IMPLEMENTATION AND ANALYSIS OF A BUBBLE PACKING METHOD FOR SURFACE
MESH GENERATION

By

SAURABH SAINI

A thesis submitted to the

Graduate School—Camden

Rutgers, The State University of New Jersey

In partial fulfilment of the requirements

For the degree of Master of Science

Graduate Program in Computer Science

Written under the direction of

Dr. Suneeta Ramaswami

And approved by

__________________________
Dr. Suneeta Ramaswami

__________________________
Dr. Jean-Camille Birget

__________________________
Dr. Steve Alessandrini

Camden, New Jersey

January 2019
THESIS ABSTRACT

Implementation and Analysis of a Bubble Packing Method for Surface Mesh Generation

by SAURABH SAINI

Thesis Director:
Dr. Suneeta Ramaswami

Abstract: This thesis presents an implementation and analysis of the bubble packing method used for surface mesh generation. The bubble packing method for surface mesh generation was first proposed by Dr Kenji Shimada in 1993 in his PhD thesis. The method generates a triangle mesh, a set of connected triangles that satisfy requirements of well-shaped triangles, compatible with a given domain, and with controlled node spacing or triangle size. This type of mesh is mainly used in finite element analysis and utilised in many simulations. There has not been much research on the bubble packing method since the 90s. The goal of this thesis is to implement and analyse the bubble packing method in order to create a path for possible future improvements and applicability of this approach to meshing. The approach in this method consists of three stages: placing nodes by spatial subdivision, moving nodes on proximity-based repulsive/attractive internode forces and finally, after nodes reach the required stability, connect nodes to create a set of well-shaped triangles by Delaunay triangulation. The first two stages, placement and movement of nodes, are carried out by placing a spherical region around each node and packing the spheres in the given domain as tightly as possible. The original description of the bubble packing method lacks many important details that are necessary for its practical implementation. This thesis also presents approaches to fill the gaps in the original algorithm so that it can be implemented.
# Table of Contents

THESIS ABSTRACT ........................................................................................................................................................................... ii  
List of Figures ...................................................................................................................................................................................... v  
List of Tables ...................................................................................................................................................................................... vii  
List of Equations .................................................................................................................................................................................. vii  
List of Pseudo Code ............................................................................................................................................................................... vii  
Chapter 1 Introduction ................................................................................................................................................................. 1  
Chapter 2 Background ................................................................................................................................................................. 3  
  2.1 Different Meshing (Grid Generation) Techniques .................................................................................................................. 3  
  2.2 Boundary Representation (B-rep or BREP) .............................................................................................................................. 7  
  2.3 Mass Spring Damper (MSD) System ........................................................................................................................................ 13  
Chapter 3 Terminology and Definitions ......................................................................................................................................... 16  
  3.1 Bubble .......................................................................................................................................................................................... 16  
  3.2 Parametric and Object Space ...................................................................................................................................................... 19  
  3.3 Rhombic Quadtree ......................................................................................................................................................................... 20  
  3.4 Relative Displacement in Parametric and Object Space ........................................................................................................ 22  
  3.5 Node spacing function ................................................................................................................................................................. 23  
Chapter 4 Bubble Packing Surface Triangle Meshing .................................................................................................................... 24  
  4.1 Summary of Algorithm ............................................................................................................................................................... 24  
  4.2 Additional Work .............................................................................................................................................................................. 29  
Chapter 5 Implementation Details ................................................................................................................................................... 38  
  5.1 Expected Input .................................................................................................................................................................................. 38  
  5.2 Node (Bubble) Data Structure .................................................................................................................................................. 38  
  5.3 Mapping B-Rep entity and Bubbles ........................................................................................................................................... 40  
  5.4 Population Control ....................................................................................................................................................................... 41  
  5.5 Force Balancing .............................................................................................................................................................................. 43  
  5.6 Stopping Criteria ......................................................................................................................................................................... 45  
Chapter 6 Results and Analysis ....................................................................................................................................................... 46  
  6.1 Quality parameters ....................................................................................................................................................................... 46  
  6.2 Outputs .......................................................................................................................................................................................... 47  
  6.3 Determining a suitable stopping criterion .................................................................................................................................. 53  
  6.4 Comparison between different Spring Constant ......................................................................................................................... 57  
  6.5 Conclusion ................................................................................................................................................................................... 58
List of Figures

Fig. 1.1 Mesh representation of a model with a different level of geometric preservation. ..... 1
Fig. 1.2 Mesh used in Structural analysis and Fluid dynamics ................................................... 2
Fig. 2.1 Different meshing techniques ....................................................................................... 3
Fig. 2.2 Structured mesh vs Unstructured mesh (credit: Point Wise [3]) ................................. 4
Fig. 2.3 Voronoi Polygon and Delaunay Triangulation ................................................................. 5
Fig. 2.4 Advancing front mesh generation [5] ........................................................................... 5
Fig. 2.5 Mesh generation by packing ........................................................................................ 6
Fig. 2.6 Boundary representation (B-rep) of Solid Model .......................................................... 7
Fig. 2.7 Parts of Boundary representation ................................................................................. 8
Fig. 2.8 Topological Vertex ......................................................................................................... 8
Fig. 2.9 Topological Edge ............................................................................................................ 9
Fig. 2.10 Topological Loop ......................................................................................................... 9
Fig. 2.11 Topological Face ......................................................................................................... 10
Fig. 2.12 Topological Shell ....................................................................................................... 10
Fig. 2.13 Topological Region .................................................................................................... 11
Fig. 2.14 Geometric Curve ........................................................................................................ 12
Fig. 2.15 Geometric Surface ..................................................................................................... 12
Fig. 2.16 Mass Spring Damper (MSD) System ......................................................................... 13
Fig. 2.17 The effect of varying damping ratio on an MSD System ............................................ 14
Fig. 3.1 A Bubble ...................................................................................................................... 16
Fig. 3.2 Force direction and spring motion .............................................................................. 16
Fig. 3.3 Bubble motion analogous to spring motion ............................................................... 16
Fig. 3.4 Region of Influence ...................................................................................................... 17
Fig. 3.5 Bubble Overlap Ratio ................................................................................................... 17
Fig. 3.6 Stable Bubble configuration ......................................................................................... 18
Fig. 3.7 Excess Bubble configuration ....................................................................................... 19
Fig. 3.8 Open Bubble configuration ......................................................................................... 19
Fig. 3.9 Parametric space and Object space ............................................................................ 20
Fig. 3.10 Square Quadtree ....................................................................................................... 21
Fig. 3.11 Rhombic Quadtree ..................................................................................................... 21
Fig. 3.12 Packing of bubbles on Quadtree cell ........................................................................ 21
Fig. 3.13 Relative displacement on Edge ................................................................................ 22
Fig. 3.14 Node spacing function and graded meshes ............................................................... 23
Fig. 4.1 Surface triangulation via bubble packing ................................................................. 24
Fig. 4.2 Spatial Subdivision on Edge ....................................................................................... 25
Fig. 4.3 Spatial Subdivision Cases on Edge ........................................................................... 26
Fig. 4.4 Spatial Subdivision on Face ....................................................................................... 26
Fig. 4.5 Rhombic cell coordinates calculation w.r.t. mid-point \( u \), \( v \) and cell length \( l \) ......... 26
Fig. 4.6 Spatial Subdivision Cases on Face ............................................................................. 27
Fig. 4.7 Open Bubble near the boundary ................................................................................ 29
Fig. 4.8 Layer of outside Bubble .............................................................................................. 29
Fig. 4.9 Node relations ............................................................................................................. 30
Fig. 4.10 Removing Nodes ........................................................................................................ 31
Fig. 4.11 Possible new bubbles location on the edge .............................................................. 32
Fig. 4.12 Open Bubble on edge with no existing neighbour .................................................... 33
Fig. 4.13 Open Bubble on edge with one existing neighbour .................................................. 33
Fig. 4.14 Open Bubble on edge with both neighbours ............................................................ 33
Fig. 4.15 Open Bubble on the face with no existing neighbour ............................................... 34
Fig. 4.16 Open Bubble on the face with one existing neighbour ............................................. 34
Fig. 4.17 Open Bubble on the face with multiple existing neighbours .................................... 35
Fig. 4.18 possible candidate if the angle between neighbours is less than π/2. ................. 35
Fig. 4.19 possible candidate if the angle between neighbours is greater than π/2. ............. 36
Fig. 4.20 Order of Population Control cause unnecessary removal and insertion. ............... 36
Fig. 5.1 Mapping B-Rep entity and Bubbles ............................................................................. 40
Fig. 5.2 Initial Data (\(Y_0\)) .................................................................................................. 43
Fig. 5.3 First-derivate (\(dY/dt\)) for given data (\(Y\)) .......................................................... 43
Fig. 5.4 Finding the total spring force on a bubble ................................................................. 44
Fig. 6.1 Bubble population and mesh irregularity. ................................................................. 46
Fig. 6.2 Aspect ratio of a triangle ........................................................................................... 47
Fig. 6.3 Output 1 ................................................................................................................... 48
Fig. 6.4 Output 2 ................................................................................................................... 49
Fig. 6.5 Output 3 ................................................................................................................... 50
Fig. 6.6 Output 4 ................................................................................................................... 51
Fig. 6.7 Output 5 ................................................................................................................... 52
Fig. 6.8 Standard Deviation of bubble movement Vs Number of iterations ......................... 53
Fig. 6.9 The 2% and 5% settling time in the unit-step response ............................................ 54
Fig. 6.10 Different energies Vs Number of iterations ............................................................. 54
Fig. 6.11 Mesh irregularity Vs Number of iterations ............................................................... 55
Fig. 6.12 Total energy with Mesh irregularity Vs Number of iterations ................................. 55
Fig. 6.13 Aspect Ratio Vs Number of iterations ..................................................................... 56
Fig. 6.14 Triangle Count with different min angle Vs Number of iterations ....................... 56
Fig. 6.15 Triangle Count with different max angle Vs Number of iterations ....................... 56
Fig. 6.16 Number of nodes added or deleted vs Number of iterations ............................... 56
Fig. 6.17 Min iteration for mesh irregularity vs Spring constant (k) ..................................... 57
Fig. 6.18 Min iteration for aspect ratio vs Spring constant (k) .............................................. 57
Fig. 7.1 Multi-core bubble neighbour update ....................................................................... 59
Fig. 7.2 Challenges with small features in the model .......................................................... 60
Fig. 7.3 Movement of bubble across face boundaries .......................................................... 60
List of Tables

Table 5.1 Bubble flags used by the algorithm ................................................................. 39

List of Equations

Eq. 2.1 Newton’s second law the force balancing ......................................................... 13
Eq. 2.2 The force balancing equation with no external force ........................................ 13
Eq. 2.3 Kinetic energy for the MSD system ................................................................. 13
Eq. 2.4 Potential energy for the MSD system .............................................................. 13
Eq. 2.5 Rate of change of energy for MSD system ..................................................... 14
Eq. 2.6 Damping ratio ............................................................................................... 14
Eq. 2.7 System of first Order Equation for MSD system .......................................... 15
Eq. 3.1 Space Curve .................................................................................................. 19
Eq. 3.2 Surface ......................................................................................................... 20
Eq. 3.3 Trimming curve in $\mathbb{R}^2$ .......................................................................... 20
Eq. 3.4 Trimming Space Curve .................................................................................. 20
Eq. 3.5 Relative displacement between mapped variables ....................................... 22
Eq. 4.1 Criteria to stop Spatial Subdivision on Face ............................................... 27
Eq. 5.1 Acceleration of each bubble ......................................................................... 43
Eq. 6.1 Mesh Irregularity ......................................................................................... 46
Eq. 6.2 Aspect ratio ................................................................................................. 47

List of Pseudo Code

P. Code 3.1 Calculate the overlap ratio for the bubble .............................................. 18
P. Code 5.1 Population Control ................................................................................. 41
P. Code 5.2 Removal of excess bubbles ..................................................................... 42
P. Code 5.3 New Bubble around Open Bubble .......................................................... 42
Chapter 1 Introduction

Recent decades have seen enormous improvement in high-performance computing and hardware which has enabled the rapid and cost effective prototyping of three-dimensional objects on a computer as a proof concept as well as efficient modification of initial prototyping. Many of these proofs of ideas need simulations where objects are placed in specified conditions; for example, air drag on a car in a windy environment or stress analysis on the wings of a plane at a given velocity and altitude. To carry out most of these kinds of simulations, we need approximate computer aided models for the objects and their environments. Due to the improvement in computer aided designing techniques, we are now able to produce complex models with very minute details preserved. However, as a result of the increased complexity of model geometry, directly solving the analytical equation on these complex geometries is either too time consuming or in some cases, impossible.

