Lecture 11

Mesh Generation

An essential step in scientific computing is to find a proper discretization of a continuous domain. This is the problem of *mesh generation*. Once we have a discretization or sometimes we just say a "mesh", differential equations for flow, waves, and heat distribution are then approximated by finite difference or finite element formulations. However, not all meshes are equally good numerically. Discretization errors depend on the geometric shape and size of the elements while the computational complexity for finding the numerical solution depends on the number of elements in the mesh and often the overall geometric quality of the mesh as well.

The most general and versatile mesh is an unstructured triangular mesh. Such a mesh is simply a triangulation of the input domain (e.g., a polygon), along with some extra vertices, called *Steiner points*. A triangulation is a decomposition of a space into a collection of interior disjoint simplices so that two simplices can only intersect at a lower dimensional simplex. We all know that in two dimensions, a simplex is a triangle and in three dimensions a simplex is a tetrahedron. A triangulation can be obtained by triangulating a point set, that form the vertices of the triangulation.

Even among triangulations some are better than others. Numerical errors depend on the *quality* of the triangulation, meaning the shapes and sizes of triangles.

In order for a mesh to be useful in approximating partial differential equations, it is necessary that discrete functions generated from the mesh (such as the piecewise linear functions) be capable of approximating the solutions sought. Classical finite element theory [18] shows that a sufficient condition for optimal approximation results to hold is that the minimum angle or the aspect ratio of each simplex in the triangulation be bounded independently of the mesh used; however, Babuska [4] shows that while this is sufficient, it is not a necessary condition. See Figure **??** for a triangulation whose minimum degree is at least 20 degree.

In summary, properties of a good mesh are:

- Fills the space
- Non-overlapping ($\sum$ areas = total area)
- Conforming Mesh (every edge shared by exactly 2 triangles)
- High Quality Elements (approximately equilateral triangles)

Automatic mesh generation is a relatively new field. No mathematically sound procedure for obtaining the 'best' mesh distribution is available. The criteria are usually heuristic.

The input description of physical domain has two components: the geometric definition of the domain and the numerical requirements within the domain. The geometric definition provides the

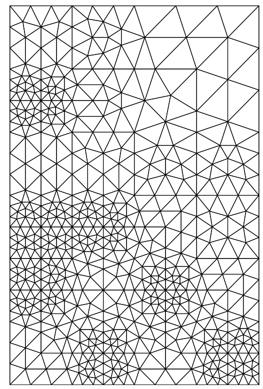

Figure 11.1: A well-shaped triangulation

boundary of the domain either in the form of a continuous model or of a discretized boundary model. Numerical requirements within the domain are typically obtained from an initial numerical simulation on a preliminary set of points. The numerical requirements obtained from the initial point set define an additional local spacing function restricting the final point set.

An automatic mesh generator try to generate an additional points to the internally and boundary of the domain to smooth out the mesh generation and concentrate mesh density where necessary - to optimize the total number of mesh points.

11.1 How to Describe a Domain?

The most intuitive and obvious structure of a domain (for modeling a scientific problem) is its geometry.

One way to describe the geometry is to use *constructive solid geometry* formula. In this approach, we have a set of basic geometric primitive shapes, such as boxes, spheres, half-spaces, triangles, tetrahedra, ellipsoids, polygons, etc. We then define (or approximate) a domain as finite unions, intersections, differences, and complementation of primitive shapes, i.e., by a well-structured formula of a finite length of primitive shapes with operators that include union, intersection, difference, and complementation.

An alternative way is to discretize the boundary of the domain, and describes the domain as a polygonal (polyhedral) object (perhaps with holes). Often we convert the constructive solid geometry formula into the discretized boundary description for mesh generation.

For many computational applications, often, some other information of a domain and the problem are equally important for quality mesh generation.

