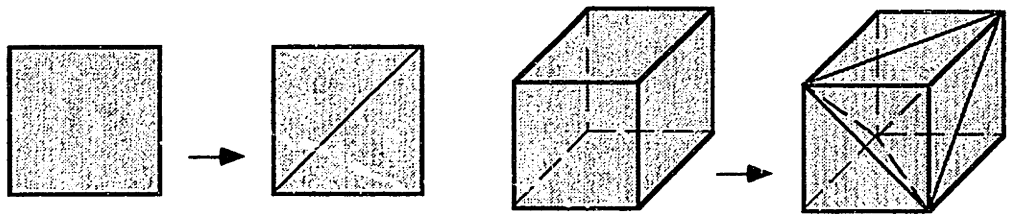
2D and 3D mesh elements are classified into two types: triangular elements and quadrilateral elements. Most mesh generation algorithms generate either triangular elements or quadrilateral elements. Only mesh template mapping followed by a coarse domain subdivision can easily generate both types of elements by selecting a proper template for each element type. All other methods have strong preference: node placement and connection algorithms using Delaunay triangulation generate only triangular elements; advancing front methods can be devised to create both type of elements, but with quite different algorithms and implementations; grid-based special subdivision methods generate triangular elements or a combination of two types, i.e., quadrilateral inside and triangular around the boundary.

Since some analysis methods and computer analysis software exclusively require one of these two types, two-way automatic conversion methods are desired: (a) from quadrilateral to triangular; and (b) from triangular to quadrilateral. The processes can be performed by introducing some cuts. In the former conversion, a square is divided into two triangles by introducing a diagonal edge, and a cube is divided into five tetrahedra by three cutting planes, as shown in Figure 2.13. These types of cutting operations can be employed to convert triangular elements to quadrilateral elements, as shown in Figure 2.14.

In the first method, a triangle is divided into three quadratic elements and a tetrahedron into four quadrilateral elements. Although the idea is simple as is the implementation, this simple process introduces a significant topology irregularity into a mesh -- when applied to a 2D hexagonal grid, only new nodes inserted on edges of triangles have the regular number of neighbors, four, while an original node is surrounded by six neighbors and a new node inserted in the middle of a triangle has only three neighbors.

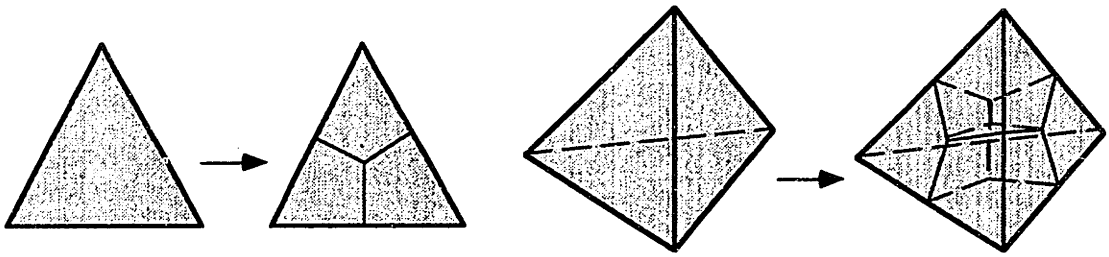
For 2D problems, more sophisticated ways to convert triangles to quadrilateral elements are studied in [Heighway 83, Johnston 91]. Heighway presents a technique for combining two adjacent triangles into a quadrilateral. Isolated triangles remaining in the mesh are then combined by moving them toward each other until they become adjacent and can be combined. Proposed by Johnston is a three-step procedure; (1) initialization, in which boundary information is extracted from mesh data and Laplacian smoothing is applied; (2) corner- and boundary-elements are identified and prioritized, then element by element conversion is performed by coupling elements, splitting a coupled element, and

propagating the split to maintain the conformity, while avoiding angles below 45 degrees; and (3) combining all isolated triangles into adjacent quadrilaterals and dividing the combined five-sided elements into three quadrilaterals by introducing nodes inside. Figure 2.15, adapted form [Johnston 91], illustrates an example of such triangular-to-quadrilateral conversion.



(a) 2D element conversion (b) 3D element conversion

Figure 2.13 Convertion from a quadrilateral element to triangular elements by introducing cuts.



(a) 2D element conversion (b) 3D element conversion

Figure 2.14 Convertion from a triangular element to quadrilateral elements by introducing cuts.

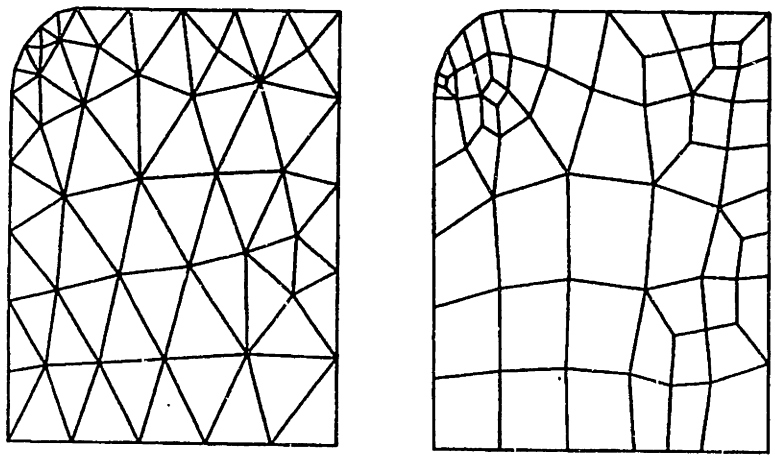


Figure 2.15 An example of converting 2D trianguluar mesh to quadrilateral mesh. (Adapted from [Johnston 91].)

2.2.3 Polygonal Surface Remeshing

This type of post-processing repolygonizes a polygonal surface with application-specific criteria, such as: (a) represent a surface with fewer polygons, or (b) uniformly distribute nodes, i.e. evenly spaced nodes. Most of the methods proposed in the computer graphics community deal with the transformation of one triangulated surface into another one for rendering.

[Turk 91] converts an irregularly sized/shaped polygonal mesh to a uniformly-sized/well-shaped polygonal mesh for texture mapping. He also presents a method called *re-tiling*, a process of simplifying a polygonal model in [Turk 92]. In this process, repelling forces are defined around vertices, and their locations are decided by a relaxation procedure. [Schroeder 92] proposes a *decimation algorithm*, which reduces the total number of triangles while preserving important form features as accurately as possible.

2.3 LIMITATIONS IN PREVIOUS METHODS

In Section 2.1, we have reviewed the key sub-processes in mesh generation: (a) node placement and connection; (b) coarse domain decomposition; (c) mesh template mapping; (d) element-level domain decomposition; (e) grid-based spatial decomposition; and (f) faceting parametric surfaces in parametric space. Many variations of complete meshing procedures have been proposed using one or a combination of these sub-processes with post-processes, such as a smoothing operation and/or element-type conversion.

Each method has its own strength and weakness, and none of them satisfies all of the requirements listed in Section 1.5. The following are critical limitations in existing methods.

Poor Node Spacing Control

Many methods do not allow continuous control over node spacing based on arbitrarily defined functions. Some methods have only node spacing control on the boundary, and internal node spacing is determined by interpolating the boundary node spacing. Others

like hierarchical spatial subdivision based methods only provide discrete element sizes. A method that can create a continuously graded mesh is desired.

Ill-Shaped Elements

All methods create some ill-shaped elements, or topologically irregular nodes, i.e., nodes connected to too many or too few neighboring nodes. To eliminate these ill-shaped elements, a smoothing procedure--in many cases Laplacian smoothing with slight modification--is employed. However, there is no guarantee that the smoothing operation will eliminate the ill-shaped elements, and smoothing may possibly destroy the node spacing distribution intended in the original mesh generation process -- in the worst case, it creates an invalid mesh. Ideally, the original mesh should be so well-shaped that no smoothing is required.

Inconsistent Applicability to Planar, Surface, Volume, and Hybrid Domains

Most algorithms are specially devised for only one problem domain, and not applicable to the other domains consistently. A procedure consistently applicable to all domains, planar, surface, volume, and hybrid, is desired. Meshing 3D geometry bounded by free-form curves and surfaces has not been well sutudied.

No Effective Adaptive Remeshing Capability

Most methods do not have effective remeshing capability for cases when domain geometry and the prescribed node spacing are continuously changing, for example in the large deformation analysis problem. There is no effective mechanism for utilizing the previous mesh in the remeshing process, so a new mesh has to be created completely from scratch every time a domain shape or a prescribed node spacing changes.

The goal of this thesis is to develop a new mesh generation scheme that can solve all the problems mentioned above.

3

Approach Overview

As described in the previous Chapter, the general meshing problem is very demanding, and it is not easy to devise a geometric algorithm that creates a high-quality mesh for domains of arbitrary shape. However, you can often see naturally-occurring objects with geometry and topology similar to that of an ideal mesh. A repeating hexagonal pattern is an especially common one -- an equilateral triangular mesh is obtained by connecting the center points of the hexagons.

What kind of physical effects create these patterns? Is it possible to make a computer model that mimics nature's shape creation process, and use it to generate a high-quality mesh? Questions such as these motivated our development of a physically-based mesh generation method, called *bubble mesh generation* .

This Chapter describes the framework of this new approach to the mesh generation problem. Section 3.1 introduces the idea of physically-based approach to geometric problems, presents shapes in nature that show repeating hexagonal patterns, and proposes a novel mesh generation method via bubble packing, or sphere packing. In Section 3.2, a physically-based model, a bubble system, is defined, and then a procedure of triangulation based on this model is described.

3.1 MOTIVATIONS

3.1.1 Physically-Based Approach

Some geometric problems can be solved by physical experiments. One such example is Steiner's problem, in which three points A, B and C are to be joined by a system of paths of minimal total length on the plane [Kappraff 90]. Mathematically, this is called a minimal network problem for three points. The solution of this problem is proved by Steiner in the following way:

- If three points form a triangle with no angle greater than 120 degrees, $\overline{AP}+\overline{BP}+\overline{CP}$ is the minimum path, where P is the joint with surrounding three angles of 120 degrees. (See Figure 3.1 (a).)

- If three points form a triangle with an angle greater than 120 degrees, either $\overline{AB}+\overline{BC}$, $\overline{BC}+\overline{CA}$, or $\overline{CA}+\overline{AB}$ is the minimum path. (See Figure 3.1 (b).)

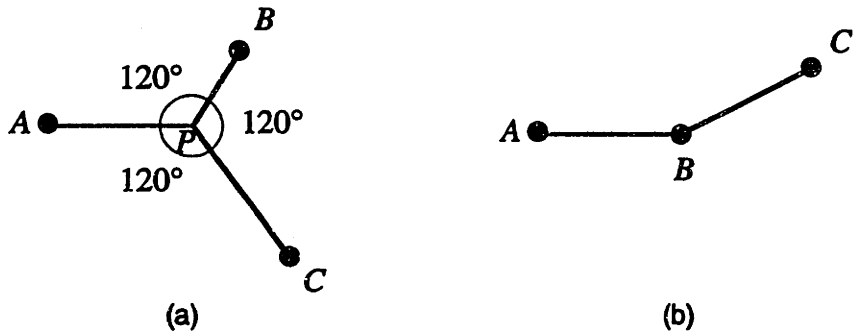
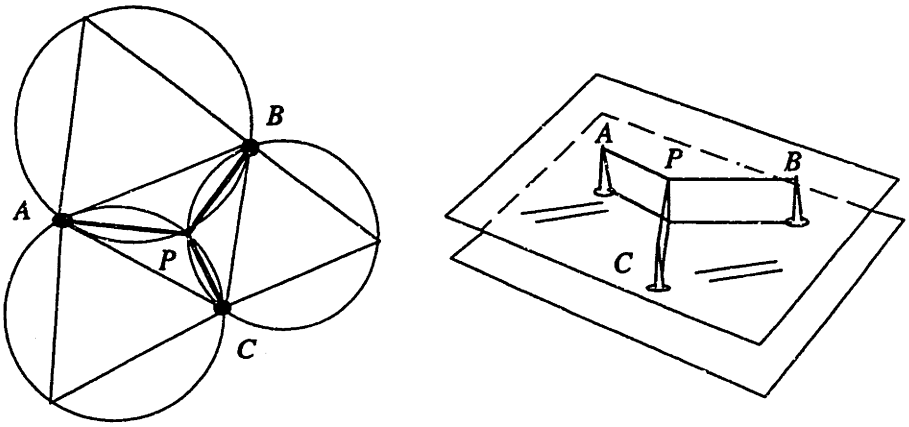


Figure 3.1 Minimal networks for three points. (a) If three points form a triangle with no angle greater than 120 degrees, $\overline{AP}+\overline{BP}+\overline{CP}$ is the minimum path. (b) If three points form a triangle with an angle greater than 120 degrees, either $\overline{AB}+\overline{BC}$, $\overline{BC}+\overline{CA}$, or $\overline{CA}+\overline{AB}$ is the minimum path (Redrawn from [Kappraff 90].)



(a) Geometric approach (b) Physically-based approach

Figure 3.2 A geometric approach and a physically-based approach.

The point P in the former case can be found with compass and straight-edge by the following steps, also shown in Figure 3.2 (a):

(1) Draw an equilateral triangle on each edge of triangle ABC.

(2) Draw a circle that circumscribes each equilateral triangle.

(3) The point P is found as the intersection point of the three circumscribing circles.

(Note that angles, $\angle APB$, $\angle BPC$, and $\angle CPA$ are all 120 degrees, as they are twice the inscribed angle, 60 degrees, that intercepts the same arc.)

It is also known that the problem can be solved by a physical experiment using soap films. Place three thumbtacks between two glass plates as shown in Figure 3.2 (b). By submerging the glass plates in a soap solution, we obtain the same solution found by the geometric approach mentioned above. If the three thumbtacks form a triangle with no angle greater than 120 degrees, the soap films will construct a Y-shaped joint with 120 angles. The reason that the soap films prefer the 120 degree angle can be explained by a static force balance -- the three films AP, BP, and CP pull the joint P into film directions since the surface tension is the a same magnitude. These three force vectors are balanced out only if all three angles are 120 degrees.

Steiner's problem is a simple case in which a geometric problem can be successfully solved using physics. In a *physically-based approach*, we establish the governing equations of the underlying physics and solve the system numerically on computer. Usually a physically-based approach is computationally much more expensive than a

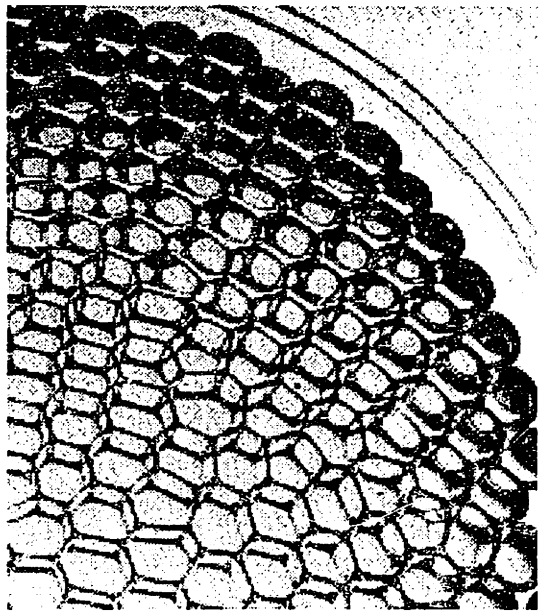
geometric approach, thus it is not an attractive approach when an efficient geometric solution scheme is known. It can, however, be advantageous when no efficient geometric algorithm is known, as is the case for the general mesh generation problem.

3.1.2 Shapes In Nature

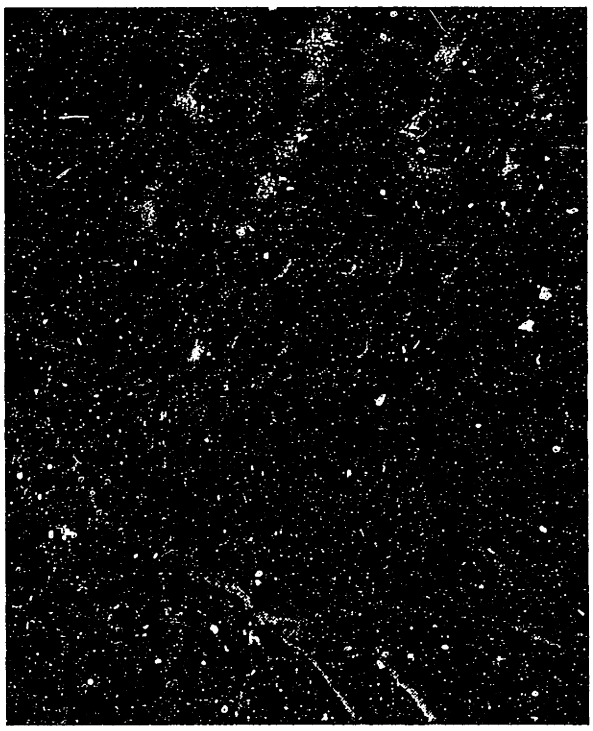
Topology and geometry similar to an ideal mesh are often found in nature. A repeating hexagonal pattern is a common one, observed in soap bubbles floating in liquid, compound eyes of insects, metal bonding, and many other natural phenomena. (See Figure 3.3a and Figure 3.3b.) The hexagonal pattern may form as a result of a variety of different physical factors, such as repelling/attracting forces, surface tension forces, thermo/fluid dynamics, and so on. It is therefore impossible to deduce solely from the pattern what forces acted to form it.

The symmetry and regularity in nature's forms and patterns have attracted many scientists and philosophers since ancient times. Why are there so many spherical shapes in nature? Why do bees construct hexagonal hives? Why do soap bubbles form a regular pattern? Although it is not easy to answer those questions, there is one common reason that explains these shapes, that is, optimization -- finding forms or pattern that minimize (or maximize) a particular quantity. For example, a stable configuration of soap films is called a "minimal surface," since it is a surface of minimal area. The quantity minimized is the potential energy of the surface tension. In a stable configuration of soap films, like the one in Steiner's problem, the potential energy must be at a local minimum. The corresponding mathematical branch concerning this optimization is called the calculus of variations, which is one of the most basic mathematical tools in modern science and engineering.

As we often see in nature, regular hexagons of the same size are capable of filling a 2D space without gaps. There are only two more regular polygons that can fill a 2D space without gaps, i.e., equilateral triangles and squares. These three space filling, or tiling, patterns are closely related to spherical packing. If we closely pack circles of the same size on a plane, we will obtain a hexagonal configuration as shown in Figure 3.4 (a), in which each circle touches six neighbors. This configuration corresponds to space filling by triangles and hexagons; a mesh of equilateral triangles is obtained by connecting the center points of neighboring bubbles, and a mesh of regular hexagons by drawing a circumscribing hexagon for each circle. On the other hand, space filling with squares corresponds to the packing of circles in a square array as shown in Figure 3.4 (b).

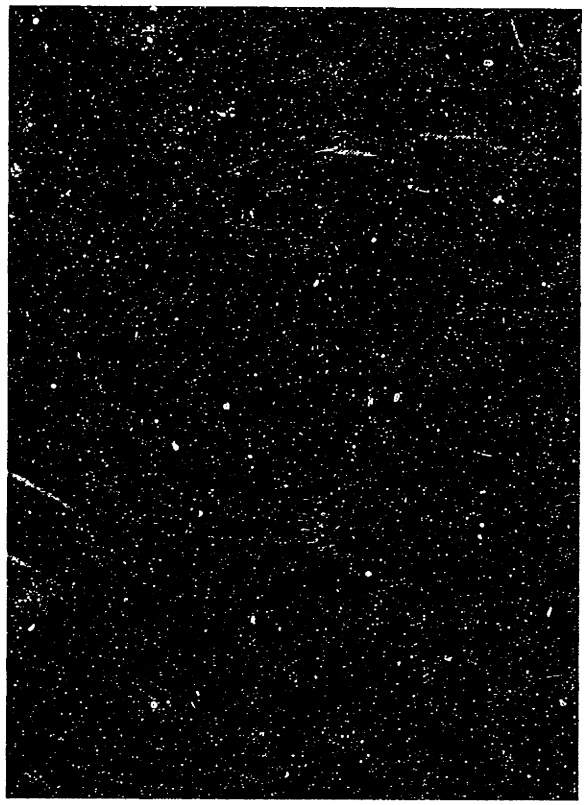


(a)

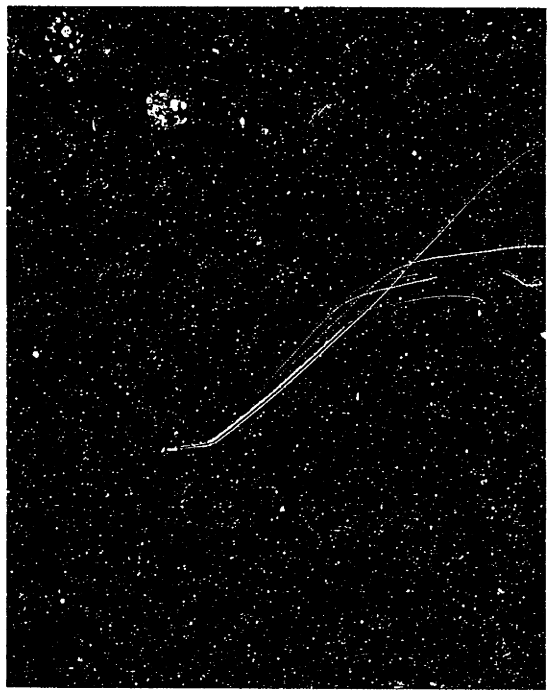


(b)

Figure 3.3a Repeating hexagonal patterns in nature. (a) Soap bubbles floating in liquid. (b) Cross-section of plant stem. (Adapted from [Burgess 87])

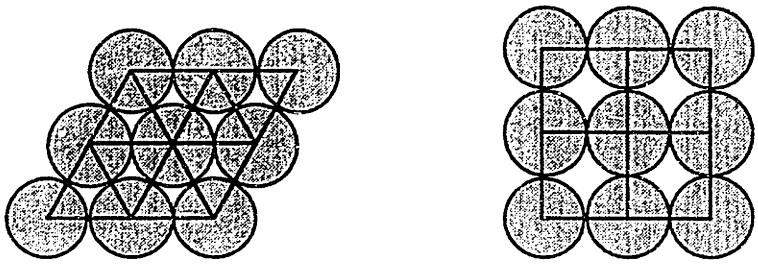


(a)



(a)

Figure 3.3b Repeating hexagonal patterns in nature. (a) Compound eyes on an insect. (b) Petals. (Adapted from [Burgess 87])



(a) Hexagonal packing (b) Square packing

Figure 3.4 Tightest packing of circles.

Note that the hexagonal packing is optimal in the sense that it minimizes the area not covered by circles -- the area in a hexagon not covered by a circle is smaller than the area in a square not covered by a circle. This is another reason why we see hexagons, with their 120 degree angles and Y-shaped linkages, so often in nature. The compound eye is composed of tightly packed light-sensitive units called ommatidia, each of which is cone-shaped, with the lens at the wider surface and the light-sensitive cells tapering inward. It is known that these ommatidia are arranged in hexagonal pattern, which makes sense since this is the most efficient arrangement in that it allows as many ommatidia as possible in a given surface area. The hexagonal packing is also observed in the process of cell growth, e.g., petals consists of closely packed cells. When cells grow at the same speed, they repel each other and eventually form a close hexagonal packing.

3.1.3 Mesh Generation via Bubble Packing

The physically-based model for mesh generation, presented as a "bubble system" in this thesis, was originally inspired by watching real soap bubbles floating in liquid as shown in Figure 3.3 (a). These bubbles are closely packed and they form a virtually regular hexagonal pattern. By connecting the bubbles' center points, a mesh of basically equilateral triangles is produced. To change the size of triangles, a temperature distribution can be applied, causing soap bubbles in the warm area to grow larger, and soap bubbles in the cold area to shrink.

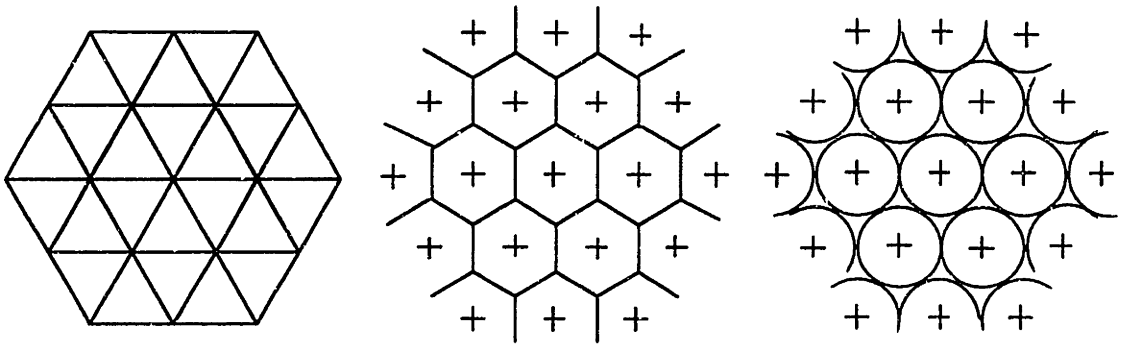
Thus the basic idea of the new meshing method proposed in this thesis can be summarized in two steps: (1) pack spheres closely in the domain, and (2) connect the center points by Delaunay triangulation, which selects the "best" topological connection for a set of nodes by avoiding the creation of triangles with small included angles

whenever possible. The latter step is the same procedure used in most of the *node placement and connection* algorithms reviewed in Section 2.1. The real challenge is the former step, namely, to obtain the optimum packing of spheres which satisfies a given node spacing.

The novelty of the proposed bubble method is that the close packing of bubbles mimics a pattern of Voronoi polygons, corresponding to well-shaped Delaunay triangles. Figure 3.5 depicts how Delaunay triangulation, Dirichlet tessellation, and sphere packing are related each other. As described in Section 2.1, Delaunay triangulation and Dirichlet tessellation are geometrical dual. In the case of uniform node spacing, a mesh of equilateral triangles is the best, and the corresponding Dirichlet tessellation is composed of regular hexagons. As shown in the upper example of Figure 3.5, the closest packing of uniformly-sized spheres corresponds to the regular hexagonal tessellation of the space.

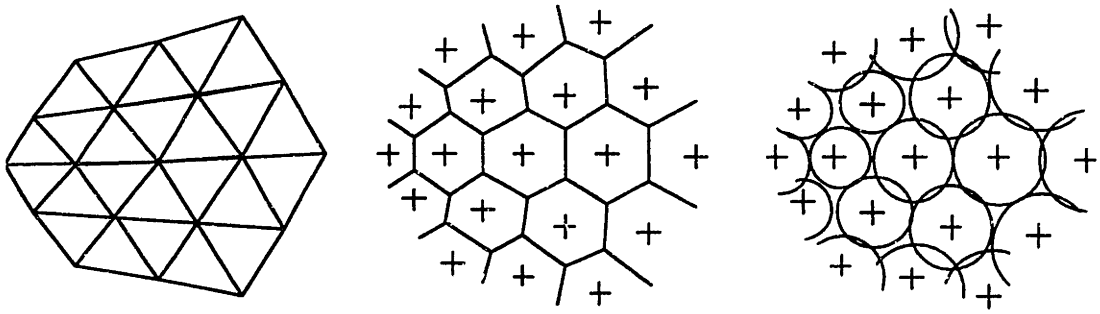
This relationship between Delaunay triangulation, Dirichlet tessellation, and sphere packing also holds for situations where spacing is non-uniform. The only difference is that spheres cannot always touch six neighbors so that some gaps and overlaps exist in the variable node spacing case (e.g., see the lower example of Figure 3.5). Our purpose, therefore, is to minimize these gaps and overlaps as much as possible using a physically-based model. In the actual implementation this is performed by defining repelling and attracting interbubble forces based on the proximity. With such a force field, a force balancing configuration of bubbles is a virtually closely packed configuration. In the context of the optimization mentioned in Section 3.1.2, the quantity minimized is the potential energy of the interbubble forces.

Thus far we have only considered two dimensional problems, i.e., how to generate a mesh by packing spheres on plane. The same concept, however, can be applied to one dimensional and three dimensional problems. Figure 3.6 shows unit of sphere packing in 1D, 2D, and 3D meshing. In this thesis, a computational model of a sphere with repelling and attracting forces is called a *bubble*.



Delaunay triangles Voronoi polygons Packed bubbles

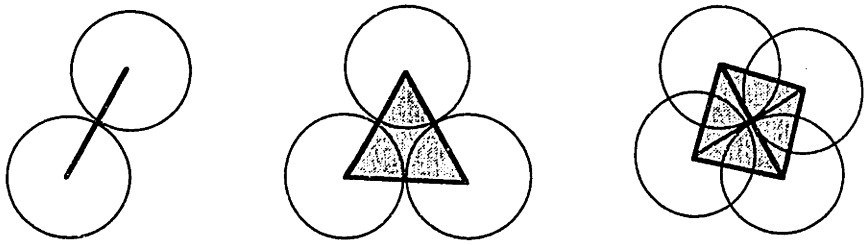
(a) Uniform node spacing



Delaunay triangles Voronoi polygons Packed bubbles

(b) Non-uniform node spacing

Figure 3.5 Delaunay triangulation, Dirichlet tessellation, and sphere packing.



(a) 1D element (b) 2D element (c) 3D element

Figure 3.6 Sphere packing units for 1D, 2D, and 3D meshing.

3.1.4 Tight Sphere Packing in 1D, 2D, and 3D

In this section, some "known" facts about dense sphere packing in 1D, 2D, and 3D are reviewed. There have been detailed studies of how uniformly sized spheres, which cannot intersect each other, are stacked to fill space, yielding various regular pattern [Williams 72]. Let us define two measurements: (a) the *kissing number*, which counts how many neighboring spheres can touch a single sphere; and (b) the *packing ratio*, which measures tightness, or efficiency, of packing. The packing ratio in 3D is defined as:

$$\text{Packing ratio} = \frac{\text{volume occupied by spheres}}{\text{whole volume}} \tag{3.1.1}$$

Similarly in 1D and 2D, ratios are defined by taking the ratio of length and area respectively instead of volume in Equation (3.1.1). The tightest, or closest, packing is the one with the highest packing ratio.

In 1D, there is only one way of tight packing, in which a sphere has two kissing neighbors, as shown in Table 3.1. Since the whole line is covered by spheres, the packing ratio is one. In 2D, there are two tight packing configurations as also discussed in the previous Section: the hexagonal lattice and the square lattice, the packing ratios of which are 0.9069 and 0.7854, respectively. The hexagonal lattice is the tightest 2D sphere packing.

In 3D, there exist many regular ways to pack equal spheres, some examples of which are shown in Table 3.2. Depicted in the top row of the table is the configuration called face-centered cubic lattice (fcc) or cubic close-packing (ccp) in crystallography. This is the tightest sphere packing in 3D space with the largest packing ratio of 0.7405, also used to pile oranges outside a green grocer's. In this configuration, each sphere touches 12 neighbors. Crystal structures based on this lattice are observed in many elements, including aluminum, calcium, copper, gold, lead, nickel, platinum, radon, and silver.

Shown in the second row of Table 3.2 is the body-centered cubic lattice (bcc), with a slightly smaller packing ratio of 0.6981. The structure is observed in barium, cesium, chromium, iron, lithium, molybdenum, niobium, rubidium, tungsten, and vanadium. The third row shows simple hexagonal packing, the structure in americium, hydrogen,

neodymium, promethium, and tellurium. In the last row is simple cubic packing with the smallest packing ratio of the above four packing configurations, 0.5236.

Table 3.1 Tight packing of spheres in 1D and 2D

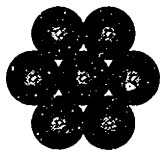
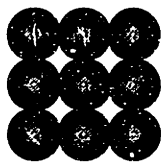
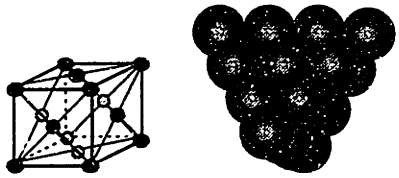
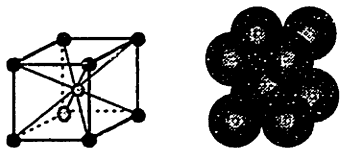
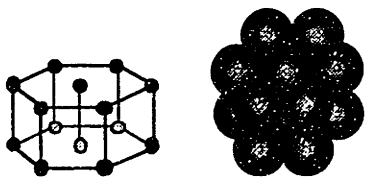
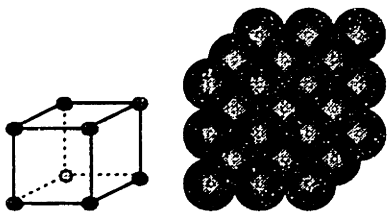
Dimension	Configuration	Kissing number	Packing ratio
1D	(tightest) 	2	1.0
2D	hexagonal lattice (tightest) 	6	$\frac{\pi}{\sqrt{12}} = 0.9069$
	square lattice 	4	$\frac{\sqrt{3}\pi}{8} = 0.6801$

Table 3.2 Tight packing of spheres in 3D

Dimension	Configuration	Kissing number	Packing ratio
3D	face-centered cubic packing (tightest) 	12	$\frac{\pi}{\sqrt{18}} = 0.7404$
	body-centered cubic packing 	8	$\frac{\sqrt{3}\pi}{8} = 0.6801$
	simple hexagonal packing 	8	$\frac{\pi}{3\sqrt{3}} = 0.6046$
	simple cubic packing 	6	$\frac{\pi}{6} = 0.5236$

In terms of the bubble meshing method described earlier, the important point here is that the face-centered cubic lattice, or cubic close-packing, corresponds to the node locations of an ideal 3D mesh consisting of regular tetrahedra. Connecting such node points by the 3D Delaunay triangulation algorithm will create a regular tetrahedral mesh. This is the tightest among the four packing configurations in the Table 3.2, and (probably) the tightest among all possible 3D sphere packing configurations including random ones. Although this claim seems intuitively correct and widely believed, a mathematical proof has remained a controversial problem, surviving unsolved for at least 380 years [Stewart 92].

This tighest sphere packing problem, called Kepler's Sphere-Packing Problem, was first posed by Johannes Kepler, famous for his discovery of elliptical orbits of planets, in his paper titled "Johann Kepler: Mathematician to His Imperial Majesty: A New Year's Gift or On the Six-Cornered Snowflake" in 1611. Since then, in spite of tremendous efforts of scientists and mathematicians, the proof of the tightest packing had survived unsolved nearly four centuries, until a solution was announced by Wu-Yi Hsiang in 1991. The proof is currently being checked by the mathematical community. (See Chapter 6 of [Stewart 92] for more details and references.)

