Bubble Mesh: Automated Triangular Meshing of Non-Manifold Geometry by Sphere Packing

Kenji Shimada † and David C. Gossard ‡

†IBM Research, Tokyo Research Laboratory
1623-14, Shimo-tsuruma, Yamato-shi, Kanagawa-ken 242, Japan
‡Department of Mechanical Engineering
Massachusetts Institute of Technology
77 Massachusetts Avenue, Cambridge, MA 02139, U.S.A.

Abstract

This paper presents a new computational method for fully automated triangular mesh generation, consistently applicable to wire-frame, surface, solid, and non-manifold geometries. The method, called *bubble meshing*, is based on the observation that a pattern of tightly packed spheres mimics a Voronoi diagram, from which a set of well-shaped Delaunay triangles and tetrahedra can be created by connecting the centers of the spheres. Given a domain geometry and a node-spacing function, spheres are packed on geometric entities, namely, vertices, edges, faces, and volumes, in ascending order of dimension. Once the domain is filled with spheres, mesh nodes are placed at the centers of these spheres and are then connected by constrained Delaunay triangulation and tetrahedrization. To obtain a closely packed configuration of spheres, the authors devised a technique for physically based mesh relaxation with adaptive population control. The process of mesh relaxation significantly reduces the number of ill-shaped triangles and tetrahedra.

1 Introduction

A great deal of design time in industry is devoted to analysis, especially when physical experiments are performed on real components. In order to reduce the whole product development time, it is therefore desirable to computerize analysis by using numerical methods such as the finite element method (FEM) and the boundary element method (BEM). Various kinds of commercial software based on these methods 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 that must satisfy a number of geometric and topological conditions dictated by the method [1]. The operation of transforming a geometric model, especially a 3D model, into a valid mesh is highly labor-intensive. Thus a fully automated mesh generation scheme is desirable.

Conversion of a CAD model into a mesh is performed in two steps, shown in Figure 1: (1) simplification of the geometry, and (2) discretization of the geometry into a mesh. In the first step, to reduce the computational time and storage space, a geometry is simplified by means of the following two procedures:

Dimensional thinning. Pipe-like or beam-like geometries are often modeled as one-dimensional curves, as shown in Figure 1(a). Shell-like geometries in which the thickness is small in comparison with the whole component size are approximated as two-dimensional shells, as shown in Figure 1(b). Many mechanical sheet-metal components and ship hulls have this type of geometry.

Insignificant feature removal. If a geometric feature, such as a hole, protrusion, or groove, is not meaningful for analysis, it is removed from the model, as shown in Figure 1(c). The criteria for determining which features can be removed are not straightforward. For example, a very small groove can cause a fatal stress concentration in structural analysis, but may negligible in heat transfer analysis.

After the original geometry has been simplified, it is usually represented as a wire-frame model, a surface model, or a solid model. In the most complicated case, however, the geometry is simplified to a non-manifold geometry, or a union of 1D, 2D, and 3D geometries, as shown in Figure 1 (d). Non-manifold situations also arise when multiple materials are used in a single component; in this case, the material boundaries must be represented as internal faces or edges.

The simplified geometry is then discretized into a

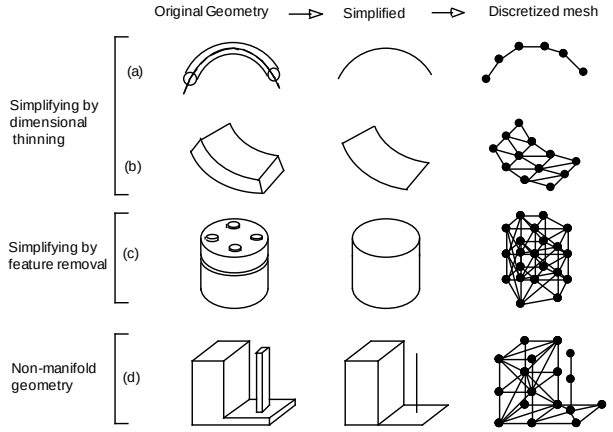


Figure 1: Conversion of a CAD model into a mesh

mesh. A curve is meshed into a series of one-dimensional beam elements with nodes that lie on the curve. A surface is subdivided into two-dimensional shell elements, or triangles. A volume is meshed into a collection of tetrahedral elements.

This paper presents a new triangular meshing procedure, called *bubble meshing*, that can handle various input geometries, including wire-frame, surface, solid, and non-manifold, in a consistent manner. In this method, node locations are obtained by closely packing spheres in the domain to be meshed and placing nodes at the centers of the packed spheres. Node connections are then decided for a complete mesh topology by using constrained Delaunay triangulation and tetrahedrization.

This new scheme of node placement contributes to a significant reduction in the number of ill-shaped elements produced in Delaunay triangulation and tetrahedrization.

2 Preliminaries

2.1 Problem Statement

As mentioned above, possible geometric inputs to the meshing procedure include wire-frame, surface, solid, and non-manifold geometries. Because the non-manifold model, by definition, can represent all types of geometry, we can simply say that the input to the mesher is a non-manifold model.

There are various definitions of non-manifold geometry [23, 15, 7]; we adopt here the definition proposed by Masuda, Shimada, Kawabe, and Numao [9, 10].

Non-manifold geometries, G, are *cell complexes* that are subsets of 3D Euclidean space. Cell complexes are mathematically defined as sets of *n-cells* that satisfy the three conditions below:

$$G = \bigcup_{\lambda \in \Lambda} e_\lambda \tag{1}$$

$$\langle e_\lambda \rangle - e_\lambda \subset \{e_\mu \mid dim(e_\mu) < dim(e_\lambda), \mu \in \Lambda\} \quad \lambda \in \Lambda \tag{2}$$

$$e_\lambda \cap e_\mu = \phi \qquad \lambda \neq \mu,\ \lambda \in \Lambda,\ \mu \in \Lambda \tag{3}$$

where G and e_λ represent a cell complex and an n-cell, respectively, and $dim(e_\lambda)$ and $\langle e_\lambda \rangle$ represent the dimension of e_λ and the closure of e_λ, respectively.