To handle these complex models, scientists and engineers have devised methods like Finite Element Analysis (FEA) and Finite Volume Methods (FVM) [1], which break the complex model geometry into a collection of small well defined and simple geometric objects like triangles and quadrilaterals (quads), while preserving the shape of the object within the given tolerance (Fig. 1.1). The process of dividing a geometry into a set of simple elements, like triangles and quads, is known as meshing. The resulting set of elements that satisfy basic requirements, such as precise control over element size and quality, compatible with domain boundaries, etc, is known as mesh.

![Sup. Image 01](credit: Stanford Bunny)

![Sup. Image 02](credit: Lp Centroidal Voronoi Tessellation and its applications by Bruno, Yang [17])

![Sup. Image 03](credit: CMU School of Computer Science [16])

Fig. 1.1 Mesh representation of a model with a different level of geometric preservation.
Since every element of a mesh is well-shaped, and has a simple geometry, and approximates a small region on an object, it can be used to solve the physical equation for that region quickly. The collective solution on each element in a mesh gives a good approximation to the solution one would obtain by applying the same equations to the entire object directly (Fig. 1.2). There are many different algorithms currently being used for meshing and a lot of research in this area focuses on the ways to improve the speed, quality of mesh generation, or the stability of the algorithm to handle complex geometries.

This thesis presents an implementation and analysis on one of these algorithms, namely the bubble packing algorithm. This algorithm applies a physics-based approach of packing bubbles in a region, then using the final position of these bubbles, generates the mesh. In Chapter 2, we build a general understanding about background information on meshing, B-Rep and mass-spring-damper system, which help in understanding the algorithm. In Chapter 3, we define some terminology and give definitions of the terms extensively used in the algorithm. In Chapter 4, we provide comprehensive details about the algorithm and explain some new approaches that are taken to complete the implementation. In Chapter 5, we give brief details about the code implementation like the data structures used, the search algorithm used, etc. In Chapter 6, we present some analysis of the results produced by our implementation of the algorithm. Finally, in Chapter 7, we discuss possibilities for future work that can be done on this algorithm.
Chapter 2 Background

In this chapter, we provide some background information on meshing, which will provide the appropriate context for the main algorithm in this thesis, namely, the bubble packing approach.

2.1 Different Meshing (Grid Generation) Techniques

A mesh is a critical part of the simulation and there are various meshing techniques in the literature [2]. Since the approximation error of a solution depends on the shape of elements in a mesh, some quality parameters are defined for elements in a mesh to quantify the overall quality of the mesh. The quality parameters are typically element size, element aspect ratio, mesh irregularity, etc. No single meshing technique can satisfy all possible requirements. Meshing is sometimes also referred to as grid generation. Meshes are divided into two main categories: structured and unstructured.

![Fig. 2.1 Different meshing techniques](image)

2.1.1 Structured Mesh Generation

For the implementation of numerical methods such as finite difference methods, each node in the computational domain must have easily identifiable neighbouring nodes [2]. A grid or mesh that satisfies this demand is the structured mesh. In other words, there is regular connectivity between elements which can easily be represented by $i$, $j$, $k$ indices (see in Fig. 2.2 left). The general approach for generating structured meshes for complex geometries is to map a complex physical domain on to a simple computational domain, and then apply a simple method to find structures subdivision of the computational domain.
For example, every node in a 2D structured mesh has corresponding integer $i$ and $j$ index values which are unique. The physical locations of the nodes are stored in a table or are functionally related to the mesh space (i.e. $(x, y) = f(i,j)$). It is also implied then that the neighbours of node $(i,j)$ are $(i-1,j)$, $(i+1,j)$, $(i,j-1)$, $(i,j+1)$, $(i-1,j-1)$, and $(i+1,j+1)$. A structured mesh makes it very easy to loop through neighbours and can be efficient with memory.

![Structured mesh vs Unstructured mesh](image)

Since this thesis deals with an unstructured mesh, we restrict our attention to unstructured mesh generation here. However, a general understanding of the difference between structured and the unstructured meshes is needed as the packing algorithm starts with node placements that resemble a structured grid, but which get transformed into our unstructured grid in later steps.

### 2.1.2 Unstructured mesh generation

Unstructured grids are more general than structured ones and can be made to conform to nearly any desired geometry (Fig. 2.2 right) [2]. This generality, however, comes with a price. The grid generation process is not completely automatic and may require considerable user interaction to produce grids with acceptable degrees of local resolution while at the same time having a minimum of element distortion. Unstructured grids require more information to be stored and recovered than structured grids (e.g., the neighbour connectivity list) and changing element types and sizes can increase numerical approximation errors. Though there is a major positive in unstructured. Since unstructured mesh generation doesn’t need mapping like in structured mesh generation, unstructured meshes are much faster to generate even for complicated geometry.

#### a) Delaunay mesh generation

Delaunay mesh generation is the most famous and widely used unstructured grid generation method. In this method, the domain is decomposed into a set of convex (Voronoi) polygons whose dual graph gives rise to a triangulation [4]. The Delaunay triangles have the property that each of their vertices is surrounded by a polygon (called Voronoi polygon) whose sides are formed by perpendicular bisectors of the sides of the triangles (Fig. 2.3). The Voronoi polygons are non-intersecting and cover the whole domain consisting of the node points. In the Delaunay
triangulation, every simplex satisfies the Delaunay property. In two dimensions, the Delaunay triangulation ($DT(P)$) for a given discrete set $P$ of points in the plane is a triangulation such that no point in $P$ is inside the circumcircle of any triangle in $DT(P)$. Similarly, in three dimensions, every tetrahedron of the DT has the empty circum-sphere property. The most widely used method of generating Delaunay triangles is by via the Circumcircle test or via Watson’s algorithm.

b) Advancing-Front mesh generation

The second most common method for mesh generation is a heuristic approach called advancing front method. Classical advancing-front approaches start from a discretization of the domain boundaries as a set of edges in two dimensions or a set of triangular faces in three dimensions [5]. The name of this class of methods refers to a strategy that consists of creating the mesh sequentially, element by element, creating new points and connecting them with previously created elements, thus marching into as yet unmeshed space and sweeping a front across the domain. The process stops when the front is empty, i.e., when the domain is entirely meshed. The front is the region(s) separating the part (or parts) of the domain already meshed from those that are still unmeshed (Fig. 2.4). Hence, depending of the strategy, the front can have multiple connected components.
c) Mesh generation by packing

Mesh generation by packing is not commonly used due to high computation time with no concrete guarantees of element quality. However, these methods are very useful for simulations which are based on partial structures as it the method closely resemble to the packing states of particles in nature. In mesh generation by packing, the algorithm starts with primitive object shapes such as circles or spheres etc. and tries to pack them closely in a domain. This packed configuration of shapes is then used to generate a mesh (Fig. 2.5). The size and shape of these primitive object depend on the nature of simulation for which the mesh is generated. For example, Fig. 2.5 a) is taken from a paper on “Micromagnetic modelling and small-angle neutron scattering characterization of magnetic nanocomposites [6]” and mesh generation by random close packing of spheres with different diameters (large spheres correspond to the magnetically hard inclusions). Fig. 2.5 b) is taken from a paper “An automatic method for generating carbon nanostructure atomistic models using hexagonal meshes with properly distributed defects [7]” and mesh generated via the sphere-packing method.
### 2.2 Boundary Representation (B-rep or BREP)

A B-rep is a data structure to store solid model representations. Different representation techniques are in use to represent and store solid models [8]. Some of the main ones are:

- **Cell decomposition**: Explicitly represents the material of which the solid is composed by dividing space into a set of elements, each full of material;

- **General sweeping**: Represents objects in terms of two-dimensional shapes extruded along general curves;

- **Set-theoretic**: There are two variants: one representing solids as combinations of primitive shapes known as Constructive Solid Geometry (CSG), and the other representing objects by defining a set of surfaces bounding the object;

- **Boundary representation (B-rep)**: This represents objects in terms of their ‘skin’, the boundary between model and non-model [9] (as shown in Fig. 2.1). Though we do not directly use all aspects of B-rep, a general understanding is necessary for the algorithm. In our algorithm since we restrict ourselves to surface meshing, we need to use only a subset of the whole B-rep data structure.

![Fig. 2.6 Boundary representation (B-rep) of Solid Model](image)

The basic method for B-rep was developed independently in the early 1970s by both Ian C. Braid in Cambridge (for CAD) and Bruce G. Baumgart at Stanford (for computer vision). Braid continued his work with the research solid modeller BUILD which was the forerunner of many research and commercial solid modelling systems. Braid worked on the commercial systems ROMULUS, the forerunner of Parasolid, and on ACIS. Parasolid and ACIS are the basis for many of today's commercial CAD systems.
2.2.1 Parts of Boundary representation

Boundary representation of models is composed of two parts: Topology and Geometry.

![Diagram of B-rep](image)

Fig. 2.7 Parts of Boundary representation

a) Topology

Topology is responsible for defining a rubber structure of the object (sometimes known as *rubber geometry*), i.e. it defines the connectivity of one entity with other, these entities don’t have fix shape or position in the space (hence known as rubber geometry). Topology in combination with Geometry (explained in next section) can define a physical object. Basic topological entities are Vertex, Edge, Loop, Face, Shell, Region and Body.

*Vertex*

Vertex is the lowest possible topological entity in a topological structure and sometimes called as a zero-dimensional entity (Fig. 2.8). A Vertex is mapped to geometrical entity *Point* to define a physical position in Euclidean space. A Vertex can only map to one geometrical Point whereas a Point can be mapped to many vertices hence there is a *many to one* \(n:1\) mapping relation between Vertex and Point.

![Diagram of Topological Vertex](image)

Fig. 2.8 Topological Vertex

*Edge*

An edge represents the one-dimensional entity and is generally bounded with vertices (Fig. 2.9). An edge can be bounded with 0, 1 or 2 vertices, and is mapped to geometrical entity *Curve* to define a physical curve like object in Euclidean space. Edge holds *length property* for an object. Edge with 0 or 1 vertices are called *periodic edge* and has a map to a closed curve that has the same start and end. An Edge can only map to a geometrical Curve whereas a Curve can be mapped to many edges hence there is a many to one \(n:1\) mapping relation between Edge and Curve. Since Edge can have at most two vertices but a vertex can have many edges, therefore, there is a *many to two* \(n:2\) mapping relation between Edge and Vertex.
Loop
Loop is a collection of directed-edge in closed circuits (Fig. 2.10). There are some degenerated loops where the loop is just a vertex like the apex of a cone, etc. Loops are defined to bound a face or in other words; loops define the boundary of a face. A Loop can only be part of one topological Face, i.e. every face is bounded (or defined) by a set of unique loops. A Loop has many edges, and an edge can be a part of many different loops hence there is a many to many \((n: n)\) mapping relation between Loop and Edge.