3.2 BUBBLE SYSTEMS

3.2.1 Bubble Size

Since the sizes of the packed bubbles decide the final mesh element sizes, it is essential to control the bubble diameters according to a given node spacing over the domain. Bubble diameters are specified by the node spacing $d(x,y,z)$ or $d(u,v)$, a function of the bubble's location in object space or parametric space. As shown in Figure 3.7, the stable configuration of two bubbles occurs when they are tangent to each other. The *stable distance* r_0, therefore, is given as the summation of the radii of the bubbles,

$$r_0 = \frac{d(u_i,v_i)}{2} + \frac{d(u_j,v_j)}{2}, \tag{3.2.1}$$

or

$$r_0 = \frac{d(x_i,y_i,z_i)}{2} + \frac{d(x_j,y_j,z_j)}{2}, \tag{3.2.2}$$

where the locations of bubbles i and j are represented by (u_i,v_i) and (u_j,v_j) in parametric space, or (x_i, y_i, z_i) and (x_j, y_j, z_j) in object space.

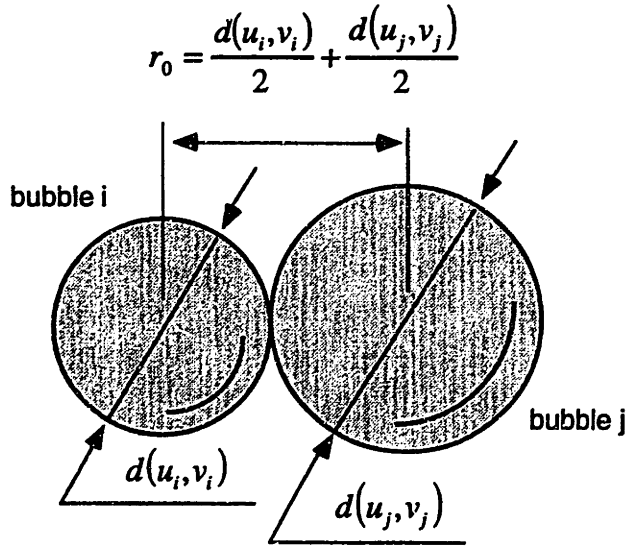


Figure 3.7 The stable distance between two bubbles defined as the summation of two radii. The bubble radius of a bubble is specified by node spacing, a function of the bubble's center location.

3.2.2 Repelling and Attracting Forces

As discussed in Section 3.1, repelling and attracting forces play a key role in constructing repeating hexagonal patterns in nature. The essential characteristics of such forces are:

- Repelling forces are exerted if two points are located closer than a specified distance.
- Attracting forces are exerted if two points are located farther apart than a specified distance.

In nature, the intermolecular, or van der Waals, interaction provides an example of such a force field. A commonly used mathematical model for this interaction is the Lennard-Jones potential, with the associated force law:

$$f(r)=\frac{12A}{r^{13}}-\frac{6B}{r^{7}}, \tag{3.2.3}$$

where A and B are positive constants, and r is the distance between two points. The first term describes the repulsion force, and the second the attraction force. The general form of this function is shown in Figure 3.8. The repulsive force increases rapidly to infinity as the distance between the two points approaches zero. The attractive forces decrease asymptotically as the two points are separated.

Since the van der Waals force creates a regular layout of points, as observed in metal bonding, we could simply take one of the popular mathematical models of this force and implement it as the interbubble force field. However, this is not a good idea, because our concern is not a realistic simulation of natural phenomena. In terms of speed, we will benefit by the selection of a simplified force model that is just sufficient for realizing a close packing of spheres. Two simplifying assumptions were made:

- The force is saturated near $r=0$, where two bubbles are located extremely close, so that interbubble forces do not grow infinitely large.
- The force interaction is active only within a specified distance to reduce computation by neglecting force effects at large distances, i.e., when two bubbles are not neighbors.

The latter is especially important to reduce a computational amount of the interacting force. A naive pairwise force calculation costs $O(n^2)$; it, however, can be reduced to $O(n\log n)$ by localizing the force effect.

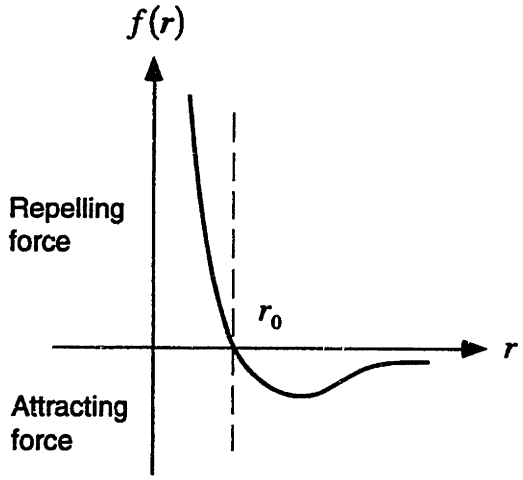


Figure 3.8 The van der Waals force, which exerts a repelling force when two points are located closer than a specified distance r_0, and an attracting force when two points are farther away.

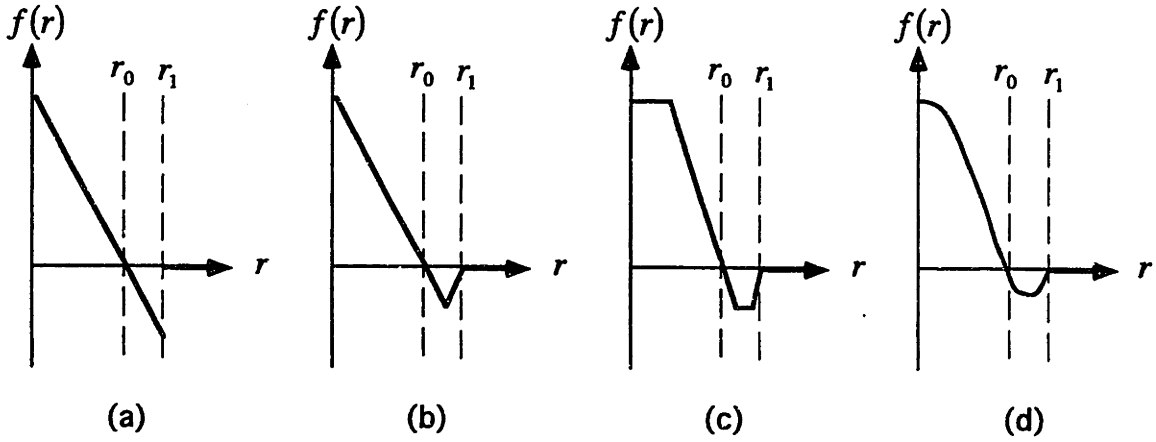


Figure 3.9 Possible forces with two simplifying assumptions: the force is saturated near $r=0$, and the force interaction is active only within a specified distance.

Many interbubble forces that satisfy these conditions can be defined as shown in Figure 3.9. A important quantity is the slope of the force at $r=r_0$

$$f'(r_0)=-k_0 \tag{3.2.4}$$

where k_0 represents the corresponding linear spring constant at the stable distance. The constant k_0 should be positive in order to create the repelling/attracting force. The force function implemented in the bubble system is a cubic function of the distance r, and is elaborated in Section 4.2.

3.2.3 Meshing Procedure with Bubble Systems

In the proposed method, as shown in Figure 3.10, meshing is performed in the order of dimensions. The four steps of meshing are:

(1) Place bubbles on all Vertices.

These bubbles are called Vertex-bubbles, and they do not move.

(2) Edge meshing

Bubbles are packed on all Edges. These bubbles and called Edge-bubbles, which can move only on Edges. After all bubbles, or mesh nodes, are placed, they are ready to be connected for a 1D mesh consisting of only line segments.

(3) Face meshing

Bubbles are packed on all Faces, and are called Face-bubbles. The movement of these bubbles are constrained to lie on the Faces. At the end of this process, nodes are distributed on Vertices, Edges, and Faces. Connecting the nodes by constrained Delaunay triangulation generates 2D mesh for a Face and 1D mesh for an Edge.

(4) Volume meshing

We pack bubbles in all Volumes. These bubbles are called Volume-bubbles and they can move freely in 3D space. This process completes node placement on Vertices, Edges, Faces, and Volumes. Connecting the nodes by constrained Delaunay triangulation will generate a 3D mesh for a Volume, a 2D mesh for a Face, and a 1D mesh for an Edge. These meshes of different dimension can be combined to a hybrid mesh.

The 1D meshing problem is solved by performing two steps, (1) and (2); the 2D meshing problem by (1), (2) and (3) as shown in Figure 3.11; and the 3D meshing problem by (1), (2), (3), and (4). Note that meshing processes in lower dimensions are sub-processes of meshing in higher dimensions. Figure 3.12 summarizes three meshing steps according to the dimension of a meshed domain.

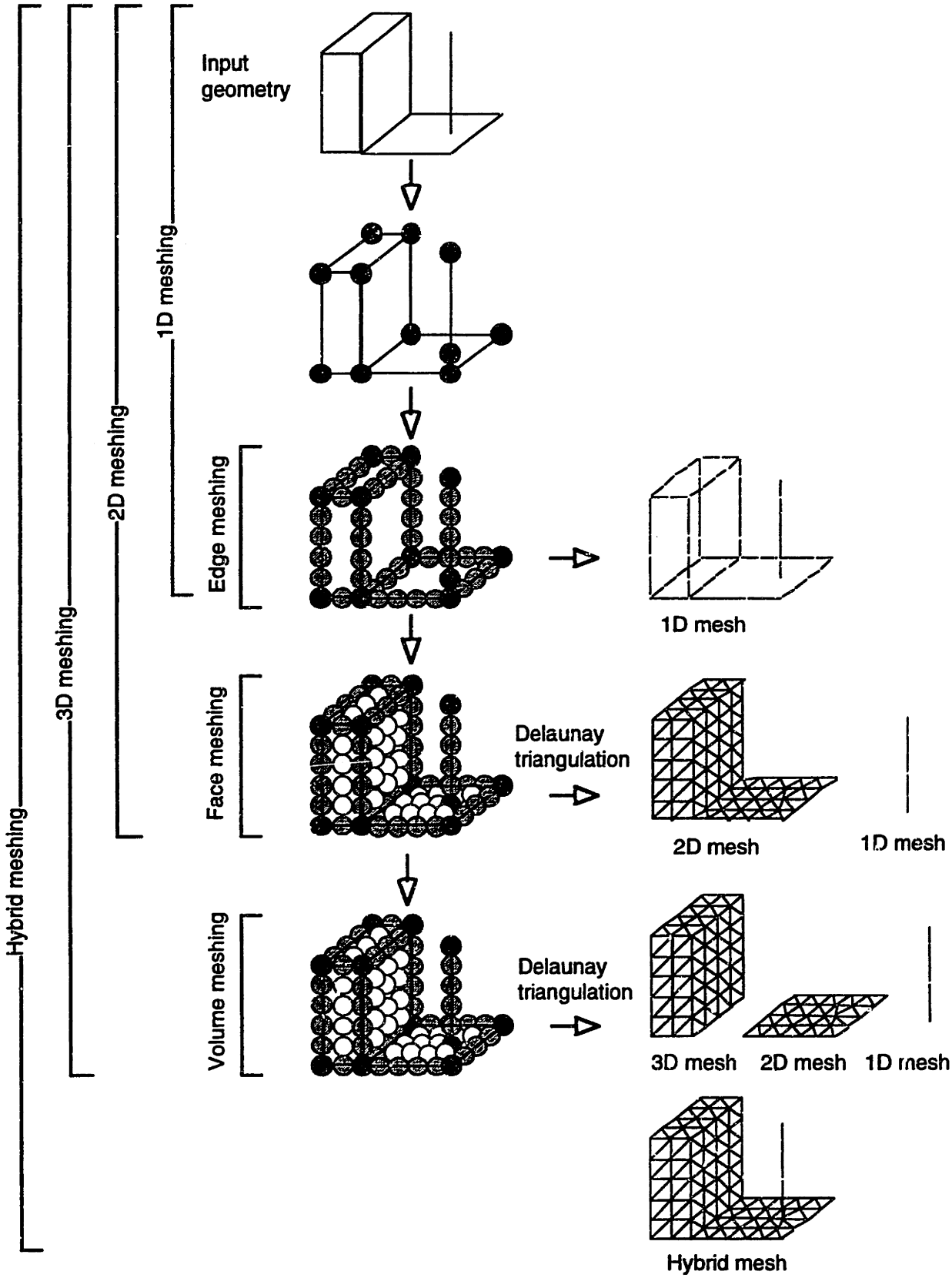
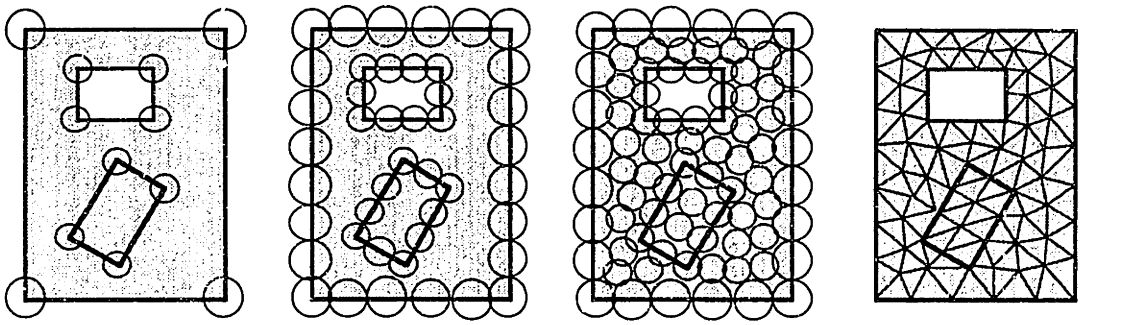


Figure 3.10 Hybrid meshing procedures.



(a) Vertex bubbles (b) Edge bubbles (c) Face bubbles (d) Delaunay triangulation

Figure 3.11 2D/surface meshing procedure.

	domain	domain perimeter	element	force-balanced bubble config.
edge-meshing	curve	vertices	line segment	
face-meshing	surface	curves	triangle	
volume-meshing	volume	surfaces	tetrahedron	

Figure 3.12 Three meshing steps according to the dimension of a meshed domain.

In Edge meshing, Face meshing, and Volume meshing, there are two major concerns, *bubble population control* and *bubble placement.* How many bubbles should be placed in each step? Except for Vertex-bubbles, the number of which is always same as the number of Vertices, an appropriate number of bubbles must be injected for each Edge, Face, and Volume, so that bubbles cover the geometry without significant overlaps and gaps. This is necessary to satisfy a given node spacing function while maintaining element shape regularity. If there are too many overlaps, the final mesh will contain an excessive number of mesh elements. On the other hand, if too many gaps exist, mesh elements will be larger than the specified size, and element shapes can be badly distorted as shown in Figure 3.13. This problem is called *bubble population control*.

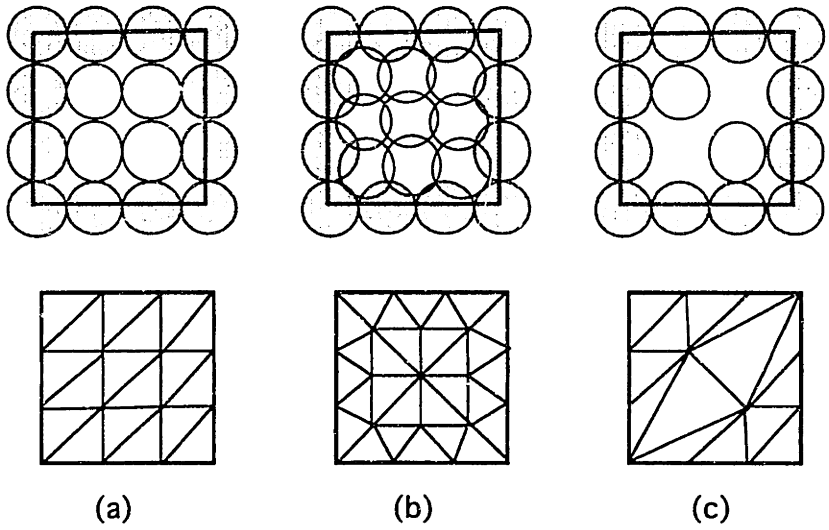


Figure 3.13 Bubble population control. (a) Packing of the right number of bubbles. (b) If there are too many overlaps, the final mesh contains too many mesh elements. (c) If too many gaps exists, mesh elements are larger than the specified size, and element shapes can also be badly distorted

The second problem is *bubble placement*, or node placement. After a proper number of bubbles are injected, a force balancing bubble configuration must be found in order to obtain regular mesh elements. This is not straightforward when an arbitrary geometry and a node spacing function is specified. As shown in Figure 3.14, incorrect bubble placement will lead to a failure to satisfy the node spacing and badly distorted elements.

In brief, the proposed bubble meshing is a procedure to place the right number of bubbles at appropriate positions on the given geometry, satisfying a given node spacing and maintaining element shape regularity.

3.2.4 Speed and Quality

Various algorithms have been crafted thus far on a case by case basis to satisfy one of two requirements, speed and quality. In the bubble method, one primary goal is to devise a unifying algorithm in which the user can select and adjust between speed and quality -- this is essential to make the method applicable to a variety of applications.

Edge meshing, Face meshing, and Volume meshing are performed in two stages: (Stage 1) meshing by hierarchical spatial subdivision, and (Stage 2) meshing by physically-based relaxation, in which the output of the first stage is utilized as an input. In the first stage, after bubbles are located using a hierarchical spatial grid, the center points of the bubbles are then connected by simple spatial ordering to give a complete

mesh topology. Since no pairwise interaction between bubbles is involved, this first stage of meshing is performed in times of order $O(n)$, where n designates the number of nodes in the triangular mesh. The mesh generated in the first stage is useful for many applications, but its major advantage is speed; the mesh quality is relatively inferior to that generated in the second stage. (Detailed in Section 4.1)

In the second stage, a force-balancing, nearly tangent configuration of bubbles is produced using physically-based relaxation. Although this includes a pairwise interaction between bubbles and thus more computational complexity, $O(n \log n)$, the quality of the final mesh is much superior. In the simulation process, each mesh node is modeled as a bubble with a point mass and viscous damping; also included in the model are interbubble forces. The equilibrium bubble configuration is obtained by solving numerically the governing equations of motion to yield the static force balance. A high quality mesh is created because the force-balancing process generates an almost equilateral triangulation, and the node spacing is well controlled over the region. Bubble population is adaptively controlled in the simulation by checking the sizes of overlaps and gaps. This process is detailed in Section 4.2 ~ 4.7.

Figure 3.15 portrays resultant meshes generated in the two stages of the bubble method. Note that gaps and overlaps created in the first stage are all fixed in the second stage.

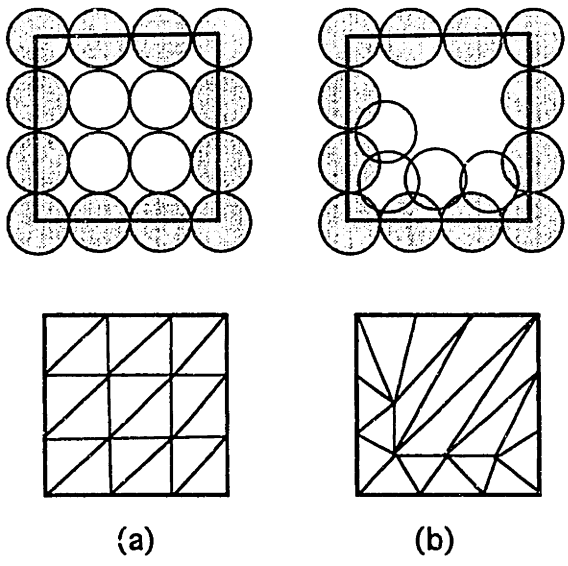
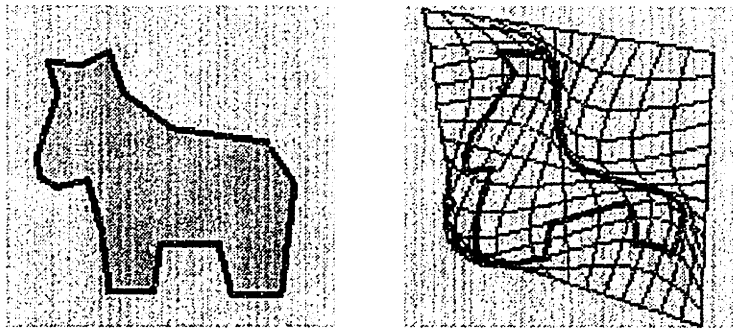
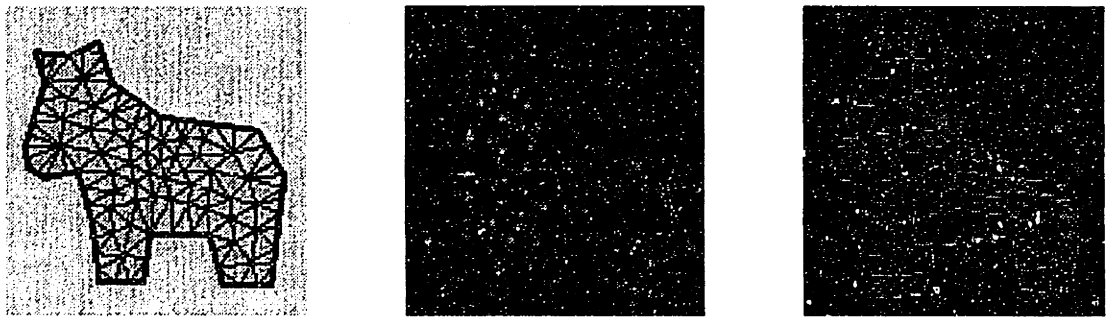


Figure 3.14 Bubble placement. (a) Bubbles are packed in a force-balancing configuration, causing a regular mesh. (b) When forces on bubbles are not balanced out, a distorted mesh can be created and node spacing is not correct.



(a) Input Surface

(b) Stage 1: Initial node placement using hierarchical spatial subdivision



(c) Stage 2: Final node placement using physically-based relaxation

Figure 3.15 Speed and quality. In the bubble method, a primary goal is to devise a unifying algorithm in which the user can select and adjust between speed and quality. To realize this, meshing is performed in two stages: (Stage 1) meshing by hierarchical spatial subdivision, and (Stage 2) meshing by physically-based relaxation, in which the output of the first stage is utilized as an input.

4

Mesh Generation with Bubble Systems

This chapter presents the technical details of the bubble method. As discussed in the previous chapter, there are two major problems: (a) where to place bubbles to minimize gaps and overlaps, and (b) how many bubbles to inject to fill the region.

The proposed approach to the first problem is to obtain a force-balancing configuration via a dynamic simulation assuming repelling/attracting interbubble forces similar to intermolecular van der Waals forces. Section 4.1 explains how bubbles are initially placed by hierarchical spatial subdivision. Section 4.2 presents examples of appropriate interbubble forces, and an initial value problem with the equation of motion is formulated in Section 4.3, followed by a numerical solution scheme to the equation in Section 4.4. Sections 4.5 and 4.6 present other important mechanisms for obtaining a force-balancing configuration: (a) confining nodes on edges and surfaces, and (b) node spacing control.

To address the problem of finding an appropriate number of bubble to inject, adaptive bubble population control is proposed, as discussed in Section 4.7.

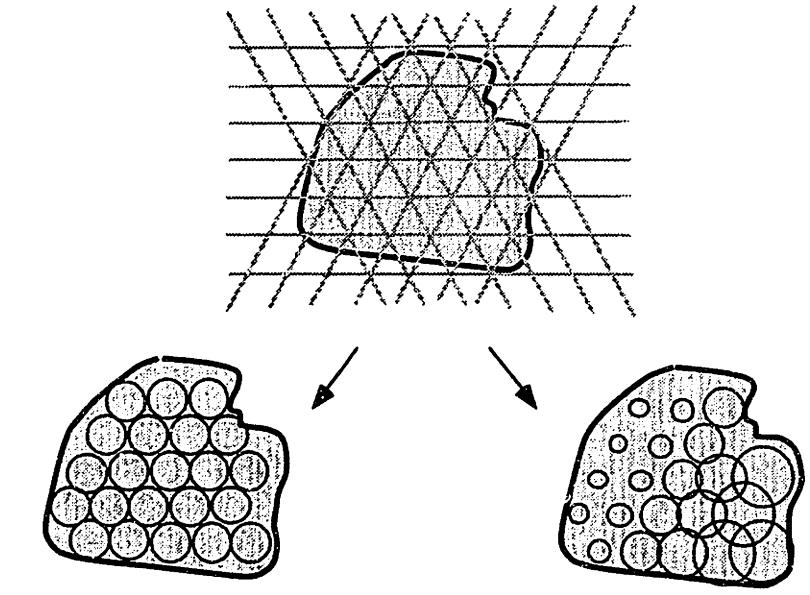
Section 4.8 discusses remeshing capabilities, one of the bubble method's major advantages.

4.1 INITIAL BUBBLE CONFIGURATION

It is essential to obtain a good initial bubble configuration before physically-based relaxation for two reasons. First, when speed is most critical, the initial bubble configuration itself can serve as a quick triangulation solution. Second, a good first guess will greatly reduce the convergence time of the lengthy relaxation process.

The simplest distribution method is to place bubbles at regular grid points in parametric space and to remove bubbles located outside the parametric boundary. Figure 4.1 shows such an example for the 2D case, where bubbles are placed at regular hexagonal grid points. This works fine for the uniform node spacing case as shown in Figure 4.1(a). The procedure, however, does not give a very desirable initial bubble configuration for the general node spacing case as can be seen in Figure 4.1(b).

To handle general node spacing, a bubble placing method based on hierarchical spatial subdivision has been devised. The basic idea is to subdivide a curve, a surface, or a volume hierarchically using a binary tree, a quadtree, and an octree respectively. Then bubbles are inserted until they cover the entire region without significant gaps or overlaps.



(a) Uniform node spacing (b) Varying node spacing

Figure 4.1 An example of initial face bubble configuration based on a regular grid. (a) Uniform node spacing. (b) Varying node spacing

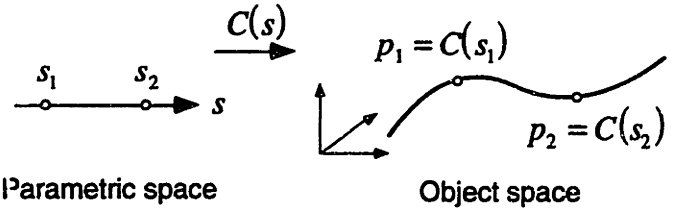
4.1.1 Edge Bubbles Initial Configuration

To place bubbles on a curve $C(s)$ defined in the parameter range of $s_1 \le s \le s_2$ as shown in Figure 4.2(a), we first place bubbles at the two end points, $p_1 = C(s_1)$ and $p_2 = C(s_2)$. The diameters of these end bubbles are d_1 and d_2 respectively. The distance in the object space between the two end points, $\overline{p_1 p_2}$, is then compared with the summation of the two radii, $d_1/2 + d_2/2$. Two cases exist. For Case 1, $\overline{p_1 p_2}$ is less than or equal to $d_1/2 + d_2/2$, meaning that the two end bubbles completely cover the curve segment, and thus no further subdivision is required. For Case 2, $\overline{p_1 p_2}$ is larger than $d_1/2 + d_2/2$, and thus further subdivision is required. In Case 2, a new bubble is placed at the midpoint of the range $s_1 \le s \le s_2$. The range is then subdivided into two sub-ranges, $s_1 \le s \le (s_1 + s_2)/2$ and $(s_1 + s_2)/2 \le s \le s_2$, on which the same distance check is performed. This process is repeated recursively until the whole curve is covered by bubbles. A procedural statement of the process is given as follows:

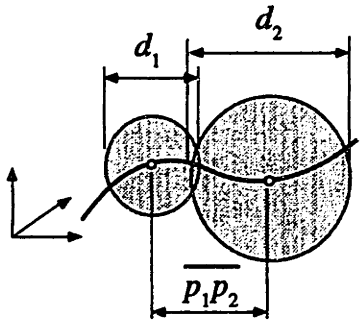
initial_edge_bubble(s_1, s_2)
  {d denotes node spacing, or bubble diameter}
  if $\frac{d_1}{2} + \frac{d_2}{2} \le \overline{p_1 p_2}$
  then {Case 1}
    return
  else {Case 2}
    place_bubble_at$\left(\frac{s_s + s_e}{2}\right)$
    initial_curve_bubble$\left(s_s, \frac{s_s + s_e}{2}\right)$
    initial_curve_bubble$\left(\frac{s_s + s_e}{2}, s_e\right)$
  endif

An example of applying this recursive procedure is shown in Figure 4.3. The resultant hierarchical subdivision can be represented as a binary tree.

This process will necessarily generate overlapping bubbles (i.e. on a relatively straight curve) or gaps between bubbles (i.e. on a sharp curve). These will be corrected later in the second stage of the algorithm.

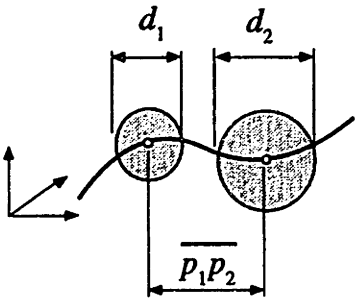


(a) Parametric curve definition



$$\frac{d_1}{2} + \frac{d_2}{2} \geq \overline{p_1 p_2}$$

(b) Case 1: Further subdivision is not necessary.



$$\frac{d_1}{2} + \frac{d_2}{2} < \overline{p_1 p_2}$$

(c) Case 2: Further subdivision is required.

Figure 4.2 Initial configuration of edge bubbles based on hierarchical spatial subdivision.

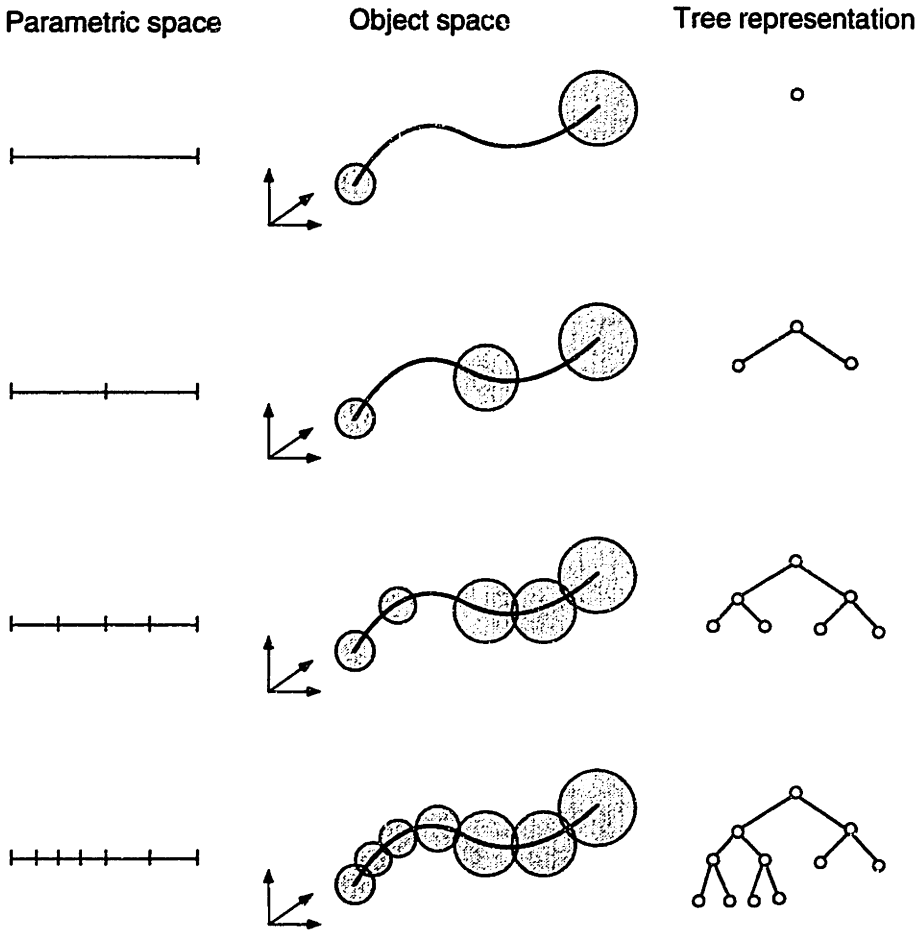


Figure 4.3 An example of hierarchical spatial subdivision of a curve.

4.1.2 Face Bubble Initial Configuration

After bubbles are placed on all curves, bubble placement on the surface $S(u,v)$ is performed using an *oblique quadtree*. Instead of square cells, the oblique octree uses rhombuses with inside angles of 60° and 120° as shown in Figure 4.4. The rhombus cell shape is adopted because it permits a regular hexagonal arrangement of bubbles when sizes are uniform and the surface is planar. The trimmed surface region in parametric space must be entirely enclosed in an initial rhombus. Let l be the edge length, and (u,v) the center point of a rhombus as shown in Figure 4.4. The center point mapped in the object space is denoted by $p_0 = S(u,v)$. The coordinates of the four corner points in the parametric space are:

$$
\begin{aligned}
(u_1, v_1) &= \left(u - 3l/4, v - \sqrt{3}l/4\right) \\
(u_2, v_2) &= \left(u + l/4, v - \sqrt{3}l/4\right) \\
(u_3, v_3) &= \left(u + 3l/4, v + \sqrt{3}l/4\right) \\
(u_4, v_4) &= \left(u - l/4, v + \sqrt{3}l/4\right)
\end{aligned}
\tag{4.1.1}
$$

For these corner points, the corresponding points mapped into the object space are denoted by p_1, p_2, p_3, and p_4 respectively. The coordinates of these points are calculated with a mapping function S:

$$p_i = S(u_i, v_i), i = 1..4 \tag{4.1.2}$$

First, bubbles are placed at the center point, and their diameters are denoted by d_1, d_2, d_3, and d_4 respectively. To judge whether or not a further subdivision is required, the summations of the two diameters of adjacent bubbles are compared to the distances between the bubbles' center points. There are two cases as shown in Figure 4.4(b) and (c):

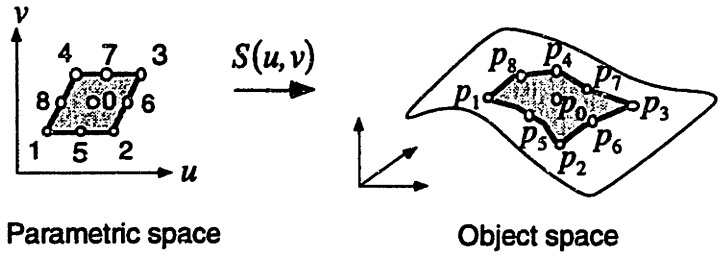
We denote as Case 1, the case in which the former is always larger than the latter, i.e.

$$\forall \binom{i}{j} = \binom{1}{2}, \binom{2}{3}, \binom{3}{4}, \binom{2}{4} \qquad \frac{d_i}{2} + \frac{d_j}{2} \geq \overline{p_i p_j} \tag{4.1.3}$$

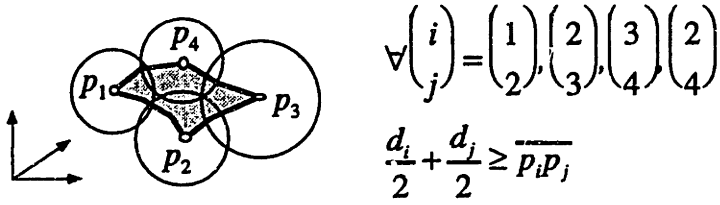
For Case 2, the former is not always larger than the latter, i.e.

$$\exists \binom{i}{j} = \binom{1}{2}, \binom{2}{3}, \binom{3}{4}, \binom{2}{4} \qquad \frac{d_i}{2} + \frac{d_j}{2} < \overline{p_i p_j} \tag{4.1.4}$$

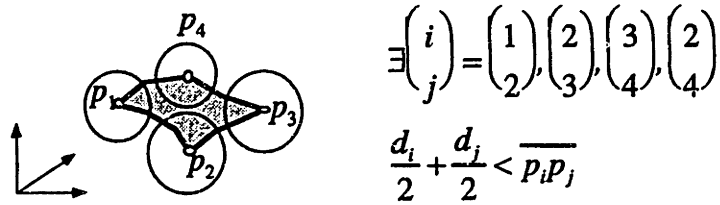
In Case 1, the four bubbles at the corners of the rhombus sufficiently cover the region, so the process is stopped. In Case 2, the region is subdivided into four sub-regions, then four additional bubbles are placed at the midpoints of all the edges of the rhombus, i.e.