The first condition means that 3D cell complexes can be represented by a collection of 0-cells, 1-cells, 2-cells, and 3-cells. In a geometric modeling system, these cells correspond to topological entities, namely, vertices, edges, faces, and volumes, respectively. The second condition specifies that the boundary of each entity consists of lower-dimensional entities, making a cell complex always closed. The third condition prohibits the mutual intersection of topological entities.

According to this definition, an n-cell is a bounded subset of 3D Euclidean space that is homeomorphic to an n-dimensional open sphere.

As in most solid modeling systems with boundary representation, geometric information is stored under three categories of geometric entities: points, curves, and surfaces, which correspond to vertices, edges, and faces, respectively.

A point is a coordinate triple in object space, $(x, y, z) \in R^3$. A curve is defined over a bounded R^1 parametric space, which is then transformed into object space. In other words, a curve geometry is given as a mapping from parametric space to object space,

$$C(s) = (x(s), y(s), z(s)), \tag{4}$$

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 mapped into object space:

$$S(u, v) = (x(u, v), y(u, v), z(u, v)). \tag{5}$$

The surface can be trimmed by means of *trimming curves*.

The actual curve and surface representations can be of any form, as long as they are continuous and a derivative vector can be calculated everywhere on the curves and surfaces.

Also given to the mesher as input is a desired distribution of mesh element size. In this paper, the element size is also referred to as the *node spacing*, since the size of a mesh element is measured by the distance between two adjacent nodes. We denote a desired node spacing

over the domain as $d(x, y, z)$, given as a function of the node location in object space.

A mesh M is a set of *mesh elements*, defined as n-cells: (1) nodes, which are 0-cells in the non-manifold model; (2) line segments, which are 1-cells; (3) triangles, which are 2-cells; and (4) tetrahedra, which are 3-cells. Thus,

$$M = (M_0, M_1, M_2, M_3), \tag{6}$$

where M_0, M_1, M_2, and M_3 are sets of nodes, line segments, triangles, and tetrahedra, respectively. A mesh node is also referred to as the center of a bubble in this paper; we use the terms mesh node and bubble interchangeably in later sections.

In summary, the mesh generation problem we are interested in can be stated as follows:

Given:

- a non-manifold geometric domain, G
- a desired node spacing distribution, $d(x, y, z)$

Generate:

- a graded, well-shaped, compatible, triangular mesh, M, of hybrid dimension.

2.2 Previous Methods

There have been several reviews of mesh generation methods [21, 17, 8]. Ho-Le, in his comprehensive survey paper [8], gives a systematic classification based on the temporal order in which nodes and elements are created. The resultant classification is well-accepted and has been referred to by many other researchers. One problem, as he also acknowledges in the paper, is that placement of individual methods into categories is not easy because many proposed methods consist of several sub-processes representing different categories in the classification.

Sub-processes commonly used in existing meshing methods include node placement and connection; coarse domain decomposition; mesh template mapping; element-level domain decomposition; grid-based spatial subdivision; and faceting of parametric surfaces in parametric space. Typically, one complete meshing scheme is characterized by a combination of these sub-processes, performed sequentially or merged into a single process.

Node placement and connection, however, can serve by themselves as a complete meshing process. In this process, a mesh is constructed in two stages: (1) node placement, and (2) node connection. Meshing algorithms of this type have recently become popular on account of their conceptual simplicity and the availability of a robust mathematical algorithm for node connection, called Delaunay triangulation or tetrahedrization. The bubble method proposed in this paper also falls into the node placement and connection category.

2.3 Delaunay Triangulation

This section briefly reviews Delaunay triangulation/tetrahedrization and Voronoi tessellation, since they are closely related to the fundamentals of bubble meshing.

Efficient algorithms for Delaunay triangulation have been intensively reviewed and studied in various textbooks [13, 4] and papers [2].

Consider N distinct points $\mathbf{p}_i, 1 \leq i \leq N$, in the two-dimensional space R^2 or the three-dimensional space R^3, and define the sets $V_i, 1 \leq i \leq N$, as

$$V_i = \{\mathbf{x} \mid \| \mathbf{x} - \mathbf{p}_i \| < \| \mathbf{x} - \mathbf{p}_j \| \text{ for all } i \neq j\}, \tag{7}$$

where $\| \cdot \|$ denotes Euclidean distance in R^2 or R^3. The set V_i is considered to consist of Voronoi polygons in R^2 and Voronoi polyhedra in R^3.

The collection of Voronoi polygons or polyhedra is called a Voronoi diagram or Dirichlet tessellation. The boundaries of the Voronoi polygon or polyhedron are portions of the perpendicular bisectors of the lines joining point $\mathbf{p}_i$ to point $\mathbf{p}_j$, when V_i and V_j are contiguous.

A vertex of a Voronoi polygon is shared by two other neighboring polygons, and a vertex of a Voronoi polyhedra is shared by three other neighboring polyhedra. We can, therefore, construct a triangle by connecting three points, defined above, in three adjacent polygons, and a tetrahedron by connecting four points, defined above, in four adjacent polyhedra. The set of such triangles is called the Delaunay triangulation, and the set of such tetrahedra the Delaunay tetrahedrization. Another Delaunay criterion is that a circumscribing circle or sphere of a Delaunay triangle or tetrahedron does not contain any other points inside.

Delaunay triangulation is considered suitable for finite element analysis, because it maximizes the sum of the smallest angles of the triangles. It creates triangles as nearly equilateral as possible for the given set of points. The same property holds in three-dimensional Delaunay tetrahedrization; the triangular faces of the tetrahedra are as nearly equilateral as possible.