Face
Face represents a two-dimensional entity and generally bounded by a set of unique loops (Fig. 2.11). A loop in a face can be to two type exterior (outer) or interior(inner) and the orientation of exterior loops is always in reverse direction to the inner loops. Exterior loop bound the maximum extent of a face whereas interior loop bound holes in a face. Due to the orientation of loops, a face has a sense of direction, i.e. a face point in a direction using the right-hand rule on Exterior loop (this referencing varies system to system). A Face can have any number of exterior or interior number of loops. Face holds area property for an object. A Face is mapped to geometrical entity Surface to define a physical shape in Euclidean space and can only map to a geometrical Surface whereas a Surface can be mapped to many faces hence there is a many to one \((n: 1)\) mapping relation between Face and Surface. Similarly, Face can have many loops, but a loop can have only one face therefor there is a one to many \((1:n)\) mapping relation between Face and Loop.
**Shell**
Shell is a closed set of directed faces in the object. Just as a loop is a transition from a one-dimensional element to a two-dimensional element, similarly, Shell is a transition from a two-dimensional element to a three-dimensional element. Due to the directed nature of face in a shell, the shell itself has a sense of direction it points (Fig. 2.12). A Shell has many faces, and a face can be a part of many different shells hence there is a many to many \((n:n)\) mapping relation between Shell and Face.

**Region**
The region represents a three-dimensional entity and is generally bounded by Shells (Fig. 2.13). In some cases, Region is also referred as a lump, since it is usually representing entities that have mass in the physical sense, but that is only possible if and only if all the entities in a region is mapped to corresponding geometrical entities so that a region can have fix shape in Euclidean space. A Region can be bounded with any number of shells; just like Face, Region also has exterior and interior shells. Exterior shell bound the maximum extent of a region whereas interior shell bound voids (3d analogous to holes in 2d) in a region, for example, a solid cube has only one outer shell, whereas a cube with a cubical void region inside it, has two shells, one exterior and one interior. The region holds the **volumetric property** for an object and can have a material assigned to it. Region with no external shell some time referred to as
an infinite region since it has no bound for the extent. Region can have many shells, but a shell can only be a part of one region therefore there is a one to many (1: n) mapping relation between Region and Shell.

**Body**

Body usually is the topmost topological entity in a topological structure and usually defines the model in focus in a system (Fig. 2.14). There are three possible types of body, Solid Body, Sheet Body and Wireframe body.

- **Solid Body** is a collection (set) of regions. Solid body has volumetric properties that differentiate from other bodies.
- **Sheet Body** is a collection (set) of faces. In Sheet body, there is no concept of inside and outside. Sheet body has area properties that differentiate it from other bodies.
- **Wireframe Body** is a collection (set) of Edges.
b) Geometry

Geometry is responsible for defining the shape of the object. Geometry entities can be infinite or finite. Geometric entities are trimmed by topology to give shape and physical attributes to the object. Geometry entities are usually analytically defined by parametric equations. Basic geometric entities are *Point*, *Curve*, and *Surface*.

**Point**

Point defines the specific position in three-dimensional space of Euclidean geometry and has explicit values of \(x, y, z\) co-ordinates.

**Curve**

A curve is defined by a continuous function \(\gamma: I \rightarrow X\) from an interval \(I\) of the real numbers into a topological space \(X\). Depending on the context, it is either \(\gamma\) or its image \(\gamma(I)\) which is called a curve (Fig. 2.15). A curve can be a *Plane Curve* for which \(X\) is the Euclidean plane or a *Space Curve* for which \(X\) is of three dimensions, usually Euclidean space. A curve \(\gamma\) is called closed curve if for \(I = [a, b]\) we have \(\gamma(a) = \gamma(b)\).  

![Fig. 2.15 Geometric Curve](image)

**Surface**

Often, a surface is defined by equations that are satisfied by the coordinates of its points. The set of the zeros of a function of three variables is a surface, which is called an *implicit surface*. If the defining three-variate function is a polynomial, the surface is an *algebraic surface*. For our algorithm, we focus on the *parametric surface* (Fig. 2.16). A parametric surface is a surface in Euclidean space \(\mathbb{R}^3\) which is defined by a parametric equation with two parameters \(\vec{r}: \mathbb{R}^2 \rightarrow \mathbb{R}^3\) or can be represented as \(\vec{r}(u, v)\).

![Fig. 2.16 Geometric Surface](image)
2.3 Mass Spring Damper (MSD) System

A simple Mass Spring Damper (MSD) system is a very common system in vibration analysis [10]. The **mass** of the dynamic system is lumped into a single point mass in the MSD system. The inertial effect of the dynamic system is related to this lumped mass that corresponds to **kinetic energy** of the system. The stored energy/internal energy of the dynamic system is modelled as a one-dimensional **spring** in the MSD system. The spring can store energy inside when it is stretched or compressed from its original length, namely, **potential energy**. The energy dissipated out of the dynamic system is modelled through a one-dimensional **damper** in the MSD system. The viscous damper, for instance, can dissipate energy as heat outside the dynamic system. The system shown in the figure that has one spring, one damper, and one mass has one degree of freedom. By Newton’s second law the force balancing equation can be:

\[ m \cdot \frac{d^2x}{dt^2} + c \cdot \frac{dx}{dt} + k \cdot x = F(t) \]

**Eq. 2.1** Newton’s second law the force balancing

Where \( m \cdot \frac{d^2x}{dt^2} \) : is the force exerted by the motion of the mass, \( c \cdot \frac{dx}{dt} \) : is the force exerted by the damper and the damper’s force is directly proportional to the velocity of the mass, \( k \cdot x \): is the force exerted by the spring and the spring’s force is directly proportional to the displacement of the mass; and lastly \( F(t) \) is any external force exerted on the system. Now since there is no external force in our system \( F(t) = 0 \), hence the equation can be re-written:

\[ m \cdot \frac{d^2\ddot{x}}{dt^2} + c \cdot \frac{d\ddot{x}}{dt} + k \cdot \ddot{x} = 0 \]

**Eq. 2.2** The force balancing equation with no external force

The Kinetic energy, \( K \), for the system is given by,

\[ K = \frac{1}{2} m v^2 = \frac{1}{2} m \cdot \left( \frac{dx}{dt} \right)^2 \]

**Eq. 2.3** Kinetic energy for the MSD system

And Potential energy, \( U \), for the system is given by,

\[ U = \int_0^x k \cdot y dy = \frac{1}{2} k \cdot x^2 \]

**Eq. 2.4** Potential energy for the MSD system
Since, the total energy of the system is $E = K + U$, by using Eq. 2.2, Eq. 2.3, Eq. 2.4 and simple calculus the rate of change of energy $(dE/dt)$ of the system is given by:

$$\frac{dE}{dt} = -c \cdot \left(\frac{dx}{dt}\right)^2 \leq 0$$

Eq. 2.5 Rate of change of energy for MSD system

Thus, the total energy is decreasing for all time, and the rate is directly proportional to $c$ (Damping Coefficient) as well as the square of the velocity. If the system configuration remains constant (i.e. no external force, no change mass, no change spring constant or no change in damping coefficient over time), system will guaranties to achieve an equilibrium state but since the rate of change of energy $(dE/dt)$ is directly proportional to the square of the velocity, $(dE/dt)$ goes down rapidly causing true equilibrium to occur at $t \to \infty$.

Now if we start with initial conditions of the position $\bar{x}_0$, velocity $\dot{\bar{x}}_0$ and acceleration $\ddot{\bar{x}}_0$, solution to the differential equation Eq. 2.2 can govern the motion of the mass in $\mathbb{R}^3$ space. In our system $\ddot{x}_0 = 0$ and $\dot{x}_0 = 0$, and solution for the above equation can be achieved in an analytical method or by numerical method. While solving the above equation analytically by assuming a solution of the type $x(t) = Ce^{st}$, and solving for $s$, we define a dimensionless positive number $\zeta$, called the damping ratio which given by

$$\zeta = \frac{c}{2\sqrt{mk}}$$

Eq. 2.6 Damping ratio

If $0 \leq \zeta \leq 1.0$, we have an underdamped system which implies the system oscillates before reaching equilibrium or steady state. If $\zeta = 0$, i.e. $c = 0$ and there is no damper in the system, the system become a harmonic oscillator and system never reaches a steady state.

If $\zeta > 1.0$, we have an overdamped system which implies there will be no oscillation in the system and system reaches equilibrium or steady state directly and there is no overshoot. In an overdamped system the bigger the $\zeta$, the slower the system will be in reaching the steady state. If $\zeta = 1.0$, it is called a critically damped system which implies the system will reach a steady state as quickly as possible without oscillating (although overshoot can occur).

Fig. 2.18 The effect of varying damping ratio on an MSD System.
The equation governing the motion in MSD system defined above is second-order differential equation. Which can easily be converted into the system of first-order differential equations. Then, a 4-Order Runge-Kutta Methods \[11\] can be applied to the system first-order differential equation to solve it numerically with the desired accuracy. Let \( x = y_0 \) then \( \left(\frac{dx}{dt}\right) = y_0 = y_1 \) and then \( \left(\frac{d^2x}{dt^2}\right) = y_1 = y_0^\prime \), and rearranging the variable in the above equation.

\[
\begin{align*}
\dot{y}_0 &= y_1 \\
\dot{y}_1 &= -\frac{1}{m}(c \cdot y_1 + k \cdot y_0)
\end{align*}
\]

Eq. 2.7 System of first Order Equation for MSD system

Now we can define a derivative function \( g(t,y) \) such that for \( Y = [y_0, y_1] \) and \( \dot{Y} = [\dot{y}_0, \dot{y}_1] \) derivative function will be \( \dot{Y} = G(t,Y) \).
Chapter 3 Terminology and Definitions

This chapter will introduce or define some terms that are used intensively in the algorithm. A better understanding of these terms is required before we start discussing the steps in the algorithm.

3.1 Bubble

The bubble is a term given to a point location in Euclidean space that has three spherical regions defined around it (shown in Fig. 3.1).

3.1.1 Bubble Boundary

A bubble boundary is a boundary of an invisible spherical region around the point in space. At this boundary, the direction of force changes signs, i.e. if we consider a spring (Fig. 3.2), it will represent the resting position of the spring.

![Fig. 3.2 Force direction and spring motion](image)

Hence, if the spring is moved in either direction (pressed or stretched), it will exert a force that will try to bring the spring back to the resting position. Therefore, the direction of the force exerted by the spring gets reversed at the resting position. A spring is a one-dimensional entity with one degree of freedom. We can extend this example to 3 dimensions by considering rubber balls (Fig. 3.3) that have the property to attract or repel other balls. A similar phenomenon exists in nature, called van der Waals force, between molecules and atoms.

![Fig. 3.3 Bubble motion analogous to spring motion](image)
The force causes the motion of a point. The value of force depends on the physical properties assigned to the points and a system such as the mass of point, the spring constant between the points and the damping coefficient of the system, etc. this is explained in the Mass Spring Damper (MSD) System.

3.1.2 Region of Influence

The region of influence is an invisible spherical region around the point in space and beyond the point’s bubble boundary. This region defines the virtual boundary to restrict the number of points that are to be considered for the force balancing calculation. If the distance between two points is greater than the sum of their radii of the region of influence, those two points will not affect each other’s motion (Fig. 3.4).

![Fig. 3.4 Region of Influence](image)

3.1.3 Overlap Boundary and Bubble Overlap Ratio

The overlap boundary is an invisible spherical region around the point in space and beyond the point’s region of influence. The overlap boundary is used to find the overlap between the given bubble to the bubbles around it. This overlap is quantified by the Overlap Ratio for the given bubble. The overlap ratio is used to determine the bubble density around a given bubble. The overlap ratio is calculated by first adding the distances to which the overlap boundary of the given bubble penetrates its neighbours and then dividing the result by the original bubble radius (Fig. 3.5).