(a) Parametric surface definition



(b) Case 1: Further subdivision is not necessary.



(c) Case 2: Further subdivision is required.

Figure 4.4 Initial face bubble configuration based on spatial subdivision.

$$
\begin{aligned}
(u_5, v_5) &= \left(u - l/4, v - \sqrt{3}l/4\right) \\
(u_6, v_6) &= \left(u + l/2, v\right) \\
(u_7, v_7) &= \left(u - l/4, v - \sqrt{3}l/4\right) \\
(u_8, v_8) &= \left(u - l/2, v\right)
\end{aligned}
\tag{4.1.5}
$$

which are mapped to the points in the object space, denoted p_5, p_6, p_7, and p_8 respectively in Figure 4.4 (a). This divides the original rhombus into four half-sized rhombuses, and new bubbles are placed on all rhombus corners. Then the same distance checking procedure is repeated for each region recursively until bubbles are distributed over the whole surface without significant gaps and overlaps.

During the quadtree building process, all new bubbles will be generated within the initial cell but they may be placed outside of the trimmed geometry. Only bubbles inside the geometry should actually be considered. For this reason, we keep a record of the positional relationship between the rhombus cell and the geometry. A cell is classified into three categories: (a) IN, the cell is inside the geometry; (b) OUT, the cell is outside the geometry; and (c) BOUNDARY, the cell is partially inside and partially outside of the geometry.

Once a cell is found to be OUT, this region is ignored, and further subdivision is not required. If a cell is classified as IN, then all the sub-regions created by subdividing this cell will be IN, and bubbles must be placed on all corners. An in-out check is required for each sub-cell every time a BOUNDARY cell is subdivided into four sub-regions -- they can be either IN, OUT, or BOUNDARY. At the conclusion of the process, a quadtree structure and bubble configuration are created, an example of which is shown in Figure 4.5. The procedural statement of the process is given as follows:

initial_face_bubble(u,v,l,in_out)

{d denotes node spacing, or bubble diameter}

{*in_out* flag shows whether the rhombus is :

IN : inside of the geomtry

OUT : outside of the geometry

BOUNDARY : partially inside and partially outside of the geometry}

if *in_out* = OUT

then

return

else

if $\forall\binom{i}{j}=\binom{1}{2},\binom{2}{3},\binom{3}{4},\binom{2}{4} \quad \frac{d_i}{2}+\frac{d_j}{2} \geq \overline{p_i p_j}$

then {Case 1}

if *in_out* = BOUNDARY

then

place_bubble_if_inside_geoemtry(u,v)

place_bubble_if_inside_geoemtry$\left(u-l/4\,,\,v-\sqrt{3}l/4\right)$

place_bubble_if_inside_geoemtry$(u+l/2\,,\,v)$

place_bubble_if_inside_geoemtry$\left(u+l/4\,,\,v-\sqrt{3}l/4\right)$

place_bubble_if_inside_geoemtry$(u-l/2\,,\,v)$

return

else

place_bubble_at(u,v)

place_bubble_at$\left(u-l/4\,,\,v-\sqrt{3}l/4\right)$

place_bubble_at$(u+l/2\,,\,v)$

place_bubble_at$\left(u+l/4\,,\,v-\sqrt{3}l/4\right)$

place_bubble_at$(u-l/2\,,\,v)$

return

endif

```
else {Case 2}
  if  in_out = BOUNDARY
  then
    in_out1 ← check_in_out_subregion1
    in_out2 ← check_in_out_subregion2
    in_out3 ← check_in_out_subregion3
    in_out4 ← check_in_out_subregion4
  else
    in_out1 ← in_out2 ← in_out2 ← in_out2 ← IN
  endif
  initial_face_bubble(u − 3l/8, v − √3l/8, l/2, in_out1)
  initial_face_bubble(u − l/8, v − √3l/8, l/2, in_out2)
  initial_face_bubble(u + 3l/8, v + √3l/8, l/2, in_out3)
  initial_face_bubble(u − l/8, v + √3l/8, l/2, in_out4)
 endif
endif
```

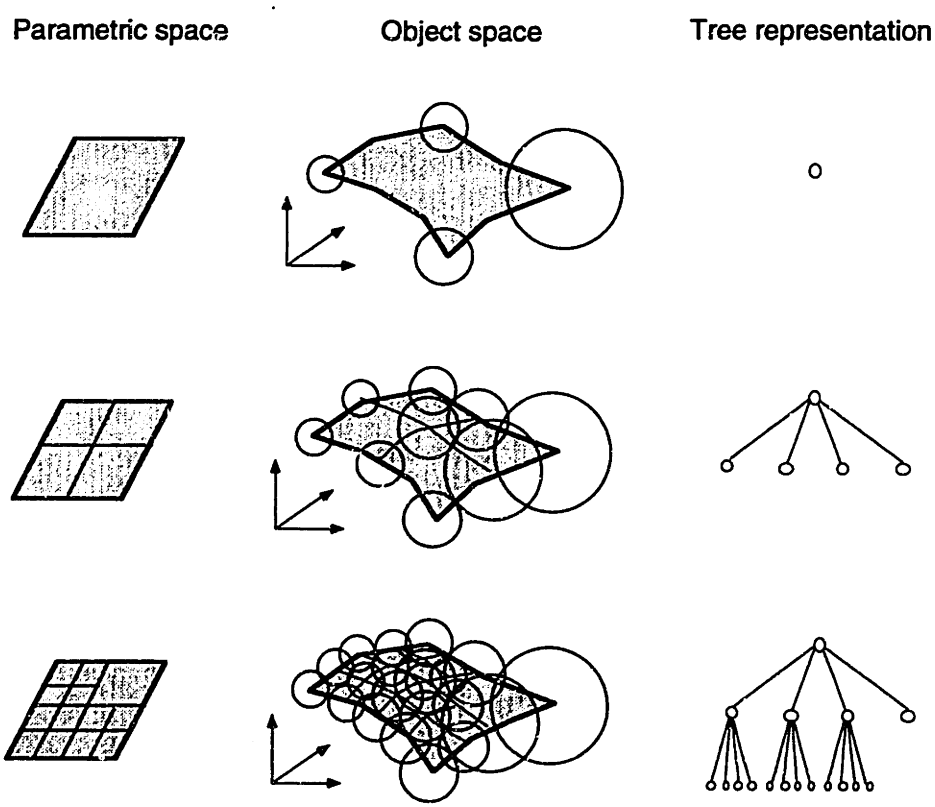


Figure 4.5 An example of hierarchical spatial subdivision of a curve.

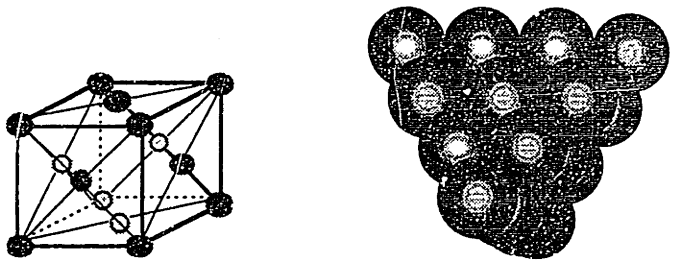


Figure 4.6 Initial volume bubble configuration based on spatial subdivision.

4.1.3 Volume Bubble Initial Configuration

A similar procedure for volume meshing, $initial\_volume\_bubble(x, y, z, l, in\_out)$, can be defined by using an octree, yielding face-centered cubic packing when uniform node spacing is specified, as shown in Figure 4.6. The procedure starts off by placing eight bubbles on a rhomboid cell, then the cell is divided into eight octants if these bubbles do not cover the whole volume. The process is recursively repeated until the whole volume is covered by bubbles, without significant gaps and overlaps.

4.1.4 Computational Complexity of Stage 1

Let us evaluate the computational complexity involved in the procedures:

$$initial\_edge\_bubble(s_1, s_2)$$

$$initial\_face\_bubble(u, v, l, in\_out)$$

$$initial\_volume\_bubble(x, y, z, l, in\_out)$$

In edge meshing with $initial\_edge\_bubble(s_1, s_2)$, one bubble is placed every time a region is subdivided. Thus, the subdivision (Case 2) occurs a total of n times, where n designates the final number of edge bubbles placed. The recursive procedure ends with Case 1 for each of the $n+1$ sub-regions. In the whole process, the subroutine $initial\_edge\_bubble(s_1, s_2)$ is called $2n+1$ times, and each time we calculate the

distance $\overline{p_1p_2}$, evaluate the two bubble diameters, and compare the distances. The computational complexity involved in edge-meshing, therefore, is estimated as $O(n)$.

The complexities involved in face-meshing and volume-meshing are also $O(n)$ based on similar analyses. One major difference, however, is that the extra, costly *in-out evaluation* must be performed for face-meshing and volume-meshing to check whether the cell is inside, outside or on the boundary of the geometry.

After the bubble locations are decided, we can quickly generate a complete triangulated mesh topology by connecting the bubbles' center points using simple spatial ordering with a computational cost of $O(n)$.

In summary, the process of bubble placement and topology generation in stage 1 costs $O(n)$, which is, as seen later in Section 4.3, much lower than the average cost of stage 2, $O(n\log n)$.

4.2 INTERBUBBLE FORCES

In stage 2, the system elimates the overlaps and gaps that exist between bubbles due to the initial bubble configuration generated in stage 1. To make the bubbles as close to tangent as possible, we have developed interbubble forces that cause the bubble system to move to a closely-packed equilibrium configuration.

Here we want to define an *interbubble force* f, much like the van der Waals force, which applies a repelling force when two bubbles are located closer than the stable distance r_0 (an overlap), or an attracting force when the bubbles lie further apart than r_0 (a gap). The force differs from the physical van der Waals force in two ways: (a) the force is saturated around $r=0$; and (b) the force effect is local. As pointed out in Section 3.2.2 and illustrated in Figure 3.9, many feasible force fields exist. In this section, two types of interbubble forces are derived, as already shown in Figures 3.9 (a) and (d) respectively.

4.2.1 Linear Interbubble Force

The simplest repelling and attracting force field is represented by a single linear function. Let the force function f be a linear function of the distance r between two bubbles i and j, and let r_0 be the stable distance between them. The force function f and its derivative are written using the ratio $w = r/r_0$

$$
\begin{aligned}
f(w) &= \frac{k_0}{r_0}(1-w) && 0 \le w \le 1.5 \\
f'(w) &= -\frac{k_0}{r_0} && 0 \le w \le 1.5 \\
f(w) &= f'(w) = 0 && 1.5 \le w
\end{aligned}
\tag{4.2.1}
$$

where k_0 represents the linear spring constant. This simple force field will work fine when the initial bubbles are distributed so that there are no significant gaps. A more smoothly changing relationship that diminishes toward the local limit of the force, e.g. $w = 1.5$, must be utilized when the number of bubbles is not precisely controlled.

4.2.2 Cubic Interbubble Force

A more smoothly changing force field can be defined by a single cubic function, which gives the force field shown in Figure 3.9 (d). A cubic function is utilized because it is the polynomial with the minimum degree which satisfies all the requirements discussed in Section 3.2.2. It also looks most similar to the original van der Waals force.

Let the force function f be a cubic function of the distance r between two bubbles i and j, and let r_0 be the stable distance between them. The force function f and its derivative are written using the ratio $w = r/r_0$

$$
\begin{aligned}
f(w) &= Aw^3 + Bw^2 + Cw + D \\
f'(w) &= 3Aw^2 + 2Bw + C
\end{aligned}
\tag{4.2.2}
$$

where A, B, C, and D are coefficients calculated later using the boundary conditions. The distance r_0 is given as a summation of two bubble diameters as mentioned in Section 3.2

$$
r_0 = \frac{d_i}{2} + \frac{d_j}{2} \tag{4.2.3}
$$

Figure 4.7 depicts the interbubble force f with a stable distance of r_0, and a local force limit at the distance $1.5r_0$. Such a cubic function is computed by satisfying the following boundary conditions:

$$f(1)=0,\ \ f(1.5)=0,\ \ f'(0)=0,\ \ f'(1)=-k_0/r_0 \tag{4.2.4}$$

where k_0 represents the corresponding linear spring constant. The above four boundary conditions are written as

$$\begin{cases} f(1)=A+B+C+D=0 \\ f(1.5)=(1.5)^3A+(1.5)^2B+1.5C+D=0 \\ f'(0)=C=0 \\ f'(1)=3A+2B+C=-k_0/r_0 \end{cases} \tag{4.2.5}$$

By solving the above simultaneous equations, the force function is obtained as

$$f(w)=1.25\frac{k_0}{r_0}w^3-2.375\frac{k_0}{r_0}w^2+1.125\frac{k_0}{r_0}=0 \qquad 0\le w\le 1.5 \tag{4.2.6}$$

$$f(w)=0 \qquad 1.5<w$$

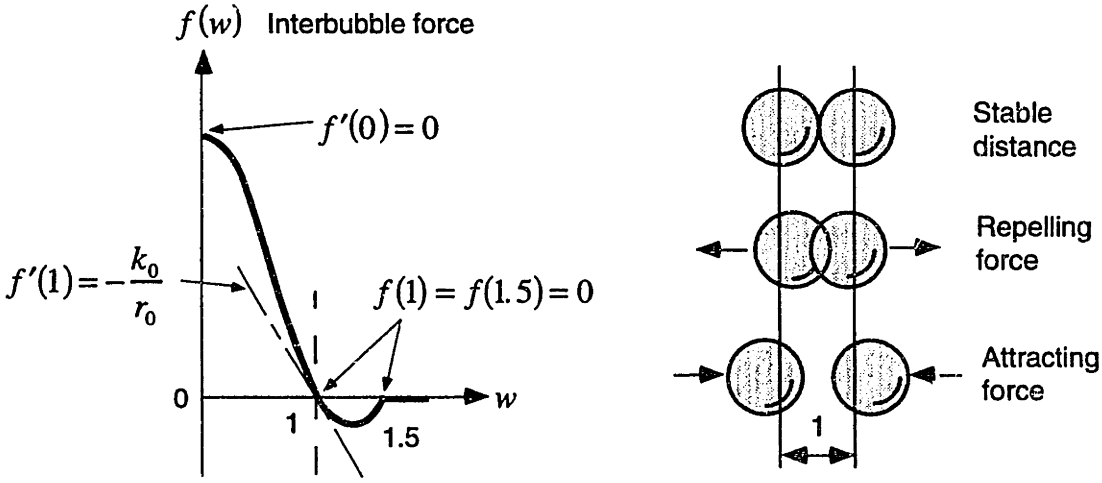


Figure 4.7 Interbubble force defined by a single cubic function. This is a smoothly changing function that satisfies four requirements: (a) it exerts a repelling force for $w<1$; (b) it exerts an attracting force for $1\le w$; (c) the force is saturated around $w=0$; and (d) the force is local and zero for $1.5<w$.

4.3 BUBBLE STATICS AND DYNAMICS

Given the interbubble forces, our goal is to find a bubble configuration that yields a static force balance. This is not an easy task when an arbitrary node spacing is specified and some bubbles are constrained on curves and surfaces. The bubble dynamics are solved in order to find a force-balancing configuration. In this section, the statics and dynamics of the bubble systems are discussed.

4.3.1 Bubble Statics

A summation of interbubble forces can be balanced out in several ways as shown in Figure 4.8: (a) bubbles are perfectly tangent to each other so that there are no repelling and attracting forces applied between them; (b) the density of the bubble population is so high that overlapping bubbles apply repelling forces; (c) the density of the bubble population is so low that gaps exist and yield attracting forces among bubbles; or (d) both attracting and repelling forces are applied due to overlaps and gaps.

Although the packing of bubbles with perfect tangency is most desirable, it is not always possible for the following two reasons:

• Boundary conditions

In each meshing step, i.e. edge-meshing, face-meshing, and volume-meshing, fixed bubbles are placed on the boundaries of the meshed region. For example, in edge-meshing, two vertex bubbles are placed on the end points of a curve and they cannot be moved. Similarly, in face-meshing, fixed locations of edge bubbles and vertex bubbles are given, and in volume-meshing, fixed locations of face, edge, and surface bubbles are given.

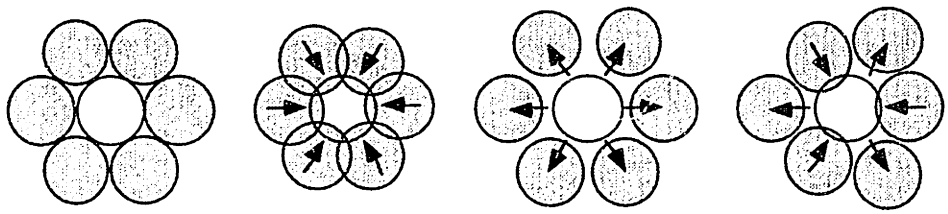


Figure 4.8 Example of a stable bubble configuration with repelling and attracting interbubble forces.

• Variable node spacing

When a varying node spacing is desired, it is not always possible to place bubbles so that all the neighbors are tangent to each other. For example, when node spacing must change rapidly, bubbles cannot be packed closely and overlaps and gaps are likely to be present.

These same factors also create geometric and topological irregularity in a mesh. In 2D meshing, triangles are called geometrically regular when they are equilateral, and topologically regular when all nodes have six neighbors. In a perfectly regular mesh, nodes are distributed at equal intervals everywhere and form a regular hexagonal grid. This is not possible with boundary conditions and varying node spacing.

The goal, therefore, is not to find a bubble configuration that yields perfect tangency between neighbors, but to obtain a force balancing configuration. Let us think about a simple example, a bubble system consisting of three fixed bubbles and one free bubble as shown in Figure 4.9. We assume for simplicity that these bubbles are constrained to a plane and that node spacing is given as a function of the 2D coordinates of the center points of the bubbles. The positions and diameters of the three fixed bubbles are denoted by (x_i, y_i) and $d_i = d(x_i, y_i)$, $i = 1..3$ respectively. The position of the free bubble (x, y) must be found.

What is to be emphasized here is the fact that there is no single force balancing configuration. Actually an infinite number may exist as explained below. Some representative configurations are given in Figure 4.9. In Figure (a), the free bubble is located outside of the interbubble force field of the three fixed bubbles, so that there is no force interaction and the static force balance is satisfied. In (b) and (c), one or two fixed bubbles are tangent to the free bubble and there is no applied interbubble force. The three repelling forces are balanced out in (d).

Among these alternative, positions (c) and (d) are stable in the sense that a small perturbation from these positions will be restored to its original position by interbubble forces. Usually during the course of mesh generation we are interested in solutions that are similar to that shown in Figure 4.9(d), where the set of repelling and attracting forces are balanced out. When free bubbles are packed closely within a region bounded by fixed bubbles, only this type of solution is feasible.

To solve the force balance, we can employ two basic strategies, (a) direct formulation and equation solving of the static balance and (b) solving the force balance as an energy minimization problem. Here let us try to apply the former method to the four bubble

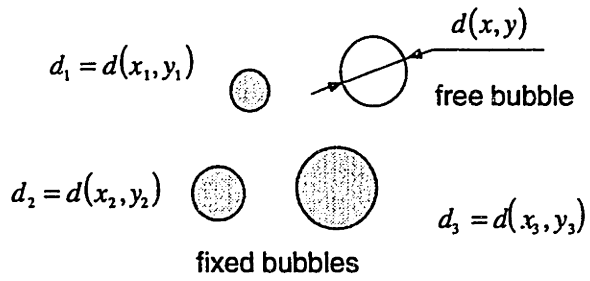
system illustrated in Figure 4.9(d). The distance and the stable distance between the free bubble and the fixed bubbles are written respectively as

$$r_i = \sqrt{(x_i - x)^2 + (y_i - y)^2} \qquad i = 1..3$$

$$r_{0i} = \frac{d(x,y)}{2} + \frac{d_i}{2} \qquad i = 1..3 \tag{4.3.1}$$

Since the magnitude of the interbubble force is given by the function $f(r_i/r_{0i})$, the simultaneous equations to be solved for a force balance in the x and y directions are:

$$\begin{cases} \sum_{i=1}^{3} f\left(\frac{r_i}{r_{0i}}\right)\frac{(x_i - x)}{r_i} = 0 \\ \sum_{i=1}^{3} f\left(\frac{r_i}{r_{0i}}\right)\frac{(y_i - y)}{r_i} = 0 \end{cases} \tag{4.3.2}$$



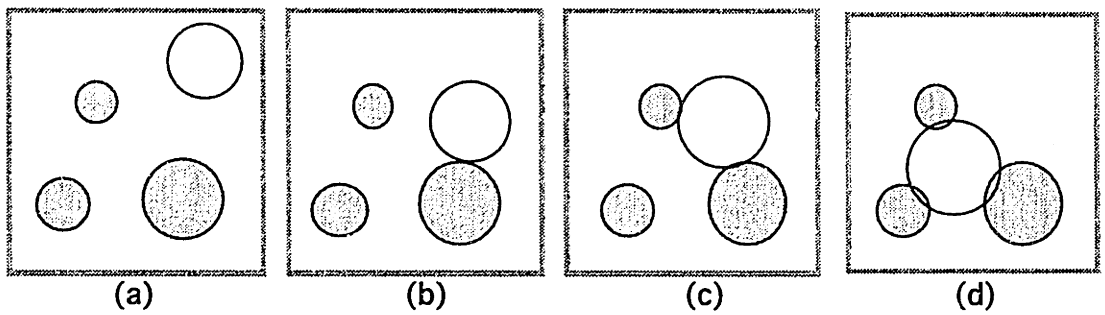


Figure 4.9 Multiple force-balancing solutions. With a system of three fixed bubbles and one free bubble, there are many force-balancing configurations. Among them, only (c) and (d) are stable in the sense that a free bubble will return to its original position when a small perturbation is applied.

As can be seen from the above equations, with an arbitrary defined node spacing, we must solve a system of simultaneous non-linear equations, for which no direct, general solution methods are available. The more complicated the node spacing and force definition, the more difficult it is to solve the system of equations. For this reason, we employ energy minimization instead of a direct solution by solving the system dynamics to achieve a force balance in the proposed bubble meshing method. The next section elaborates how the dynamic models are constructed for our bubble systems, and Section 4.4 discusses how the dynamic equations are solved numerically.

4.3.2 Bubble Dynamics

As explained in the previous section, our concern is the final force balancing configuration of bubbles, obtained by solving a system of non-linear equations. Two widely-used approaches to solve this type of equation are: (a) multidimensional root finding techniques such as Newton-Raphson, and (b) multidimensional minimization (optimization). Since we place strict geometric constraints on each degree of freedom (to keep all bubbles within the given geometry), it is difficult to implement an algorithm with approach (a).

In our implementation, therefore, an energy minimization scheme, a multidimensional minimization with derivative information, is employed. The governing equation of motion is constructed, and the initial value problem is solved numerically. As time is incremented, the bubble system approaches equilibrium, a force-balancing configuration.

Let n be the number of free bubbles in the system. Since each bubble has three degrees of freedom (d.o.f.), x, y, and z, in 3D space, the whole system has $3n$ d.o.f. A point mass is assumed at each bubble's center point, and viscous damping is modeled to absorb the system's motion energy and eventually yield a static system. The equation of motion of ith d.o.f is written as follows:

$$m_i \frac{d^2 x_i(t)}{dt^2} + c_i \frac{dx_i(t)}{dt} = f_i(t), \quad i = 1 \ldots 3n \tag{4.3.3}$$

where m_i denotes mass, c_i is the damping coefficient, and x_i is the position of the ith d.o.f in object space. Given an initial configuration, the bubble motions are calculated by solving the governing equations numerically.

4.3.3 Physical Parameters

Another important issue in the physically-based approach is the selection of physical parameters such as the mass, the damping and the strength of interbubble forces. They determine the time response of the system, and we want to design a "well-behaved" physical model by carefully choosing a combination of parameters. In this section, we discuss some criteria used in designing the bubble system, then describe standard control theories and methodologies that are applied to decide upon good physical parameters.

There are two ways to measure the time response of a system: the transient and the steady-state response. The transient response means how the system behaves from the initial state to the final state as time passes. We want the bubbles to quickly move to the force-balancing configuration without too much oscillatory behavior. The system's response when time approaches infinity is referred to as the steady-state response.

In designing the bubble system, the most important characteristic of the dynamic behavior is absolute stability -- the system must be stable, that is, the sytem must eventually return to its equilibrium state when the system is subjected to a disturbance. Unstable systems, which generate continuing oscillations or diverging outputs from the equilibrium state, must be avoided. In brief, we want both a quick response and stability.

The bubble system dynamics are represented in equation (4.3.3). The system is similar to a standard second-order, single degree system consisting of a mass, viscous damping, and a linear spring. The only difference is that the interbubble forces are not linear springs, but non-linear springs with non-constant force-displacement ratios. In order to directly apply the standard control theory of systems with linear springs, we can define representative linear spring constants for our non-linear springs, assuming a closely packed bubble configuration. Note that the converted linear spring constants are not utilized in solving the equation, but utilized only in determining good physical parameter values.

First, in edge-meshing, assuming three free bubbles are tangent to each other as shown in Figure 4.10(a), the spring constants of two virtual linear springs are both k_0 (defined in Section 4.2). In a basic model where two outside bubbles move in the same direction and the center bubble moves in the opposite direction by the same amount, the center points of the two springs stand still, and the center bubble is supported by two springs of half-length, each of which has a spring constant of $2k_0$. The total equivalent linear spring constant in edge-meshing, therefore, is estimated as $4k_0$. In face-meshing, a hexagonal arrangement is assumed to calculate the equivalent spring constant, in volume-

meshing, a face-centered cubic packing is assumed. The equivalent spring constants k in the three stages of meshing are calculated as follows:

edge-meshing $\quad k = 4k_0$

face-meshing $\quad k_x = 8k_0 \qquad k_y = 4\sqrt{3}k_0$

$$k = \frac{k_x + k_y}{2} = \left(2\sqrt{3} + 4\right)k_0 \tag{4.3.4}$$

volume-meshing $\quad k_x = 12k_0 \qquad k_y = \left(4\sqrt{3} + 4\right)k_0 \qquad k_z = 4\sqrt{6}k_0$

$$k = \frac{k_x + k_y + k_z}{3} = \frac{4\sqrt{3} + 4\sqrt{6} + 16}{3}k_0$$

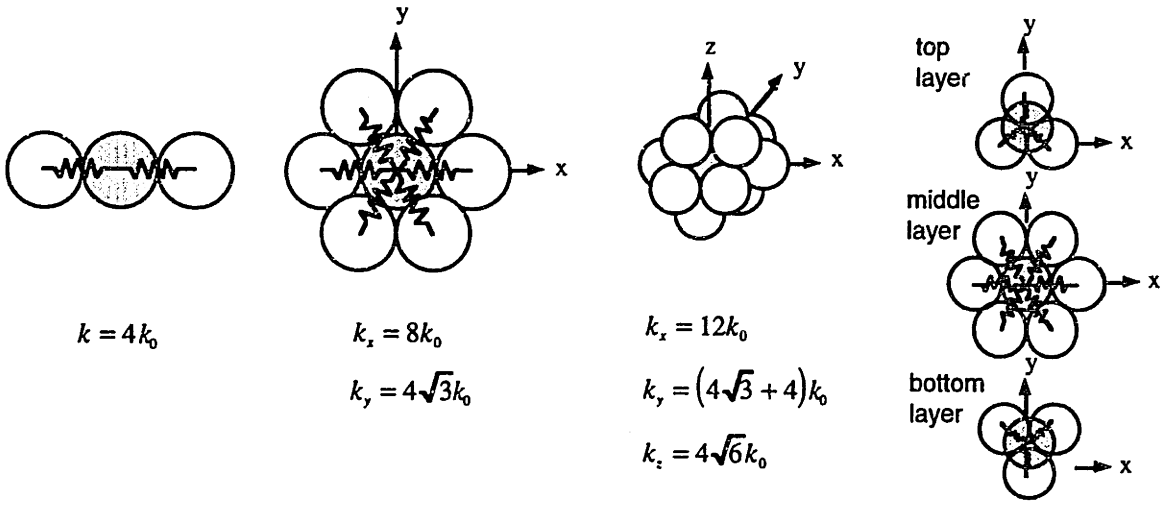


Figure 4.10 Equivalent linear spring constant at stable distance.

For face-meshing and volume-meshing, spring constants are slightly different in each direction. Therefore, we average all of the spring constants to obtain an equivalent spring constant. With the equivalent spring constant k, one degree of freedom of bubble motion is governed by the following equation of motion:

$$m\frac{d^2x(t)}{dt^2}+c\frac{dx(t)}{dt}+kx(t)=0 \tag{4.3.5}$$

The original equation of motion, which includes a non-linear force term, was converted to the above standard form consisting of mass, viscous damping, and a linear spring. Stability and a quick response time are contradicting requirements for a second order system; we obtain more stability by increasing the damping c, but some response speed is lost. Let us review the well-known transient response of the system to find a good set of physical parameters that strike a balance between stability and speed.

To solve equation (4.3.5), we assume a solution in the form

$$x(t)=Ce^{st} \tag{4.3.6}$$

Inserting equation (4.3.6) into (4.3.5) yields

$$ms^2+cs+k=0 \tag{4.3.7}$$

with the roots:

$$s_{1,2}=\frac{-c\pm\sqrt{c^2-4mk}}{2m}=-\frac{c}{2m}\pm\sqrt{\left(\frac{c}{2m}\right)^2-\frac{k}{m}} \tag{4.3.8}$$

These roots give two solutions to equation (4.3.5), thus the general solution is given by a combination of the two solutions:

$$x(t)=C_1e^{s_1t}+C_2e^{s_2t} \tag{4.3.9}$$

where C_1 and C_2 are arbitrary constants to be determined fro
m the initial conditions of the system. The critical damping coefficient c_c is defined as the value of the damping constant for which the radical in equation (4.3.7) vanishes, i.e.

$$\sqrt{\left(\frac{c_c}{2m}\right)^2-\frac{k}{m}}=0$$
$$c_c=2\sqrt{km}=2m\omega_n \tag{4.3.10}$$

where ω_n represents the natural frequency of the system. Now we can define the damping ratio ζ, which governs the system's damped transient response [Rao 90].

$$\zeta = \frac{c}{c_c} = \frac{c}{2\sqrt{mk}} \tag{4.3.11}$$

The dynamic behavior of second-order systems can therefore be described in terms of two parameters ζ and ω_n. As the case $\zeta = 0$ leads to undamped vibrations that are not appropriate for our purposes, we assume $\zeta \neq 0$ and consider the following three cases, the representative responses of which are also graphically presented in Figure 4.11.

Underdamped case $(0 < \zeta < 1)$

The response in this case is an oscillation with exponentially diminishing amplitude. The two roots of equation (4.3.7) can be expressed as

$$s_{1,2} = \left(-\zeta \pm i\sqrt{1-\zeta^2}\right)\omega_n \tag{4.3.12}$$

and the solution, equation (4.3.8), can be written in several forms:

$$\begin{aligned} x(t) &= C_1 e^{\left(-\zeta + i\sqrt{1-\zeta^2}\right)\omega_n t} + C_2 e^{\left(-\zeta - i\sqrt{1-\zeta^2}\right)\omega_n t} \\ &= e^{-\zeta\omega_n t}\left[C_1 e^{i\sqrt{1-\zeta^2}\omega_n t} + C_2 e^{-i\sqrt{1-\zeta^2}\omega_n t}\right] \\ &= X_0 e^{-\zeta\omega_n t}\sin\left(\sqrt{1-\zeta^2}\omega_n t + \Phi_0\right) \end{aligned} \tag{4.3.13}$$

where C_1, C_2, X_0, and Φ_0 are arbitrary constants to be determined from the initial conditions. The motion described by equation (4.3.12) is a damped harmonic motion of frequency

$$\omega_d = \sqrt{1-\zeta^2}\omega_n \tag{4.3.14}$$

Critically damped case $(\zeta = 1)$

The response in this case is non-periodic, and the motion will monotonically diminish to zero as time goes to infinity. The two roots of equation (4.3.9) can be expressed:

$$s_1 = s_2 = -\omega_n \tag{4.3.15}$$

and the solution, equation (4.3.8), can be written in the form:

$$x(t) = (C_1 + C_2 t)e^{-\omega_n t} \tag{4.3.16}$$

Overdamped case ($\zeta > 1$)

The response in this case is also non-periodic, and the motion will monotonically diminish to zero as time goes to infinity. The two roots of equation (4.3.8) are two distinct negative real numbers:

$$s_{1,2} = \left(-\zeta \pm \sqrt{\zeta^2 - 1}\right)\omega_n \tag{4.3.17}$$

and the solution, equation (4.3.9), can be written in the form:

$$x(t) = C_1 e^{\left(-\zeta + \sqrt{\zeta^2 - 1}\right)\omega_n t} + C_2 e^{\left(-\zeta - \sqrt{\zeta^2 - 1}\right)\omega_n t} \tag{4.3.18}$$

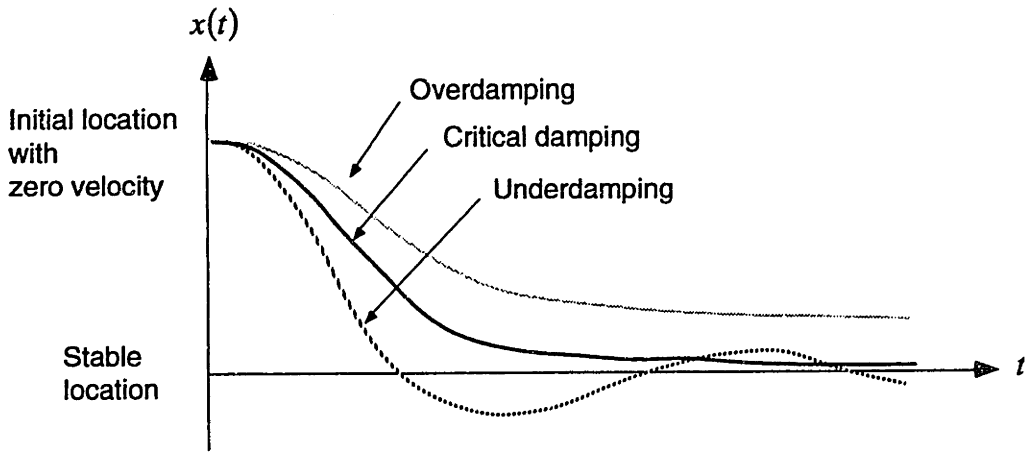


Figure 4.11 The dynamic behavior of second-order systems described in terms of the damping ratio ζ. The three cases are: (a) underdamped ($0 < \zeta < 1$); (b) critically damped ($\zeta = 1$); and (c) overdamped ($\zeta > 1$).