One problem to be noted here is that since the union of Delaunay triangles or tetrahedra is a superset of the domain, some extraneous triangles or tetrahedra must be deleted. It is also necessary to ensure that pairs of adjacent points on boundaries are connected. Triangulation or tetrahedrization with such constraints is sometimes called constrained Delaunay triangulation or tetrahedrization. Algorithms for this procedure have been proposed by several researchers, such as: Fang and Piegl [5]; Sapidis [16]; and Meshkat et. al. [11].

2.4 Bad Triangles and Tetrahedra

During node placement, an appropriate number of nodes must be inserted in a well-distributed configu-

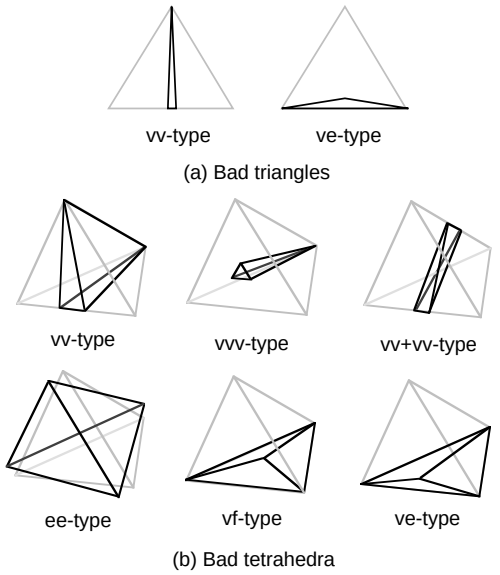


Figure 2: Ill-shaped triangles and tetrahedra

ration, so that no badly distorted or skinny triangles or tetrahedra are created. Delaunay tessellation does "optimize" the element shapes, but the actual quality of these shapes depends totally on the given configuration of nodes.

As Dey [3] and Meshkat [11] pointed out, bad triangles and tetrahedra are either thin (i.e. needle-like) or flat. Such ill-shaped elements must be avoided in analysis because they increase the analysis error and slow the solution convergence.

Starting from an equilateral triangle and a tetrahedron, one could create bad elements by moving some vertices, edges, and faces close to each other. As Figure 2 indicates, there are two types of bad triangle: (1) *vv-type*, created by moving two vertices close together, and (2) *ve-type*, created by moving a vertex and an edge close together. Bad tetrahedra have more variety: (1) *vv-type*, (2) *vvv-type*, (3) *vv+vv-type*, (4) *ee-type*, (5) *vf-type*, and (6) *ve-type*. Among these six types of ill-shaped tetrahedron, the vvv-type and the vv+vv-type are thin, or needle-like, while the others are flat.

Previous approaches resolve bad elements by post-processing, either by moving nodes or by adding and deleting nodes. In bubble meshing, without using such post-processing, we can greatly reduce the possibility of creating these bad elements by defining proximity-based internode forces and finding a force-balancing configuration; node configurations of bad elements in Figure 2 cause large overlaps and gaps between bubbles, and thus cannot be stable.

2.5 Mesh Quality Measurement

For quantifying mesh quality in triangulation and tetrahedrization, we define two kinds of irregularity: *topological mesh irregularity* and *geometric mesh irregularity*.

For topological mesh irregularity, we define the following measure, similar to that defined by Frey and Field [6]:

$$\varepsilon_t = \frac{1}{n}\sum_{i=0}^{n} |\delta_i - D|\,, \quad D = \begin{cases} 6 & \text{for triangles} \\ 12 & \text{for tetrahedra,} \end{cases} \tag{8}$$

where δ_i represents the *degree*, or the number of neighboring nodes, connected to the ith interior node, and n represents the total number of interior nodes in the domain. Thus, in general, as elements become more equilateral, the mesh irregularity approaches 0, but vanishes only when all the nodes have D neighbors, a rare situation. Otherwise, it has a positive value that designates how much the mesh differs from a perfectly regular triangular lattice.

For geometric mesh irregularity, we define the following measure, ε_g, which is the aspect ratio of an inscribed circle or sphere to a circumscribing circle or sphere:

$$\varepsilon_g = \frac{1}{m}\sum_{i=0}^{m}(A - \frac{r_i}{R_i}), \quad A = \begin{cases} 0.5 & \text{for triangles} \\ \sqrt{2/11} & \text{for tetrahedra,} \end{cases} \tag{9}$$

where m represents the number of elements, and r_i and R_i are the radii of inscribed and circumscribing circles or spheres, respectively. The ratio r_i/R_i is at maximum 0.5 for an equilateral triangle and $\sqrt{2/11}$ for an equilateral tetrahedron. The smaller the value of ε_g, the more regular the mesh.

3 Meshing via Bubble Packing

3.1 Method Overview

The bubble method can be summarized as a sequence of two steps: (1) pack spheres, or bubbles, closely in the domain, and (2) connect their centers by constrained Delaunay triangulation and tetrahedrization, which select the best topological connection for a set of nodes by avoiding small included angles while preserving the compatibility of the mesh with the domain. The novelty of the method is that the close packing of bubbles mimics a pattern of Voronoi tessellation, corresponding to well-shaped Delaunay triangles or tetrahedra [18, 19, 20]. Figure 3(a) depicts this relationship in two dimensions.

In packing bubbles, some gaps and overlaps are inevitable, so our aim is to minimize these gaps and overlaps as much as possible by injecting an appropriate number of bubbles and placing them at suitable locations. In implementation, this is realized by (1) mak-

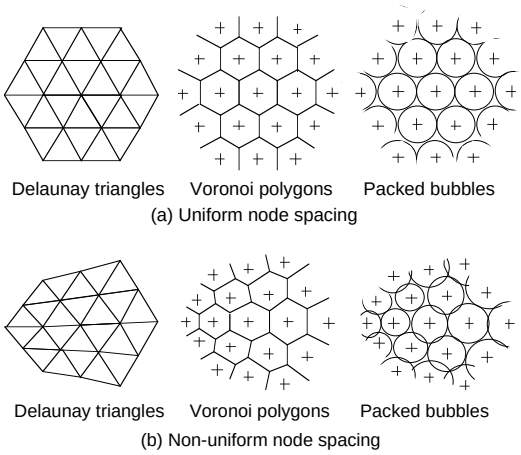


Figure 3: Delaunay triangles, Voronoi polygons, and packed bubbles

ing an initial guess, using hierarchical spatial subdivision (described in detail in Section 3.2); (2) defining proximity-based repulsive/attractive interbubble forces (described in detail in Section 3.3); and (3) performing dynamic simulation for a force-balancing configuration (described in detail in Section 3.4), while adaptively controlling the bubble population (described in detail in Section 3.5).