![Fig. 3.5 Bubble Overlap Ratio](image)

The overlap ratio is calculated differently for a bubble at the vertex, a bubble at the edge, and a bubble at face, since not all neighbouring bubbles around a given bubble are considered for overlap calculation. Bubbles are considered for overlap will be discussed later in detail in
section 4.2.1 and 4.2.2 where we discuss about the relations between nodes. If \( p \) is a position in Euclidean space, \( r \) is bubble boundary radius, \( r_o \) is overlap boundary radius and \( \text{neighbours} \) is list of bubbles that are consider for overlap for and given bubble, following is pseudo code to calculate the overlap ratio for the bubble (Fig. 3.5):

```plaintext
calculateOverlap(p, r, r_o, neighbours):
    totalOverlap = 0.0
    foreach p', r' in neighbours:
        d = distance(p, p')
        overlap = r_o + r' - d
        totalOverlap = totalOverlap + overlap
    end
    return (totalOverlap / r)
end
```
P. Code 3.1 Calculate the overlap ratio for the bubble

Once the overlap ratio is calculated for a bubble, we can categorise the bubble state into three types: Stable Bubble, Excess Bubble, and Open Bubble.

a) Stable Bubble

Stable Bubble is a bubble configuration where the bubble makes the best possible packing configuration, which depends on the location of the bubble (The best packing configuration for an edge bubble differs from that for a face bubble). For a face bubble (as shown in Fig. 3.6) a best possible configuration packs bubbles all around the bubble, whereas for an edge bubble’s (as shown in Fig. 3.6) best possible configuration packs bubbles along the edge. By using an appropriate cut-off value for a bubble’s overlap ratio, we can categorise it as an “Excess” and “Open” bubble.

![Stable edge Configuration](image)

![Stable Face Configuration](image)

Fig. 3.6 Stable Bubble configuration

b) Excess Bubble

An excess bubble is one that significantly overlaps its neighbours and should consequently be deleted. The cut-off ratio for an excess bubble differs for vertex, edge, and face bubble since the maximum number of neighbouring bubbles depends on the location of the bubble as discussed in above. Examples of excess bubbles are shown in Fig. 3.7.
c) Open Bubble

An open bubble is one that lacks an appropriate number of neighbouring bubbles, thus creating a gap in the bubble configuration. One or more additional bubbles must be added around an open bubble. As for an excess bubble, the Cut-off overlap ratio for an open bubble is different for vertex, edge, and face bubble. Examples of open bubbles are shown in Fig. 3.8.

3.2 Parametric and Object Space

A parametric equation defines a group of quantities as functions of one or more independent variables called parameters. Parametric equations are commonly used to express the coordinates of the points that make up a geometric object such as a curve or surface. In this case, equations are collectively called a parametric representation or parameterisation of the object [12]. Generally, the space that bounds all possible values of parameters is called Parametric Space; similarly, the space that bounds all possible values that make up a geometric object is Object Space. Parametric equations are the mapping functions between these spaces. In most cases, Object Space is $\mathbb{R}^3$, where a point is specified as $(x, y, z)$. The collection of these points as a one-dimensional entity; i.e., zero area entity is called a Space Curve, represented as $C$. The parametric space for a space curve is bounded in $\mathbb{R}^3$ and the general notation for a variable is $t$ or $s$. In our case, we will use $s$ such that the parametric equation for space curve is

$$C(s) = (x(s), y(s), z(s)).$$

Eq. 3.1 Space Curve
Similarly, a two-dimensional entity in object space; i.e., zero volume entity is called **Surface**. The parametric space for a surface is bounded in \( \mathbb{R}^2 \) where the general notation for a point in \( \mathbb{R}^2 \) is \((u, v)\). Therefore, the parametric equation for the surface is

\[
S(u, v) = (x(u, v), y(u, v), z(u, v)).
\]

Eq. 3.2 Surface

The mapping between these spaces is shown in Fig. 3.9.

### 3.2.1 Trimmed Surfaces

To create a complex surface, surfaces are sometimes trimmed by restricting the rectangular region in parametric space to a subset called the **trimmed region** and its boundary is called the **trimming curve**, denoted by

\[
c(s) = (u(s), v(s))
\]

Eq. 3.3 Trimming curve in \( \mathbb{R}^2 \)

Since trimming curves are planar curves, they are defined in \( \mathbb{R}^2 \). They can be interpreted as a mapping from \( \mathbb{R}^1 \rightarrow \mathbb{R}^2 \rightarrow \mathbb{R}^3 \) to actual **Trimming Space Curve** \( C \).

\[
C(s) = \left( x(u(s), v(s)), y(u(s), v(s)), z(u(s), v(s)) \right)
\]

Eq. 3.4 Trimming Space Curve

![image](image1.png)

**Fig. 3.9 Parametric space and Object space**

### 3.3 Rhombic Quadtree

A quadtree is a tree data structure in which each internal node has exactly four children. It is most often used to partition a two-dimensional space by recursively subdividing it into four quadrants or regions. A quadrant is subdivided whenever it contains some geometric data. The data associated with a leaf cell varies by application, but the leaf cell represents a "unit of interesting spatial information"(Fig. 3.10).
The subdivided regions may be square or rectangular or may have arbitrary shapes. In a Rhombic Quadtree, the subdivided regions are rhombic in shape (Fig. 3.11).

In our case, we create a rhombic cell with $60^\circ$ angle as this type of rhombic cells create a best packing for bubbles of equal radii placed at each node, as shown in Fig. 3.12.
3.4 Relative Displacement in Parametric and Object Space

As we are working with two spaces, we need to devise a mapping method to map displacement in object space to parametric space and vice-versa. Our surface is defined as an equation of two parameters (Eq. 3.2), and a given condition is that the surface must be continuous and differentiable at every point. Using simple calculus, we can create an approximation of relative displacement.

Let

\[ y_0 = f(x). \]

Then, using Taylor's expansion

\[ y_1 = f(x + \Delta x) = f(x) + \Delta x \cdot f'(x) + \frac{(\Delta x)^2}{2!} \cdot f''(x) + \frac{(\Delta x)^3}{3!} \cdot f'''(x) + \ldots \]

So, if we have small \( \Delta x \) change, we can approximate

\[ y_1 = y_0 + \Delta x \cdot f'(x) \]

\[ y_1 - y_0 = \Delta x \cdot f'(x) \]

\[ \Delta y = f'(x) \cdot \Delta x \]

Eq. 3.5 Relative displacement between mapped variables

3.4.1 Relative displacements on Edge

We can apply this relative displacement method (Eq. 3.5) to a space curve (Eq. 3.1).

\[ C(s) = (x(s), y(s), z(s)) = Y \]

\[ \Delta Y = C'(s) \cdot \Delta s \]

\( C'(s) \) is the tangent to the curve at given \( s \), \( \Delta Y \) the net displacement in object space and \( \Delta s \) the relative displacement in parametric space (Fig. 3.13). Applying a dot product of \( C'(s) \) to both side we get

\[ C'(s) \cdot \Delta Y = C'(s) \cdot C'(s) \cdot \Delta s \]

\[ C'(s) \cdot \Delta Y = \|C'(s)\|^2 \cdot \Delta s \]

\[ \Delta s = \frac{C'(s) \cdot \Delta Y}{\|C'(s)\|^2} \]

Fig. 3.13 Relative displacement on Edge
3.4.2 Relative displacements on Surface

Apply the relative displacement method (Eq. 3.5) to a surface (Eq. 3.2), we get

$$S(u, v) = (x(u, v), y(u, v), z(u, v)) = Y$$

$$\Delta u = \frac{S_u'(u, v) \cdot \Delta Y}{\|S_u'(u, v)\|^2}$$

$$\Delta v = \frac{S_v'(u, v) \cdot \Delta Y}{\|S_v'(u, v)\|^2}$$

$S_u'(u, v)$ and $S_v'(u, v)$ is the tangents to the surface with respect to $u$ and $v$, respectively, at given $u, v; \Delta Y$ the net displacement in object space and $(\Delta u, \Delta v)$ the relative displacement in parametric space.

3.5 Node spacing function

Localized control on triangle size in a domain is an important part of any Mesher. Depending on the region of interest in a domain mesh consumer wants variation from denser (high number of smaller size triangles) to coarser (low number or bigger size triangles) mesh in domain. The triangle size is dependent on the spacing between the nodes in a domain and distance between node in bubble packing depends on the size (radius) of the bubble. In this algorithm, the radius of a bubble at a given location in domain is control by a node spacing function. A node spacing function is a user defined function which return the radius of the bubble for a given location in the space. By providing this function, a mesh consumer can generate a graded mesh.

![Node spacing function and graded meshes](image)
Chapter 4 Bubble Packing Surface
Triangle Meshing

The bubble packing meshing algorithm was first devised and described as a PhD thesis of prof. Kenji Shimada in 1993 [13]. The PhD thesis gives an algorithm for a surface mesh as well as a volume mesh. In this thesis, we only focus on surface meshes.

4.1 Summary of Algorithm

The bubble method can be summarised simply in two steps:

(1) Pack spheres, or bubbles, closely in the domain.
(2) Connect their centres by constrained Delaunay triangulation.

Fig. 4.1 Surface triangulation via bubble packing

Packing of the spheres can be done by Physically Based Relaxation which we will discuss later in section 4.1.3. Before applying relaxation methods on bubbles, we need to find an intial position for each bubble in the domain. A quick initial guess for bubble configuration is obtained by using hierarchical spatial subdivision which discussed in detail in the following section. A good initial guess bubble configuration is important because it reduces the computational time necessary for the normally time-consuming process of dynamic simulation, or physically based relaxation. A population control mechanism is used during relaxation to remove any superfluous bubble that is largely overlapped by its neighbours and to add new bubble around any lone bubble missing some neighbours so that a given domain is filled with an appropriate number of bubbles. The automatic population control mechanism feature drastically reduces the time taken for the system to converge to a force-balancing configuration.

The whole process of packing bubbles in the domain can be divided into three independent steps, Initial Bubble Placement, Adaptive Bubble Population Control and Physically Based
Relaxation.

4.1.1 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 solution. Second, a good initial guess will greatly reduce the convergence time of the lengthy relaxation process later. The algorithm makes an initial guess a good one by using a hierarchical spatial subdivision.

a) Hierarchical Spatial Subdivision on Edge

To place bubbles on edge with a curve \( C(s) \) defined in the parameter range \( s_1 < s < s_2 \), the algorithm first places bubbles at the two end points \( p_1 = C(s_1) \) and \( p_2 = C(s_2) \) as shown in Fig. 4.2.

![Fig. 4.2 Spatial Subdivision on Edge](image-url)

The radii of these end bubbles are denoted as \( r_1 \) and \( r_2 \) respectively. The distance in object space between the two end points \( p_1 \) and \( p_2 \) is then compared with the sum of the two radii, \( (r_1 + r_2) \). Two cases exist. In Case 1, shown in Fig. 4.2 (a), the distance \( p_1 \) to \( p_2 \) is less than or equal to \( (r_1 + r_2) \), meaning that the two end bubbles approximately cover the curve segment (Edge), and thus no further subdivision is required. In Case 2 shown in Fig. 4.2 (b), the distance \( p_1 \) to \( p_2 \) is larger than \( (r_1 + r_2) \), indicating that further subdivision is required and a new bubble is then placed at the midpoint of the range \( s_1 < s < s_2 \). The range is then subdivided into two sub-ranges \( s_1 < s < (s_1 + s_2)/2 \) and \( (s_1 + s_2)/2 < s < s_2 \) on which the same distance check is performed, and another bubble is added at the midpoint if necessary. This process is repeated recursively until the whole curve is covered by bubbles.
b) Hierarchical Spatial Subdivision on Face

After bubbles have been placed on all curves, bubble placement on the surface $S(u, v)$ is performed by using a rhombic quadtree. The face (Trimmed surface region) in parametric space must be entirely enclosed in an initial rhombic cell.

![Spatial Subdivision Cases on Face](image)

Let $l$ be the edge length, and $(u, v)$ the center of a rhombic cell. The center mapped in object space is denoted by $p_0 = S(u, v)$.

![Spatial Subdivision on Face](image)

The coordinates of the four corners can be calculated as shown in Fig. 4.5. The coordinates of the four corners in parametric space are

$$(u_1, v_1) = (u - 3l/4, v - \sqrt{3}l/4)$$

$$(u_2, v_2) = (u + 3l/4, v - \sqrt{3}l/4)$$

$$(u_3, v_3) = (u + 3l/4, v + \sqrt{3}l/4)$$

$$(u_4, v_4) = (u - 3l/4, v + \sqrt{3}l/4)$$

For these corners, the corresponding points mapped into object space are denoted by $p_1, p_2, p_3$, and $p_4$ respectively.
First, bubbles are placed at the four corners, and their radii are denoted by $r_1$, $r_2$, $r_3$, and $r_4$ respectively. To judge whether the further subdivision is required, the sum of the radii of two adjacent bubbles is compared with the distance between the bubbles centers. There are two cases, as shown in Fig. 4.6.