Among the three categories, the critical and the overdamped cases are more stable than the underdamped case in the sense that there isn't periodic oscillation. They require a much longer settling time however, i.e. the time before the output settles down within an allowable tolerance; note that all damped motion asymptotically approaches the goal position, so that we have to wait an infinite time for a perfect settling. It is, therefore, desirable to select a set of physical parameters that yields the minimum settling time.

In the unit-step response, the 2% and 5% settling time of an underdamped second-order system are defined as the time necessary for the system to settle down to a tolerance band of $\pm 2\%$ and $\pm 5\%$ respectively, as depicted in Figure 4.12. It is known that the settling time t_s reaches a minimum value around $\zeta = 0.76$ for $\pm 2\%$ range, and $\zeta = 0.68$ for $\pm 5\%$ [Ogata 70].

In conclusion, after the long review presented above, we can say that physical parameters should be chosen such that they give approximately

$$\zeta = \frac{c}{2\sqrt{mk}} \approx 0.7 \tag{4.3.19}$$

Once a set of values for m, c, and k are selected such that they meet equation (4.3.19), equation (4.3.4) calculates the appropriate value of k_0 for edge-meshing, face-meshing, and volume-meshing, and then the actual force function is given by either (4.2.1) or (4.2.2).

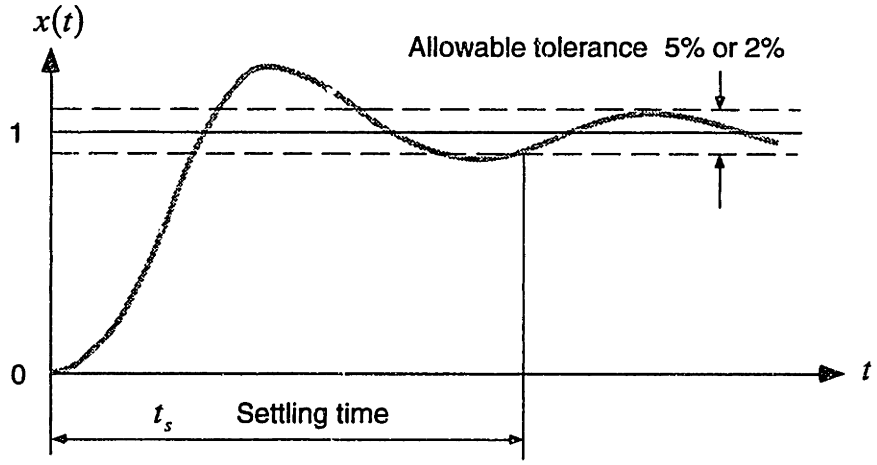


Figure 4.12 The 2% and 5% settling time of an underdamped second-order system in the unit-step response, defined as the time necessary for the system to settle down to within a tolerance band of $\pm 2\%$ and $\pm 5\%$ respectively.

4.4 NUMERICAL SOLUTION SCHEME

Given the equation of motion (4.3.3), the question of how the equation is solved numerically now arises. This is an initial value problem governed by a second-order ordinary differential equation, where all locations and velocities are given at some starting time, and we want to find the locations (and velocities in some cases) at some discrete points in time.

With a proper set of physical parameters, the movement of the bubbles should settle down as time goes by, and they should stand virtually still after some time yielding a static force balance. For a single degree of freedom, the initial value problem is written as

$$\begin{aligned} x = x(0),\ \frac{dx}{dt} = 0 \qquad & t = 0 \\ m\frac{d^2x(t)}{dt^2} + c\frac{dx(t)}{dt} = f(t) \qquad & t > 0 \end{aligned} \tag{4.4.1}$$

where $x(0)$ represents an initial bubble position obtained by the method described in Section 4.1. At the initial time, all the velocities of bubbles are assumed to be zero.

A good place to start is to rewrite our equation of motion (4.4.1) into a standard form of coupled first-order equations, so that we can directly apply various numerical integration schemes, such as Euler's method and the Runge-Kutta method. Let us introduce two new variables p_1 and p_2 to replace locations, x, and velocities, dx/dt, respectively. The original second-order equation for $t > 0$ in (4.4.1) is written as a set of coupled two first-order equations.

$$\begin{cases} \dfrac{dp_1}{dt} = p_2 \\ \dfrac{dp_2}{dt} = \dfrac{1}{m}\left[f(t) - cp_2(t)\right] \end{cases} \tag{4.4.2}$$

Using vector notation, the equation (4.4.1) can be further written

$$\begin{aligned} P = \begin{bmatrix} p_1(0) \\ 0 \end{bmatrix} \qquad & t = 0 \\ \frac{dP}{dt} = G(t, P) \qquad & t > 0 \end{aligned} \tag{4.4.3}$$

where

$$P=\begin{bmatrix} p_1 \\ p_2 \end{bmatrix}$$

$$G(t,P)=\begin{bmatrix} g_1 \\ g_2 \end{bmatrix}=\begin{bmatrix} p_2 \\ \frac{1}{m}[f(t)-cp_2(t)] \end{bmatrix} \tag{4.4.4}$$

Now our concern is how equation (4.4.3) can be solved by step-by-step integration. The easiest way to solve the problem is called *Euler's method,* obtained by rewriting dP and dt in the second equation of (4.4.3) as finite steps ΔP and Δt [Press 88]. Multiplying both sides of the equations by Δt gives the formula for calculating the change of P for the time step Δt.

$$\Delta P = \Delta t\, G(t,P) \tag{4.4.5}$$

This leads to the following recurrence formula with a constant time step of $\Delta t = h$:

$$P_0=\begin{bmatrix} p_1(0) \\ 0 \end{bmatrix}$$

$$P_{n+1} = P_n + hG(t_n,P_n) \qquad n=1,2,... \tag{4.4.6}$$

for marching out an approximate solution to (4.4.3). In the limit where the stepsize is very small, a good approximation to the original differential equation is achieved.

Although a complete estimation of error created by various types of finite difference approximations may be found in other references, a comparison between (4.4.6) and the following Taylor's series expansion will prove that the truncation error of the above Euler's method is $O(h^2)$.

$$P_{n+1} = P_n + hP_n^{'} + \frac{1}{2!}h^2P_n^{''} + \frac{1}{3!}h^3P_n^{'''} + \ldots \tag{4.4.7}$$

Generally, in most practical applications, Euler's method is not recommended due to the relatively large truncation error as compared to other more sophisticated methods such as Runge-Kutta. In our meshing problem, however, we are not concerned with an exact solution of the time trajectories of bubbles, but concerned with the final stable configuration only. In fact, our meshing procedure is one of the few applications where the most simple method, Euler's method, is still useful.

We can, however, use a more sophisticated method so that we get less truncation error, or higher-order truncation error. Let us consider procedures which, at somewhat

more computation per step, provide less errors. Although various methods have been devised, all recurrence formulas for integrating the propagation problem given in the form of equation (4.4.3) can be considered as special cases of the formula

$$P_{n+q} = P_n + qh\,(\text{weighted average of } G \text{ values}) \tag{4.4.8}$$

which approximates the slope between P_n and P_{n+q} by a weighted average of slopes of tangents to curves.

Among these methods, the *fourth-order Runge-Kutta method* has been widely used in many engineering and science simulations [Press 88]. The method evaluates G values four times in each time step: once at the initial point, twice at midpoints, and once at the end point. By using the combination of weights shown in equation (4.4.9), the method provides a truncation error of $O(h^5)$.

$$\begin{aligned}
k_1 &= hG(t_n, P_n) \\
k_2 &= hG\left(t_n + \frac{h}{2}, P_n + \frac{k_1}{2}\right) \\
k_3 &= hG\left(t_n + \frac{h}{2}, P_n + \frac{k_2}{2}\right) \\
k_4 &= hG(t_n + h, P_n + k_3) \\
P_{n+1} &= P_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}
\end{aligned} \tag{4.4.9}$$

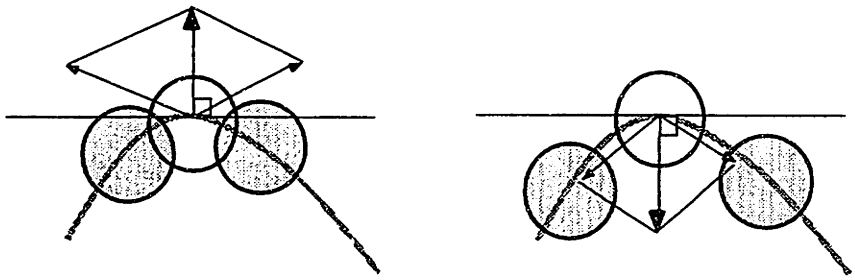
Thus far we have considered two representative methods for integrating ordinary differential equations: (a) Euler's method, the simplest method with larger truncation error; and (b) fourth-order Runge-Kutta, the most widely-used method with less truncation error. Although more sophisticated methods such as the Runge-Kutta method with adaptive stepsize control, predictor-corrector methods, and the Bulirsch-Stoer method may yield even less truncation error and more solution stability [Press 88], the two basic methods discussed in this Section were proved to be feasible as a result of actual implementation in the bubble meshing system.

4.5 NODE CONSTRAINTS

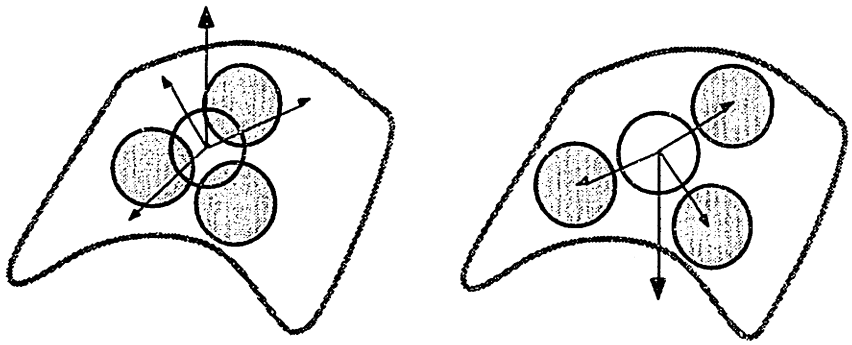
Thus far we have discussed interbubble forces, the force balancing configuration, dynamics of the bubble system, and the numerical solution scheme for the dynamic equation. These are important technical elements common to all steps of meshing, i.e. edge-meshing, face-meshing, and volume-meshing.

There is one other thing that is essential to edge-meshing and face-meshing, namely the node constraints. We require all edge bubbles to move only on their associated curve, and all surface bubbles must remain on the surface during the course of simulation. Generally bubble locations calculated by the numerical integration in Section 4.4 are not geometrically-compatible, i.e., not constrained on curves and surfaces.

Consider the curve and surface bubbles shown in Figure 4.13. In both configurations, the middle bubbles are repelled or attracted by the surrounding bubbles, such that a summation of those forces is directed perpendicular to the tangent vectors.



(a) Stable configuration of edge bubbles



(b) Stable configuration of surface bubbles

Figure 4.13 Stable bubble configuration with non-zero interbubble force applied. Bubbles are stable if a summation of applied forces is perpendicular to the tangent vector of the curve or the surface.

With the numerical solution in Section 4.4, if the force term is not zero, the bubble cannot stand still. These configurations, however, should be considered stable, because the bubble should not be allowed to move off of the curve or the surface. In each time step, therefore, we need to constrain the bubble movement such that all the bubbles lie exactly on the curves and surfaces. This is made possible by using the tangency information of the curves and surfaces.

To explain how bubbles are confined on a curve, let us denote the geometrically compatible location of the ith bubble on a curve as $\mathbf{x}_i(t)$ in object space and $s(t)$ in parametric space, and a new, unconstrained location $\mathbf{x}_i(t+\Delta t)$ in object-space, which was calculated by solving the equation of motion. The correct, confined location of a bubble on a curve is obtained as follows:

(1) Calculate the unconstrained displacement vector $\Delta\mathbf{x}_i = \mathbf{x}_i(t+\Delta t) - \mathbf{x}_i(t)$ in object space.

(2) Calculate the normalized tangent vector at the bubble location at time t, $C^s/|C^s|$.

(3) Take the dot product of the unconstrained displacement vector and the normalized tangent vector, and divide it by the length of the tangent vector:

$$\Delta s = \frac{\Delta\mathbf{x}_i \cdot C^s}{|C^s|^2} \tag{4.5.1}$$

This value gives the corresponding displacement in parametric space.

(4) Using the displacement in parametric space, the constrained bubble location on a curve at time $t+\Delta t$ is recalculated as $\mathbf{x}_i(t+\Delta t) = C(s(t)+\Delta s)$.

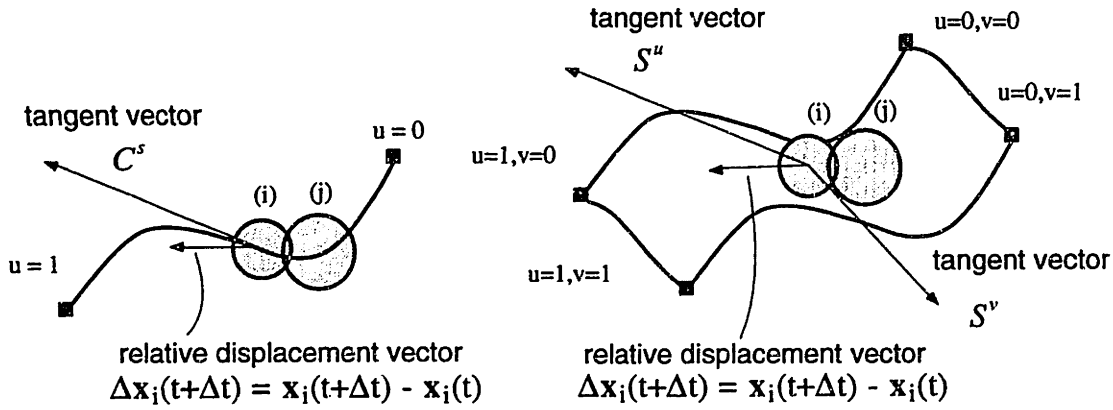


Figure 4.14 Confined bubble locations calculated by projecting an unconstrained displacement vector on the tangent vectors and calculating the corresponding displacement in parametric space.

Bubbles are constrained to remain on surfaces in a similar way using two normalized tangent vectors S^u and S^v, instead of C^s as in the curve case. The procedure is described as follows:

(1) Calculate the unconstrained displacement vector $\Delta\mathbf{x}_i = \mathbf{x}_i(t+\Delta t) - \mathbf{x}_i(t)$ in object space.

(2) Calculate the two normalized tangent vectors at the bubble location at time t, $S^u/|S^u|$ and $S^v/|S^v|$.

(3) Take the dot product of the unconstrained displacement vector and the normalized tangent vectors:

$$\Delta u = \frac{\Delta\mathbf{x}_i \cdot S^u}{|S^u|^2}$$
$$\Delta v = \frac{\Delta\mathbf{x}_i \cdot S^v}{|S^v|^2} \qquad (4.5.2)$$

These values give the corresponding displacement in parametric space.

(4) Using the displacement in parametric space, the constrained bubble location on a curve at time $t+\Delta t$ is recalculated as $\mathbf{x}_i(t+\Delta t) = S\big(u(t)+\Delta u, v(t)+\Delta v\big)$.

4.6 NODE SPACING

In most applications, node spacing must be controlled according to a given distribution function over the domain, since a uniformly small mesh size requires a larger number of mesh elements. Variable node spacing is one of the major difficulties for general 2D/3D mesh generation; some existing methods generate only discretely stepped node spacing, or allow node spacing only on boundaries. In this respect, the bubble mesh generation method is superior -- it allows the definition of an arbitrary, continuously varying node spacing all over the domain by modulating bubble sizes and packing them closely.

In this section, some representative node spacing functions are presented for various applications, such as analysis and computer graphics.

Analysis Error Based Node Spacing

The feasibility and accuracy of the finite element method is highly dependent on how the node spacing is controlled over the domain. Although a uniformly fine mesh will generate a feasible solution, it is usually too computationally costly for practical use. This is the reason why companies need experienced and diligent engineers for mesh creation.

For the same reason, adaptive mesh refinement procedures have been studied intensively. The main idea in adaptive mesh refinement procedures is to evaluate a new node spacing distribution from the results of a previous analysis, such that the newly refined mesh will improve the solution in the next analysis. A typical procedure is described as follows:

(1) Perform an initial analysis using a starting mesh, which can be uniformly sized, i.e. $d_{init}(x,y,z) = const$.

(2) Evaluate an error estimate $\|e\|$ based on the results of the analysis.

(3) If the error is within an allowable tolerance, stop the mesh refinement. Otherwise continue.

(4) Calculate a new node spacing distribution based on the error estimate in step (2), $d_{new}(x,y,z) = function\_of\_\|e\|$.

(5) Generate a new mesh which meets the node spacing calculated in step (4).

(6) Perform an analysis using the new mesh, and go to step (2).

There is a significant amount of technical content in the above procedure, such as how to give a quantitative measure to the magnitude of error in the analysis and how to relate the error to the node spacing. Intensive research efforts have been made on these issues and they are still active [LeeNS 93]. Although readers can refer to [Hinton 91], let us take a brief look at the underlying idea of one of these methods, residual based error estimation.

In a displacement based FE method, an approximate displacement field is given. For a compatible solution, the displacement should be continuous. However, the derivatives of the displacement, which imply strain and hence stress, are discontinuous across element boundaries. The error estimates in residual methods are calculated by using the stress jump around the element boundaries and the residual terms in the governing equation over the interior of the elements; residual terms remain in the force balancing equations, as all FE solutions are only approximations. Another major category of error estimation methods are labelled as the *best guess stress methods*, for details see [Zienkiewicz 87].

In brief, in all adaptive meshing methods, the errors in the analysis are evaluated based on the current discretization, then an improved node spacing distribution for the next analysis is predicted. From the point of view of mesh generation, the important fact is that all of these adaptive procedures provide a desired node spacing distribution over a domain, and a sophisticated meshing algorithm that can satisfy the node spacing distribution is required.

Curvature-Based Node Spacing

In surface rendering and rapid prototyping applications, a smooth curve or surface appears choppy and segmented (a polygonal silhouette) if the size of triangles in the generated mesh is too large. Although this problem can be solved by using uniform, fine triangles throughout the mesh, this increases the total number of triangles and requires more computational time and memory space.

A better solution, therefore, is to adjust the triangle size based on the maximum principal curvature. A surface region of high curvature is meshed with finer triangles, and a region of low curvature with larger triangles. This makes triangulation error constant throughout the surface, i.e., the distance between the original surface and the triangular mesh is controlled within a given tolerance ε everywhere on the surface.

To define a node spacing function that maintains constant error, a small portion of the surface around the point (u,v) in parametric space is approximated by a sphere with a radius equal to the minimum radius of curvature $\rho_{\min}(u,v)$ at that point, as shown in Figure 4.15. Two bubbles are then placed on this sphere such that they are tangent to each other. The maximum distance between: (a) the spherical surface, and (b) the line segment connecting the center points of the bubbles, corresponds to a given tolerance ε. From this simple geometric relationship, the node spacing, or bubble diameter, is given as follows:

$$d_c(u,v) = 2\sqrt{2\varepsilon\rho_{\min}(u,v) - \varepsilon^2} \tag{4.6.1}$$

See Figure 5.10 (a) for an example of meshing using this node spacing function.

View Based Node Spacing

Another useful node spacing definition for computer graphics applications is view-dependent node spacing, in which all the mesh triangles on the graphics screen are created with a uniform size. This prevents the system from drawing many unnecessarily small triangles for surface portions that are located at a far distance. This is made possible by defining the node spacing to be proportional to the distance between the viewpoint and the bubble's location in object space, yielding finer triangles for regions near the viewpoint and larger triangles for distant regions;

$$d_v(u,v) = a \cdot \overline{V_0 S(u,v)} \tag{4.6.2}$$

where V_o denotes a viewpoint in object space. (See Figure 5.10 (b).)

Curvature and View Based Node Spacing

By multiplying the curvature-based and view-dependent node spacing functions, $d_{cv}(u,v) = d_c(u,v) \cdot d_v(u,v)$, we combine both characteristics. (See Figure 5.10(c).)

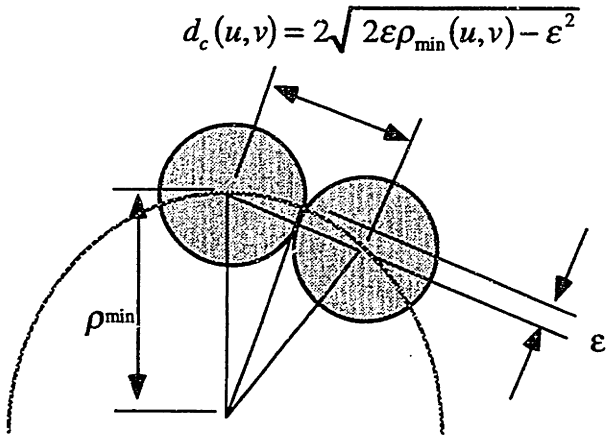


Figure 4.15 Curvature-based node spacing. The node spacing distribution is defined such that the error between the original surface and the planar approximation is maintained within an allowable tolerance ε.

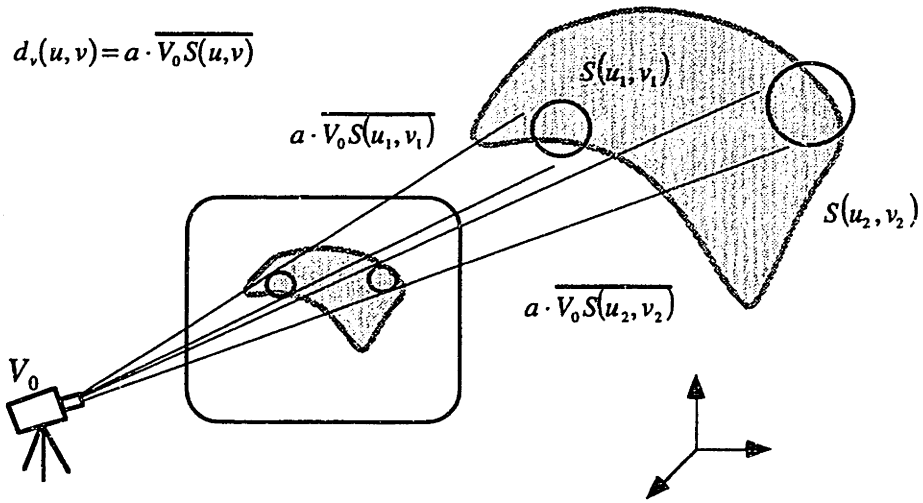


Figure 4.16 View based node spacing. The node spacing is controlled such that all the mesh triangles on the graphics screen are drawn with a uniform size. This helps to avoid drawing unnecessarily small triangles for surface portions that are distant from the viewer.

4.7 ADAPTIVE BUBBLE POPULATION CONTROL

For the meshing procedure to be fully automated, we must calculate the right number of bubbles to insert prior to or during the dynamic simulation. When a proper number of bubbles are placed in the region there are no significant gaps and overlaps among bubbles. A bubble population that is too small has many gaps, yielding larger mesh elements than will satisfy the given node spacing, and an excessive bubble population has many overlaps, yielding unnecessary small mesh elements.

One way to estimate an appropriate number of bubbles is to calculate the weighted length, area, and volume of the domain being meshed using the node spacing function as the weighting factor.

(1) Assume the force-balancing bubble configuration shown in Figure 3.12.

(2) Calculate the weighted length, area and volume -- with a specified mesh density of d, the weighted length is the one divided by d, the area d^2, and the volume d^3.

(3) Divide the total weighted length, area or volume by the amount occupied by a unit bubble, a bubble with a unit diameter.

In edge meshing, the length occupied by a unit edge bubble is equal to the bubble diameter, $L_0 = 1$. Therefore the right number of edge bubbles for an edge is estimated by the following equation:

$$(\#of\ edge-bubbles) = \frac{1}{L_0}\left[\int_{curve} \frac{1}{d(\mathbf{x})}dl - \frac{L_0}{2}\left(\frac{1}{d(\mathbf{x}_{end1})} + \frac{1}{d(\mathbf{x}_{end2})}\right)\right] \qquad (4.7.1)$$

The first term of the right hand side of the equation calculates the weighted length of the edge, the second term and the third term calculate the weighted length occupied by the two vertex-bubbles at the edge perimeters.

In face meshing, a hexagonal bubble configuration is assumed, making the area occupied by a unit face bubble $A_0 = \sqrt{3}/2$; in volume meshing, the unit volume-bubble occupies the volume $V_0 = \sqrt{2}/2$. The following equations estimate the right number of bubbles in face-meshing and volume-meshing.

$$(\#\,of\ face-bubbles) = \frac{1}{A_0}\left[\int_{surface} \frac{1}{d(\mathbf{x})^2}\,dA - \frac{A_0}{2}\sum_{\substack{edge-bubbles\\ vertex-bubbles}} \frac{1}{d(\mathbf{x})^2}\right] \tag{4.7.2}$$

$$(\#\,of\ volume-bubbles) = \frac{1}{V_0}\left[\int_{volume} \frac{1}{d(\mathbf{x})^3}\,dV - \frac{V_0}{2}\sum_{\substack{face-bubbles\\ edge-bubbles\\ vertex-bubbles}} \frac{1}{d(\mathbf{x})^3}\right] \tag{4.7.3}$$

One more problem here is where to place the bubbles once the proper number of bubbles has been calculated. For a simple convex 2D or 3D domain, all bubbles can be injected anywhere inside the domain; the bubbles will eventually be distributed in the domain due to the repelling and attracting forces. This, however, does not always work for domains with constrictions as shown in Figure 4.17. Although, it might be possible to find a narrow constriction beforehand and cut the domain into convex sub-domains [Patrikalakis 90, Gursoy 89], it is not straightforward; the size of the bubbles and the width of the constriction would have to be compared to see whether or not the bubbles could pass through the constriction.

Another more promising way to obtain the right number of bubbles, actually implemented in our system, is to adjust the number of bubbles adaptively during the course of the dynamic simulation, called *adaptive bubble population control.* It is implemented to erase *excess bubbles*, and to add bubbles around *open bubbles*. An excess bubble is one which significantly overlaps its neighbors and consequently should be deleted. An open bubble is one that lacks an appropriate number of neighboring bubbles, creating a gap in the bubble configuration. Additional bubble(s) must be added around an open bubble. A given region is always filled with a sufficient number of bubbles, without gaps or overlaps, through this method.

This automatic process enables the system to reach equilibrium very quickly, by avoiding the slow, natural dispersion of bubbles from densely packed to sparse areas. In order to check whether a bubble is an open bubble or an excess bubble, it is necessary to evaluate the "overlapping ratio" for each case as shown in Figure 4.18. The overlapping ratio is based upon the length of penetration of one bubble into the neighboring bubble. The penetration length is expressed in terms of the radius of a neighboring bubble overlapped, or penetrated, by a center bubble.

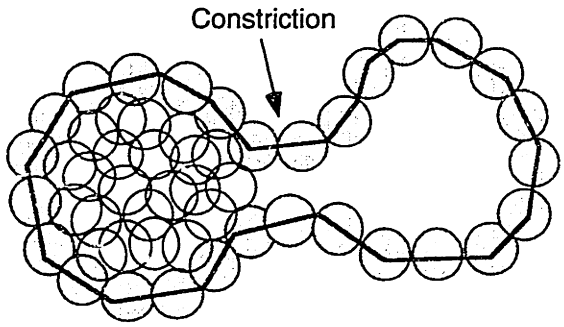


Figure 4.17 Bubble placement in a 2D region with a constriction. Even if the right number of bubbles to fill the region is known, it is not always straightforward to allocate the right number of bubbles to each sub-region. The same problem occurs in a 3D region.

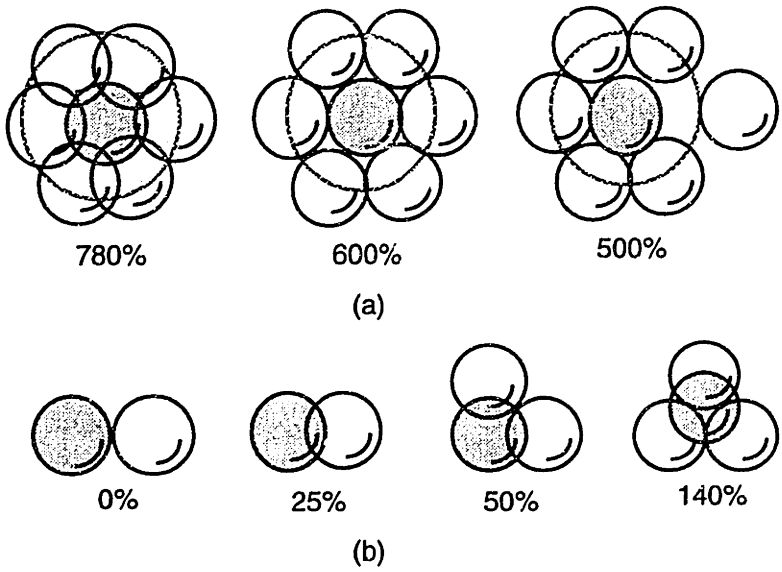


Figure 4.18 Criteria for adding and deleting bubbles. (a) An open bubble with an overlap ratio less than 500% is subdivided into two. (b) An excess bubble with an overlap ratio greater than 140% is erased.

To determine whether a bubble is open or not, we consider a bubble with a diameter twice as large as the bubble under evaluation, and then check the overlapping ratio of its neighboring bubbles. The overlapping ratio is calculated by first summing the penetration lengths of the double-sized bubble into its neighbors and then dividing by the original bubble radius.

Thus, if the original bubble were surrounded by six neighboring bubbles, all tangent to each other and all of uniform size, the overlapping ratio would be 600%. The bubble is

not open, and, therefore, there are no gaps to fill. As shown in Figure 4.18(a), however, if the overlapping ratio is too small, e.g., approximately 500% or less, the bubble is open. It does not have enough neighbors, and thus another bubble should be added to fill the gap.

The criterion for an excess bubble is a similar overlapping ratio between the bubble and its neighbors. For example, if three bubbles tangent to each other are overlapped by a bubble placed at the center of the grouping, and all bubbles are of uniform size, the overlapping ratio is approximately 140%. Thus, the overlapping bubble is excess and should be deleted (See Figure 4.18 (b)). Thus, in this evaluation, if the ratio is too large, e.g., approximately 140% or more, the bubble is excess and should be erased.

4.8 REMESHING

4.8.1 Local Remeshing

One advantage of the bubble method is its local remeshing capability. Local remeshing allows one to re-generate a mesh only in the partial regions where meshes are merged, while the remaining portions of meshes are utilized as they are. At least two motivations make this capability practically important.

- The amount of computation required for local remeshing is smaller than for remeshing the whole domain.

- A particular pattern of mesh, or node spacing distribution, may be required in a local region.

The second point is especially important in engineering analysis, in which form-features, such as holes, cracks, and sharp corners, are associated by analysis engineers to pre-defined mesh patterns. An example of such patterns is a mesh with increasing node spacing in the radial direction, specified in the vicinity of a hole or a crack where a stress concentration is expected.

Figure 4.19 shows a simple example of local remeshing performed by merging two node spacing functions. Shown in Figure 4.19(a) is a base geometry with a uniform node spacing. The task is to merge a form feature with varying node spacing as shown in Figure 4.19(b) onto the base geometry.

In figures (a) and (b), node spacing functions are shown on a horizontal cross section, and they are merged in (c) by taking the minimum values of the node spacing functions shown in (a) and (b). The value of the merged node spacing is equal to (a) in sub-regions AE and FB, and equal to (b) in sub-regions EC and DF. As shown in (c), if two bubbles are simply combined, bubbles are highly overlapped causing ill-shaped mesh elements around the connection regions, points E and F. In order to avoid the ill-shaped elements, the dynamic simulation discussed in Section 4.3 must be performed.

Instead of solving the dynamics for all of the bubbles, we can move bubbles selectively as shown in Figure 4.19(d). Bubbles around points E and F are selectively specified as free bubbles, which can move freely in the dynamic simulation, and the rest of the bubbles are fixed. Note that the population of bubbles is controlled adaptively using the mechanism described in Section 4.7. In the dynamic simulation, the degrees of freedom of the bubble system are reduced, so the amount of computation required is less than that required for remeshing the whole domain from scratch.

Figure 4.20 illustrates an example of an incremental merge of pre-defined meshes. In the example, two rectangles with pre-defined meshes, shown in (a), are merged to yield a single mesh as shown in (e). In Figure 4.20(b) two sets of bubbles are simply superimposed. The dynamic simulation with adaptive bubble population control is also performed during this step on the bubbles in the overlapped region in order to move them to a force balancing configuration. After a satisfactory bubble configuration is obtained, the center points of the bubbles are connected for a complete mesh topology as shown in Figure 4.20 (e).

Procedures for local remeshing can be best summarized in the following steps:

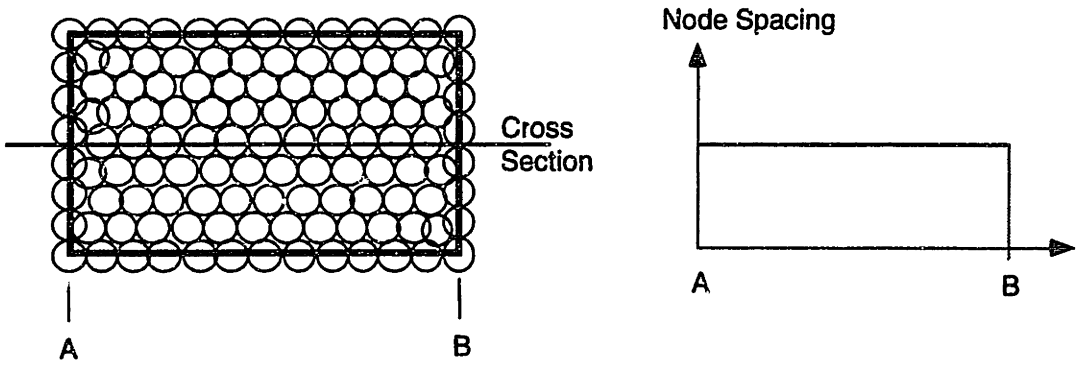
(1) Superimpose two sets of bubbles.