The numerical spacing functions, typically denoted by $h(\boldsymbol{x})$, is usually defined at a point $\boldsymbol{x}$ by the eigenvalues of the Hessian matrix of the solution u to the governing partial differential equations (PDEs) [4, 95, 69]. Locally, u behaves like a quadratic function

$$u(\boldsymbol{x} + d\boldsymbol{x}) = \frac{1}{2}(\boldsymbol{x}H\boldsymbol{x}^T) + \boldsymbol{x}\nabla u(\boldsymbol{x}) + u(\boldsymbol{x}),$$

where H is the *Hessian matrix* of u, the matrix of second partial derivatives. The spacing of mesh points, required by the accuracy of the discretization at a point $\boldsymbol{x}$, is denoted by $h(\boldsymbol{x})$ and should depend on the reciprocal of the square root of the largest eigenvalues of H at $\boldsymbol{x}$.

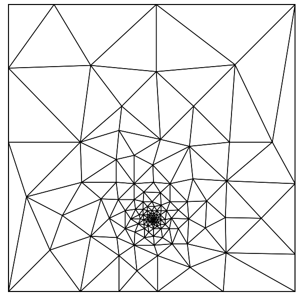

Figure 11.2: Triangulation of well-spaced point set around a singularity

When solving a PDE numerically, we estimate the eigenvalues of Hessian at a certain set of points in the domain based on the numerical approximation of the previous iteration [4, 95]. We then expand the spacing requirement induced by Hessian at these points over the entire domain.

For a problem with a smooth change in solution, we can use a (more-or-less) uniform mesh where all elements are of roughly equal size. On the other hand, for problem with rapid change in solution, such as earthquake, wave, shock modeling, we may use much dense grinding in the area of with high intensity. See Fig 11.2.So, the information about the solution structure can be of a great value to quality mesh generation.

Other type of information may come in the process of solving a simulation problem. For example, in adaptive methods, we may start with a much coarse and uniform grid. We then estimate the error of the previous step. Based on the error bound, we then adaptively refine the mesh, e.g., make the area with larger error much more dense for the next step calculation. As we shall argue later, unstructured mesh generation is more about finding the proper distribution of mesh point then the discretization itself (this is a very personal opinion).

11.2 Types of Meshes

- **Structured grids** divide the domain into regular grid squares. For examples finite difference gridding on a square grid. Matrices based on structured grids are very easy and fast to assemble. Structured grids are easy to generate; numerical formulation and solution based on structured grids are also relatively simple.

- **Unstructured grids** decompose the domain into simple mesh elements such as simplices based on a density function that is defined by the input geometry or the numerical requirements (e.g., from error estimation). But the associated matrices are harder and slower to assemble compared to the previous method; the resulting linear systems are also relatively hard to solve. Most of finite element meshes used in practice are of the unstructured type.

- **Hybrid grids** are generated by first decomposing the domain into non-regular domain and then decomposing each such domain by a regular grid. Hybrid grids are often used in domain decomposition.

Structured grids are much easy to generate and manipulate. The numerical theory of this discretization is better understood. However, its applications is limited to problems with simple domain and smooth changes in solution. For problems with complex geometry whose solution changes rapidly, we need to use an *unstructured mesh* to reduce the problem size. For example,

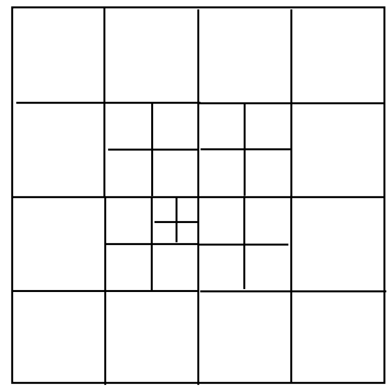

Figure 11.3: A quadtree

when modeling earthquake we want a dense discretization near the quake center and a sparse discretization in the regions with low activities. It would be waste to give regions with low activities as fine a discretization as the regions with high activities. Unstructured meshes are especially important for three dimensional problems.

The adaptability of unstructured meshes comes with new challenges, especially for 3D problems. However, the numerical theory becomes more difficult – this is an outstanding direction for future research; the algorithmic design becomes much harder.