In case 1, the sum of the radii is larger than the distance between the centres, namely, where

$$r_i + r_j \geq \overline{p_ip_i} \quad \forall \ (i, j) \in \{(1,2), (2,3), (3,4), (4,1)\}$$

Eq. 4.1 Criteria to stop Spatial Subdivision on Face

If the above equation is not satisfied, Case 2 applies. In Case 1, the four bubbles at the corners of the rhombus approximately cover the region, so the process is stopped. In Case 2, the region is further subdivided into four sub-cells, whose centres are

$$(u_{01}, v_{01}) = (u - 3l/8, v - \sqrt{3}l/8)$$
$$(u_{02}, v_{02}) = (u + l/8, v - \sqrt{3}l/8)$$
$$(u_{03}, v_{03}) = (u + 3l/8, v + \sqrt{3}l/8)$$
$$(u_{04}, v_{04}) = (u - l/8, v + \sqrt{3}l/8)$$

Four additional bubbles are then placed at the midpoints of all the edges of the original rhombus:

$$(u_5, v_5) = (u - l/4, v - \sqrt{3}l/4)$$
$$(u_6, v_6) = (u + l/2, v)$$
$$(u_7, v_7) = (u + l/4, v + \sqrt{3}l/4)$$
$$(u_8, v_8) = (u - l/2, v)$$

which are mapped to points in object space, denoted as $p_5, p_6, p_7,$ and $p_8$ respectively, as shown in Fig. 4.3. A fifth bubble is placed at the center of the original rhombus $p_0$. The same distance-checking procedure is then repeated for four sub-cells, $p_1p_5p_0p_8, p_5p_2p_6p_0, p_0p_6p_3p_7,$ and $p_8p_0p_7p_4,$ recursively until bubbles are distributed over the whole surface without significant gaps and overlaps. Note that in Eq. 4.1 we don’t check for the diagonally opposite bubbles, even if they overlap we must subdivide the cell till the equation is not satisfied.

During the above quadtree building process, rhombic cells are classified into three categories: (1) **IN**, where the cell is inside the geometry; (2) **OUT**, where the cell is outside the geometry; and (3) **BOUNDARY**, where the cell overlaps the boundary and is therefore partly inside and partly outside the geometry. Because only bubbles inside the geometry need be considered,
once a rhombic 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-cells created by subdividing this cell will be IN, and bubbles must be placed on all corners. Every time a BOUNDARY cell is subdivided, an in-out check is required for each of the four sub-cells - they can be either IN, OUT, or BOUNDARY.

4.1.2 Adaptive Bubble Population Control

Adaptive bubble population control is an automatic feature implemented to remove an Excess Bubble and to add bubbles around an Open Bubble. As discussed previously under the topic Bubble Overlap Ratio, the overlap ratio for each bubble is calculated, and each bubble is classified as Stable Bubble, Excess Bubble, or Open Bubble. In the original paper [13], a bubble on a face classified as an excess bubble if its overlap ratio is greater than 8.0 as shown in Fig. 3.7. It is classified as an open bubble if its overlap ratio is less than 5.0 shown in Fig. 3.8. If the bubble is an open bubble, it does not have enough neighbours, and therefore another bubble should be added to fill the gap. Similarly, if the bubble is an excess bubble, the bubble is superfluous and should be erased.

Using adaptive population control method, a given region is always filled up with a necessary and sufficient number of bubbles, without significant gaps or overlaps, even in continuous remeshing and local remeshing. The method also enables the system to reach equilibrium more quickly, avoiding the slow natural dispersion of bubbles from a packed area to a sparse area.

4.1.3 Physically Based Relaxation

Given the inter-bubble 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. 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 the approach in (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.

We can imagine each bubble as a spring placed in a viscous medium. Then the whole system can be represented as a mass-spring damper (MSD) system. Hence the motion of each bubble can be given by the solution of Eq. 2.2, and we can set up a system of 1st order differential equations (Eq. 2.4) which can be numerically solved by fourth-Order Runge-Kutta Methods. Physically Based Relaxation is explained in detail in section 2.3.
4.2 Additional Work

When implementing bubble packing algorithm [13] described in the previous section, we discovered several details that were not explicitly stated in Shimada’s thesis, but which are necessary for a complete implementation. These additional specifications are described below.

4.2.1 A layer of outside Bubble

The number of bubbles around a bubble define the overlap ratio for that bubble which in turn defines whether that bubble should be removed, or a new bubble must be created around. This works well for a bubble which is surrounded by the bubbles that lie on the face, but when a bubble lies near a boundary, then bubble may get marked as an open bubble even if it does not need any new bubble, as shown in Fig. 4.5.

![Fig. 4.7 Open Bubble near the boundary](image)

This case sometimes gets resolved when bubbles on edge move close so that the open bubble has enough overlap that it will be marked as stable. However, it is possible this resolution does not occur. But we can easily avoid this case if we can generate a layer of outside bubbles around the border of the face. These outside bubbles will not influence the motion of any bubble which is part of our active system, i.e. bubbles on the face and edge. They can only be used to calculate the overlap.

![Fig. 4.8 Layer of outside Bubble](image)

Sometimes when the underlying surface is not defined outside the boundary of the face, it may happen that we can’t generate outside bubble layer. The algorithm can still work, but since the bubble near boundary will be marked as an open bubble, new bubble insertion part of the algorithm will try to insert a new bubble and fail every time. We can ignore the bubble where bubble insertion algorithm fails and look for the next possible candidate for the open bubble.
4.2.2 Active and Passive Nodes

As described previously, there are some nodes which take part in both Physically Based Relaxation and overlap ratio calculation, and some nodes take part in just overlap ratio calculation hence we must define a relationship between nodes. Node relations between nodes are by the location of the nodes in domain, i.e. vertex node, edge node, face node or outside node. We divide nodes into three types of relations active, only overlap and passive.

- **Active Node Relation**: When for a given node if a neighbouring node is considered for both relaxation and overlap calculation, then given node has an “active relation” with that neighbour node. Hence that neighbour node will be placed in a list of active neighbours in inside the given node.

- **Only Overlap Relation**: When for a given node if a neighbouring node is only considered for overlap calculation, then given node has “only overlap relation” with that neighbour node. Hence that neighbour node will be placed in a list of overlapping neighbours in inside the given node example a layer of outside bubble described above.

- **Passive Node Relation**: When for a given node if a neighbouring node is not considered for both relaxation and overlap calculation, but if that neighbour node has an “active relation” or “only overlap relation” with a given node, then given node has a “passive relation” with that neighbour node. Hence that neighbour node will be placed in a list of passive neighbours in inside the given node. Passive list is important to maintaining for backtracking as in some step if a given node gets deleted we must remove that node form all its neighbours’ neighbour list.

![Fig. 4.9 Node relations](image)

a) Node relations For Vertex Node

Since there should be a node at every vertex, we make sure that vertex node doesn’t move. This is done by making all neighbouring nodes of the vertex node passive neighbours so that there will not be any force on the vertex node. (Fig. 4.9)
b) Node relations For Edge Node

Since a node at an edge should always be on edge, to make sure that an edge node moves only on edge, all face nodes neighbouring the edge node must be passive neighbours, and all edge and vertex nodes must be active nodes (Fig. 4.9). However, this condition alone doesn’t restrict node to the edge. After each displacement, the edge node must be pulled back to the edge which can be done as explained in section 3.4.1 (Relative displacement for Edge Motion).

c) Node relations For Face Node

All nodes on a face should always move freely on the face. Hence all neighbouring nodes of a face node must be active neighbours except the outside nodes since they are only used for overlap-ratio calculation as explained in section 4.2.1 (Fig. 4.9). However, this alone does not restrict a node to the face after each displacement; a face node must be pulled back to the face which can be done as explained in section 3.4.2 (Relative displacement on Surface Motion).

4.2.3 Removing a Bubble

Removing an excess bubble causes a change in overlap ratio for all its neighbour. Hence, sometimes a neighbouring bubble which is an excess bubble at first can become a stable bubble or a stable bubble can become an open bubble. As a result, removing an excess bubble at random may cause more than the required bubble to be removed as shown in Fig. 4.10. A better approach is to remove the bubble with the highest overlap ratio first, and then to update the overlap ratio of all its neighbours. Since we have a nearly constant number of neighbours to a bubble, this deletion and updating process will have constant time complexity.

![Fig. 4.10 Removing Nodes](image-url)
4.2.4 Creating a new Bubble

Creating a new bubble is a critical part of the bubble packing algorithm. A new bubble at the wrong place can make a bubble packing system unstable. If we can generate a bubble at the best possible location, we can drastically reduce the number of iterations needed to reach a stable state. Most of the new bubble creation happens in the first few iterations and after that the new bubble creation algorithm is rarely called. The total number of new bubble creations across the process depends on the quality of our initial bubble configuration. The better the quality of the initial placement, the fewer the number of new bubble creation. Since the total number of new bubble creation calls are not that large, we can afford to expend the time required to find the best possible placement for a new bubble around an open bubble. But at the same time, we must be careful that we do not spend unnecessary time finding the location because the subsequent force balancing step can adjust the bubble automatically.

a) New bubble on Edge

An open bubble on an edge occurs when there is a vacancy for a new bubble on one or both sides of the bubble. Since an edge is always continuous and has direction (direction of edge usually gets defined by the variation of a parameter on edge), an open bubble divides an edge into two regions. We refer to these regions as “+” and “−” with respect to the open bubble, and we can add new bubbles on either side (Fig. 4.11). All calculations are done in parametric and object space simultaneously, using the method (Relative displacement for Edge Motion) explained in section 3.4.1.

![Possible new bubbles location on the edge](image)

Fig. 4.11 Possible new bubbles location on the edge

Now the decision on where to place a new bubble depends on the location of already existing neighbours. Since we maintain the neighbour list for a node as per the region of influence, there can be three possible cases:

- **No neighbour exists on either sides of the open bubble.** In this case, we can create a new bubble on either side of the open node at a distance twice the radius of the open bubble (Fig. 4.12). This can be done by finding the Relative displacement for Edge Motion explained in section 3.4.1. The new bubble may or may not be overlapping the open bubble since the bubble radius at this location may be different from that of the open bubble.
• **One neighbour exists on exactly one side of the open bubble.** In this case, we can create a new bubble on the side without a neighbour at a distance twice the radius of the open bubble (Fig. 4.13).

![Fig. 4.13 Open Bubble on edge with one existing neighbour](image)

• **Both neighbours exist.** In this case, we first get the parameter of each neighbour from the open bubble and mark them as $t^{-}$ and $t^{+}$. Then we find the parameter for half of the parametric distance $t^{-}/2$ and $t^{+}/2$. Using $t^{-}/2$ and $t^{+}/2$ as the parametric values, we calculate possible bubble locations $p^{-}$ and $p^{+}$, respectively. Now the position ($p^{-}$ or $p^{+}$) that is farthest from the open bubble will be accepted as a valid position for a new bubble.

![Fig. 4.14 Open Bubble on edge with both neighbours](image)

**b) New bubble on Face**

Creating a new bubble around an open bubble on a face is a bit more involved than placing new bubble on an edge. The number of neighbours for a face bubble can vary from 0 to $n$ where the value of $n$ depends on the radii of the bubbles that are neighbours of the given face bubble. But, usually, the neighbouring bubbles have radii that are approximately equal to that of the given bubble and hence $n$ generally has a value around 6. All the calculations for the face are done in parametric space with two assumption:

- the given surface will not have any high sporadic variation in object space for a given small parametric space.
• the relative displacement between object space and parametric space are comparably similar.