(2) Erase unnecessary bubbles.

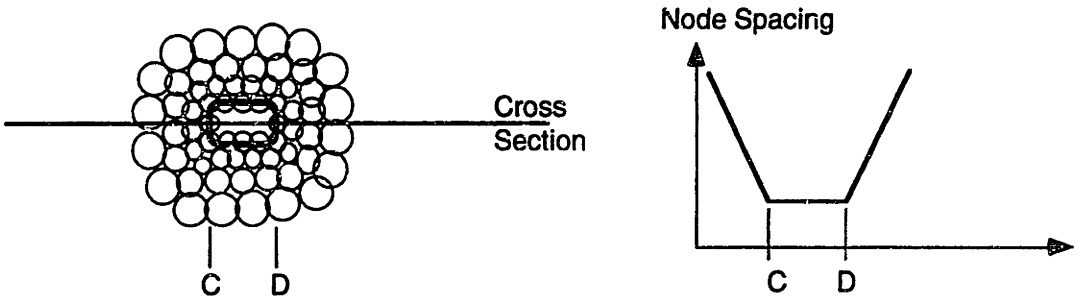
(3) Select bubbles to be moved for a force balance.

(4) Perform the dynamic simulation with adaptive population control.

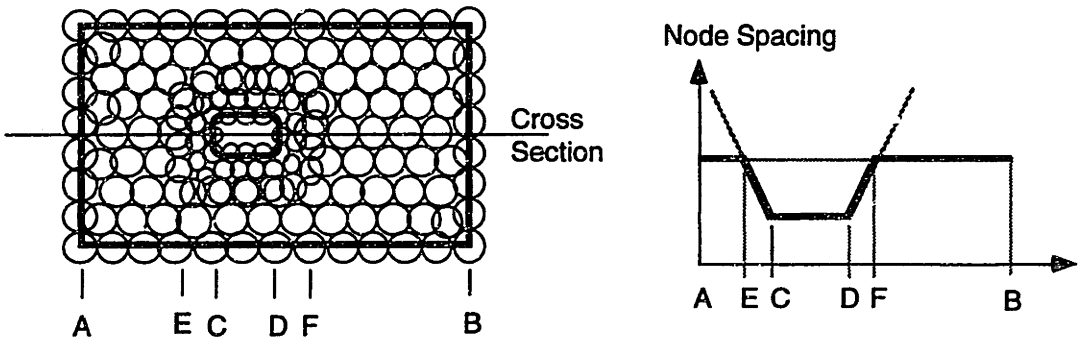
(5) Connect center points of bubbles for a complete mesh topology.



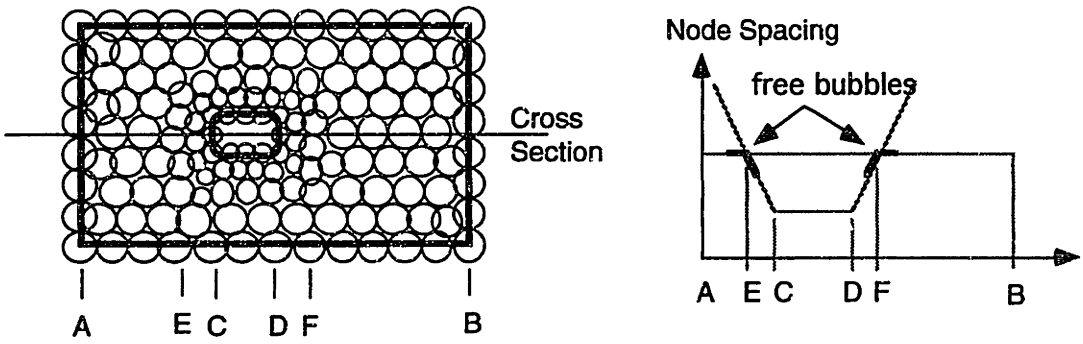
(a) Base geometry with constant node spacing



(b) Form feature with varying node spacing



(c) Base geometry and form feature with combined node spacing



(d) Force balancing bubble configuration

Figure 4.19 Feature-based mesh generation with bubble systems. Two node spacing functions are merged by taking their minimum.

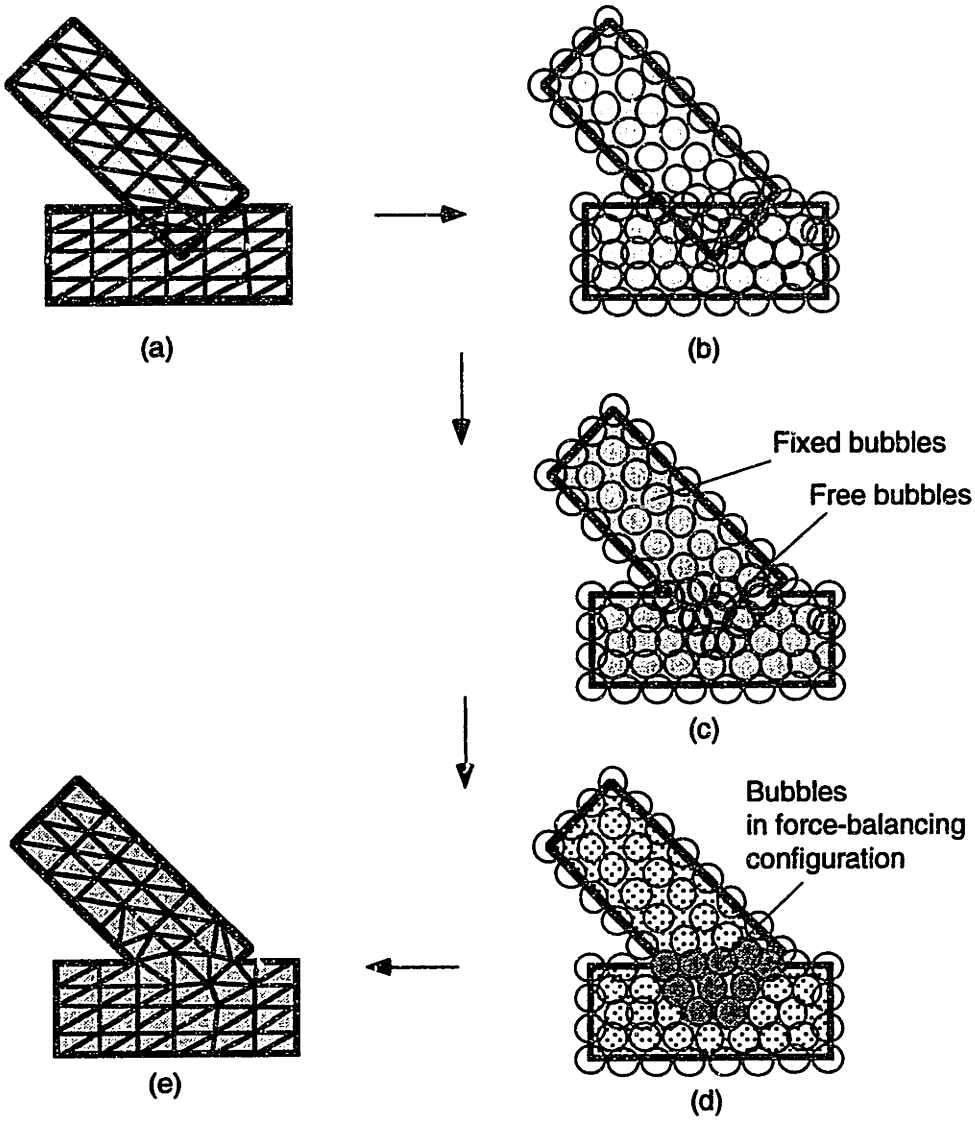


Figure 4.20 Merging two meshes using bubble systems. Two sets of bubbles are combined and overlapping bubbles are deleted. The rest of the bubbles are fixed and new free bubbles are filled in the gap, or intersection region of the two geometries. In this way, only a partial region needs to be remeshed.

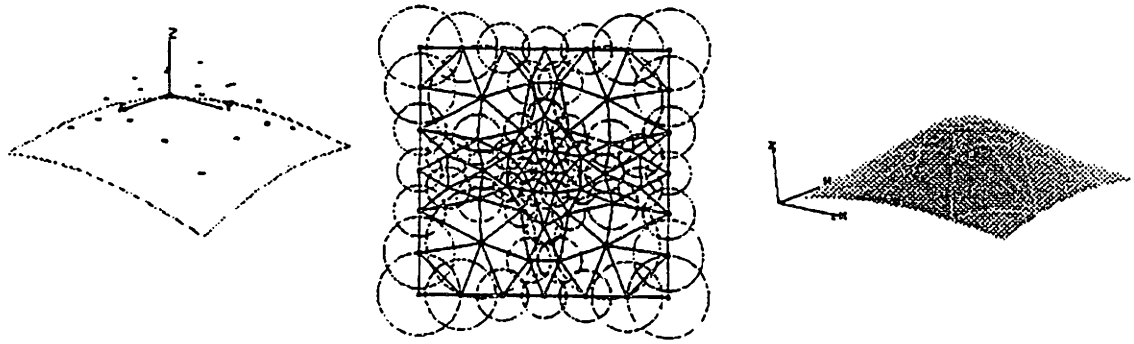
4.8.2 Continuous Remeshing

The bubble system is also very powerful when continuous remeshing is required, for example when the surface geometry and the node spacing are changing incrementally. The original bubble configuration can be used as an initial condition for the next meshing step; we need only a few relaxation steps to reach the next equilibrium. Applications requiring continuous remeshing include surface reconstruction and large deformation FEM/BEM analysis.

Surface reconstruction, in which a surface is generated to fit a set of scattered 3D data points, is a common requirement in numerous applications including machine vision, medical image processing, and geometric design. The bubble mesh generation scheme has been used to aid in a new method of surface reconstruction. (See [Fang 92] for details.)

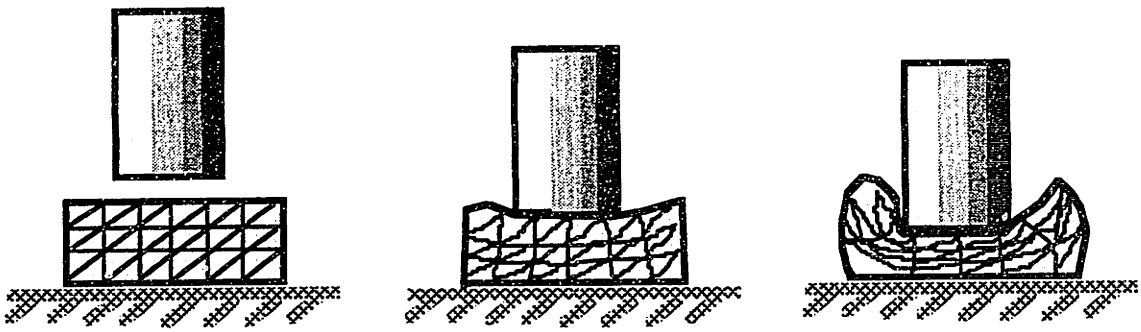
In this method, a deformable surface is represented as a set of connected triangular patches, whose deformation (stretching and bending) is defined by the minimization of an energy functional. As no mathematical surface representation exists, we start with a flat surface with triangles of uniform size. A set of imaginary springs is defined to connect the 3D data points and the surface, and they pull and deform the surface. As the surface is deformed, its triangular mesh is reconstructed using curvature-based node spacing to produce more triangles around high-curvature, highly-stretched regions. Figure 4.21 shows an example of this process. Given a set of scattered data points as shown in (a), triangular patches are created in parametric space as shown in (b), and the surface is deformed in object space to fit the given data points as shown in (c).

Continuous remeshing over changing geometry and node spacing is similarly necessary in large deformation FEM/BEM analysis. When deformed, a domain must be remeshed because its original triangles often become too thin and distorted to yield an accurate FEM/BEM solution. Furthermore, the critical region of high stress/strain may shift such that a different node spacing must be defined.

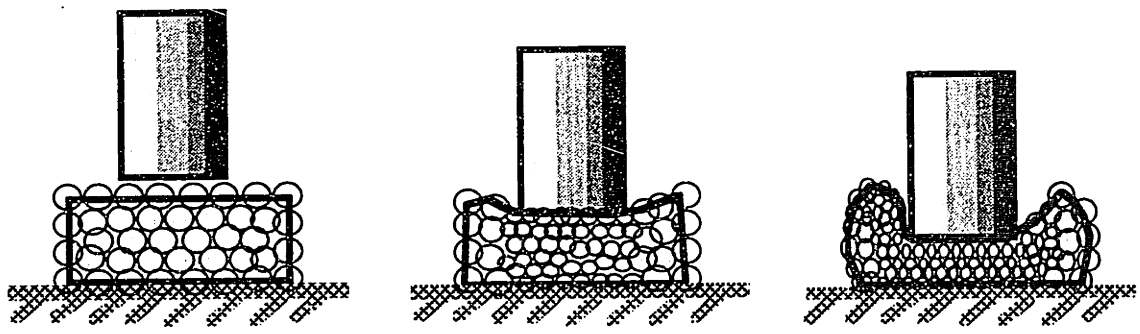


(a) Scattered data points (b) Reconstructed mesh in parametric space (c) Reconstructed surface in object space

Figure 4.21 Surface reconstruction from unorganized data points using a deformable surface. A set of imaginary springs is defined to connect the 3D data points and the surface, and they pull and deform the surface. Underlying triangular patches of the deformable surface in parametric space were obtained using a bubble system.



(a) Large deformation FEM/BEM analysis



(b) Incremental remeshing with bubble systems

Figure 4.22 Large deformation FEM/BEM. The bubble method is especially powerful when continuous remeshing is required such as in large deformation analysis, in which an original mesh is deformed so much that some triangles are too skinny for an accurate analysis.

5 Results and Implementations

This chapter presents some sample output generated by the bubble method described in Chapter 3 and Chapter 4. In Section 5.1, results of 1D, 2D, surface, and 3D meshes are shown to prove the advantages over the conventional methods explained in Chapter 2. Note the regularity of the geometry and topology of the elements, the flexible remeshing capability, and the precise, continuous node spacing control. No post-processes for improving mesh quality, such as Laplacian smoothing, were applied for the results shown in this chapter; elements generated by the bubble method are so well-shaped that no smoothing operations are required.

Section 5.2 presents an implementation of the bubble method employed in an interactive surface design system. Since we utilize an underlying surface representation of FEM-based triangular patches, the geometry and topology of the triangular mesh must be reconstructed every time a user deforms or trims a portion of a surface This is a good example of the bubble method's application, where element shape regularity and node spacing are essential requirements for achieving an accurate FEM solution in a small computational time.

5.1 RESULTS OF BUBBLE MESH GENERATION

In this section, results are shown in the order of dimension. Note that a higher dimension meshing requires lower dimension processes; i.e. 2D meshing and surface meshing require 1D meshing; and 3D requires 1D, 2D, and surface meshing. The complete meshing procedure is stated in Section 3.2.3.

5.1.1 1D Meshing

In 1D meshing, a planar or space curve is subdivided into a series of line segments according to a given node spacing. Figure 5.1 shows the result of performing 1D meshing on a cubic spline. A uniform node spacing is specified in Figure 5.1(a), yielding uniformly sized bubbles distributed on the curve. One problem of the uniform segment size is that if the subdivision is too coarse, as in the picture, a smooth curve becomes choppy and segmented in appearance (a polygonal silhouette). Although this problem can be solved by using uniform, small segments throughout the curve, this causes inefficiency by increasing the total number of elements.

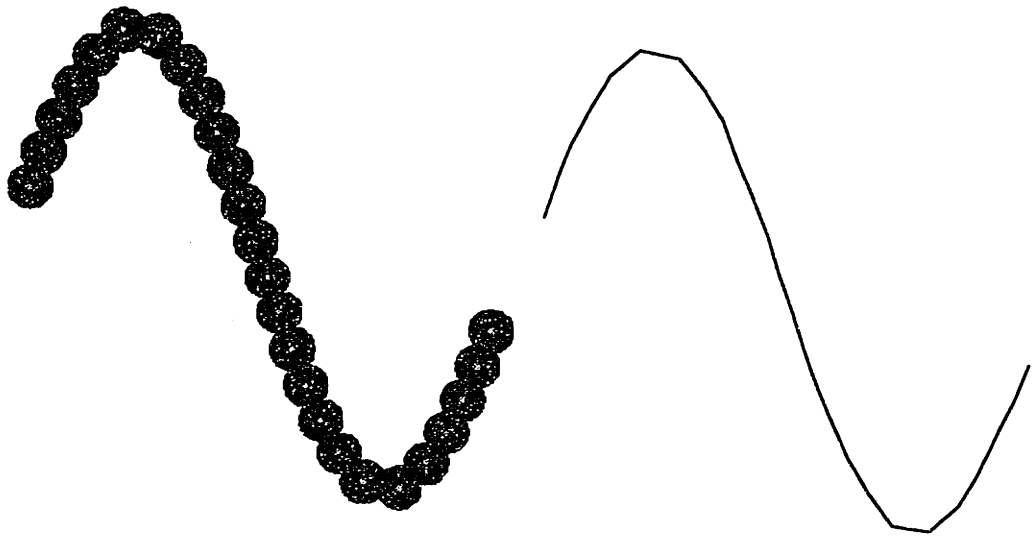
A better solution, therefore, is to adjust the element size based on the curvature so that high curvature regions are approximated with shorter line segments, and low curvature regions with longer line segments. This makes the approximation error constant throughout the curve; i.e., the distance between an original curve and a polyline is controlled within a given tolerance ε everywhere on the surface. The result in Figure 5.1(b) was obtained using a curvature-based node spacing, which was calculated as:

$$d_c(s) = 2\sqrt{2\varepsilon\rho(s) - \varepsilon^2} \tag{5.1.1}$$

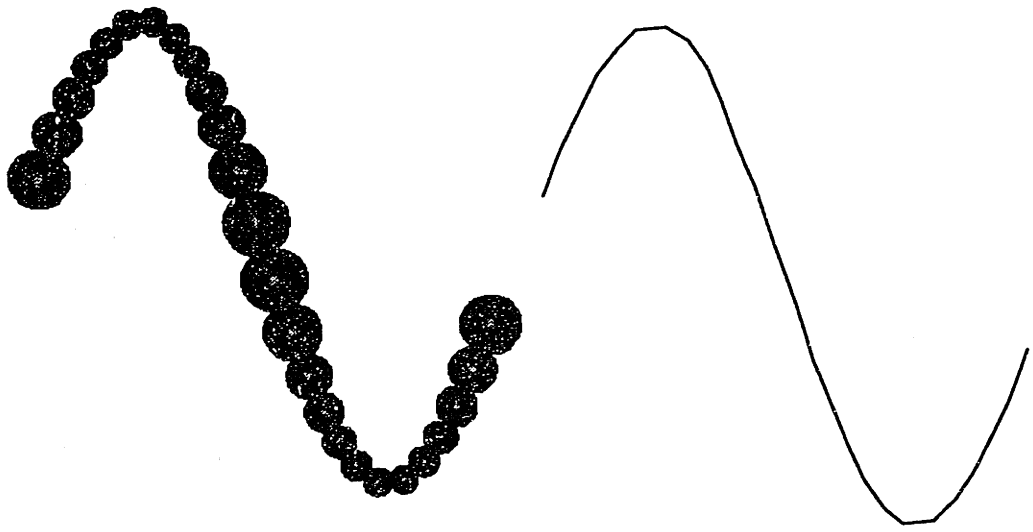
where $\rho(s)$ represents the curvature. (See Figure 4.15 for the derivation of this equation.)

5.1.2 2D Meshing

As also discussed in Section 4.7, packing the right number of bubbles is key to the regularity of a mesh -- using more or fewer bubbles than sufficient and necessary will introduce some irregularity into a mesh. To illustrate this problem, six different meshing examples are depicted in Figure 5.2, where a regular hexagon is subdivided into triangles using a uniform node spacing, $d(x,y) = C_0$.



(a) Uniform node spacing



(b) Curvature-based node spacing

Figure 5.1. 1D meshing examples with a curvature-based node spacing, $d_c(s)=2\sqrt{2\varepsilon\rho(s)-\varepsilon^2}$.

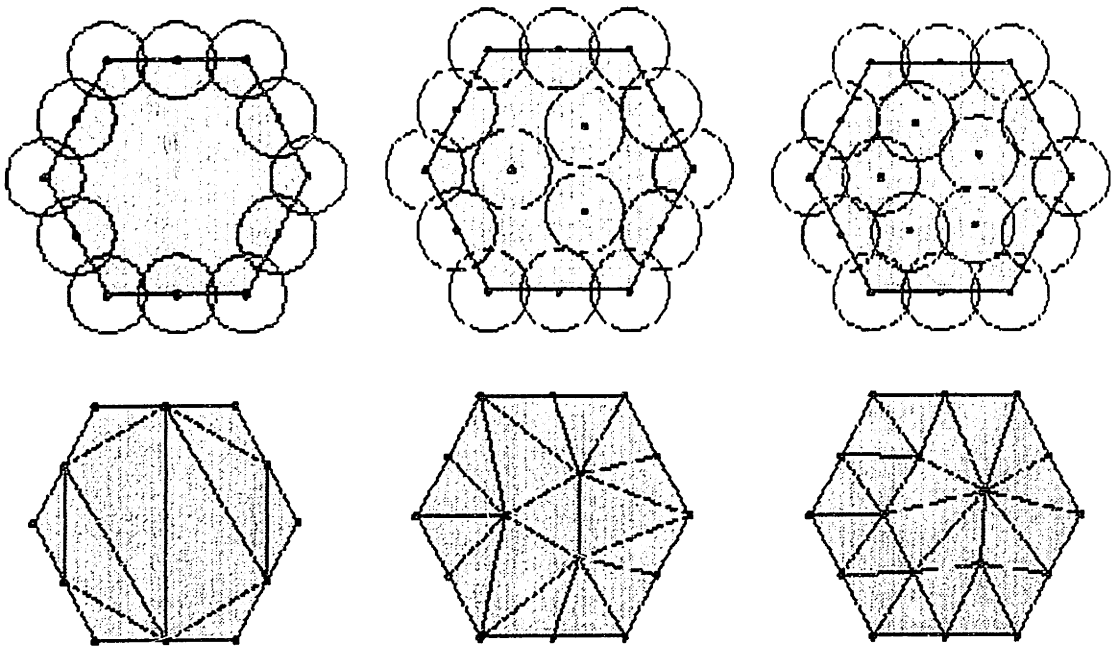
Since node spacing, or bubble diameter, is specified to be a little longer than half of a hexagon edge, each edge is filled with three bubbles. Seven bubbles are sufficient to fill the inside region as shown in Figure 5.2 (d), yielding a nearly perfect tangency. The resultant mesh is geometrically and topologically regular; all triangles are equilateral, and all inside nodes have six neighbors. If some bubbles are intentionally erased or added from this best packing, some irregularity is introduced as shown in Figure 5.2 (a), (b), (c), (e) and (f), where inside nodes have an irregular number of neighbors.

Figure 5.3 shows meshes from two stages of bubble methods with a radially increasing node spacing function, $d(x,y)=0.1+(x-0.5)^3+(y-0.5)^3$. Note that significant gaps and overlaps are created in the first stage, yielding badly distorted, or thin, triangles. These gaps and overlaps are minimized during the course of the dynamic simulation in the second stage.

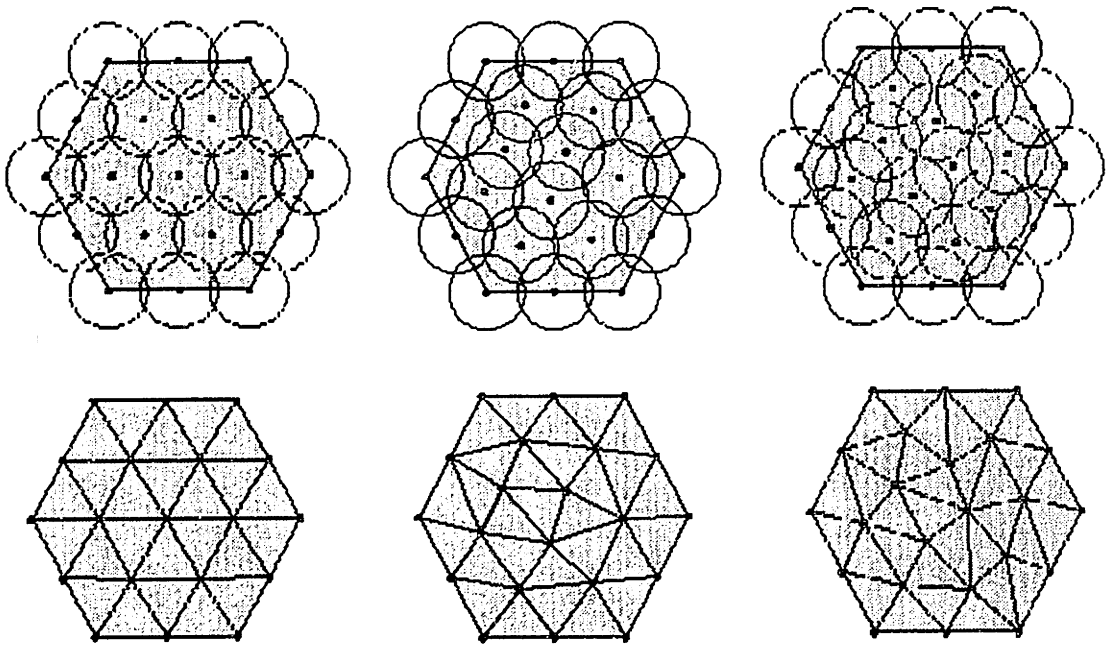
Figures 5.4, and 5.5 illustrate some results of 2D triangulation with uniform node spacing, shown on the left hand side, and with varying node spacing, $d(x,y)=x^2$ and $d(x,y)=x^3+y^3$, shown on the right hand side. In both figures, the first rows illustrate specified node spacing; the second rows packed bubbles in 2D domains; the third shows the corresponding constrained Delaunay triangulation found by using the center points of bubbles as the nodes; and in the bottom each triangle is further subdivided into four triangles by introducing three nodes on the edges of each original triangle. Note that no significant gaps and overlaps are created in the bubble packing configuration, the irregularity in topology and geometry is minimized so that all the triangles are well-shaped.

A process converging to a force-balancing, or minimum energy, configuration of bubbles is illustrated in Figure 5.6. In the figure, the configuration shown in the top row is an initial condition generated by random bubble placement, yielding many ill-shaped triangles. As the equations of motion are integrated numerically, the bubbles move to a minimum energy configuration. Note that the number of bubbles is adjusted adaptively by removing highly-overlapping ones and by dividing lone ones in wide openings.

Figure 5.7 illustrates how the total energy, the summation of kinetic energy and potential energy, is minimized through this dynamic simulation. Since we assume that bubbles are standing still in the initial configuration, the kinetic energy at the starting time is zero. It increases, then finally is reduced to almost zero again.



(a) 0 bubbles (b) 3 bubbles (c) 5 bubbles

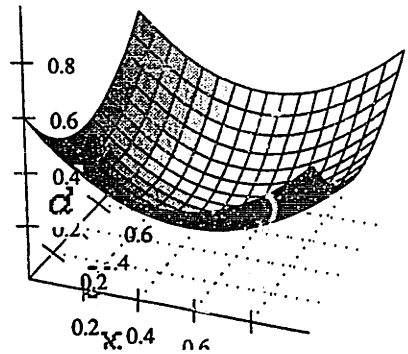


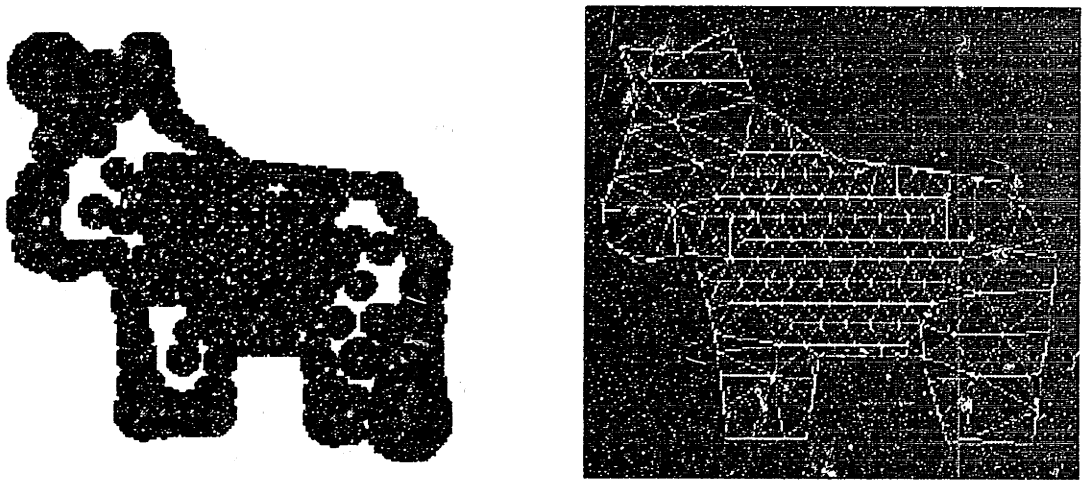
(d) 7 bubbles (e) 9 bubbles (f) 11 bubbles

Figure 5.2 Topology and geometry irregularities in 2D meshing with a constant node spacing, $d(x,y) = C_0$.

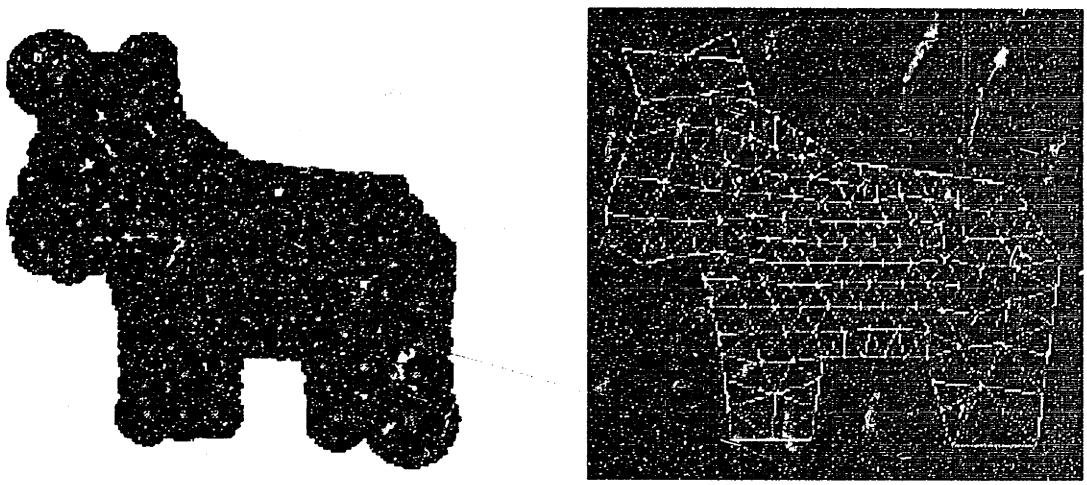
(a) Node spacing function:

$$d(x,y)=0.1+(x-0.5)^2+(y-0.5)^2$$



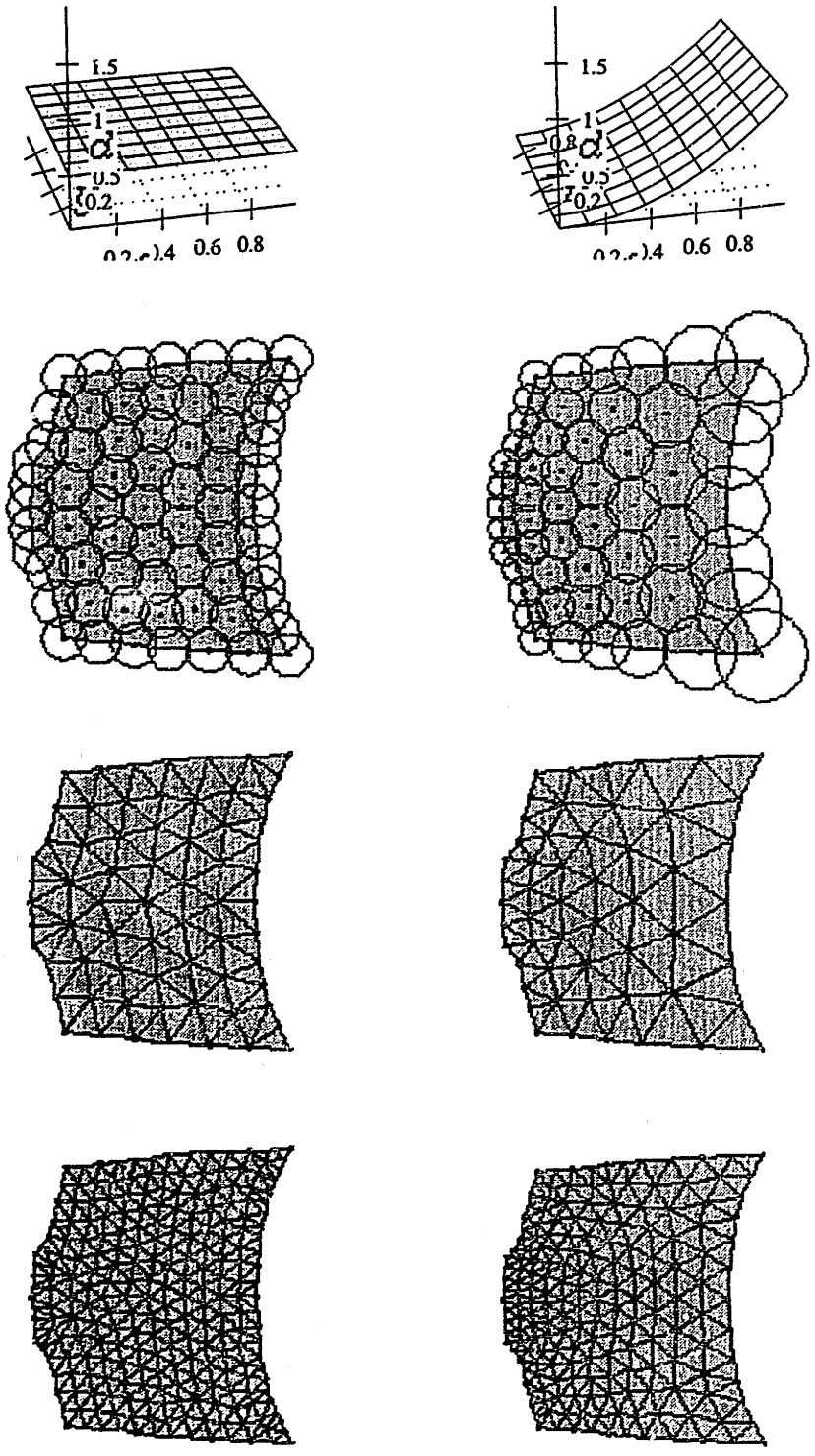


(b) Stage 1: Initial bubble configuration and the mesh



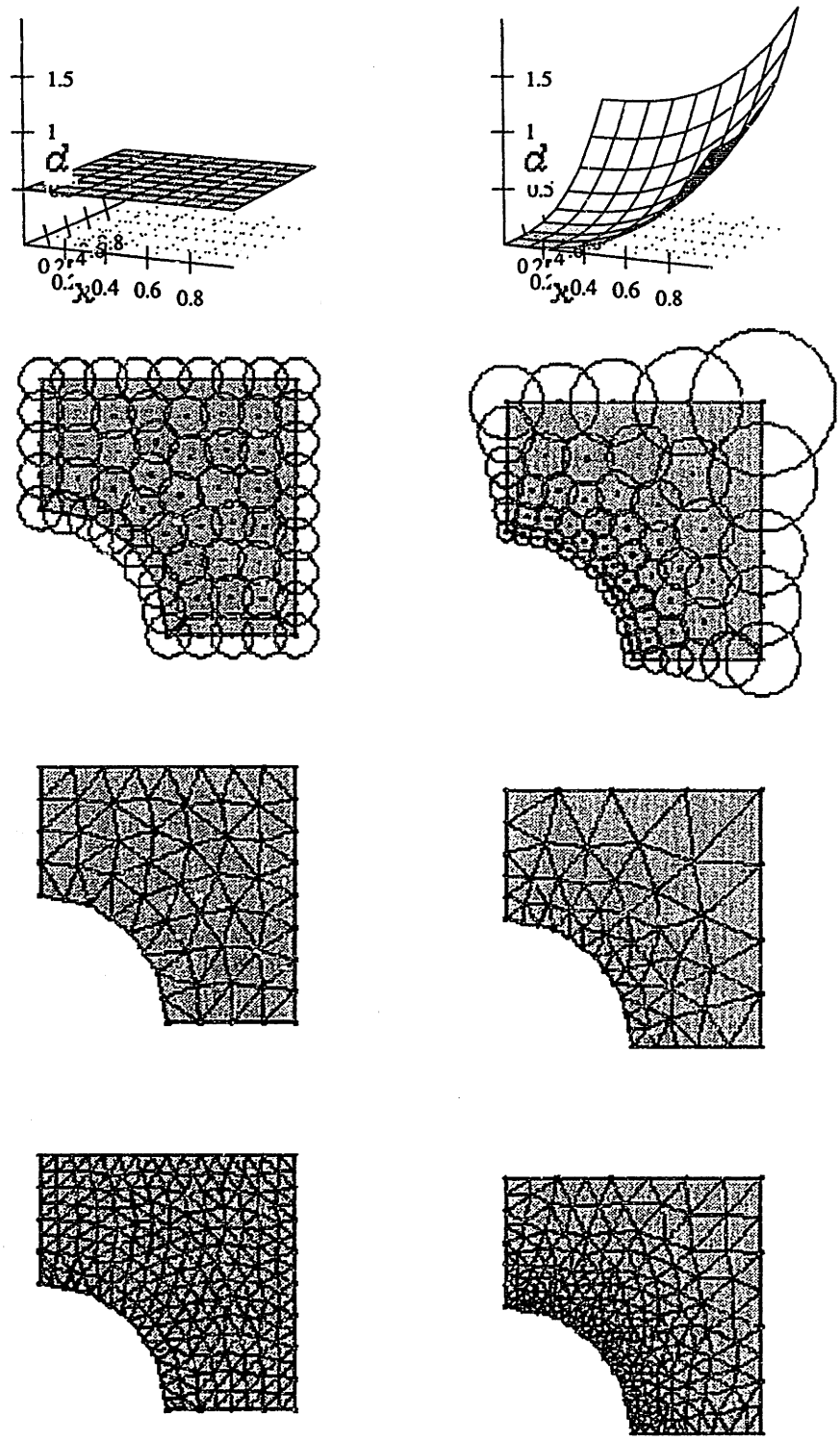
(c) Stage 2: Force-balancing bubble configuration and the mesh

Figure 5.3 Two stages of 2D meshing procedures



(a) Uniform node spacing (b) Varying node spacing

Figure 5.4 2D meshing example 1 with uniform node spacing and varying node spacing, $d(x,y)=x^2$.