As shown in Figure 4, bubbles, or mesh nodes, are placed in order of dimension; that is, (1) bubbles are placed on all vertices (0-cells) in the non-manifold model, (2) bubbles are placed on all curves (1-cells), (3) all surfaces (2-cells) are filled in with bubbles, and (4) all volumes (3-cells) are filled in with bubbles. Once all the bubbles have been closely packed in the domain, a mesh node is placed at the center of each bubble. A triangular mesh is then created by using a constrained Delaunay triangulation to connect the nodes. The algorithm used is a modification of the Delaunay triangulation proposed by Watson [22].

3.2 Initial Bubble Placement

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 or tetrahedrization solution. Second, a good initial guess will greatly reduce the convergence time of the lengthy relaxation process later.

To handle general node spacing, we devised a bubble placement method based on hierarchical spatial subdivision. This method subdivides a curve, a surface, or a volume hierarchically by using a binary tree, a quadtree, or an octree respectively. In this process, the domain

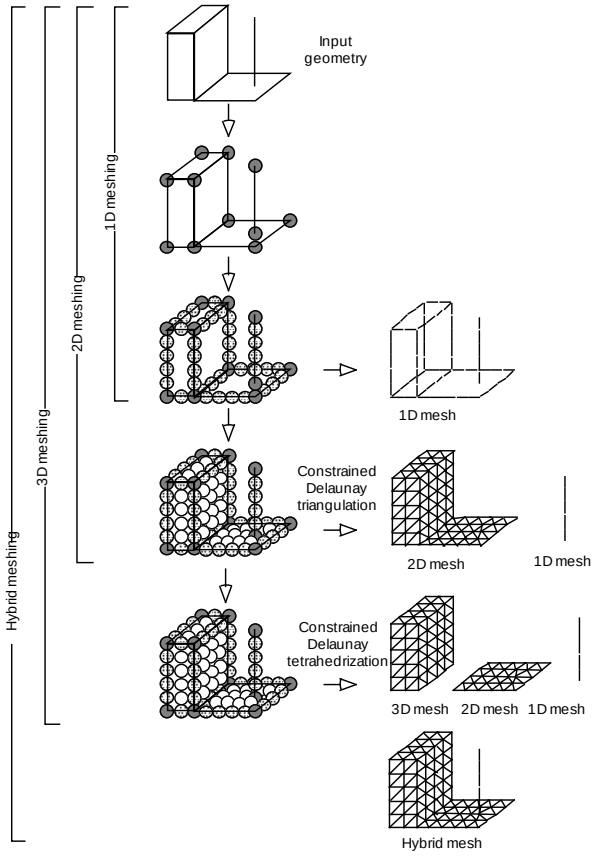


Figure 4: Non-manifold meshing procedures

is subdivided and bubbles are inserted until they cover the entire region without significant gaps or overlaps.

For example, in order to place initial bubbles on a curve segment, $C(s)$, $s_1 \leq s \leq s_2$, bubbles are first placed on the two end points of the domain, $C(s_1)$ and $C(s_2)$. The diameters of these bubbles in object space are calculated as $d_1 = d(C(s_1))$ and $d_2 = d(C(s_2))$ by using a node spacing function, $d(x, y, z)$. The length of the curve between the end points is then calculated, and this length is compared to the sum of the two radii, $d_1/2 + d_2/2$. If the curve length is longer than the sum of the radii, a new bubble is inserted at the midpoint, $C((s_1 + s2)/2)$, and the curve segment is subdivided into two sub-segments, $s_1 \leq (s_1 + s_2)/2$ and $(s_1 + s_2)/2 \leq s_2$. The same distance check is then performed on each sub-segment recursively.

For a surface and a volume, initial bubbles are placed by similar hierarchical subdivision except that we use oblique cells instead of square cells in order to realize hexagonal arrangement of the bubbles. Details of this algorithm are given elsewhere [19].

3.3 Interbubble Forces

A proximity-based interbubble force is defined so that a system of bubbles is in equilibrium when bubbles are closely packed, or "kissing" each other.

As shown in Figure 5(a), a mesh element can be generated by connecting the centers of adjacent bubbles; for example, a two-dimensional mesh element (i.e. triangle) can be generated by connecting the centers of three adjacent bubbles. Similarly, a one-dimensional mesh element (i.e. line segment) and a three-dimensional mesh element (i.e. tetrahedron) can be generated by connecting two or four bubbles, respectively.