In a manner similar to the case of a bubble on an edge, there can be three possible cases:

• **No neighbour exists.** In this case, we first decide on a displacement factor $k$ (in our case $k = 1.8$). Then we calculate a relative displacement in parametric space (explained in section 3.4.2) $\Delta u$ and $\Delta v$ which is equivalent to the displacement of $kr$ in $t_u$ and $t_v$ in object space respectively. Using $\Delta u$ and $\Delta v$, we can have 8 simple new possible locations for the bubble at parameters (Fig. 4.15):

$$(u + \Delta u, v + \Delta v) \quad (u - \Delta u, v - \Delta v) \quad (u + \Delta u, v - \Delta v) \quad (u - \Delta u, v) \\
(u, v + \Delta v) \quad (u, v - \Delta v) \quad (u + \Delta u, v) \quad (u - \Delta u, v)$$

We pick the first location by checking if each one lies on the face (if a face has a hole any of the above parameters may lie in the hole or if the given bubble is near a boundary, one or more location may lie outside). If none of the above locations lie on the face, we can create more locations using some factor and a combination of $\Delta u$ and $\Delta v$ or we can skip the node as having failed new bubble insertion.

![Fig. 4.15 Open Bubble on the face with no existing neighbour](image)

• **One neighbour exists.** In this case, we can find $\Delta u$ and $\Delta v$ by subtracting the $(u, v)$ location of the open bubble from the parametric location of neighbour bubble $(u', v')$. After this, we can apply a method similar to the one discussed in above case, while avoiding the location of the already existing neighbour (Fig. 4.16).

![Fig. 4.16 Open Bubble on the face with one existing neighbour](image)
**More than one neighbour exists.** In this case, we pick one of the neighbours as reference neighbour and order all neighbours counterclockwise in parametric space. Once ordered we find the angle between adjacent neighbours (Fig. 4.17).

![Open Bubble on the face with multiple existing neighbours](image)

There are two possible cases:

- The angle between two adjacent neighbours is less than $\frac{\pi}{2}$. Using the centres of these two neighbours and the open bubble, we create a triangle and pick centroid as a possible candidate for the new bubble location and push that location into the possible candidate list.

![possible candidate if the angle between neighbours is less than $\pi/2$](image)

- The angle between two adjacent neighbours is greater than or equal $\frac{\pi}{2}$. Let $\theta$ the angle is between the neighbours. We first decide on a displacement factor $k$ (in our case $k = 1.8$). Then we calculate a relative displacement in parametric space (explained in section 3.4.2), $\Delta u$ and $\Delta v$, which is equivalent to the displacement of $kr$ in $t_u$ and $t_v$ in object space respectively. Then we find a direction $\vec{d}$ that is angle bisector for the two neighbours. Using direction $\vec{d}$ and $(\Delta u, \Delta v)$ we find a new location. This location will be a possible candidate for the new bubble location and is pushed into the possible candidate list (Fig. 4.19). Once we collect all possible candidate locations, we find an overlap ratio for each point. We pick the candidate with the minimum overlap ratio as the final location for a new position. Note that all the candidates must be on the face.
4.2.5 Order of Population Control

Algorithms for adaptive population control affect the stability of the system, for bubble removal as well as addition. A mass-spring-damper system of a constant configuration of bubbles will always guarantee a steady state (an equilibrium state), and therefore a guaranteed output after some iterations of force balancing. But whenever we remove or add bubbles in the system, it attains a new configuration. As a result, the steady state of the system changes; in other words, the whole force balancing process restarts. Hence, the greater the number of bubble insertion and removals the more unstable the system becomes. To reduce the number of insertion and removal of bubbles, it is necessary that we start with a good configuration. However, even if we start with a good configuration sometimes the order of insertion and removal can itself cause some unnecessary bubble insertions and removals. For example, as discussed above, the overlap ratio of a bubble on an edge depends only on edge and vertex bubbles, whereas the edge bubble is considered for overlap ratio calculation for a bubble on a face. Since the cut-off for the openness of both types of bubbles is different; we may have two open bubbles nearby, one on edge and one on the face (Fig. 4.20).

Now if the face bubble gets considered first, it will cause a new bubble insertion on the face, but this new bubble will not affect the overlap state of the edge bubble. The edge bubble is still an open bubble and hence will try to create a bubble on the edge. This new edge bubble will affect the overlap ratio of the bubble created by the open face bubble, making the new face bubble an excess bubble, which will now need an unnecessary removal. This whole situation can be

\[
lu = \left( u + d_u \Delta u, v + d_v \Delta v \right)
\]
avoided if we insert a bubble on an edge before we handle any face bubbles. The order of removal of the bubble can avoid any unnecessary population control steps. So, the order of population controls is given below:

1. Remove any excess bubbles on the edges and update the overlap ratios table.
2. Remove any excess bubbles on the faces and update the overlap ratios table.
3. Insert bubbles around any open bubbles on the edges and update the overlap ratios table.
4. Insert bubbles around any open bubbles on the faces and update the overlap ratios table.
5. Remove any excess bubbles on the edges caused by the insertion step and update the overlap ratios table.
6. Remove any excess bubbles on the faces caused by the insertion step and update the overlap ratios table.

In these steps, it is observed that the removal of excess bubbles can make some excess bubbles become into stable bubbles. At the same time, in some cases, insertion of a new bubble can cause a stable bubble to change state and become an excess bubble. Therefore, we perform removal, insertion and one more removal after insertion.

4.2.6 Force Balancing in between Population Control

When a new bubble is inserted into the system, the location of the new bubble is calculated purely by geometric operations done on the location of open bubble that caused the insertion and locations of the open bubble’s neighbours. Insertion algorithm never considers the forces applied on the open bubble and its old neighbours. In some cases, this newly inserted bubble can immediately become an excess bubble. Consequently, this bubble get removed in the next step, leaving the system in the same state as it was prior to the population control. This immediate conversion of excess bubbles can make system enter an endless loop where the same bubble is getting inserted and removed. This scenario occurs because sometimes one iteration is not enough to cause a significant change in bubble location. If we add a few iterations between two consecutive population control steps, we can avoid this scenario. Allowing several iterations of bubble placement between two consecutive population control step better ensures that the system configuration differs from one step to the next. In our case, we allow 20 iterations between two consecutive population control steps.
Chapter 5 Implementation Details

This chapter provides details about the implementation of the bubble packing algorithm and consolidates the explanation of the data flow in the algorithm.

5.1 Expected Input

The input to the algorithm is a B-Rep (explained in section 2.2) of surface patches. Surface patches are defined as faces in a B-Rep, which can be a part of a Solid Body or a Sheet Body. A domain to be meshed consists of either one Face or multiple Faces.

The algorithm expects the following as input:

- A face as defined by an underlying Surface (Eq. 3.2) and loops of trimming Curves (Eq. 3.4)
- At least one exterior (outer) Loop \( (l_o) \) and any number of interior (inner) Loops \( (l_i) \) (as explained earlier a face has two type of loops).
- At most two end vertices bounding each edge, also denoted by \( V = (x, y, z) \).
- All curves and surfaces are continuous, and differentiable (at least twice) within the given domain.
- Node Spacing Function, to provide a radius of a bubble at a given location in the domain. Node spacing functions are defined as:
  - For Face: \( r = \phi_f(f, u, v) \) where \( f \) is a face and \((u, v)\) are the parameters for the underlying surface of the face.
  - For Edge: \( r = \phi_e(e, s) \) where \( e \) is an edge and \( s \) is the parameter for the underlying curve.
  - For Vertex: \( r = \phi_v(v) \) where \( v \) is a vertex.

All the inputs to the algorithm are provided by the interfaces to the entities, so the algorithm doesn’t store any unnecessary information and the caller of the algorithm can optimise the calling as needed.

5.2 Node (Bubble) Data Structure

The data structure used to store node (or bubbles) is the most critical one for the algorithm. Due to a high number of bubbles in the system, we can’t overload bubbles with data. At the same time, we must optimise the data on bubbles for both speed and memory in order to improve performance. Bubble data is divided into two parts: Bubble Properties and Area of Influence.

5.2.1 Bubble Properties

Bubble properties are the data values that belong purely to a bubble and define the identity of a bubble.
• **Position**: Location of the bubble in Euclidean space defined by $(x, y, z)$ coordinates *(Memory footprint of 24 Bytes as double precision)*.

• **Radius**: Bubble radius at the position used in force balancing. *(Memory footprint of 4 Bytes as single precision)*

• **Mass**: Bubble mass at the position used in force balancing. *(Memory footprint of 4 Bytes as single precision)*

• **Flags**: A 16 bit array of true/false flags to hold additional information required by the algorithm *(Memory footprint of 2 Bytes)*. Current flags used are:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outside</td>
<td>True</td>
<td>The bubble is not inside the face boundaries.</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>The bubble is valid and part of the solution.</td>
</tr>
<tr>
<td>Vertex</td>
<td>True</td>
<td>The bubble lies on a vertex.</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>The bubble does not lie on a vertex.</td>
</tr>
<tr>
<td>Edge</td>
<td>True</td>
<td>The bubble lies on an edge.</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>The bubble does not lie on an edge.</td>
</tr>
<tr>
<td>Face</td>
<td>True</td>
<td>The bubble lies on a face.</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>The bubble does not lie on a face.</td>
</tr>
<tr>
<td>Crowded</td>
<td>True</td>
<td>The bubble is an excess bubble.</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>The bubble overlap ratio is under excess bubble cut-off.</td>
</tr>
<tr>
<td>Deserted</td>
<td>True</td>
<td>The bubble is an open bubble.</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>The bubble overlap ratio is greater than open bubble cut-off.</td>
</tr>
<tr>
<td>New Bubble</td>
<td>True</td>
<td>The bubble is created by pervious (or current) bubble insertion operation under population control. This flag gets reset after every population control step.</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>The bubble is an old bubble.</td>
</tr>
<tr>
<td>Invalid</td>
<td>True</td>
<td>The bubble is marked for deletion.</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>The bubble is valid.</td>
</tr>
</tbody>
</table>

**5.2.2 Area of Influence (AOI)**

Area of Influence (AOI) stores data about the bubble’s relation with neighbouring bubbles. AOI stores three lists of bubbles: Active, Passive and Only Overlap neighbour bubbles (explained in section 4.2.2). AOI also stores a reference to parameter object. Parameter object stores following data:

• **Influence Radius**: to define the region around the bubble. Any other bubble whose influence region overlaps with the given bubble’s influence region will be considered to be a neighbour of the as given bubble.

• **Overlap Radius**: to be used in overlap calculation for the bubble.
• **Superfluous Cut-off**: to be used to determine if the bubble becomes an excess bubble.

• **Deficiency Cut-off**: to be used to determine if the bubble becomes an open bubble.

• **Spring Constant**: to be used in calculating spring force between the bubble and its neighbour.

• **Damping Coefficient**: to be used in calculating the damping force on the bubble.

• **Tolerances**: tolerances used for unit and unitless calculations.

### 5.3 Mapping B-Rep entity and Bubbles

As explained above, whenever we update a bubble’s position, the algorithm has to query the new radius for the bubble. In addition, to check if the new location of the bubble lies on the face or not, the algorithm must check the state of the new locations with the face (Similarly for edge and vertex). Storing the reference of the B-Rep entities within each bubble will unnecessarily increase the memory footprint because the number of B-Rep entities is significantly less than the number of bubbles. Similarly, it is an unnecessary increase in memory footprint if we store a parametric value corresponding to a bubble position on the B-Rep entity. Moreover, most of the time a bubble is a part of a single B-Rep entity, but a bubble can be a part of multiple faces as well as multiple edges at the same time (for example a bubble at the vertex). Hence, for a given bubble we need a relation between each parameter, and B-Rep entity to which it belongs. The B-Rep entity manager usually provides a way to query the parameter of a given position on an entity at runtime, but this process is not efficient, and leads to a major performance hit.

To avoid all this, we maintain a hash table of entities ($T_e$) in which each entity is mapped to another hash table ($T_b$). The table $T_b$ stores the bubbles on the corresponding entity. In addition, in table $T_b$ each bubble is mapped to the parametric value of the bubble on the corresponding entity (shown in Fig. 5.1).