(a) Uniform node spacing (b) Varying node spacing

Figure 5.5 2D meshing example 2 with uniform node spacing and varying node spacing, $d(x,y) = x^3 + y^3$

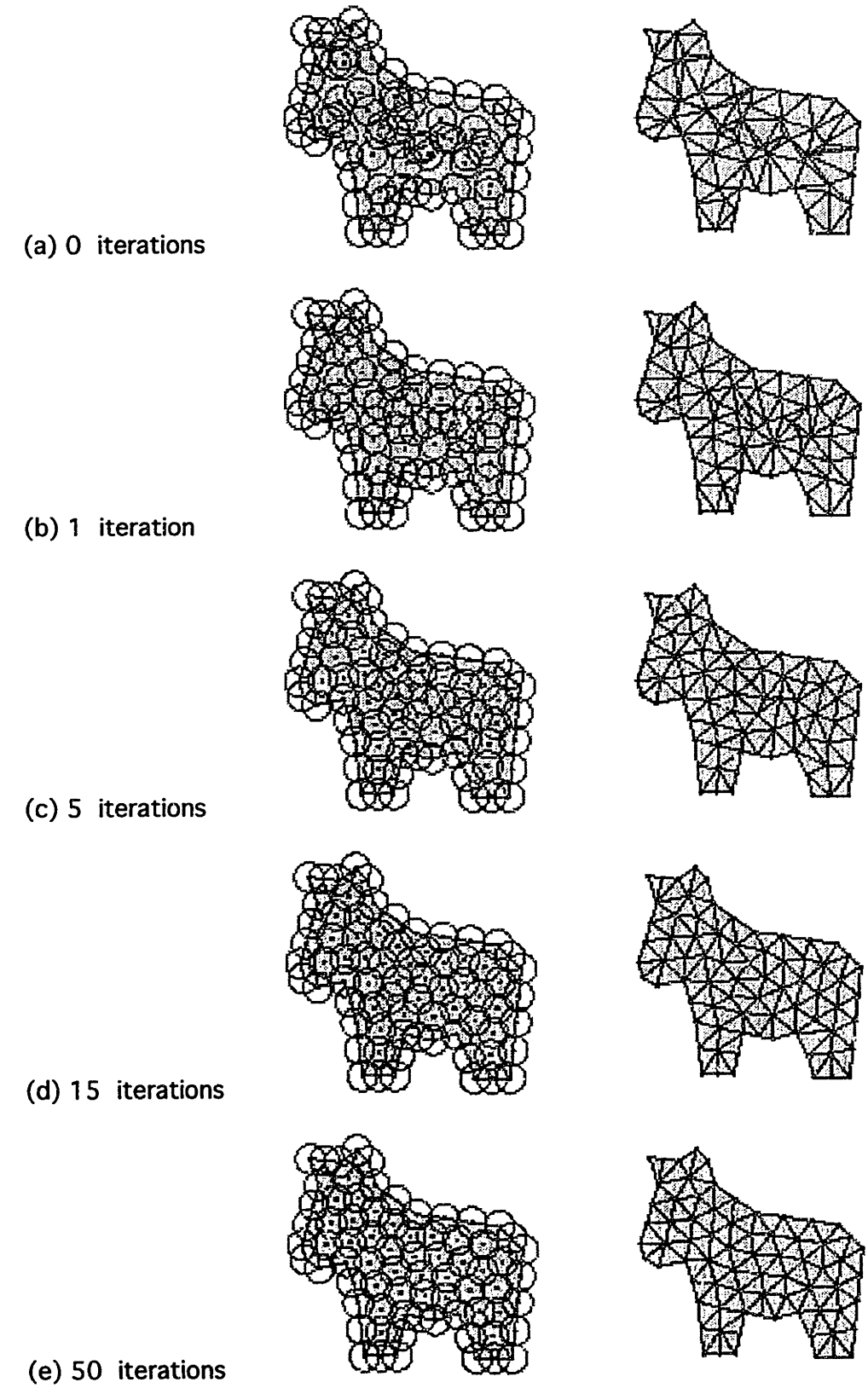
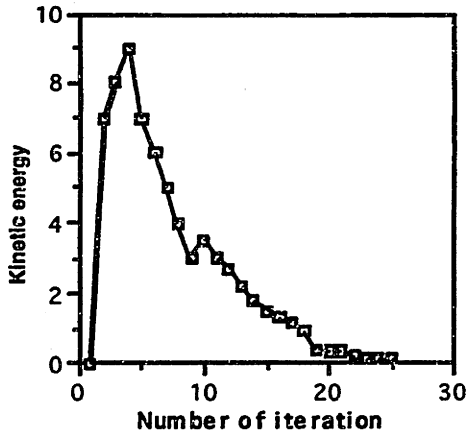
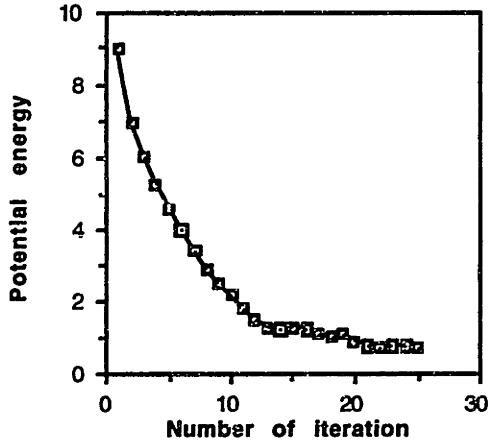


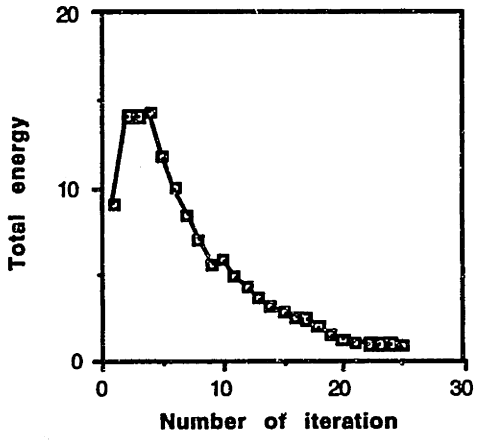
Figure 5.6 Dynamic simulation of 2D bubbles.



(a) Kinetic energy



(b) Potential energy



(c) Total energy

Figure 5.7 Energy minimization in 2D bubble dynamic simulation.

5.1.3 Surface Meshing

Figure 5.8 depicts the four stages of surface meshing: (1) vertex bubble placement; (2) edge bubble placement; (3) face bubble placement; and (4) constrained Delaunay triangulation. The result shown in the figure is obtained by the dynamic simulation done in the second stage of the bubble method.

When an adjustment between speed and quality is necessary, the two-stage meshing procedure is employed as shown in Figure 5.9. A mesh as in Figure 5.9 (a) is obtained quickly by the recursive spatial subdivision procedure described in Section 4.1, yielding a relatively inferior quality -- note that the mesh has thin elements around the boundary and in the region where the grid size changes.

A better quality mesh is produced as shown in Figure 5.9 (b) by the physically-based relaxation procedure. In the figure, node spacing is controlled based on the curvature. A surface region of high curvature is meshed with finer triangles, and a region of low curvature with larger triangles. This makes the triangulation error constant throughout the surface; the distance between the original surface and the triangular mesh is controlled within a given tolerance ε everywhere on the surface. The actual node spacing function utilized is:

$$d_c(u,v) = 2\sqrt{2\varepsilon\rho_{\min}(u,v) - \varepsilon^2} \qquad (5.1.2)$$

Figure 5.10 shows three meshes based on different node spacing functions specified over a surface: curvature-based; view-dependent; and a combination of curvature and view based. (See Section 4.6 for the derivation of the node spacing function.) Note that, in case of curvature-based node spacing shown in Figure 5.10 (a), the highly curved center portion of the surface is faceted with finer triangles. With the view-dependent node spacing shown in (b), all the triangles on the graphic screen (object space) are of uniform size.

In Figure 5.11, a mesh with uniform node spacing and a mesh with curvature-based node spacing are compared. Note that the polygonal silhouette is less obvious with curvature-based node spacing.

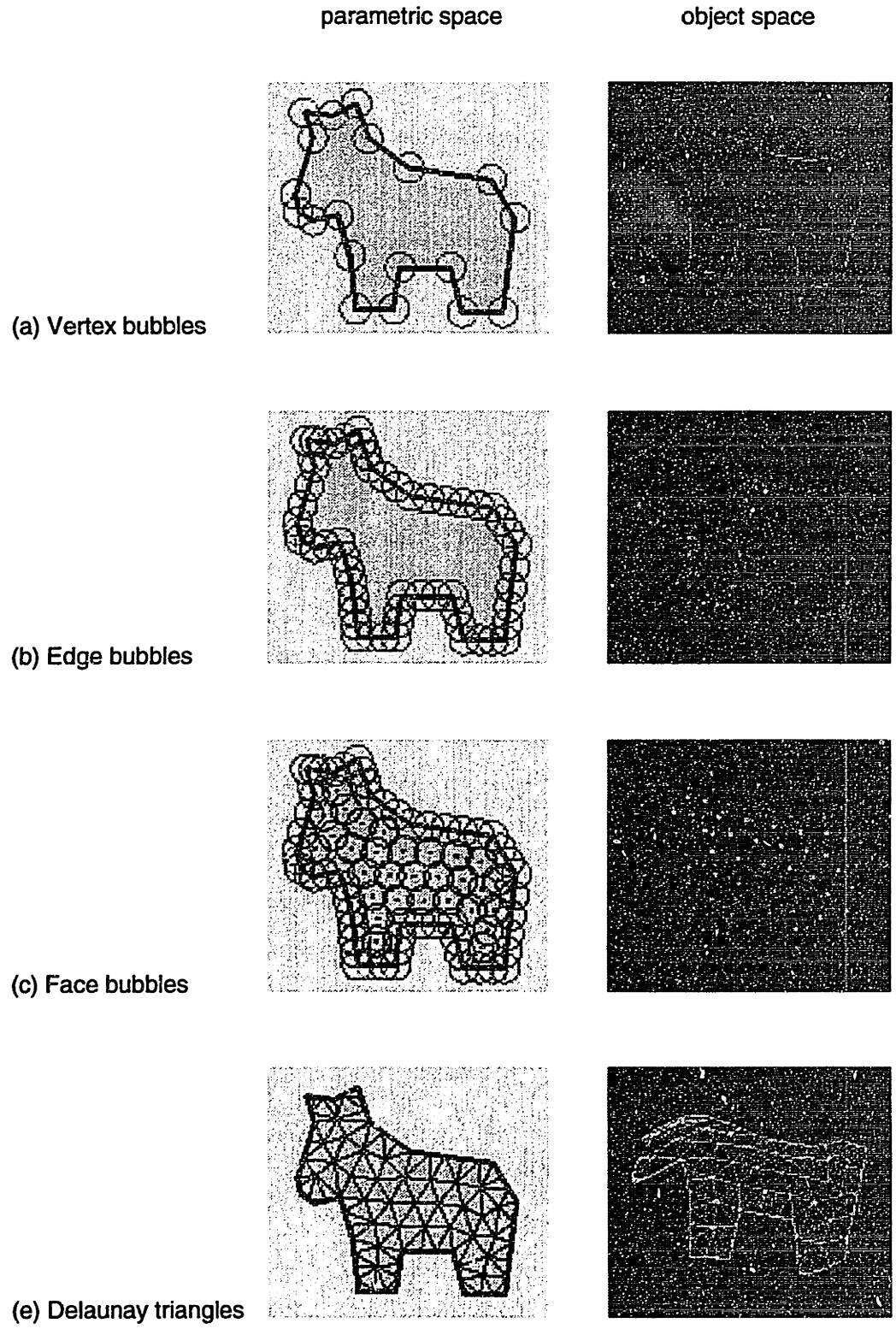
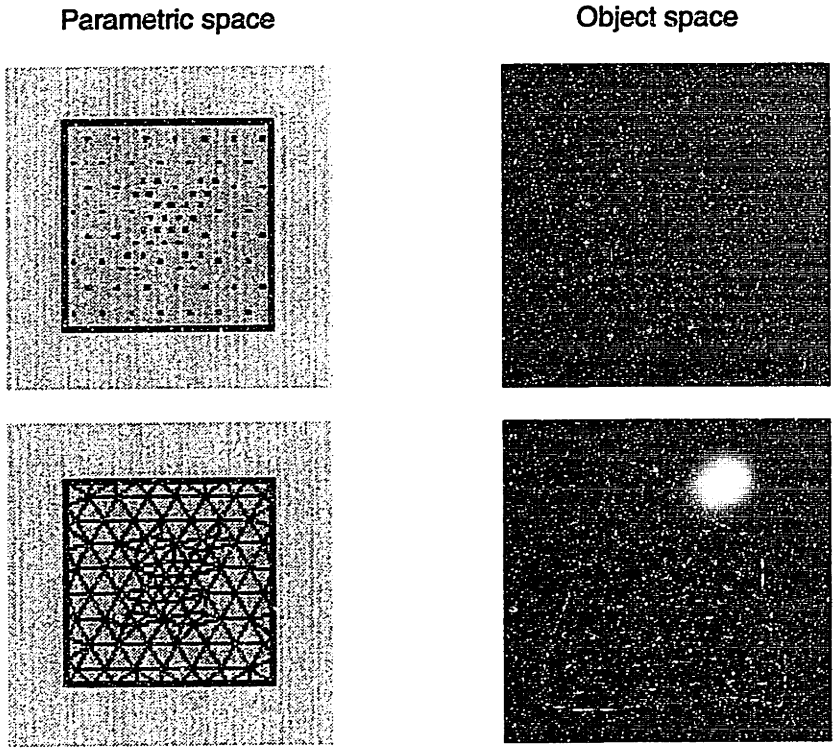
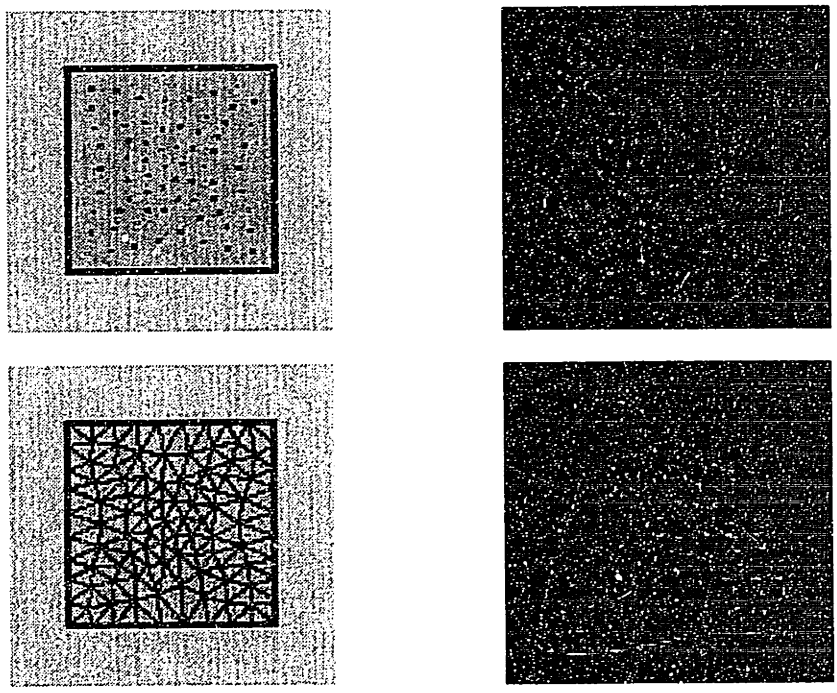


Figure 5.8 Example of surface meshing procedure with uniform node spacing.



(a) Stage 1: Initial bubble configuration and the mesh



(b) Stage 2: Final bubble configuration and the mesh

Figure 5.9 The two stages of surface triangulation. (Stage 1) is performed quickly but generates a mesh with relatively inferior quality. (Stage 2) uses physically-based relaxation and creates a high quality mesh.

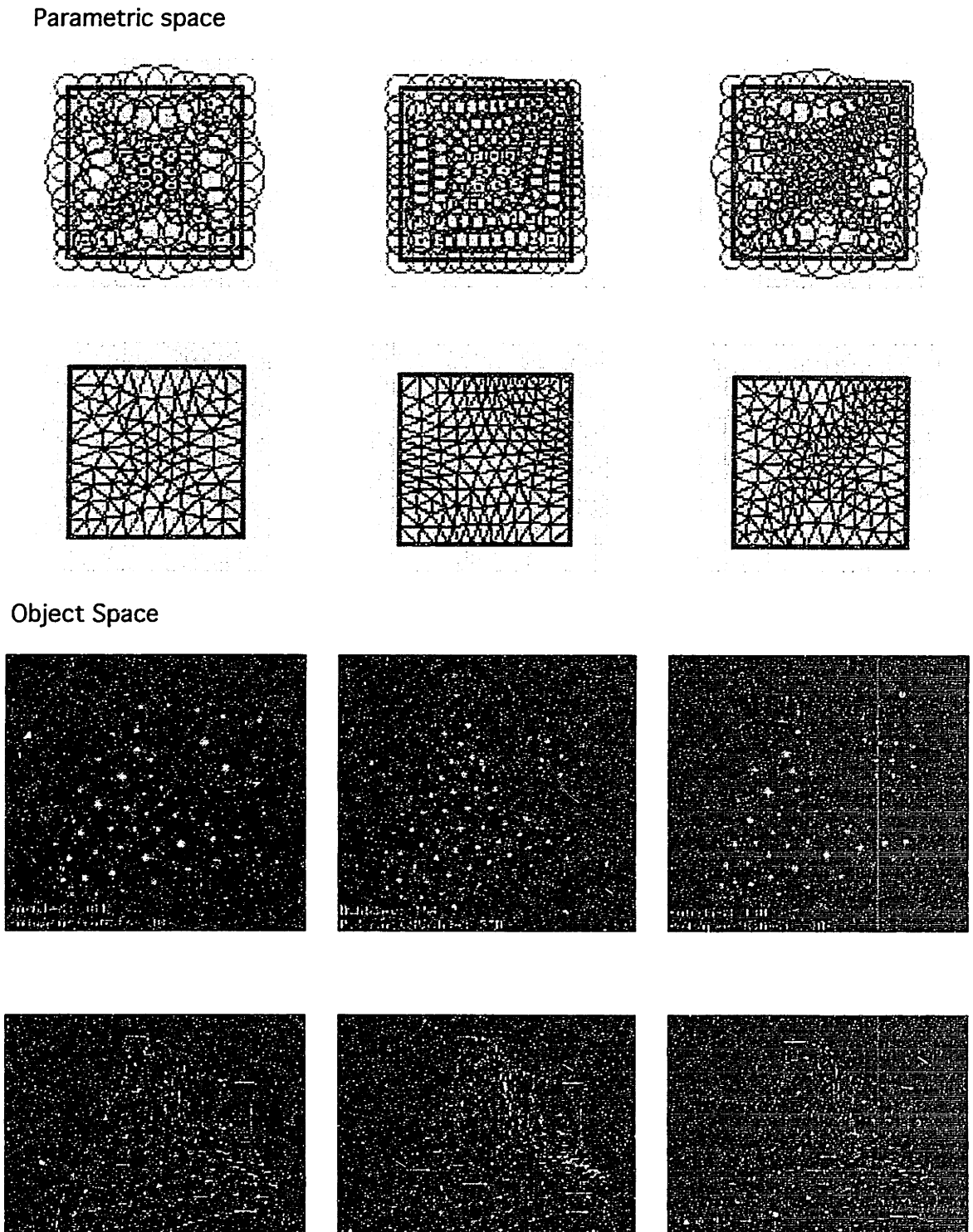


Figure 5.10 Various node spacing in surface triangulation.

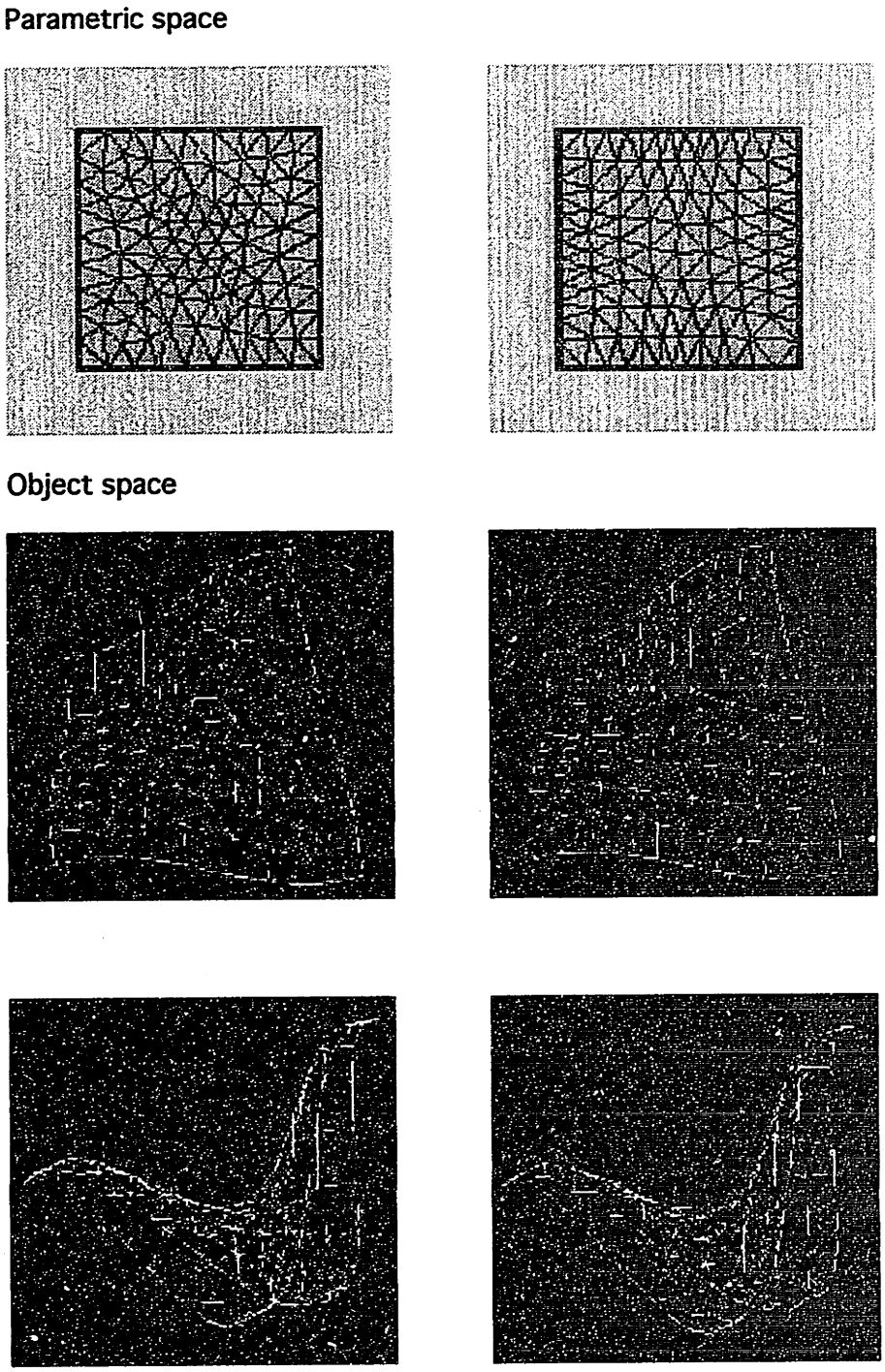
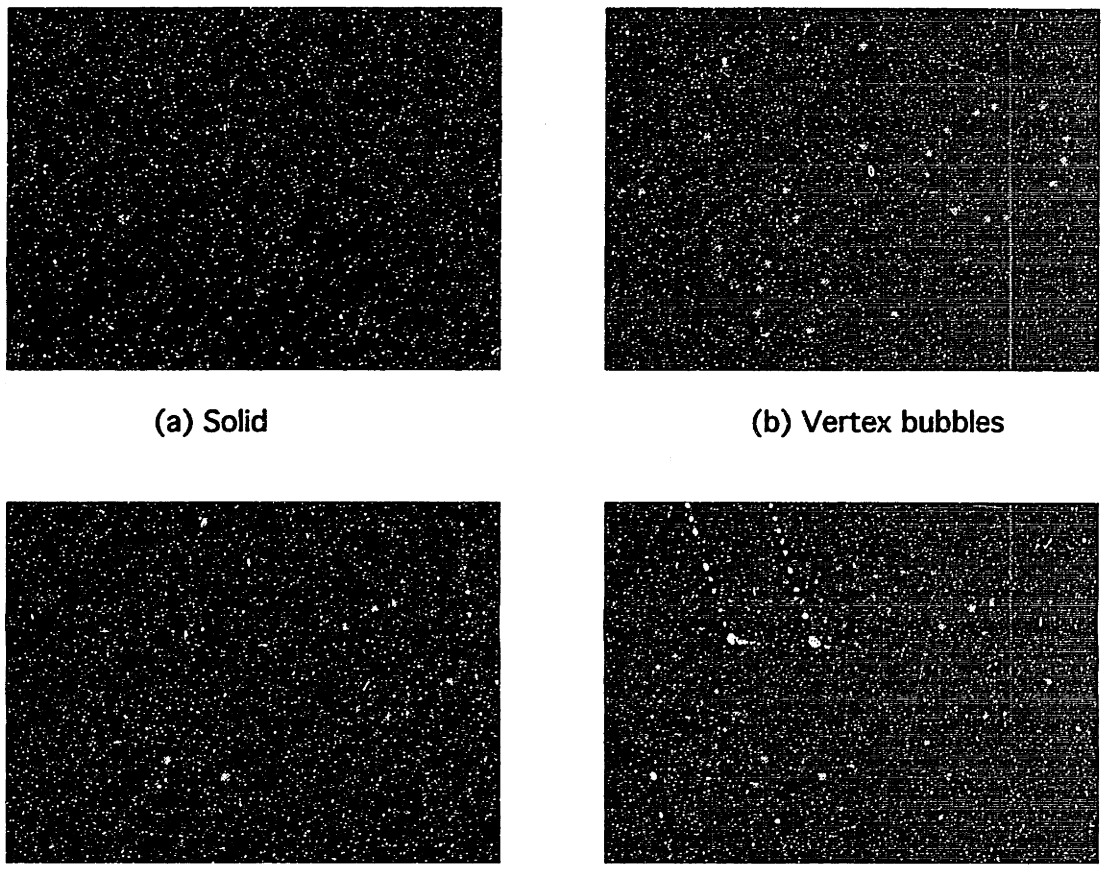


Figure 5.11 Reduced polygonal silhouette by curvature-based node spacing.

5.1.4 3D Meshing

Figure 5.12 portrays the three dimensional meshing procedure: (1) vertex bubble placement; (2) edge bubble placement; (3) face bubble placement; and (4) Delaunay triangulation. The bubble mesh generation method is incorporated with a commercial solid modeling system, DESIGNBASE [Chiyokura 90], to capture the topology and geometry information of a solid.

The 3D bubble mesh generation method can be implemented using any solid/non-manifold modeling system that provides the following geometric information, required in each meshing step.



(a) Solid (b) Vertex bubbles

(c) Edge bubbles (b) Face bubbles

Figure 5.12 3D meshing procedure.

Edge Meshing

- identify all curves in a model.
- obtain the positions of two end points of each curve.
- evaluate the position of a point on a curve, $C(s)=(x(s),y(s),z(s))$.
- evaluate the tangent vector at a point on a curve, $C^s=\frac{dC}{ds}=\left(\frac{dx}{ds},\frac{dy}{ds},\frac{dz}{ds}\right)$.

Face Meshing

- identify all faces in a model.
- identify all edges defining each face.
- evaluate the position of a point on a face, $S(u,v)=(x(u,v),y(u,v),z(u,v))$.
- evaluate the two tangent vectors at a point on a face,

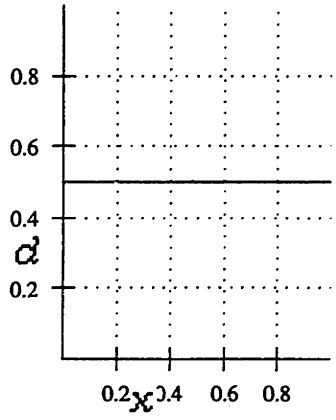
$$S^u=\frac{\partial S}{\partial u}=\left(\frac{\partial x}{\partial u},\frac{\partial y}{\partial u},\frac{\partial z}{\partial u}\right) \quad \text{and} \quad S^v=\frac{\partial S}{\partial v}=\left(\frac{\partial x}{\partial v},\frac{\partial y}{\partial v},\frac{\partial z}{\partial v}\right).$$

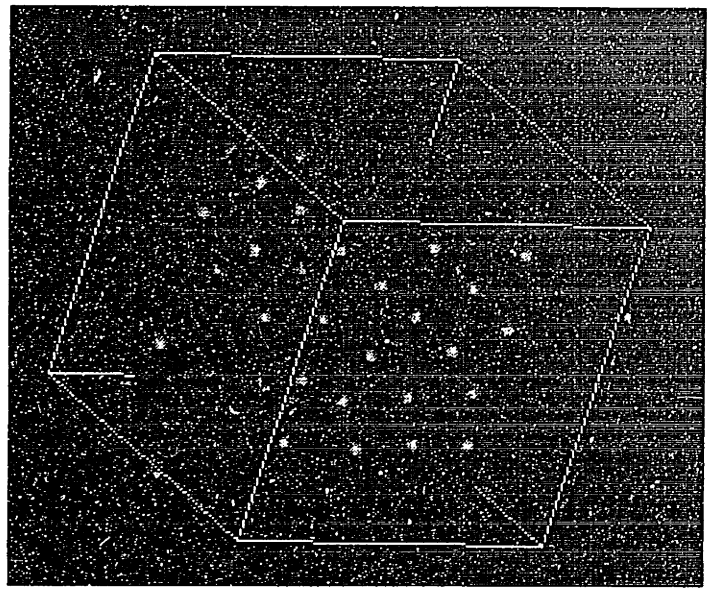
- check if a given point is inside the trimming boundary of a face.

Volume Meshing

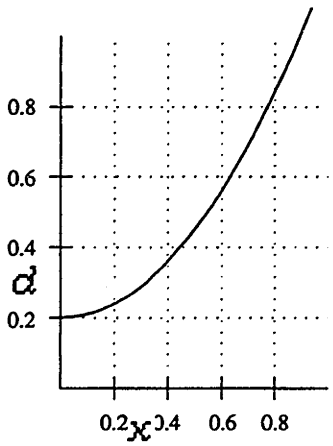
- identify all volumes in a model.
- identify all faces defining each volume.
- identify all edges defining each face.
- check if a point is inside the volume.