As noted in Section 2.1, the size of a mesh element is measured by the distance between two nodes, that is, the length of a line segment, an edge of a triangle, or an edge of a tetrahedron. In an ideally tangential configuration as shown in Figure 5(a), this length is equal to the distance between the centers of two adjacent bubbles. Because we adjust bubble diameters to equal a given node-spacing function, $d(x, y, z)$, the *stable distance* l_0 between two bubbles i and j is calculated as the sum of the radii of the bubbles,

$$l_0 = \frac{d(x_i, y_i, z_i)}{2} + \frac{d(x_j, y_j, z_j)}{2}. \qquad (10)$$

We now define an *interbubble force* f, much like the van der Waals force, such that a repulsive force is applied when two bubbles are located closer than the stable distance l_0 (bubbles are overlapping), and an attractive force is applied when the bubbles lie farther apart than l_0 (there is a gap). As shown in Figure 5(b), the implemented force f is defined as a bounded cubic function of the distance l satisfying the following boundary conditions:

$$f(l) = \begin{cases} al^3 + bl^2 + cl + d & 0 \le l \le 1.5l_0 \\ 0 & 1.5l_0 < l \end{cases}$$

$$f(l_0) = f(1.5l_0) = 0, \quad f'(0) = 0, \quad f'(l_0) = -k_0. \qquad (11)$$

Note that k_0 represents the corresponding linear spring constant at the stable distance l_0. It is one of the key physical parameters that govern the behavior of the bubble system. Also note that, unlike the van der Waals force, the force defined here has the following characteristics: (1) the saturation of the force near $l = 0$, where l is the distance between the centers of the bubbles, prevents the force from growing infinitely large; and (2) the force is effective only within a specified range, $l < 1.5l_0$.

With this interbubble force, all the ill-shaped elements shown in Figure 2 are physically unstable because they all have some geometric entities located closer or further than the stable distance predicted by the node spacing function. Thus, the chance of creating such bad elements is significantly lowered.

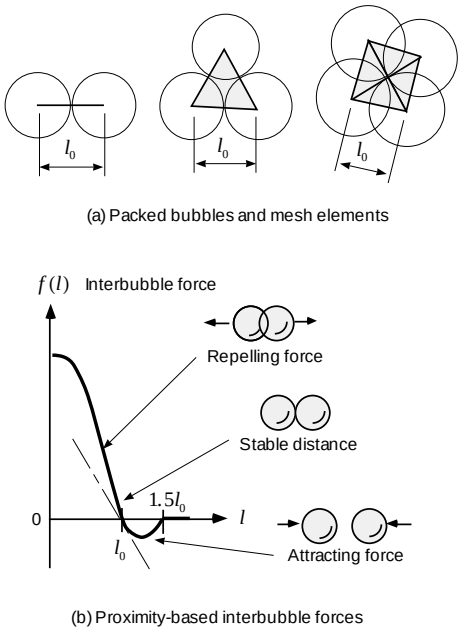


Figure 5: Interbubble repulsive/attractive forces

3.4 Physically Based Relaxation

Given the interbubble forces defined in the previous section, we must now find a bubble configuration that yields a static force balance. This is not a straightforward task, for two reasons: (1) the defined force is not linear; and (2) strict geometric constraints are imposed on each d.o.f. to keep a bubble on a specific curve or surface. Therefore, a direct solution to the static force-balance, such as the Newton-Raphson method of multidimensional root-finding, is not efficient. Instead, we use dynamic simulation, assuming a point mass at the center of each bubble and the effect of viscous damping.

Another reason for using dynamics to solve the force-balancing equation is to obtain continuous remeshing capability. Whenever the geometry and/or node spacing is slightly modified, dynamic simulation automatically produces a new stable configuration of bubbles close to the original configuration. This feature is useful in, for instance, structural analyses such as automobile crash analysis and sheet-metal forming analysis, where the geometry is continuously deformed.

The governing equation of motion of the ith bubble is written as follows:

$$m_i \frac{d^2\mathbf{x}_i(t)}{dt^2} + c_i \frac{d\mathbf{x}_i(t)}{dt} = \mathbf{f}_i(t), \qquad i = 1 \ldots n, \qquad (12)$$

where m_i denotes the mass, c_i the damping coefficient, and $\mathbf{x}_i$ the position of the ith d.o.f in object space. Given the initial locations of the bubbles, we integrate the differential equations through time at each

time step, using the standard *fourth-order Runge-Kutta* method [14]. The integration process is repeated a fixed number of times specified by the user, or until the system reaches equilibrium, a state in which the distance moved by the bubbles during one time-step in any d.o.f becomes less than a given small value.

With the above equation of motion, one of the remaining tasks is to "design" a combination of physical parameters, namely, the mass, the damping, and the strength of the interbubble force. Though this requirement is not often mentioned in publications on physically based approaches, it clearly constitutes a very important issue, because the characteristics of the system are all determined by the above parameters; if the parameters are not appropriate the system may be very slow or oscillatory, or, in the worst case, totally unstable. It is thus vital to select the parameters so that the system strikes a balance between stability and quick response. For this purpose, we first find the representative linear spring constant, k, for the non-linear interbubble forces and then apply knowledge about the standard second-order system consisting of a mass, a damper, and a linear spring.

One degree of freedom of bubble motion is approximated by the following equation of motion with the equivalent spring constant k, calculated from k_0 given in the previous section:

$$m\frac{d^2x(t)}{dt^2} + c\frac{dx(t)}{dt} + kx(t) = 0. \tag{13}$$

Although stability and a quick response time are conflicting requirements for this second-order system, we can strike a good balance between them if the damping ratio is set around 0.7 [12]. Consequently, the physical parameters of the bubble system should be chosen to satisfy the following relationship:

$$\zeta = \frac{c}{2\sqrt{mk}} \approx 0.7. \tag{14}$$

Note that the linear spring approximation mentioned above is used only to determine good physical parameter values, and not to calculate the force in dynamic simulation.

Having defined an equation of motion with a good combination of physical parameters, we consider how to confine bubbles to curves and surfaces. This is necessary because bubble locations calculated by numerically integrating the equation of motion are not geometrically compatible; that is, bubbles do not lie on the target curves and surfaces.

Hence, the movement of bubbles must be corrected in each time step, so that all the bubbles lie exactly on the target geometries. To explain how to confine bubbles to a curve, let us denote a 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 an unconstrained displacement from time t to time $t + \Delta t$ as $\Delta\mathbf{x}_i$ in object space, calculated simply by integrating the equation of motion. The correct, confined location of the bubble on a curve is obtained as follows:

1. Calculate the unconstrained displacement vector $\Delta\mathbf{x}_i$ in object space.
2. Calculate the normalized tangent vector at the bubble location at time t, $\frac{C^s}{|C^s|}$, where $C^s = \frac{dC(s)}{ds}$.
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:

$$\Delta s = \frac{\Delta\mathbf{x}_i \cdot C^s}{|C^s|^2}. \tag{15}$$

   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 $\mathbf{x}_i(t + \Delta t) = C(s(t) + \Delta s)$.