![Fig. 5.1 Mapping B-Rep entity and Bubbles](image-url)
By maintaining these hash tables, the algorithm can make many queries. Some examples are:

- **To find the entities on which the bubble lies**: Linear search on all entities and test if the bubble exists in the bubble’s hash map. If yes, the bubble exists on that entity. We can do a linear search as we have very few entities. To refine the search, we use the flag stored on the bubble to check the entity type first.

- **To find the parametric value of the bubble on the given entity**: Look for the entity in the first hash table to get the bubble’s hash table. Look for the bubble in the bubble hash and find the corresponding parameter.

### 5.4 Population Control

To run population control, we must calculate the overlap ratio (explained in section 3.1.3). After calculating the overlap ratio, we maintain two hash tables: one for bubbles which are excess bubbles and one for bubbles which are marked as open bubble this take $O(n)$ time. We then run the procedure to remove excess bubbles which may create new open bubbles, so we must update the open bubble table. The algorithm tries to create new bubbles near the open bubbles, which may cause some of the old bubbles to become excess bubbles. Hence, we must update our excess bubbles table and run the procedure to remove excess bubbles once more.

The algorithm is as follows:

```plaintext
populationControl(allBubbles):
    excessBubbles = HashTable(
    openBubbles = HashTable(
        foreach bubble in allBubbles:
            [ratio, state] = findOverlapAndState(bubble)
            if state == Excess then excessBubbles[bubble] = ratio
            else if state == Open then openBubbles[bubble] = ratio
        end
        deleteExcessBubbles(excessBubbles, openBubbles)
        addBubbleAtOpenBubbles(openBubbles, excessBubbles)
    deleteExcessBubbles(excessBubbles, openBubbles)

P. Code 5.1 Population Control
```

#### 5.4.1 Removal of excess bubbles

To remove an excess bubble, we must find the bubble with the highest overlap ratio separately for each face, edge and vertex. We first remove the vertex excess bubble, then the one for an edge and finally the face excess bubble (explained in section 4.2.5). Before deleting the bubble with the highest overlap, we mark all its neighbours, as removal of the bubble can only affect the overlap ratio of the neighbours. Once we remove the bubble, we must recalculate the overlap ratio for all the marked bubbles. We then update the hash tables for excess as well as open bubbles, since the removal of an excess bubble may change the status of a bubble from excess to normal or from normal to open. The algorithm will look like:
5.4.2 New Bubble around Open Bubble

To add a bubble around an open bubble, we find the bubble with the lowest overlap ratio separately for each face, edge, and vertex. A new bubble is inserted first for a vertex then an edge and finally a face (explained in section 4.2.5). After adding the new bubble, we mark all the neighbours for the new bubble and the bubble causing the addition of the new bubble. The addition of a new bubble can only affect the overlap ratio of its neighbours. Once we add the bubble, we must recalculate the overlap ratio and update the hash tables for excess bubbles as well as open bubbles. Adding a bubble can turn some bubbles into excess bubble. The algorithm is as follows:

```
addBubbleAtOpenBubbles(openBubbles, excessBubbles):
    while (openBubbles):
        openBubble = findMostDeserted(openBubbles)
        if isVertexBubble(openBubble) then newBubble = newBubbleAtVertex(openBubble)
        else if isEdgeBubble(openBubble) then newBubble = newBubbleAtEdge(openBubble)
        else if isFaceBubble(openBubble) then newBubble = newBubbleAtFace(openBubble)
        end
        if newBubble == null then continue
        end
        updateNeighbours(newBubble)
        bubblesToCheck = neighbours(newBubble)
        bubblesToCheck.append(newBubble)
        openBubbles = erase(bubblesToCheck)
        foreach bubble in bubblesToCheck:
            [ratio, state] = findOverlapAndState(bubble)
            if state == Excess then excessBubbles[bubble] = ratio
            else if state == Open then openBubbles[bubble] = ratio
            end
        end
    end
```

P. Code 5.3 New Bubble around Open Bubble
5.5 Force Balancing

The algorithm applies a 4th order Runge-Kutta method in order to do the force balancing in the MSD system (explained in section 2.3). The whole process of force balancing is divided into three phases, described below:

5.5.1 Initial Data \((Y_0)\)

Before we start the iteration under 4th order Runge-Kutta, we must create the initial boundary condition denoted as \(Y_0\) at \(t_0\), where \(Y_0\) is a continuous array of initial locations \(\vec{x} = (x, y, z)\) and initial velocities \(d\vec{x}/dt = \vec{\dot{x}} = (\dot{x}, \dot{y}, \dot{z})\) for all bubbles as shown in Fig. 5.2. Since, all the bubbles are stationary at the initial boundary condition, velocity is zero i.e. \(\dot{x} = 0\) for all bubbles.

![Fig. 5.2 Initial Data \((Y_0)\)](image)

5.5.2 Finding first-derivate \((dY/dt)\) for given data \((Y)\)

The first derivative of \(Y\) is a continuous array of all velocity \(\dot{\vec{x}} = (\dot{x}, \dot{y}, \dot{z})\) and acceleration \(d^2\vec{x}/dt^2 = \ddot{\vec{x}} = (\ddot{x}, \ddot{y}, \ddot{z})\) values for all bubbles, as shown in Fig. 5.3.

![Fig. 5.3 First-derivate \((dY/dt)\) for given data \((Y)\)](image)

Since the latter half of our data in \(Y\) contains the velocities \(\dot{\vec{x}}\) of the bubbles, we can copy the last half of \(Y\) in the first half of \(\dot{Y}\). The second half of \(\dot{Y}\) represents the acceleration of each bubble, which can be calculated by using Eq. 2.2 as

\[
\frac{d^2\vec{x}}{dt^2} = -\frac{1}{m} \left( c \cdot \frac{d\vec{x}}{dt} + k \cdot \vec{x} \right)
\]

Eq. 5.1 Acceleration of each bubble

In Eq. 5.1, \(c \cdot \frac{d\vec{x}}{dt}\) represent the force exerted on the bubble due to the velocity of the bubble, known as the damping force and \(k \cdot \vec{x}\) is the cumulative force exerted on the bubble by the neighbouring bubbles, known as spring force. Finding the damping force is simple as we can
get the velocity of the bubble from \( Y \) and the damping coefficient of the medium. Finding the spring force is a bit involved since the total spring force depends on the bubble configuration around the given bubble. The algorithm first finds the force on the bubble caused by one bubble-neighbour pair, and then adds all these forces to find the cumulative force on the bubble. In each bubble-neighbour pair, the neighbouring bubble is considered as static or instantaneously rigid as shown in Fig. 5.4. Once we have both the damping and spring forces, we add them and divide the net force by the mass of the bubble to find the acceleration of the bubble. After calculating the acceleration of each bubble, we fill the second half of \( \dot{Y} \).

![Fig. 5.4 Finding the total spring force on a bubble](image)

5.5.3 Update bubbles position by the delta of data (\( \Delta Y \))

The last question to be addressed for force balancing concerns the confinement of bubbles to curves and surfaces. In general, bubble locations calculated by numerically integrating the equation of motion are not geometrically compatible; that is, bubbles do not lie on the curves and surfaces to be meshed. In each time step, therefore, the bubble movement must be corrected so that all the bubbles lie exactly on the target geometries.

Let \( Y_i \) be the solution of the system at \( t_i \) i.e. after the \( i^{th} \) iteration. Similarly, \( Y_{i-1} \) is the solution of the system at \( t_{i-1} \). The constrained delta movement of each bubbles we can be given by \( \Delta Y = Y_i - Y_{i-1} \). Since \( \Delta Y \) is an unconstrained movement of bubbles in object space, we cannot add the \( \Delta Y \) directly to each bubble to find the next position as it will cause the bubbles to move away from the geometry. To confine bubbles to a curve, we can apply the method explained in 3.4.1. Similarly, to confine the bubble to a surface, we can apply the method explained in 3.4.2. After calculating the corrected location of each bubble, the algorithm will update the location of each bubble as well as the bubble location data in \( Y_i \). The corrected \( Y_i \) will be passed as initial condition to the next iteration of force balancing.

Since we are correcting the bubble location at every iteration, our system of bubbles may never reach a state where the net force on each bubble in the system becomes zero. In other words, the algorithm cannot create a stopping criterion as \( \vec{F} = \vec{0} \) (equilibrium state) to stop the
iterations. To stop the force balancing iterations, the algorithm must create some new stopping criteria, which are explained in next section.

5.6 Stopping Criteria

A simple stopping criterion might be to utilize the bubble motion itself; that is, if the sum of the magnitude of displacement falls below a certain threshold for each bubble, the algorithm stops. However, in practices, it takes a significantly high number of iterations for magnitude of displacement of each bubble in the system to fall below the threshold. Whereas, due to the nature of an underdamped MSD system most of the high magnitude movements take place in initial part of the iterations. After that the system slows down and overshoots few times before it reaches the stable state. Since our goal is not so much to achieve a stable state, but more to reach a state where bubbles are near their final position as fast as possible. This part of the algorithm still needs more work. Our current implementation of the algorithm uses the standard deviation of the displacement. The algorithm assumes that the mean displacement of the system is zero and if the last five standard deviations are below the tolerance; the force balancing iterations stop. In section 6.3 we will analyse few more viable stopping criteria.
Chapter 6 Results and Analysis

This chapter presents some of the outputs that were produced by the bubble packing algorithm as well as the analysis of the results. It also provides a brief introduction to some of the basic quality parameters, to help in understanding and quantifying the result produced by the bubble packing algorithm.

6.1 Quality parameters

As discussed in Section 2.1, there are many quality parameters to quantify the overall mesh quality. In general, Mesh algorithms are designed to generate meshes according to the type of simulation. Hence, it’s the type of simulation that defines the quality parameters for the mesh. Since we are not designing a mesher for a particular type of simulation, we will use three general quality parameters for our meshes: mesh irregularity, aspect ratio based on inradius to circumradius ratio, and the triangle count within min and max angle ranges.

6.1.1 Mesh Irregularity

For quantifying mesh quality in a surface triangulation, we define the following measure of mesh irregularity (\(\varepsilon\)), similar to that defined by Frey and Field [14].

\[\varepsilon = \frac{1}{n} \sum_{i=0}^{n} |\delta_i - 6|\]

Eq. 6.1 Mesh Irregularity

where \(\delta_i\) represents the degree, or the number of neighbouring nodes, connected to the \(i^{th}\) interior node, and \(n\) the total number of interior nodes in the domain.

Thus, in general, as triangles become more equilateral, the mesh irregularity approaches 0 (Fig. 6.1) but will vanish only when all the nodes have six neighbours, a rare situation. Otherwise, it has a positive value that designates how much the mesh differs from a perfect hexagonal lattice. An advantage of the mesh irregularity quality parameter is that we don’t need a triangulated mesh; we can directly use the neighbour information stored in bubbles to calculated mesh irregularity. Therefore, mesh irregularity can be calculated easily while iterating through the mesh.
6.1.2 Aspect Ratio

There are multiple ways to define the aspect ratio of a triangle for example, as the ratio of smallest edge length to the largest edge length of a triangle. For our algorithm, we focus on the aspect ratio \( (ar) \) calculated by the inradius and circumradius i.e.

\[
ar = \frac{2 \times r_i}{r_c}
\]

Eq. 6.2 Aspect ratio

where \( r_i \) is the inradius and \( r_c \) is the circumradius of the triangle. As triangles become more equilateral, the aspect ratio approaches 1.

\[
\angle A = \angle B = \angle C = 60^\circ \quad ar = 1.0
\]

\[
\angle A = 170^\circ \quad ar = 0.016
\]

\[
\angle A = 90^\circ, \angle B = 85^\circ \quad ar = 0.16
\]

Fig. 6.2 Aspect ratio of a triangle

6.1.3 Min and Max angle ranges

In many applications, a simulation requires a mesh that is primarily made of equilateral triangles. A good mesh will have a very high percentage of triangles that have a min angle range between 50° to 60° and the max angle between 60° to 70°. A problem with the quality parameters discussed in the previous section is that, they depend so much on the mathematical interpretation of the shape of a triangle that few bad triangles can skew the overall quality parameter of mesh. Moreover, sometimes these quality parameters can tell you how good a triangle is but can not accurately capture the information about how bad the triangle is. For example, the aspect ratio of a triangle with angles 90°, 85° and 5° (Fig. 6.2 right) is equal to the aspect ratio of a triangle with angles 24.175°, 24.175° and 131.65°. While neither triangle is a good triangle, the first triangle is much worse as it has close to zero area. Hence, it is a good idea to keep track of the count of triangles under different ranges of min and max angle.