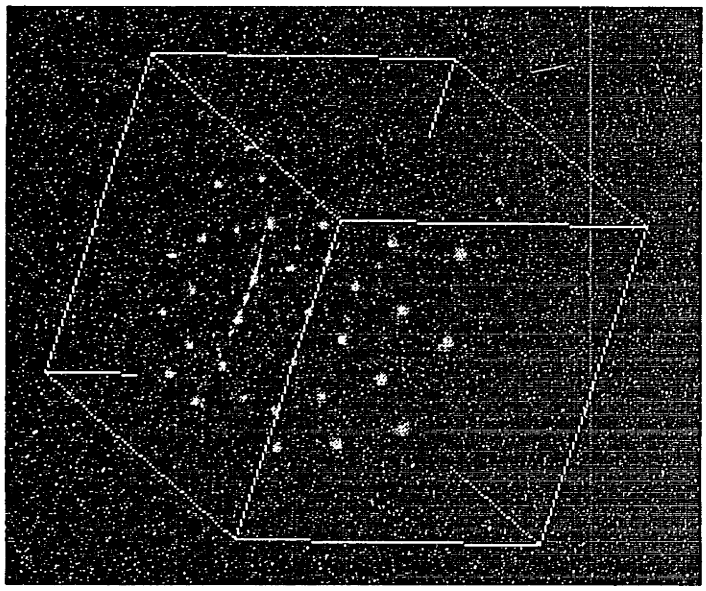
Figure 5.13 shows volume bubbles packed in a cube with different node spacings: uniform node spacing and monotonically increasing node spacing. In 3D, many regular ways exist to pack equal spheres, some examples of which are shown in Table 3.2 in Section 3.1.4. Among them, close cubic packing is the tightest one in 3D space, in which each sphere touches 12 neighbors. Configurations shown in Figure 5.13 are similar to this packing, yielding a well-shaped tetrahedral mesh.





(a) Uniform node spacing





(b) Monotonically increasing node spacing

Figure 5.13 Volume bubbles packed in a cube.

5.2 MESH RECONSTRUCTION FOR SURFACE DESIGN

This section describes how bubble mesh generation is utilized in an interactive surface design system developed at the Computer-Aided Design Laboratory at MIT. The system allows a user to shape a free-form surface interactively by trimming and deforming. Arbitrarily shaped bumps, grooves and holes that satisfy precise placement and slope requirements can be added by indicating the desired shape and location for the feature. FEM-based deformable surfaces are used as the underlying surface representation to guarantee that the shapes will be smooth while satisfying geometric constraints on positions and slopes.

In order for users to work directly with the design surface, independent of the underlying surface representation, the topology and geometry of the triangular surface patches must be automatically reconstructed in such a way that triangles are not ill-shaped, or too thin. The sizes of the triangles are controlled based on surface curvature so that the amount of shape variation over each triangle is basically uniform. This allows a good FEM-based deformable surface solution.

5.2.1 Feature-Based Deformable Surface Design

The surfaces of the stamped sheet-metal components used for automobiles can be categorized into two types: (a) *aesthetic surfaces* and (b) *functional surfaces*. Aesthetic surfaces, used to design automotive outer panels, are generally smooth and pleasing to the eye. Functional surfaces, used for structural members, typically contain many form-features such as channels, ribs and holes, since a primary goal is to increase rigidity while keeping weight low.

As pointed out in [Cavendish 91], a typical car contains well over 200 stamped sheet metal parts, and 95% of them can be characterized as functional surfaces. Although functional surface design is one of the most time-consuming processes in the entire vehicle design cycle, there has not been much research effort in the CAGD community directed towards improving the design environment for these surfaces.

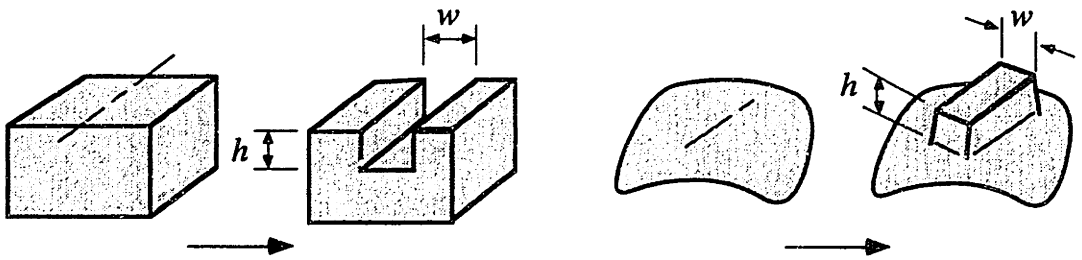
Our research goal of functional surface design is to develop an intuitive and easy-to-use design system for functional surfaces using finite-element based deformable surfaces. For that purpose, we utilized FEM-based deformable surface representation proposed in [Celniker 89, Celniker 90, Celniker 91]. These surfaces can be made to interpolate space

curves input by the user to describe feature boundaries, while also automatically providing necessary continuity between adjacent patches

In developing our surface design system, we made use of two major techniques, both of which already have been employed successfully in drafting CAD (2D) and solid CAD (3D): (a) *design-by-feature* [Dixon 88]; and (b) *dimension-driven geometry* [Kimura 87, Light 82, Suzuki 90]. A popular way to combine these two concepts in a 3D solid modeling system is to allow users first to define a base geometry, and then to add features one by one using dimension-driven geometry. This technique is implemented in some advanced solid modeling systems [Shimizu 91, Kawabe 88, Shimada 89].

For example, suppose that a groove is to be added to a base solid, such as a cube as shown in Figure 5.14 (a). The user inputs only the dimensions of the groove, i.e., width, depth, and the location of the groove on the cube. All of the necessary changes to the geometry and topology data are then managed by the system without user intervention; the updated model has 16 vertices, 24 edges, and 10 faces, while the original cube had only 8 vertices, 12 edges, and 6 faces. The lesson is that the design environment can be much easier to use and more intuitive when the underlying geometry and topology data are hidden and handled by the system.

We want to apply the same concepts to a system for adding functional features to free-form surfaces. Thus, if a user specifies the location and dimensions of a form-feature such as the channel shown in Figure 5.14 (b), the system automatically calculates the required geometric constraints and topology to incorporate the channel into the original base surface. Automatic "blending" between overlapping shape features and the base surface is achieved by computing the simplified physics of a membrane, thereby avoiding a great deal of manual geometry definition.



(a) Solid modeling (b) Surface modeling

Figure 5.14 Feature-based/dimension-driven solid modeling and surface modeling.

In a conventional spline-based system, addition of the new geometry typically requires user intervention with the underlying surface representation, by, for example (a) adjusting appropriate control point positions, or (b) specifying spatial curves and blend surfaces. Although a portion of a surface can be deformed using these methods, it is difficult to apply precise geometric constraints on a boundary of deformation, such as positional and slope requirements, constraints required in the design of a functional surface like an automobile inner panel.

On our system, the surface is represented parametrically by a mapping that translates points within a planar, bounded region into a C^1 continuous 3D surface. The planar parametric region is divided into a set of triangles that form a mesh. The 3D shape of each triangle is fully determined by 12 degrees of freedom -- position and derivative values associated with element edges and corners.

When a new feature is added, we must recreate the parametric mesh such that edges of the triangular elements match with the boundaries for the new feature. This mesh reconstruction allows a straightforward application of the geometric constraints necessary to impose the shape of the new feature on our original surface.

In this mesh reconstruction, performed each time a feature is added, we have the following requirements: (a) meshing must be performed at interactive speeds; (b) meshing must be fully automatic; (c) triangles cannot be thin or badly-distorted (since the shape equations are solved using a FEM-based method); and (d) the triangle sizes must be controlled based on the curvature distribution of the base surface. To solve this problem, the physically-based triangulation algorithm proposed in this thesis was employed.

Section 5.5.2 introduces previous work on surface sculpting. Section 5.5.3 gives an overview of the FEM-based deformable surface and its application to functional surface design. Examples of the design process are shown in section 5.5.4.

5.2.2 Previous Work on Surface Sculpting

Many surface sculpting techniques have recently been proposed in the computer graphics community. None, however, fully satisfy our need for multi-feature functional surface design. [Forsey 88] presents a surface representation scheme using hierarchical B-spline refinement. This allows a user to manipulate a surface region directly, but refinement regions are restricted to isoparametric locations, and arbitrary shape deformations cannot be achieved. [Sederberg 86] introduces the free-form deformation technique (FFD), which allows modeling while hiding the surface representation via a

FFD lattice. [Coquillart 90] extends FFD by using a non-parallelepipedical 3D lattice to allow arbitrarily shaped deformations. These FFD-based methods are powerful and useful for aesthetic surfaces, but lattice-creation, though an improvement over pulling control points, can be tedious. [Hsu 92] presents a direct manipulation technique, but the method is not suitable for defining surfaces with strict geometric constraints such as exact position and slope constraints at deforming boundary curves on a surface.

[Cavendish 91] provides a good definition of feature-based functional surfaces, presents a practical surface design approach, and shows design examples of reasonably complex functional surfaces. In the method, the transition region between the base surface and a feature's primary surface are blended using a pre-defined mathematical function. This technique uses an explicit surface representation and a feature-definition technique that is primarily suited for surfaces that are not highly curved.

5.2.3 Feature-Based Sculpting of Deformable Surfaces

In this section, the underlying deformable surface scheme is reviewed. The process of adding features is then described in order to clarify the need for mesh reconstruction and to point out various goals for the meshing system.

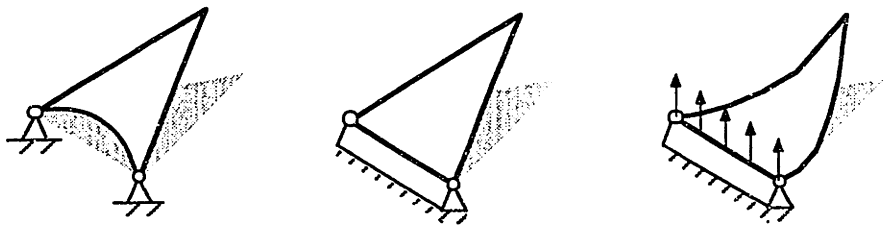
Terzopoulos et.al. [Terzopoulos 87] introduced the use of deformable curves and surfaces for computer animation and for the extraction of shape features from video images. Deformable surfaces seek shapes that minimize an internal energy subject to geometric constraints and applied loads. Their use in the design process is motivated in part by the fact that deformable surfaces can relegate fairness preservation to the surface solution, freeing the user to concentrate on shape definition.

Celniker and Gossard [Celniker 90, Celniker 91] developed a free-form deformable surface design system based on a finite-element formulation. Our feature-based surface design system continues to work with this formulation, in which surface shape is modeled after the physical behavior of two very different media: beams and soap films. Beams deform gracefully by distributing curvature along their length. Soap films, on the other hand, naturally form surfaces of minimal surface area. We construct our energy functional as a weighted sum of both stretching and bending terms, which is then minimized iteratively within the finite element representation. This model choice produces surfaces that naturally seek fair shapes.

We mesh our parametric domain with triangular patches, since triangular patches can cover arbitrarily-shaped surface domains. Each triangle has 12 degrees of freedom, which control its mapping into 3D. By setting these dofs appropriately, various boundary

conditions can be enforced along triangle edges, for example: *pinned*, *hinged*, and *fixed* edges, as shown in Figure 5.15.

A pinned edge allows any point or set of points in the surface to be set to locations in 3D space; the remainder of the shape is solved for by a minimization of the surface energy function. The position of a hinged edge is specified but the surface normal along the edge is free to vary; by matching two hinged edges, we can insert a tangent discontinuity in the interior of a surface. At fixed edges, all dofs are specified, and the edge location and the surface normals along the edge are fully controlled.



(a) pinned (b) hinged (c) fixed

Figure 5.15 Geometric constraints for a FEM-based deformable surface.

Figure 5.16 shows an example of a surface deformed with a combination of these geometric constraints. Depicted in figure 5.16 (a) is the surface before deformation. In (b) and (c), we move the center point of the pentagon while applying pinned and fixed constraints consecutively along the outer edges of the surface. Note that when the outer edges are fixed, the position and surface normals along the outer edges remain fixed at their original position regardless of the amount that the center point is moved. Note also that the deformable surface solution yields a smoothly blended surface regardless of the applied deformations.

In order to create an intuitive and easy-to-use surface sculpting environment based on the deformable surface representation, two basic feature-adding operations are implemented: *deforming* and *trimming*. A deforming operation locally molds, stretches or squeezes a base surface to create channels and bumps. A trimming operation adds holes by removing arbitrary-shaped portions of the surface.

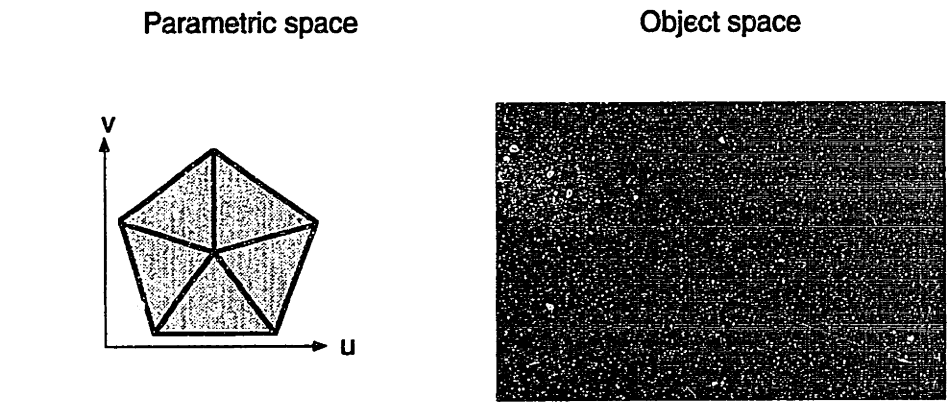
As summarized in Figure 5.17, the information a user has to input to perform these operations include: (a) characteristic dimensions, such as the height and the width of a

channel; (b) feature location, such as the starting point and the end point of a channel; and, if necessary, (c) the feature shape, such as the boundary shape of a hole.

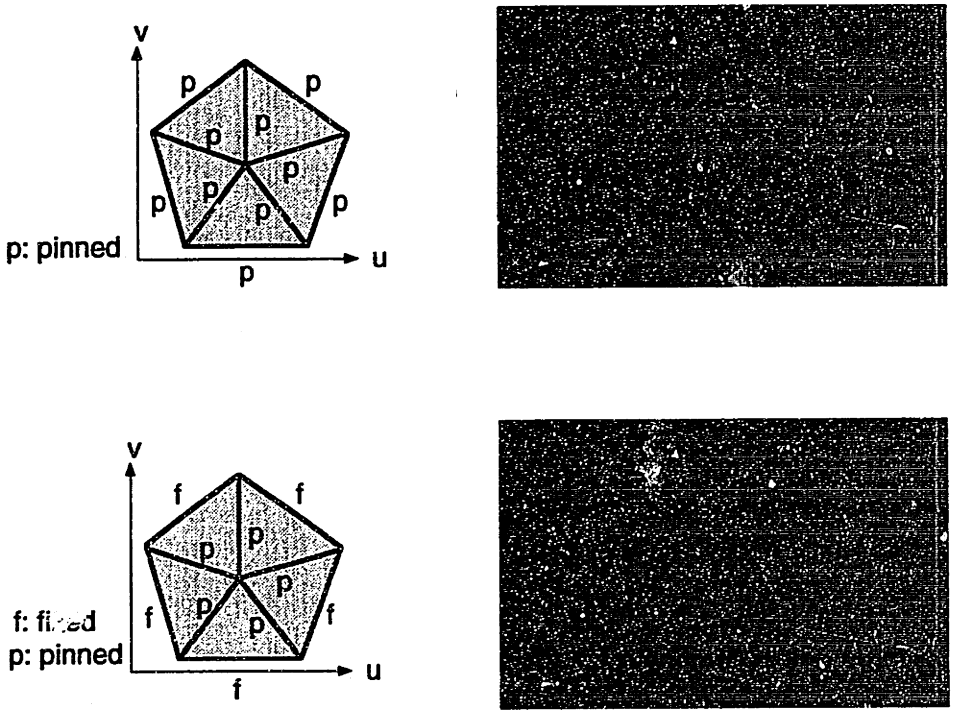
In this process of interactive surface design, the underlying triangular mesh has to be reconstructed every time a new feature is added. After remeshing, all boundaries for the new feature will be represented by one or more patch edges, allowing us to manipulate directly triangle dofs in order to apply necessary position and slope constraints along feature boundaries.

Let us now review the nomenclature which will be used to refer to the various operations used in functional surface design. Functional surface design is performed by interactively adding form-features on a *base surface*. The base surface $S(u,v) = (x(u,v), y(u,v), z(u,v))$ is defined over a bounded, planar region called the *parametric space*, which is transformed into a 3D region called *object space*. The surface can be trimmed by removing portions of the parametric domain inside *trimming curves*, denoted as $C(s) = (x(s), y(s), z(s))$. When a *trimming operation* is performed, it removes a sub-region of the surface, and creates a closed loop consisting of the new trimming curves in parametric space.

In a *deforming operation*, a deformed sub-region of the surface is represented by two loops, the inner and the outer. Curves of the loops are called *deformation-boundary curves*, also denoted as $C(s) = (x(s), y(s), z(s))$. The region inside the inner loop is called the *offset region*, a region to be offset to a constant height or depth. The region outside the outer loop remains unchanged, so that the position and slope along the outer loop match the base surface. The region between two loops is called the *transition region*, and its shape is determined by solving the deformable surface solution to blend the base surface with the offset region.



(a) Base surface before deformation



(b) Deformed surface under geometric constraints

Figure 5.16 Example of deformable surface solution with geometric constraints of fixed edges and pinned edges.

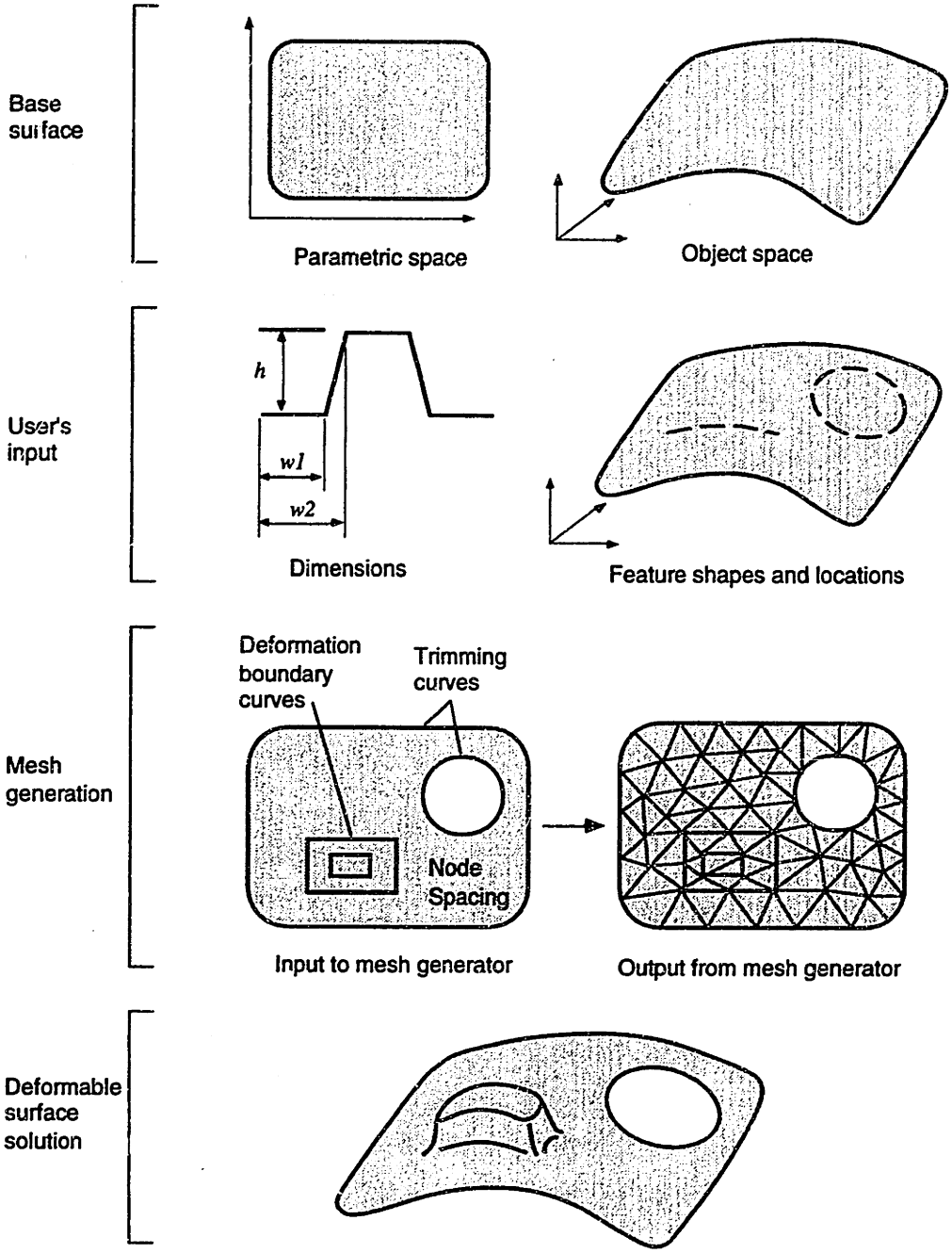


Figure 5.17 Procedure of feature-based sculpting of deformable surfaces.

5.2.4 Mesh Reconstruction in Sculpting Deformable Surfaces

Examples of reconstructed meshes are shown in Figure 5.18. Mesh reconstruction based on the physically-based triangulation occurs quickly enough for real-time use. Computation times are highly dependent on the CPU, the graphic details shown, the initial bubble distribution, and the number of bubbles. To provide some sort of estimate: the system of 80-100 node points, or bubbles, as shown in Figure 5.18 will take about 5 seconds to settle satisfactorily on a SGI 4D/GT70.

Figures 5.19a and 5.19b show results of our sculpting procedure applied to the creation of an automotive inner body panel used to support a car door. The left-hand pictures illustrate the parametric space of the surface. The right-hand pictures depict the surface presented to the user. As can be seen by the mapping of features from the parametric domain to object space, the parametric domain is highly distorted because the surface geometry was specified more precisely in critical regions such as the highly curved region at the bottom, and none of the original NURBS surfaces were reparameterized before the feature addition process.

Figure 5.19a(a) shows the result of fitting the outer body panel, a NURBS surface, using our deformable surfaces. Figure 5.19a(b) shows the surface after two simple overlapping beams (grooves of fixed cross-section) have been added. Figure 5.20a(a) shows the surface with four overlapping beams and one hole. Figure 5.20a(b) shows the same surface from another view to provide a clearer look at the meshing results. All meshes displayed in these examples were created through the automatic mesh reconstruction technique presented in Chapters 3 and 4.

Note that the nodes on the boundaries of the holes and beams were constrained while all interior nodes were free to move about until arriving at a satisfactory settling state. This is because we want mesh lines to coincide with feature boundaries. Bubbles on the boundary of the hole shown in Figure 5.20 were specified to be very dense for reasons unrelated to meshing quality.

When a user is actually working with the system, the mesh lines are not be shown--they have no relationship to the design process. The mesh lines are shown here merely to display the effectiveness of the bubble meshing procedure.

Creation of the entire panel shown in Figure 5.19b took less than four minutes. The settling times after each feature addition in the automotive example (200-400 bubbles) were always less than thirty seconds on an HP 9000 Series.

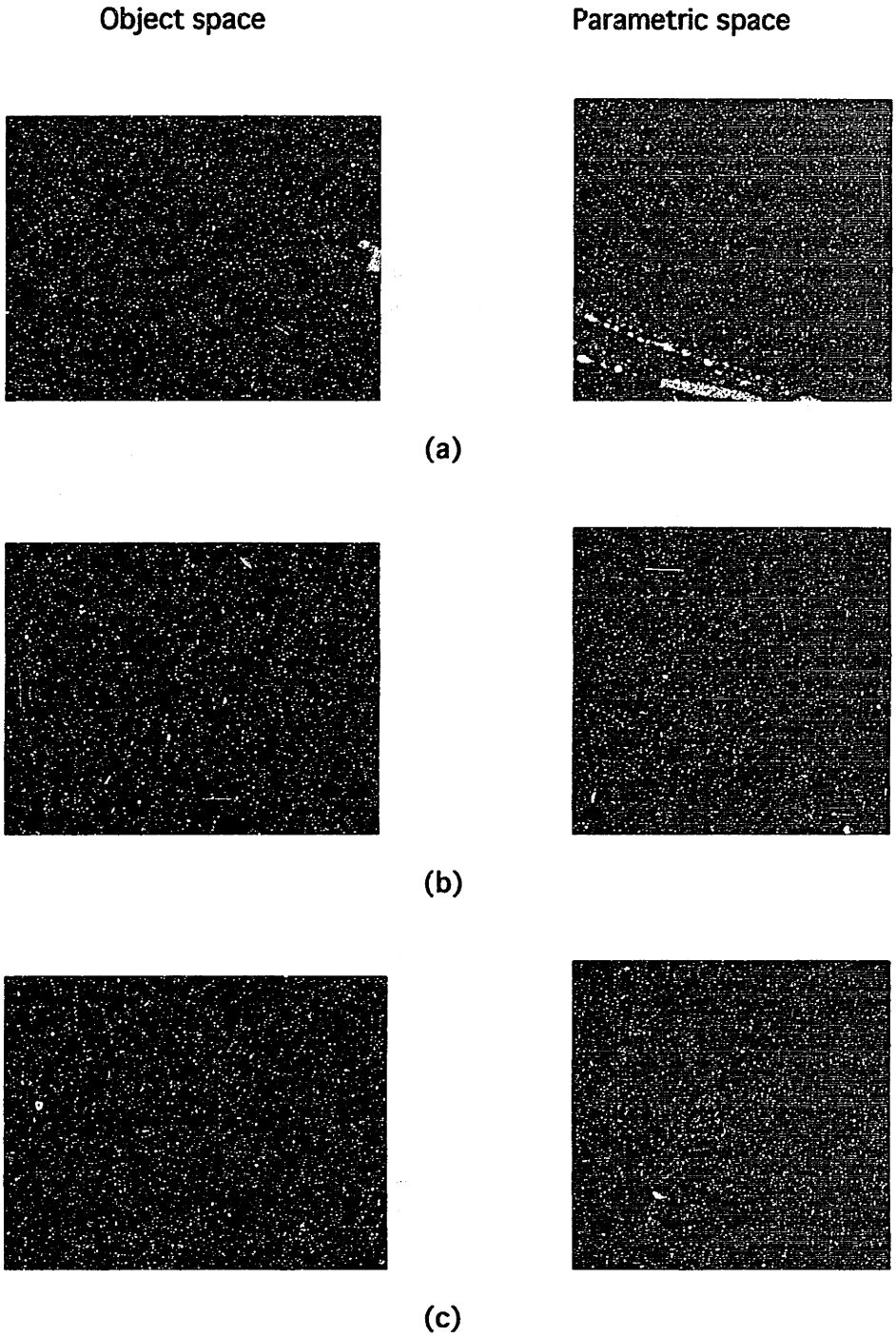


Figure 5.18 An example of the feature-based surface design procedure.

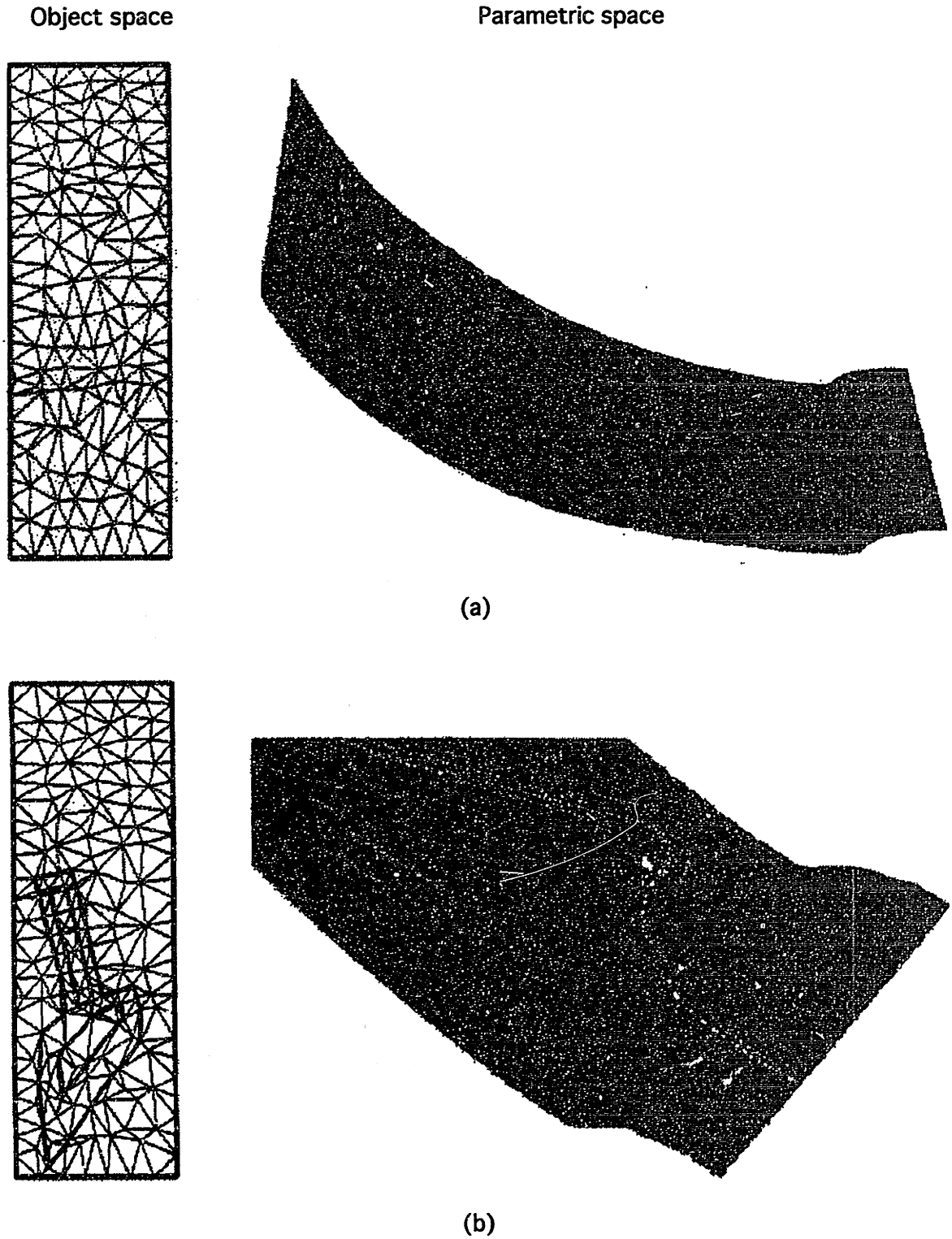
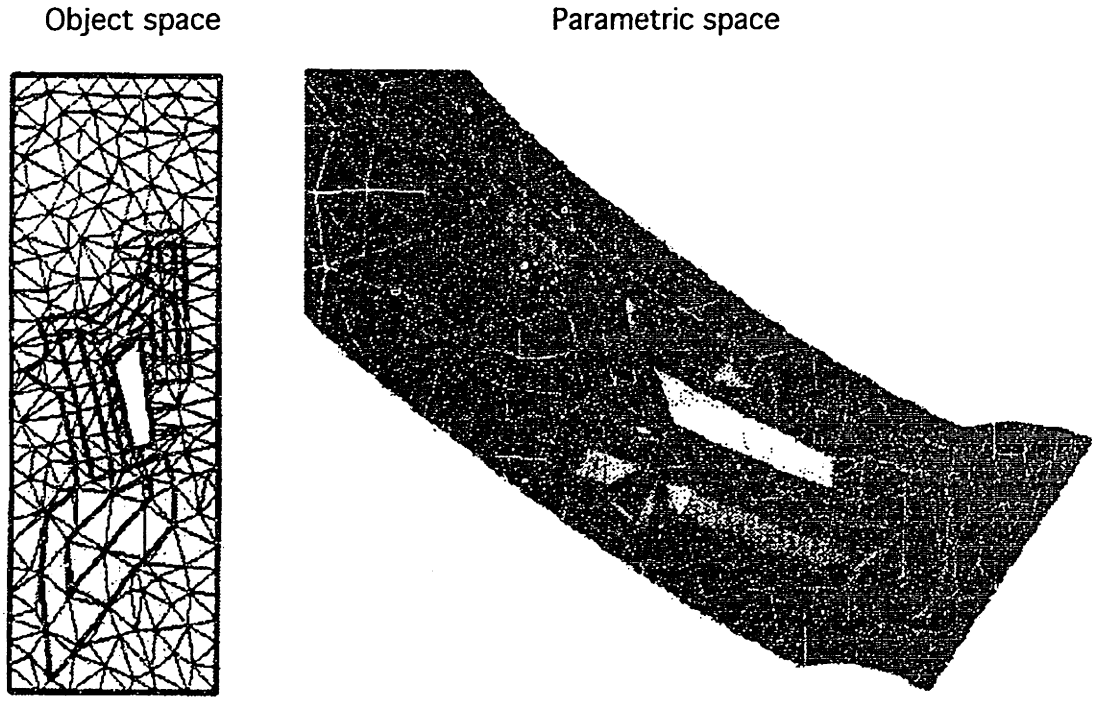


Figure 5.19a Functional surface design example 1 -- the meshed parametric domain (left) and the corresponding automotive inner body panel (right). The original NURBS surface was meshed and fit with FEM triangular patches (a), then a groove was added to the surface (b).



(a)

(b)

Figure 5.19b Functional surface design example 2 -- the meshed parametric domain (left) and the corresponding automotive inner body panel (right). Four beams and one hole have now been added to the original surface shown in 5.19a(a). The overlapping beams were blended automatically using the deformable surface technique. Figure (b) shows another view of the same surface.

6

Conclusions

6.1 THESIS SUMMARY

A new computational method for physically-based mesh generation was developed. In order to make the method application independent, only the geometry and the desired node spacing, i.e. the desired length of a triangle or tetrahedron edge, are assumed as input. The general meshing problem is highly demanding, with many requirements to be satisfied: (a) it has to handle various domains: 1D, 2D, surface, 3D, and hybrid in a consistent manner; (b) node spacing must be continuously controlled according to a user-given function; (c) badly-distorted, or thin triangles must be avoided as much as possible; (d) boundary nodes must be placed exactly on the boundary; (e) the algorithm must be computationally efficient; (f) it has to have efficient remeshing capability; and (g) it should be fully-automated.

Six categories of methods have been proposed previously: (a) node placement and connection, (b) coarse domain decomposition, (c) mesh template mapping, (d) element-level decomposition, (e) grid-based spatial subdivision, and (f) faceting parametric surfaces in parametric space. None of these methods, or combination of them, fully satisfies all of the meshing requirements listed above. The major limitations in previous methods are: (a) inconsistent applicability to 2D and 3D domains, (b) poor node spacing control, (c) ill-shaped elements, and (d) lack of effective adaptive remeshing capability. The goal of this thesis was to develop a new computational method to overcome these limitations.