Bubbles are constrained to a surface in a similar way, using two tangent vectors, $S^u = \frac{\partial S(u,v)}{\partial u}$ and $S^v = \frac{\partial S(u,v)}{\partial v}$, instead of C^s as in the case of a curve.

3.5 Adaptive Population Control

In order to pack a necessary and sufficient number of bubbles within a domain, we devised an automatic method for adaptively controlling bubble population. This method examines a local bubble population, removes *excess bubbles* which significantly overlap their neighbors, and adds bubbles around *open bubbles* which lack an appropriate number of neighboring bubbles.

In order to determine whether a bubble is excess or open, the following *overlapping ratio*, α_i, for the ith bubble is defined:

$$\alpha_i = \frac{2}{d(\mathbf{x}_i)} \sum_{j=0}^{n} (d(\mathbf{x}_i) + \frac{d(\mathbf{x}_j)}{2} - \overline{\mathbf{x}_i\mathbf{x}_j}), \tag{16}$$

where $\mathbf{x}_i$ is the location of the ith bubble in object space, $\mathbf{x}_j$ is the location of the jth adjacent bubble, and n is the number of adjacent bubbles. This equation adds the distances to which the double-sized ith bubble penetrates or is separated from its neighboring bubbles, and then divides the result by the original bubble radius.

In an ideal situation where uniformly sized bubbles are tightly packed, the standard overlapping ratio of a bubble on an edge is $\alpha = 2.0$, that of a bubble on a face is $\alpha = 6.0$, and that of a bubble in a volume is

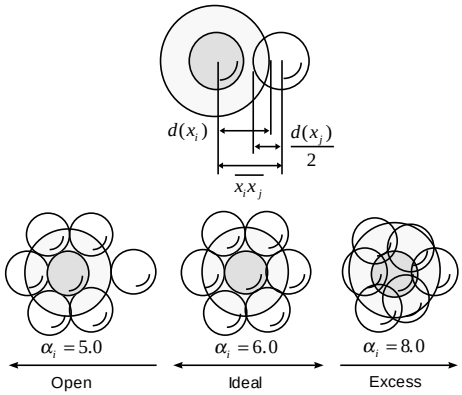


Figure 6: Overlapping ratio

$\alpha = 12.0$; these values correspond to the numbers of neighboring bubbles. When the overlapping ratio for the ith bubble is close to the above standard values, the local bubble population there is appropriate. Too small a ratio indicates that the bubble has significant gaps around it, so that one or more bubble must be added in the vicinity. Too large a ratio indicates that the bubble population there is too high, and that the bubble must be deleted. Figure 6 illustrates the relationship between the overlapping ratio and the bubble population.

The adaptive population control mechanism recognizes and eliminates the danger of deleting or adding too many bubbles. For example, if two bubbles overlap each other, each one is considered to have an overlapping neighbor that should, theoretically, be deleted. If this were done, however, both bubbles would be deleted, leaving a gap. On the other hand, adding too many bubbles between two open bubbles could be dangerous. The algorithm takes this "double-counting" into account, however, and deletes or adds just the appropriate number of bubbles, usually only one out of every few that would theoretically be required.

It is also important to note that the bubble population check is performed automatically only when the system of bubbles is relatively stable, that is, when the maximum velocity of all bubbles is within a small value.

With this adaptive population control mechanism, none of the ill-shaped elements shown in Figure 2 are likely to emerge because they are caused by significant gaps and overlaps between bubbles.

4 Results and Discussion

Figure 7 shows an example of 2D mesh relaxation. In this example, to show the effect of dynamic simulation more clearly, initial bubbles are placed randomly without using hierarchical spatial subdivision. As shown at the top of Figure 7, such a random node configuration produces many thin or flat triangles. The number of ill-shaped elements is reduced as the bubbles move into a force-balancing, or tightly packed, configuration. Given a random initial configuration with topological irregularity $\varepsilon_t = 1.03$, the irregularity is reduced to $\varepsilon_t = 0.38$ after 50 iterations of numerical integration of the equation of motion.

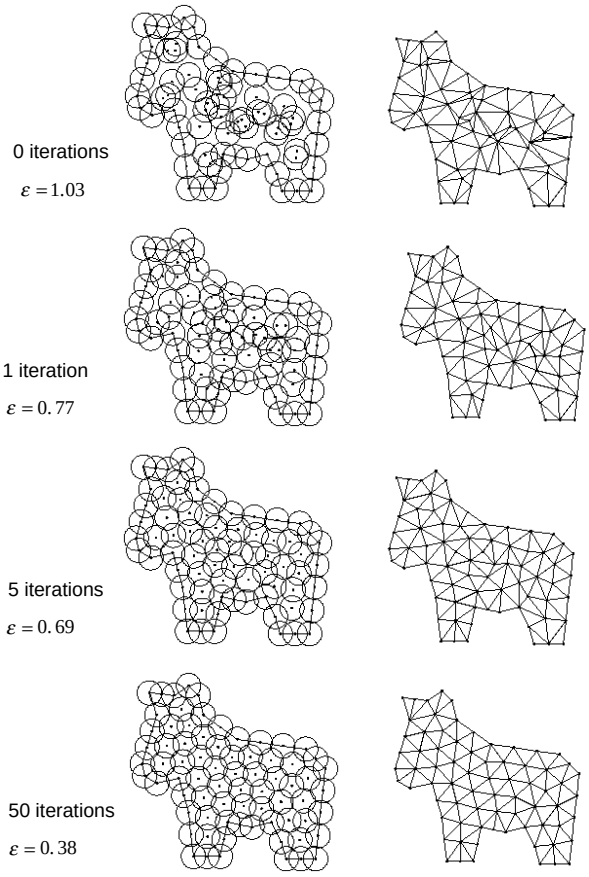
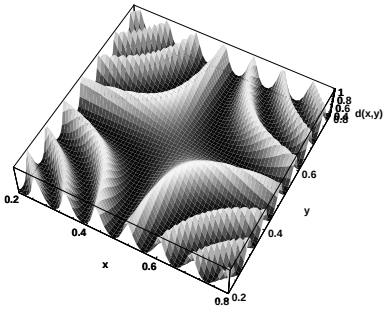