11.3 Refinement Methods

A mesh generator usually does two things: (1) it generates a set of points that satisfies both geometric and numerical conditions imposed on the physical domain. (2) it builds a robust and well-shaped meshes over this point set, e.g., a triangulation of the point set. Most mesh generation algorithms merge the two functions, and generate the point set implicitly as part of the mesh generation phase. A most useful technique is to generate point set and its discretization by an iterative refinement. We now discuss *hierarchical refinement* and *Delaunay refinement*, two of the most commonly used refinement methods.

11.3.1 Hierarchical Refinement

The hierarchical refinement uses quadtrees in two dimensions and octtrees in three dimensions. The basic philosophy of using quad- and oct-trees in meshes refinements and hierarchical N-body simulation is the same: adaptively refining the domain by selectively and recursively divide boxes enable us to achieve numerical accuracy with close to an optimal discretization. The definition of quad- and oct-tree can be found in Chapter **??**. Figure 11.3 shows a quad-tree.

In quadtree refinement of an input domain, we start with a square box encompassing the domain and then adaptively splitting a box into four boxes recursively until each box small enough with respect to the geometric and numerical requirement. This step is very similar to quad-tree decomposition for N-body simulation. However, in mesh generation, we need to ensure that the mesh is well-shaped. This requirement makes mesh generation different from hierarchical N-body approximation. In mesh generate, we need to generate a set of smooth points. In the context of quad-tree refinement, it means that we need to make the quad-tree *balanced* in the sense that no leaf-box is adjacent to a leaf-box more than twice its side length.

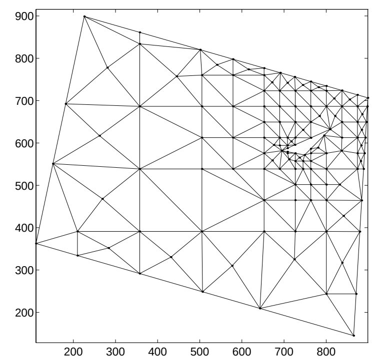

Figure 11.4: A well-shaped mesh generated by quad-tree refinement

With adaptive hierarchical trees, we can "optimally" approximate any geometric and numerical spacing function. The proof of the optimality can be found in the papers of Bern, Eppstein, and Gilbert for 2D and Mitchell and Vavasis for 3D. Formal discuss of the numerical and geometric spacing function can be found in the point generation paper of Miller, Talmor and Teng.

The following procedure describes the basic steps of hierarchical refinement.

1. Construct the hierarchical tree for the domain so that the leaf boxes approximate the numerical and geometric spacing functions.
2. Balance the hierarchical tree.
3. Warping and triangulation: If a point is too close to a boundary of its leaf box then one of the corners collapses to that point.

11.3.2 Delaunay Triangulation

Suppose $P = \{p_1, \ldots, p_n\}$ is a point set in d dimensions. The convex hull of $d+1$ affinely independent points from P forms a *Delaunay simplex* if the circumscribed ball of the simplex contains no point from P in its interior. The union of all Delaunay simplices forms the *Delaunay diagram*, $DT(P)$. If the set P is not degenerate then the $DT(P)$ is a simplex decomposition of the convex hull of P. We will sometimes refer to the Delaunay simplex as a triangle or a tetrahedron.

Associated with $DT(P)$ is a collection of balls, called *Delaunay balls*, one for each cell in $DT(P)$. The Delaunay ball circumscribes its cell. When points in P are in general position, each Delaunay simplex define a Delaunay ball, its circumscribed ball. By definition, there is no point from P lies in the interior of a Delaunay ball. We denote the set of all Delaunay balls of P by $DB(P)$.

The geometric dual of Delaunay Diagram is the *Voronoi Diagram*, which of consists a set of polyhedra $V_1, \ldots, V_n$, one for each point in P, called the *Voronoi Polyhedra*. Geometrically, V_i is the set of points $p \in \mathbb{R}^d$ whose Euclidean distance to p_i is less than or equal to that of any other point in P. We call p_i the *center* of polyhedra V_i. For more discussion, see [78, 31].

The DT has some very desired properties for mesh generation. For example, among all triangulations of a point set in 2D, the DT maximizes the smallest angle, it contains the nearest-neighbors graph, and the minimal spanning tree. Thus Delaunay triangulation is very useful for computer graphics and mesh generation in two dimensions. Moreover, discrete maximum principles will only exist for Delaunay triangulations. Chew [17] and Ruppert [84] have developed Delaunay refinement algorithms that generate provably good meshes for 2D domains.