6.2 Outputs

In this section, we show the output generated by our algorithm for mass \( (m) \) 1.0, spring constant \( (k) \) value 30, and damping ratio \( (\xi) \) 0.7. First column of output shows the mesh generated using the initial bubble position (without force balancing). The first column shows the starting mesh and its quality parameters (as explained in the previous section). The second column shows the mesh generated by the bubble with minimum mesh irregularity achieved during the run, along with the quality parameter for that mesh. Output 4, shows a mesh generated with a node spacing function \( r = c_4/(c_2 + c_3 \cdot \sin(c_4 + c_5 \cdot x \cdot y)) \) where \( c_i \) are scaler constants. Output 5, show a starting mesh with good quality initial bubble position; hence the number of iterations to achieve minimum mesh irregularity was very low.
6.2.1 Output 1

Starting with mesh irregularity $0.96875$ after 245 iteration mesh irregularity $0.5565217$.
6.2.2 Output 2

Starting with mesh irregularity 0.6997167 after 271 iteration mesh irregularity 0.4574132.
6.2.3 Output 3

Starting with mesh irregularity $0.7873016$ after 205 iteration mesh irregularity $0.5633803$. 

Fig. 6.5 Output 3
6.2.4 Output 4

Starting with mesh irregularity $0.5949074$ after 149 iteration mesh irregularity $0.5145374$. 

![Graphs showing iteration 0 and iteration 149 for mesh irregularity and aspect ratio distributions.]

Fig. 6.6 Output 4
6.2.5 Output 5

Starting with mesh irregularity \(0.1011029\) after 44 iteration mesh irregularity \(0.08969466\).
Note that, as seen in *Output 2* and *Output 3*, the output meshes may not be the best possible mesh for the bubble positions. Our concern is to create well-shaped triangles in object space; the inter-bubble forces are defined and dynamic simulation is performed in object space. Ideally, the mesh topology, or node connection, should be decided in object space. In the current implementation, however, this topology is generated in parametric space by using a 2D constrained Delaunay triangulation. The best node connection in parametric space may not be the best, however, when the surface is later mapped to object space, especially when the surface becomes highly deformed because of mapping. Nevertheless, this problem can be addressed by checking the shapes of the triangles in object space and flipping the connections of nodes where necessary.

### 6.3 Determining a suitable stopping criterion

We analyse the result produced at each iteration to find one or more distinguishing features that can be used as suitable stopping criterion. As explained in section 5.6, our algorithm is using the standard deviation of bubble movements after each iteration as a stopping criterion. However, the number of iterations required for the algorithm to stop exceeds by a very high margin the number of iterations required to achieve the desired quality. Most of the quality improvement occurs in the first few iterations. After that, the bubbles move quite slowly toward the equilibrium state, but the movement is still high enough to pass the tolerance check. In the following analysis, the figures are generated from *Output 1* (shown in Fig. 6.3) from the previous section, but the results are similar in nearly all the other configurations.

![Fig. 6.8 Standard Deviation of bubble movement Vs Number of iterations](image)

Fig. 6.8 shows that node movements of high magnitude occur in the first few hundred iterations and values are more spread out across the chart. This behaviour is expected as most of most of the bubble insertions, and deletion takes place in the early stages of the algorithm, which causes more irregularities in the bubble motion. But once the system reaches a stable number of bubbles (around 1500 iteration mark), the standard deviation of bubble movements shows a more defined shape and the magnitude of the standard deviation slowly decreases until finally goes below the tolerance and the iterations stop.
The PhD thesis of prof. Kenji Shimada [13] does not talk about the stopping criterion directly. Thesis talks about settling time, i.e. 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 ±2% and ±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 ±2% range, and \( \zeta = 0.68 \) for ±5% [15], that’s why the algorithm select \( \zeta = 0.7 \).

The prof. Kenji Shimada’s thesis also talks about the process converging to minimum energy state for configuration of bubbles. In other words, how the total energy, the summation of kinetic energy and potential energy, is minimised through this dynamic simulation. The system starts with very high potential energy which reduces over the time (Fig. 6.10 left-top). Since we assume that bubbles are standing still in the initial configuration, the kinetic energy at the
starting time is zero (Fig. 6.10 right-top), kinetic energy then increases, and slowly reduces over time. Whereas in Fig. 6.10 left-bottom where the plot is drawn from iteration 1 and onward that trend of kinetic energy is not visible this is because in our current implementations if there is a node deletion or addition operation under population control, algorithm resets the velocity of every bubble to zero hence kinetic energies have discontinuities.

Mesh irregularity may be a good candidate for a stopping criterion because we do not have to generate the mesh to calculate mesh irregularity; the neighbouring information is enough. Bubble packing always has neighbouring information ready by design. Therefore calculating mesh irregularity doesn’t cause any performance issues. When we plot the mesh irregularity against the number of iterations (Fig. 6.11), we make following major observation.

- First, that mesh irregularity behaves similarly as the total energy of the system (Fig. 6.12), therefore can be used instead for tracking the total energy of the system.

- Second, even though the algorithm achieves minimum mesh irregularity at around 250 iterations, most of the improvement in mesh irregularity occurs under 50 iterations. Moreover, if other quality parameters are plotted for the same range (Fig. 6.13, Fig. 6.14, Fig. 6.15) they agree with mesh irregularity, i.e. most of the improvement happens under 50 iteration. This makes mesh irregularity a prime candidate for the stopping criterion.

- However, the last observation is that improvement in mesh irregularity is not monotone till it reaches a required quality and hence finding a local minimum cannot be used as the stopping criterion.
Fig. 6.13 Aspect Ratio Vs Number of iterations

Fig. 6.14 Triangle Count with different min angle Vs Number of iterations

Fig. 6.15 Triangle Count with different max angle Vs Number of iterations

Fig. 6.16 Number of nodes added or deleted vs Number of iterations
6.4 Comparison between different Spring Constant

While generating a result for varying values of the spring constant while keeping the mass and node spacing function constant, it was observed that the system behaves in a stable manner within the range \([0.1, 150]\) for spring constant \((k)\). When \(k\) goes below 0.1, the bubble movement per iteration is smaller than the tolerance set for node movement cut-off, hence the system stops as soon as the iterations start. When \(k\) goes over 150, the bubble movement per iteration is so high that the whole system becomes unstable. The following results are generated for mass \((m)\) 1.0, damping ratio \((\xi)\) 0.7 and spring constant \((k)\) starting form 0.1 to 100 with a step size of 5. The algorithm was run 100 times for each value of the spring constant \((k)\); therefore, around 2000 data points were generated and are plotted below.

![Fig. 6.17 Min iteration for mesh irregularity vs Spring constant \((k)\)](image)

It was also observed that for a given node spacing function nearly all iterations were able to produce a mesh irregularity below 0.55. Hence a cut off of 0.55 was set and we noted the minimum number of iterations required to achieve mesh irregularity of 0.55 (Fig. 6.17 left). By applying an order 4 polynomial fit to predict the iteration trend we found that our algorithm produces minima around \(k\) equal to 30. The minimum iteration count first goes down as we move away from increasing for \(k > 30\). There is a similar trend for minimum mesh irregularity for a given spring constant (Fig. 6.17 middle). We also plotted the minimum number of iterations to reach the minimum mesh irregularity for a given spring constant (Fig. 6.17 right). We found that the algorithm gives the best results at \(k \sim 30\) with comparably better mesh irregularity.

![Fig. 6.18 Min iteration for aspect ratio vs Spring constant \((k)\)](image)

A similar trend was observed for the mean aspect ratio of the mesh (Fig. 6.18).
6.5 Conclusion

Based on the empirical analysis of the result presented in the previous section, we conclude with the following observations:

- The best value for the *spring constant* lies around 30 for evenly massed bubbles.
- The system becomes unstable for a spring constant above 100 for evenly massed bubbles.
- Mesh irregularity may be a more viable and efficient stopping criterion, but further analysis is needed to determine the accuracy of this approach.

The results presented in this chapter shed further light on the viability of the bubble packing algorithm in producing meshes of good quality. There is still a lot of scope for improvement and analysis in the algorithm. But even this initial investigation into the algorithm presented many positive results and insight into the algorithm. For example, as the system remains stable if the physical parameters are selected correctly in spite of the complexities of the geometry or of this algorithm. In addition, the two basic problems of meshing, namely, where to place nodes and how many nodes to place in the domain, were solved via dynamic simulation of inter-bubble forces and adaptive bubble population control, in spite of using some analytical interpretation of the underlying surface. This partial-independence from the geometric definitions safeguards the algorithm against many problems caused by the geometries.
Chapter 7 Future Work

Due to the lack of commercialisation of the bubble packing algorithm as a mainstream meshing algorithm, there is a limited amount of research focused on this topic. As a result, there are numerous opportunities to investigate improvement in the algorithm, some of which are discussed below.

7.1 Move Force Balancing to GPU (Graphics Processing Unit)

As a GPU uses the Single instruction, multiple data (SIMD) paradigm, it gives high performance boost when the same instructions (code) are applied to each data in a large dataset where the processing is mutually exclusive on data (the operation on data at position $i$ in the array does not need any information from data at $i + j$ where $j \in \{1,2,\ldots,n\}$) and the data resides contiguously in memory space. Force balancing uses the Runge-Kutta method as explained in section 2.3. Runge-Kutta method applies simple arithmetic on a large continuous array of data to find the next data set, and operations on data are mutually independent of the positions of data in the array. Utilizing the GPU for this could significantly speed up the computation. Actual statistics can only be obtained after implementation, but this approach looks promising and can be a next possible future step in the algorithm.

7.2 Multi-Core Bubble Neighbour Update

A time-consuming step of the bubble packing algorithm is updating the neighbour information for each bubble. Each bubble has a fixed region of influence and the magnitude of motion is not that large for each bubble after each iteration. Since a bubble is independent of the bubble outside its fixed region of influence, we can parallelise the node neighbour update code. We can divide the domain into regions of independent motion as shown in Fig. 7.1. Regions of independent motion can then be sent on different cores of CPU to update the neighbour information in parallel. This process can improve the overall performance of the algorithm.

![Fig. 7.1 Multi-core bubble neighbour update](image)
7.3 Single Body Meshing

One of the major challenges with meshing is handling small features in a big model (Fig. 7.2 (a)). Due to the size disparities in features in a model, most meshing algorithms either ends up with a large number of elements or crash due to the disagreement of tolerances and the nature of preserving the geometries (Fig. 7.2 (b)). Most of these small features do not affect the solution of the simulations; hence a lot of automated effort is needed to clean the model as per the simulation before it gets meshed.

Meshing using bubble-packing method can be a very useful tool in handling these problems. A small change in information handling can make information flow across geometry seamlessly. The force balancing algorithm in bubble packing depends on the position of neighbouring bubbles in the space rather than the actual shape of underlying geometry, but at the same time, positions of bubbles get corrected after each iteration to pull the bubble back to the geometry (explained in section 5.5.3). This correction step can be exploited in packing, where rather than pulling the bubble back to the same geometry we can pull back the bubble to the nearest geometry and update the bubble information accordingly. This can allow the bubble to move freely across all the surfaces of the model as if the whole model is made up of a single surface (Fig. 7.3). As a result, we can reduce the number of elements around the small feature and produces a much more reliable mesh for simulation.
References