Figure 7: Physically-based mesh relaxation

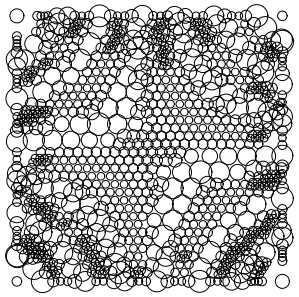
Figures 8 and 9 illustrate the result of surface-meshing. Figure 8(a) shows a complicated node-spacing function,

$$d(x, y, z) = \frac{1}{c_1 + \sin(c_2 + xy)}. \qquad (17)$$

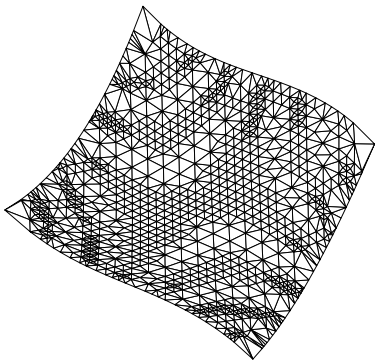
The initial bubble configuration in Figure 8 (b) was created by hierarchical spatial subdivision, causing some gaps and overlaps between bubbles. On account of these gaps and overlaps, triangulation of the initial bubble placement produces thin elements and flat elements in the regions where the node spacing changes, as shown in Figure 8(c). In dynamic simulation, however, as the bubbles move into a force-balancing configuration



(a) Node-spacing function



(b) Initial bubble placement using hierarchical spatial subdivision

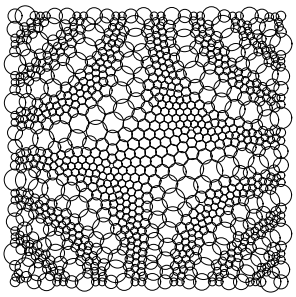


(c) Constrained Delaunay triangulation of the initial node configuration

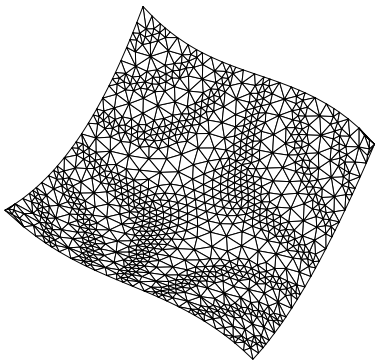
Figure 8: Initial bubble configuration and mesh

and their population is adaptively controlled, these ill-shaped elements are smoothed out, as shown in Figures 9(a) and 9(b). Table 1 summarizes how the mesh irregularity is reduced in physically based mesh relaxation.

Figures 10(a) and 10(b) illustrate the close packing of bubbles on a solid geometry and the resulting 3D mesh created by constrained Delaunay tetrahedrization. In this example, a constant node spacing is defined, yielding packed bubbles of a uniform size.



(a) Force-balancing node placement



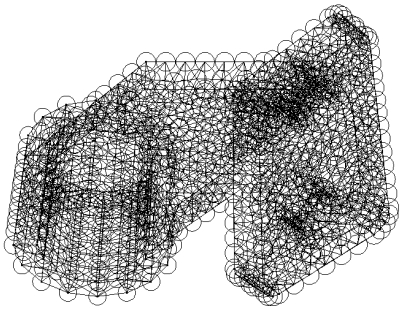
(b) Constrained Delaunay triangulation of a force-balancing node placement

Figure 9: Improved bubble configuration and mesh obtained by using physically based mesh relaxation

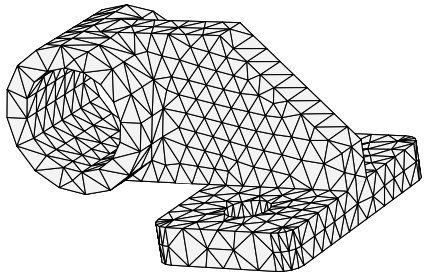
Table 1: Mesh irregularity mimimization

No. of iterations	0	10	50	100
ε_t (topological)	0.96	0.67	0.53	0.42
ε_g (geometrical)	0.31	0.24	0.11	0.04

Figure 11 shows meshing of a simple non-manifold geometry consisting of seven vertices, nine edges, five faces, and one volume. After bubbles have been tightly packed on these geometric entities, as shown in Figure 11(a), their centers are connected to give a set of line segments, triangles, and tetrahedra. Note that geometric compatibility is satisfied on (1) the joint-edge shared by the volume and the dangling face, and (2) the joint-vertex shared by the dangling face and the dangling edge.