The novelty of the proposed bubble method is that the close packing of bubbles mimics a pattern of Voronoi polygons, corresponding to well-shaped Delaunay triangles. In finding a configuration of closely packed 2D and 3D bubbles, there are two major problems to be solved: (a) where to place bubbles for regular element shapes, (b) how many bubbles to inject to fill a region.

The proposed solution to the first problem is the dynamic simulation with attracting/repelling interbubble forces, much like intermolecular van der Waals forces, a mass and a viscous damping effect. Starting with an initial bubble configuration generated by spatial subdivision, the equation of motion is integrated numerically until the system reaches an equilibrium. In the course of simulation, the size of a bubble is adjusted based on a user-given node spacing function.

For the second problem of finding the right number of bubbles, an adaptive bubble population control mechanism was proposed. With this mechanism, excess bubbles with high overlapping ratios are erased and new bubbles are added around open bubbles.

6.2 CONTRIBUTION

The major contribution of the thesis is a novel physically-based meshing algorithm that creates a quality triangular mesh. Through an actual implementation of the method, it was shown that the approach can overcome most of the serious weaknesses of previous approaches.

Consistent applicability to 1D/2D/surface/3D/hybrid domains

The method is completely consistent for all types of input geometry. Unlike most previously proposed methods, 3D domains and hybrid domains are triangulated using exactly the same procedure as for 1D and 2D domains. Bubbles are always filled in the order of dimensions, i.e.: (1) bubble placement on vertices, (2) bubble placement on edges, (3) bubble placement on faces, and (4) bubble placement in volumes. The same equations of motion, numerical solution methods, adaptive bubble population control, etc. are utilized in 1D, 2D, surface, 3D, and hybrid domains consistently.

Precise node spacing control

Unlike most previous methods, in which node spacing is changed only discretely or specified only on the boundary, the bubble method provides continuous node spacing control over a whole 2D or 3D region. A complicated mathematical function, such as curvature-based node spacing or view-dependent node spacing, can be specified. Especially for 3D problems, such precise node spacing control has never been realized by previously proposed methods.

Well-shaped elements

The process of bubble packing eliminates badly-distorted elements. This is because van der Waals-like interbubble forces keep all bubbles located a proper distance from their neighbors. Thin elements, which are created when two nodes are located too closely together, are thus avoided.

Adaptive remeshing capability

The bubble system is able to change its configuration dynamically when geometry and/or desired node spacing requirements are changed even slightly. In some applications such as large deformation FEM/BEM, a mesh must be reconstructed iteratively, because both boundary geometry and the desired node spacing predicted by analysis error change, causing the original mesh to be too distorted for correct analysis in the next step.

The bubble method is especially advantageous in situations where continuous remeshing is required; unlike other methods, the bubble method can utilize the mesh from the previous step as an initial condition for its dynamic simulation. If the change in geometry and node spacing is slight and incremental, the system of bubbles can reach a new equilibrium very quickly.

Intuitive and easy to implement

The idea of bubble meshing is quite intuitive and its computational procedure is easy to implement.

6.3 CURRENT LIMITATIONS

• Triangular elements

The bubble method proposed in the thesis has been developed for a quality triangular mesh, and it can not generate a quadrilateral mesh directly. Some conversion methods for going from a triangular to a quadrilateral mesh are available as discussed in Section 2.2.2.

• High computational cost for a quality mesh

Like all other physically-based approaches, the second stage of the bubble method, the dynamic simulation, is computationally costly due to the pairwise interbubble force interaction. For a large system of bubbles, an appropriate method of subdividing the system into sub-systems, such as bucketing or kD-trees, must be incorporated into the process of interbubble force calculation for decreased computational complexity.

6.4 FUTURE DIRECTIONS

The proposed concept of a physically-based approach to mesh generation can be extended in the future, for example, in the following directions:

• Packing squares and cubes for a quadrilateral mesh generation.

For direct generation of a quadrilateral mesh, other type of elements such as squares and cubes can be utilized. Packing such geometric units in a 2D/3D region would require a more sophisticated physical model and a more efficient numerical solution scheme.

• Meshing a domain in higher dimensions via hypersphere packing.

The bubble approach presented in the thesis can be directly extended to mesh generation in higher dimensions. After a distribution of nodes is obtained, they can be connected for a complete mesh topology by an n-dimensional Delaunay triangulation algorithm published earlier.

References

[Armstrong 91] Armstrong, C.G., *et al.*, "Automatic Generation of Well Structured Meshs Using Medial Axis and Surface Subdivision," *ASME Advances in Design Automation*, pp.139-146, 1991.

[Aurenhammer 91] Aurenhammer, F., "Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure," *ACM Computing Surveys*, vol. 23, no. 3, 1991.

[Bathe 82] Bathe, K.-J., *Finite Element Procedures in Engineering Analysis.* 1982, Prentice-Hall .

[Blacker 88] Blacker, T.D., *et al.*, "Automated Quadrilateral Mesh Generation: A Knowledge System Approach," *ASME Paper No. 88-WA/CIE-4*, vol. , 1988.

[Blacker 91] Blacker, T.D. and M.B. Stephenson, "Paving: A New Approach To Automated Quadrilateral Mesh Generation," *International Journal for Numerical Methods in Engineering*, vol. 32, pp.811-847, 1991.

[Blum 73] Blum, H., "Biological Shape and Visual Science (Part I)," *Journal of Theoretical Biology*, vol. 38, pp.205-287, 1973.

[Blum 78] Blum, H. and R.N. Nagel, "Shape Description Using Weighted Symmetric Axis Features," *Pattern Recognition*, vol. 10, no. 3, pp.167-180, 1978.

[Bowyer 81] Bowyer, A., "Computing Dirichlet Tessellations," *Comp. J.*, vol. 24, pp.162-166, 1981.

[Bowyer 81] Bowyer, A., "Computing Dirichlet Tessellations," *The Computer Journal*, vol. 9, pp.219, 1981.

[Bubska 76] Bubska, I. and A.K. Aziz, "On the Angle Criterion in the Finite Element Method," *SIAM Journal on Numerical Analysis*, vol. 13, pp.214-226, 1976.

[Buratynski 90] Buratynski, E.K., "A Fully Automatic Three-Dimensional Mesh Generator for Complex Geometries," *International Journal for Numerical Methods in Engineering*, vol. 30, pp.931-952, 1990.

[Burgess 87] Burgess, J., M. Marten, and R. Taylor, *Under the Microscope*. 1987, Cambridge University Press .

[Bykat 76] Bykat, A., "Automatic Generation of Triangular Grid: I - Subdivision of a General Polygon into Convex Subregions. II - Triangulation of Convex Polygons," *International Journal for Numerical Methods in Engineering*, vol. 10, pp.1329-1342, 1976.

[Bykat 83] Bykat, A., "Design of a Recursive, Shape Controlling Mesh Generator," *International Journal for Numerical Methods in Engineering*, vol. 19, pp.1375-1390, 1983.

[Cavendish 74] Cavendish, J.C., "Automatic Triangulation of Arbitrary Planar Domains for the Finite Element Method," *International Journal for Numerical Methods in Engineering*, vol. 8, pp.679-696, 1974.

[Cavendish 77] Cavendish, J.C., W.J. Gordon, and C.A. Hall, "Substructured Macro Elements Based on Locally Blended Interpolation," *International Journal for Numerical Methods in Engineering*, vol. 29, pp.1405-1421, 1977.

[Cavendish 84] Cavendish, J.C., "A New Class of Transitional Blended Finite Elements for Analysis of Solid Structures," *International Journal for Numerical Methods in Engineering*, vol. 20, pp.241-253, 1984.

[Cavendish 85] Cavendish, J.C., D.A. Field, and W.H. Frey, "An Approach to Automatic Three-Dimensional Finite Element Mesh Generation," *International Journal for Numerical Methods in Engineering*, vol. 21, pp.329-347, 1985.

[Cavendish 91] Cavendish, J.C. and W.H. Frey, "Feature-Based Design and Finite Element Mesh Generation for Functional Surfaces," *Advances in Engineering Software*, vol. 13, no. 5/6, pp.226-237, 1991.

[Celniker 89] Celniker, G. and D.C. Gossard, "Energy-based models for free-form surface shape design," *ASME Design Automation Conference*, 1989.

[Celniker 90] Celniker, G., "ShapeWright: Finite Element Based Free-Form Shape Design," *Ph.D. thesis Massachusetts Institute of Technology*, 1990.

[Celniker 91] Celniker, G. and D.C. Gossard, "Deformable Curve and Surface Finite Elements for Free-Form Shape Design," *Computer Graphics (SIGGRAPH '91)*, vol. 25, no. 4, pp.257-266, 1991.

[Chae 88] Chae, S.W., "On the Automatic Generation of Near-Optimal Meshs for Three-Dimensional Linear Elastic Finite Element Analysis," *Ph.D. thesis Massachusetts Institute of Technology, Cambridge, MA*, 1988.

[Cohen 80] Cohen, H.D., "A Method for the Automatic Generation of Triangular Elements on a Surface," *International Journal for Numerical Methods in Engineering*, vol. 15, pp.470-476, 1980.

[Cook 74] Cook, W.A., "Body Oriented (Natural) Coordinates for Generating Three Dimensional Meshes," *International Journal for Numerical Methods in Engineering*, vol. 8, pp.27-43, 1974.

[Coquillart 90] Coquillart, S., "Extended Free-Form Deformation: A Sculpting Tool for 3D Geometric Modeling," *Computer Graphics (SIGGRAPH '90)*, vol. 24, no. 4, pp.187-196, 1990.

[Crawford 87] Crawford, R.H., W.N. Waggenspack, and D.C. Anderson, "Composite Mappings for Planar Mesh Generation," *International Journal for Numerical Methods in Engineering*, vol. 24, pp.2241-2252, 1987.

[de Cougny 90] de Cougny, H.L., M.S. Shephard, and M.K. Georges, "Explicit Node Point Smoothing within the Finite Octree Mesh Generator," *Scientific Computation Research Center, Rensselaer Polytechnic Institute, SCOREC Report # 10-1990, 1990*

[Deljouie-Rakhshandeh 90] Deljouie-Rakhshandeh, K., "An Approach to the Generation of Triangular Grids Possessing Few Obtuse Triangles," *International Journal for Numerical Methods in Engineering*, vol. 29, pp.1299-1321, 1990.

[Dixon 88] Dixon, J.R., "Research in Designing with Features," *IFIP Workshop on Intelligent CAD*, 1988.

[Dolenc 90] Dolenc, A. and I. Makela, "Optimized Triangulation of Parametric Surfaces," in (ed.), *Mathematics of Surfaces IV*, 1990.

[Edelsbrunner 87] Edelsbrunner, H., *Algorithms in Combinatorial Geometry.* 1987, USA: Springer-Verlag .

[Fang 92] Fang, L. and D.C. Gossard, "Reconstruction of Smooth Parametric Surfaces from Unorganized Data Point," *SPIE Vol. 1830 Curves and Surfaces in Computer Vision and Graphics III*, 1992.

[FangT-P 92] Fang, T.-P. and L.A. Piegl, "Algorithm for Delaunay Triangulation and Convex-Hull Computation Using a Sparse Matrix," *Computer-Aided Design*, vol. 24, no. 8, 1992.

[Field 86] Field, D.A., "Implementing Watson's Algorithm in Three Dimensions," *Second Annual ACM Symposium on Computational Geometry*, pp.246-259, 1986.

[Field 88] Field, D.A., "Laplacian Smoothing and Delaunay Triangulations," *Commun. Appl. Numer. Methods*, vol. 4, pp.709-712, 1988.

[Field 91a] Field, D.A., "A Generic Delaunay Triangulation Algorithm for Finite Element Meshes," *Advances in Engineering Software*, vol. 13, no. 5/6, pp.263-272, 1991.

[Field 91b] Field, D.A. and W.D. Smith, "Graded Tetrahedral Finite Element Meshes," *International Journal for Numerical Methods in Engineering*, vol. 31, pp.413-425, 1991.

[Foley 90] Foley, J.D., *et al.*, *Computer Graphics: Principles and Practice*. 1990, Addison-Wesley . 734-741.

[Forsey 88] Forsey, D.R. and R.H. Bartels, "Hierarchical B-Spline Refinement," *Computer Graphics (SIGGRAPH '88)*, vol. 22, no. 4, pp.205-212, 1988.

[Frederick 70] Frederick, C.O., Y.C. Wong, and F.W. Edge, "Two-Dimensional Automatic Mesh Generation for Structural Analysis," *International Journal for Numerical Methods in Engineering*, vol. 2, pp.133-144, 1970.

[Frey 87] Frey, W.H., "Selective Refinement: A New Strategy for Automatic Node Placement in Graded Triangular Meshes," *International Journal for Numerical Methods in Engineering*, vol. 24, pp.2183-2200, 1987.

[Frey 91] Frey, W.H. and D.A. Field, "Mesh Relaxation: A New Technique for Improving Triangulations," *International Journal for Numerical Methods in Engineering*, vol. 31, pp.1121-1133, 1991.

[Fukuda 72] Fukuda, J. and J. Suhara, "Automatic Mesh Generation for Finite Element Analysis," in J.J. Oden (ed.), *Advances in Computational Methods in Structural Mechanics and Design*, UAH Press, Huntsville, Alabama, 1972.

[George 91] George, P.L., F. Hecht, and M.G. Vallet, "Creation of Internal Points in Voronoi's Type Method. Control Adaptation," *Advances in Engineering Software*, vol. 13, no. 5/6, pp.303-311, 1991.

[Gordon 71] Gordon, W.J., "Blending-Function Methods of Bivariate and Multivariate Interpolation and Approximation," *SIAM Journal Numerical Analysis*, vol. 8, no. 1, pp.158-177, 1971.

[Gordon 73] Gordon, W.J., "Construction of Curvilinear Coordinate Systems and Applications to Mesh Generation," *International Journal for Numerical Methods in Engineering*, vol. 7, pp.461-477, 1973.

[Green 78] Green, P.J. and R. Sibson, "Computing Dirichlet Tessellations in the Plane," *Computer Journal*, vol. 21, no. 2, pp.168-173, 1978.

[Gursoy 89] Gursoy, H.N., "Shape Interogation By Medial Axis Transform for Automated Analysis," *Ph.D.thesis, Massachusetts Institute of Technology, Cambridge, MA, U.S.A*, 1989.

[Gursoy 91] Gursoy, H.N., "Automated Interrogation and Adaptive Subdivision of Shape Using Medial Axis Transform," *Advances in Engieering Software*, vol. 13, no. 5/6, pp.287-302, 1991.

[Gursoy 92a] Gursoy, H.N. and N.M. Patrikalakis, "An Automatic Coarse and Fine Surface Mesh Generation Scheme Based on Medial Axis Transform: Part I Algorithms," *Engineering with Computers*, vol. 8, pp.121-137, 1992.

[Gursoy 92b] Gursoy, H.N. and N.M. Patrikalakis, "An Automatic Coarse and Fine Surface Mesh Generation Scheme Based on Medial Axis Transform: Part II Implementation," *Engineering with Computers*, vol. 8, pp.179-196, 1992.

[Harber 81] Haber, R.B., *et al.*, "A Generalized Two-Dimensional, Graphical Finite Element Preprocessor Utilizing Discrete Transfinite Mappings," *International Journal for Numerical Methods in Engineering*, vol. 17, pp.1015-1044, 1981.

[Harber 82] Haber, R.B. and J.F. Abel, "Discrete Transfinite Mappings for the Description and Meshing of Three-Dimensional Surfaces Using Interactive Computer Graphics," *International Journal for Numerical Methods in Engineering*, vol. 18, pp.41-66, 1982.

[Heighway 82] Heighway, E.A. and C.S. Biddlecombe, "Two-Dimensional Automatic Triangular Mesh Generation for The Finite Element Electromagnetics Package PE2D," *IEEE Trans. on Mag.*, vol. MAG-18, no. 2, pp.594-598, 1982.

[Heighway 83] Heighway, E.A., "A Mesh Generator for Automatically Subdividing Irregular Polygons into Quadrilaterals," *IEEE Trans. Magn.*, vol. Mag-19, pp.2535-2538, 1983.

[Hinton 91] Hinton, E., N.V.R. Rao, and M. Ozakca, "Mesh Generation with Adaptive Finite Element Analysis," *Advances in Engineering Analysis*, vol. 13, no. 5/6, pp.238-262, 1991.

[Ho-Le 88] Ho-Le, K., "Finite Element Mesh Generation Methods: A Review and Classification," *Computer-Aided Design*, vol. 20, no. 1, pp.27-38, 1988.

[Imafuku 80] Imafuku, I., Y. Kodera, and M. Sayawaki, "A Generalized Automatic Mesh Generation Scheme for Finite Element Method," *International Journal for Numerical Methods in Engineering*, vol. 15, pp.713-731, 1980.

[Jain 86] Jain, A., "Modern Methods for Automatic FE Mesh Generation," in K. Baldwin (ed.), *Modern Methods for Automating Finite Element Mesh Generation*, The American Society of Civil Engineers, pp.19-28, 1986.

[Joe 86a] Joe, B. and R.B. Simpson, "Triangular Meshes for Regions of Complicated Shape," *International Journal for Numerical Methods in Engineering*, vol. 23, pp.751-778, 1986.

[Joe 86b] Joe, B., "Delaunay Triangular Meshes in Convex Polygons," *SIAM Journal on Scientific and Statistical Computing*, vol. 7, no. 2, pp.514-539, 1986.

[Joe 91a] Joe, B., "GEOMPACK - A Software Package for the Generation of Meshes Using Geometric Algorithms," *Advances in Engineering Software*, vol. 13, no. 5/6, pp.325-331, 1991.

[Joe 91b] Joe, B., "Delaunay Versus Max-Min Solid Angle Triangulations for Three-Dimensional Mesh Generation," *International Journal for Numerical Methods in Engineering*, vol. 31, pp.987-997, 1991.

[Joe 91c] Joe, B., "Construction of Three-Dimensional Delaunay Triangulations Using Local Transformations," *Computer Aided Geometric Design*, vol. 8, pp.123-142, 1991.

[Johnston 91] Johnston, B.P., J.M. Sullivan, and A. Kwasnik, "Automatic Conversion of Triangular Finite Element Meshes to Quadrilateral Elements," *International Journal for Numerical Methods in Engineering*, vol. 31, pp.67-84, 1991.

[Kappraff 90] Kappraff, J., *Connections: The Geometric Bridge Between Art and Science*. 1990, McGraw-Hill .

[Kawabe 88] Kawabe, S., K. Shimada, and H. Masuda, "A Framework for 3D Modeling: Constraint-Based Description and Non-Manifold Geometric Modeling," in (ed.), *Organization of Engineering Knowledge for Product Modelling in Computer Integrated Manufacturing*, Elsevier, pp.325-354, 1988.

[Kikuchi 86] Kikuchi, N., "Adaptive Grid-Based Methods for Finite Element Analysis," *Compu. Methods in Applied Mech. Eng.*, vol. 55, pp.129-160, 1986.

[Kimura 87] Kimura, F., *et al.*, "Variational Geometry Based on Logical Constraints and Its Applications to Product Modelling," *CIRP*, vol. 37, no. 1, pp.65-68, 1987.

[Lawson 86] Lawson, C.Y., "Properties of n-Dimensional Triangulations," *Computer Aided Geometric Design*, vol. 3, pp.231-246, 1986.

[LeeDT 84] Lee, D.T. and F.P. Preparata, "Computational Geometry - A Survey," *IEEE Transaction on Computer*, vol. C-33, no. 12, pp.1072-1101, 1984.

[LeeDT 86] Lee, D.T., "Generalized Delaunay Triangulation for Planar Graphs," *Discrete and Computational Geometry*, vol. 1, pp.201-217, 1986.

[LeeNS 93] Lee, N.S., "Adaptive Procedures for Nonlinear Elastic and Elasto-Plastic Finite Element Analysis," *Ph.D. Thesis thesis Massachusetts Institute of Technology*, 1993.

[LeeYT 83] Lee, Y.T., "Automatic Finite Element Mesh Generation Based on Constructive Solid Geometry," *Ph.D. thesis Univ. of Leeds, Leeds, England*, 1983.

[LeeYT 84] Lee, Y.T., "Automatic Finite-Element Mesh Generation," *ACM Transactions on Graphics*, vol. 3, no. 4, pp.287-311, 1984.

[Lewis 77] Lewis, B.A. and J.S. Robinson, "Triangulation of Planar Regions with Applications," *Computer Journal*, vol. 21, no. 4, pp.324-332, 1977.

[Light 82] Light, R.A. and D.C. Gossard, "Modification of Geometric Models through Variational Geometry," *Computer-Aided Design*, vol. 14, no. 4, pp.209-214, 1982.

[Lindholm 83] Lindholm, D.A., "Automatic Triangular Mesh Generation on Surfaces of Polyhedora," *IEEE Trans. Mag.*, vol. MAG-19, no. 6, pp.1539-1542, 1983.

[Lo 85] Lo, S.H., "A New Mesh Generation Scheme for Arbitrary Planar Domains," *International Journal for Numerical Methods in Engineering*, vol. 21, pp.1403-1426, 1985.

[Lo 89] Lo, S.H., "Delaunay Triangulation of Non-Convex Planar Domains," *International Journal for Numerical Methods in Engineering*, vol. 28, pp.2695-2707, 1989.

[Lohner 88] Lohner, R. and P. Parikh, "Generation of Three-Dimensional Unstructured Grids by The Advancing-Front Method," *AIAA-88-0515*, vol. , 1988.

[Mascardini 83] Mascardini, A.O., B.A. Lewis, and M. Cross, "AGTHOM - Automatic Generation of Triangular and Higher Order Meshes," *International Journal for Numerical Methods in Engineering*, vol. 19, pp.1331-1353, 1983.

[Masuda 89] Masuda, H., *et al.*, "A Mathematical Theory and Applications of Non-Manifold Geometric Modeling," in (ed.), *Advanced Geometric Modelling for Engineering Applications*, North-Holland, pp.89-103, 1989.

[Masuda 93] Masuda, H., "Topological Operators and Boolean Operations for Complex-Based Nonmanifold Geometric Models," *Computer-Aided Design*, vol. 25, no. 2, pp.119-129, 1993.

[Meshkat 91] Meshkat, S. and H. Li, "Three-Dimensional Automatic Unstructured Grid Generation Based on Delaunay Tetrahedrization," *Third International Conference on Numerical Grid Generation*, pp.841-851, 1991.

[Nguyen-Van-Phai 82] Nguyen-Van-Phai, "Automatic Mesh Generation with Tetrahedron Elements," *International Journal for Numerical Methods in Engineering*, vol. 18, pp.273-289, 1982.

[Ogata 70] Ogata, K., *Modern Control Engineering*. 1970, Prentice-Hall .

[Patrikalakis 90] Patrikalakis, N.M. and H.N. Gursoy, "Shape Interrogation by Medial Axis Transform," *Advances in Design Automation*, pp.77-88, 1990.

[Patrikalakis 93] Patrikalakis, N.M., "Surface-To-Surface Intersections," *IEEE Computer Graphics and Applications*, vol. 13, no. 1, pp.89-95, 1993.

[Peraire 88] Peraire, J., *et al.*, "Finite Element Euler Computations in Three Dimensions," *International Journal for Numerical Methods in Engineering*, vol. 26, pp.2135-2159, 1988.

[Perucchio 89] Perucchio, R., M. Saxena, and A. Kela, "Automatic Mesh Generation from Solid Models Based on Recursive Spatial Decompositions," *International Journal for Numerical Methods in Engineering*, vol. 28, pp.2469-2502, 1989.

[Preparata 85] Preparata, F. and M. Shamos, *Computational Geometry - An Introduction*. 1985, USA: Springer-Verlag .

[Press 88] Press, W.H., *et al., Numerical Recipes in C: the Art of Scientific Computing*. 1988, Cambridge University Press .

[Rao 90] Rao, S.S., *Mechanical Vibrations*. 1990, Addison-Wesley .

[Rock 91] Rock, S.J., "Solid Freeform Fabrication and CAD System Interfacing," *Ph.D. Thesis thesis Rensselaer Polytechnic Institute*, 1991.

[Rockwood 89] Rockwood, A., K. Heaton, and T. Davis, "Real-Time Rendering of Trimmed Surfaces," *Computer Graphics (SIGGRAPH '89)*, vol. 21, no. 4, pp.107-116, 1989.

[Ruppert 92] Ruppert, J., "Results on Triangulation and High Quality Mesh Generation," *Ph.D. thesis, University of California at Berkeley, CA, U.S.A.*, 1992.

[Ruppert 93] Ruppert, J., "A New and Simple Algorithm for Quality 2-Dimensional Mesh Generation," *The Fourth Annual Symposium on Discrete Alrorithms*, pp.83-92, 1993.

[Sadek 80] Sadek, E., "A Scheme for the Automatic Generation of Triangular Finite Elements," *International Journal for Numerical Methods in Engineering*, vol. 15, pp.1813-1822, 1980.

[Sadek 80] Sadek, E.A., "A Scheme for the Automatic Generation of Triangular Finite Elements," *International Journal for Numerical Methods in Engineering*, vol. 15, pp.1813-1822, 1980.

[Sapidis 89] Sapidis, N. and R. Perucchio, "Advanced Techniques for Automatic Finite Element Meshing form Solid Models," *Computer-Aided Design*, vol. 21, no. 4, pp.248-253, 1989.

[Sapidis 91a] Sapidis, N. and R. Perucchio, "Delaunay Triangulation of Arbitrarily Shaped Planar Domains," *Computer Aided Geometric Design*, vol. 8, no. 6, pp.421-437, 1991.

[Sapidis 91b] Sapidis, N. and R. Perucchio, "Domain Delaunay Tetrahedrizaion of Solid Models," *Intern. Journal of Computational Geometry & Applications*, vol. 1, no. 3, pp.299-325, 1991.

[Sapidis 92] Sapidis, N. and R. Perucchio, "Solid/Solid Classification Operations for Recursive Spatial Decomposition and Domain Triangulation of Solid Models," *Computer-Aided Design*, vol. 24, no. 10, pp.517-528, 1992.

[Schoofs 79] Schoofs, A.J.G., L.H.T.M. Van Beukering, and M.L.C. Sluiter, "A General Purpose Two-Dimensional Mesh Generator," *Advances in Engineering Software*, vol. 1, no. 3, pp.131-136, 1979.

[Schroeder 88] Schroeder, W.J., "Geometry-Based Fully Automatic Mesh Generation and the Delaunay Triangulation," *International Journal for Numerical Methods in Engineering*, vol. 26, pp.2503-2515, 1988.

[Schroeder 90] Schroeder, W.J., "A Combined Octree/Delaunay Method for Fully Automatic 3-D Mesh Generation," *International Journal for Numerical Methods in Engineering*, vol. 29, pp.37-55, 1990.

[Schroeder 92] Schroeder, W.J., J.A. Zarge, and W.E. Lorensen, "Decimation of Triangle Meshes," *SIGGRAPH '92*, vol. 25, no. 4, pp.247-250, 1992.

[Sederberg 86] Sederberg, T.W. and S.R. Parry, "Free-Form Deformation of Solid Geometric Models," *Computer Graphics (SIGGRAPH '86)*, vol. 20, no. 4, pp.151-160, 1986.

[Sheng 92] Sheng, X. and B.E. Hirsch, "Triangulation of Trimmed Surfaces in Parametric Space," *Computer-Aided Design*, vol. 24, no. 8, pp.437-444, 1992.

[Shephard 86] Shephard, M.S., M.A. Yerry, and P.L. Baehmann, "Automatic Mesh Generation Allowing for Efficient A Priori and A posteriori Mesh Refinement," *Computer Methods in Applied Mechanics and Engineering*, vol. 55, pp.161-180, 1986.

[Shephard 88] Shephard, M.S., *et al.*, "Trends in Automatic Three-Dimensional Mesh Generation," *Computers & Structures*, vol. 30, no. 1/2, pp.421-429, 1988.

[Shephard 91] Shephard, M.S. and M.K. Georges, "Automatic Three-Dimensional Mesh Generation by the Finite Octree Technique," *International Journal for Numerical Methods in Engineering*, vol. 32, pp.709-749, 1991.

[Shephard 91] Yuen, M.M.F., S.T. Tan, and K.Y. Hung, "A Hierarchical Approach to Automatic Finite Element Mesh Generation," *International Journal for Numerical Methods in Engineering*, vol. 32, pp.501-525, 1991.

[Shimada 89] Shimada, K., *et al.*, "Constraint-Based Object Description for Product Modelling," *The 3rd International Conference on Computer Applications in Product and Engineering (CAPE'89)*, pp.95-106, 1989.

[Shimada 91] Shimada, K., "Grid-Based Two-Dimensional Mesh Generation," *Massachusetts Institute of Technology, CADLAB Internal Document, 1991*

[Shimada 92] Shimada, K. and D.C. Gossard, "Computational Method for Physically-Based FE Mesh Generation," *IFIP TC5/WG 5.3 Eight International PROLAMAT Conference*, pp.411-420, 1992.

[Shimada 93] Shimada, K., "Bubble Mesh Generation: Automated High-Quality Triangulation of Surfaces and Volumes," *Ph.D. thesis Massachusetts Institute of Technology, Cambridge, MA*, 1993.

[Shimizu 91] Shimizu, S., K. Inoue, and M. Numao, "An ATMS-Based Geometric Constraint Solver for 3D CAD," *The 3rd International Conference on Tools for Artificial Intelligence (TAI'91)*, pp.282-290, 1991.

[Sibson 78] Sibson, R., "Locally Equiangular Triangulations," *Computer Journal*, vol. 21, pp.243-245, 1978.

[Sloan 84] Sloan, S.W. and G.T. Houlsby, "An Implementation of Watson's Algorithm for Computing Two-Dimensional Delaunay Triangulations," *Advances in Engineering Software*, vol. 6, pp.192, 1984.

[Sloan 87] Sloan, S.W., "A Fast Algorithm for Constructing Delaunay Triangulations in the Plane," *Advances in Engineering Software*, vol. 9, no. 1, pp.34-55, 1987.

[Sluiter 82] Sluiter, M.L.C. and D.L. Hansen, "A General Purpose Automatic Mesh Generator for Shell and Solid Finite Elements," in L.E. Hulberd (ed.), *Computers in Engineering*, New York, 1982.

[Srinivasan 90] Srinivasan, V., L.R. Nackman, and J.-M. Tang, "Automatic Mesh Generation Using the Symmetric Axis Transformation of Polygonal Domains," *IBM Research Report, RC16132*, vol. , 1990.

[Stephenson 89] Stephenson, M.B. and T.D. Blacker, "Using Conjoint Meshing Primitives to Generate Quadrilateral and Hexahedral Elements in Irregular Regions," *ASME Computations in Engineering Conference*, 1989.

[Stewart 92] Stewart, I., *The Problems of Mathematics*. 1992, Oxford University Press .

[Suzuki 90] Suzuki, H., H. Ando, and F. Kimura, "Synthesizing Product Shapes with Geometric Design Constraints and Reasoning," in (ed.), *Internal CAD*, Elsevier Science Publishers B. V., pp.408-423, 1990.

[Tam 91] Tam, T.K.H. and C.G. Armstrong, "2D Finite Element Mesh Generation by Medial Axis Subdivision," *Advances in Engieering Software*, vol. 13, no. 5/6, pp.313-324, 1991.

[Terzopoulos 87] Terzopoulos, D., *et al.*, "Elastic Deformable Models," *Computer Graphics (SHIGGRAPH '88)*, vol. 21, no. 4, 1987.

[Thacker 80a] Thacker, W.C., "A Brief Review of Techniques for Generating Irregular Computational Grids," *International Journal for Numerical Methods in Engineering*, vol. 15, pp.1335-1341, 1980.

[Thacker 80b] Thacker, W.C., A. Gonzales, and G.E. Putland, "A Method for Automating the Construction of Irregular Computational Grids for Storm Surge Forecast Models," *Journal of Computational Physics*, vol. 37, pp.371-387, 1980.

[Tracy 77] Tracy, F.T., "Graphical Pre- and Post-Processor For Two Dimensional Finite Element Method Programs," *SIGGRAPH '77*, pp.8-12, 1977.

[Turk 91] Turk, G., "Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion," *SIGGRAPH '91*, vol. 25, no. 4, pp.289-298, 1991.

[Turk 92] Turk, G., "Re-Tiling Polygonal Surfaces," *SIGGRAPH '92*, vol. 26, no. 2, pp.55-64, 1992.

[Van-Phai 92] Van-Phai, N., "Automatic Mesh Generation with Tetrahedron Elements," *International Journal for Numerical Methods in Engineering*, vol. 18, pp.273-280, 1992.

[Watson 81] Watson, D.F., "Computing the n-Dimensional Delaunay Tesselation with Applications to Voronoi Polytopes," *Computer Journal*, vol. 24, pp.167-172, 1981.

[Wellford 88] Wellford, L.C. and M.R. Gorman, "A Finite Element Transitional Mesh Generation Procedure Using Sweeping Functions," *International Journal for Numerical Methods in Engineering*, vol. 26, pp.2623-2643, 1988.

[Williams 72] Williams, R., *The Geometrical Foundation of Natural Structure.* 1972, Dover Publications, Inc. .

[Woo 84] Woo, T.C. and T. Thomasma, "An Algorithm for Generating Solid Elements in Objects with Holes," *Computers and Structures*, vol. 18, no. 2, pp.333-342, 1984.

[Wordenweber 81] Wordenweber, B., "Automatic Mesh Generation of 2- and 3-Dimentional Curveilinear Manifolds," *Ph.D. thesis Cambridge University, UK*, 1981.

[Wordenweber 84] Wordenweber, B., "Finite Element Mesh Generation," *Computer-Aided Design*, vol. 16, no. 5, pp.285-291, 1984.

[Yerry 83] Yerry, M.A. and M.S. Shephard, "A Modified-Quadtree Approach to Finite Element Mesh Generation," *IEEE Computer Graphics & Application*, vol. 3, pp.39-46, 1983.

[Yerry 84] Yerry, M.A. and M.S. Shephard, "Automatic Three-Dimensional Mesh Generation by the Modified-Octree Technique," *International Journal for Numerical Methods in Engineering*, vol. 20, pp.1965-1990, 1984.

[Zhu 91] Zhu, J.Z., *et al.*, "A New Approach to the Development of Automatic Quadrilateral Mesh Generation," *International Journal for Numerical Methods in Engineering*, vol. 32, pp.849-866, 1991.

[Zienkiewicz 71] Zienkiewicz, O.C. and D.V. Phillips, "An Automatic Mesh Generation Scheme for Plane and Curved Surfaces by 'Isoparametric' co-ordinates," *International Journal for Numerical Methods in Engineering*, vol. 3, pp.519-528, 1971.

[Zienkiewicz 87] Zienkiewicz, O.C. and J.Z. Zhu, "A Simple Error Estimator and Adaptive Procedure for Practical Engineering Analysis," *International Journal for Numerical Methods in Engineering*, vol. 24, pp.337-357, 1987.