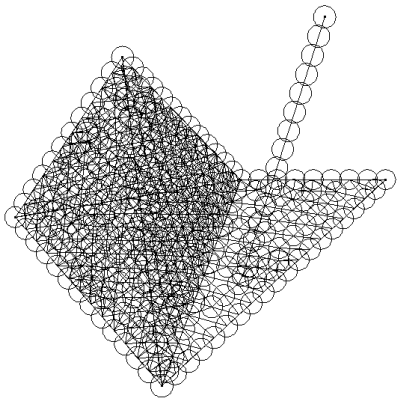


(a) Close packing of bubbles

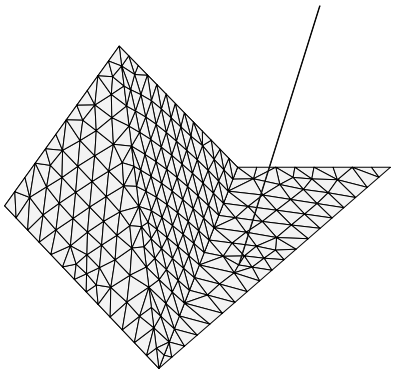


(b) 3D mesh consisting of tetrahedra

Figure 10: Uniform tetrahedrization of a 3D solid



(a) Close packing of uniformly sized bubbles



(b) Mesh consisting of line segments, triangles, and tetrahedra

Figure 11: Meshing non-manifold geometry

A rough estimate of the actual computational time may be obtained from the fact that the initial bubble placement obtained by hierarchical spatial subdivision is completed with in a few seconds for a system of up to 1000 bubbles on a typical engineering workstation such as the IBM RS/6000, while the physically based mesh relaxation for the same system requires 20 to 40 seconds to converge. The mesh relaxation stage is more time-consuming, because of the need to calculate interbubble forces and overlapping ratios. While a naive pairwise calculation of these costs $O(n^2)$, where n is the number of bubbles, it can be reduced to $O(n \log(n))$ by (1) using constrained Delaunay triangulation or tetrahedrization to find adjacent bubbles, and (2) calculating forces and overlapping ratios only between adjacent pairs of bubbles.

5 Conclusion

We developed a new computational method for physically based mesh generation. The novelty of the proposed bubble method is that the close packing of bubbles mimics a Voronoi diagram pattern, corresponding to well-shaped Delaunay triangles and tetrahedra. In order to find a configuration of closely packed 1D, 2D, and 3D bubbles, two critical problems must to be solved: (1) where to place bubbles for regular element shapes, and (2) how many bubbles to inject to fill a region.

Our proposed solution to the first problem is dynamic simulation with attractive or repulsive interbubble forces, a mass, and viscous damping effects. For the second problem of finding the right number of bubbles,

we proposed an adaptive population control mechanism.

In actual implementation, the bubble method generated node configurations that yield virtually no ill-shaped triangles or tetrahedra.

Acknowledgments

The authors would like to thank Professor Nicholas Patrikalakis and Professor Alex Pentland for their intensive feedback on this work. Thanks also go to Barbara Balents, Ashok Kumar, and Lian Fang for useful technical discussions during the development of the proposed method.

References

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

[2] Cavendish, J.C., *An Approach to Automatic Three-Dimensional Finite Element Mesh Generation*, International Journal for Numerical Methods in Engineering, vol. 21, pp. 329–347, 1985.

[3] Dey, T.K., C.L. Bajaj, and K. Sugihara, *On Good Triangulations in Three Dimensions*, International Journal of Computational Geometry & Applications, vol. 2, no. 1, pp. 75–95, 1992.

[4] Edelsbrunner, H., *Algorithms in Combinatorial Geometry*, Springer-Verlag, 1987.

[5] Fang, T.-P. and L.A. Piegl, *Algorithm for Constrained Delaunay Triangulation*, The Visual Computer, vol. 10, pp. 255–265, 1994.

[6] 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.

[7] Gursoz, E.L., Y. Choi, and F.B. Prinz, *Vertex-Based Representation of Non-manifold Boundaries*, in Geometric Modeling for Product Engineering, Elsevier, pp. 107–130, 1990.

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

[9] Kawabe, S., K. Shimada, and H. Masuda, *A Framework for 3D Modeling: Constraint-Based Description and Non-Manifold Geometric Modeling*, in Organization of Engineering Knowledge for Product Modelling in Computer Integrated Manufacturing, Elsevier, pp. 325–354, 1988. (and IBM Research Report TR87-1024, 1988.)

[10] Masuda, H., K. Shimada, M. Numao, and S. Kawabe, *A Mathematical Theory and Applications of Non-Manifold Geometric Modeling*, in Advanced Geometric Modeling for Engineering Applications, North-Holland, pp. 89–103, 1989.

[11] Meshkat, S., J. Ruppert, and V.T. Rajan, *CDSmesh: An Automatic Three-Dimensional Mesh Generator*, IBM Research Report.

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

[13] Preparata, F. and M. Shamos, *Computational Geometry – An Introduction*, Springer-Verlag, 1985.

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

[15] Rossignac, J.R., and M.A. O'Connor, *SGC: A Dimension-Independent Model for Pointsets with Internal Structures and Incomplete Boundaries*, in Geometric Modeling for Product Engineering, Elsevier, pp. 145–180, 1990.

[16] 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.

[17] Shephard, M.S., et al., *Trends in Automatic Three-Dimensional Mesh Generation*, Computers & Structures, vol. 30, no. 1/2, pp. 421–429, 1988.

[18] Shimada, K. and D.C. Gossard, *Computational Methods for Physically-Based FE Mesh Generation*, Proc. of the IFIP TC5/WG5.3 Eight International PROLAMAT Conference, edited by G.J. Olling and F. Kimura, pp. 411–420, 1992.

[19] Shimada, K., *Physically-Based Mesh Generation: Automated Triangulation of Surfaces and Volumes via Bubble Packing*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, 1993.

[20] Shimada, K., B. Balents, and D.C. Gossard, *Automatic Mesh Reconstruction for Feature-Based Sculpting of Deformable Surfaces*, IFIP Transaction B-8, Geometric Modeling for Product Realization, edited by P.R. Wilson, M.J. Wozny, and M.J. Pratt, pp. 165–188, 1993.

[21] 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.

[22] Watson, D.F., *Computing the n-Dimensional Delaunay Tessellation with Applications to Voronoi Polytopes*, Computer Journal, vol. 24, pp. 167–172, 1981.

[23] Weiler, K., *Topological Structures for Geometric Modeling*, Ph.D. thesis, Rensselaer Polytechnic Institute, NY, 1986.