Notice that an internal diagonal belongs to the Delaunay triangulation of four points if the sum of the two opposing angles is less than π.

A 2D Delaunay Triangulation can be found by the following simple algorithm: *FLIP algorithm*

- Find any triangulation (can be done in $O(n \lg n)$ time using divide and conquer.)
- For each edge pq, let the two faces the edge is in be prq and psq. Then pq is not an local Delaunay edge if the interior the circumscribed circle of prq contains s. Interestingly, this condition also mean that the interior of the circumscribed circle of psq contains r and the sum of the angles prq and psq is greater than π. We call the condition that the sum of the angles of prq and psq is no more than π *the angle condition*. Then, if pq does not satisfy the angle property, we just flip it: remove edge pq from T, and put in edge rs. Repeat this until all edges satisfy the angle property.

It is not too hard to show that if FLIP terminates, it will output a Delaunay Triangulation. A little addition geometric effort can show that the FLIP procedure above, fortunately, always terminate after at most $O(n^2)$ flips.

The following is an interesting observation of Guibas, Knuth and Sharir. If we choose a random ordering π of from $\{1, ..., n\}$ to $\{1, ..., n\}$ and permute the points based on π: $p_{\pi(1)} \dots p_{\pi(n)}$. We then incrementally insert the points into the the current triangulation and perform flip if needed. Notice that the initial triangulation is a triangle formed by the first three points. It can be shown that the expected number of flips of the about algorithm is $O(n \log n)$. This gives a randomized $O(n \log n)$ time DT algorithm.

11.3.3 Delaunay Refinement

Even though the Delaunay triangulation maximize the smallest angles. The Delaunay triangulation of most point sets are bad in the sense that one of the triangles is too 'skinny'. In this case, Chew and Ruppert observed that we can refine the Delaunay triangulation to improve its quality. This idea is first proposed by Paul Chew. In 1992, Jim Ruppert gave a quality guaranteed procedure.

- Put a point at the circumcenter of the skinny triangle
- if the circum-center encroaches upon an edge of an input segment, split an edge adding its middle point; otherwise add the circumcenter.
- Update the Delaunay triangulation by FLIPPING.

A point *encroaches* on an edge if the point is contained in the interior of the circle of which the edge is a diameter. We can now define two operations, Split-Triangle and Split-Segment

Split-Triangle(T)

- Add circumcenter C of Triangle T
- Update the Delaunay Triangulation $(P \cup C)$

Split-Segment(S)

- Add midpoint m
- Update the Delaunay Triangulation $(P \cup M)$

The *Delaunay Refinement* algorithm then becomes

- Initialize
  - Perform a Delaunay Triangulation on P
  - IF some segment, l, is not in the Delaunay Triangle of P, THEN Split-Segment (l)
- Repeat the following until $\alpha > 25°$
  - IF T is a skinny triangle, try to Split(T)
  - IF C of T is 'close' to segment S then Split-Seg(S)
  - ELSE Split Triangle (T)

The following theory was then stated, without proof. The proof is first given by Jim Ruppert in his Ph.D thesis from UC. Berkeley.

Theorem 11.3.1 Not only does the Delaunay Refinement produce all triangles so that $MIN\ \alpha > 25°$, the size of the mesh it produces is no more than $C * Optimal(size)$.

11.4 Working With Meshes

- Smoothing: Move individual points to improve the mesh.
- Laplacian Smoothing: Move every element towards the center of mass of neighbors
- Refinement: Have a mesh, want smaller elements in a certain region. One option is to put a circumscribed triangles inside the triangles you are refining. Another approach is to split the longest edge on the triangles. However, you have to be careful of hanging nodes while doing this.

11.5 Unsolved Problems

There still is no good algorithm to generate meshes of shapes. There has been a fair amount of research and this is an important topic, but at the current time the algorithms listed above are merely best efforts and require a lot of work and/or customization to produce good meshes. A good algorithm would fill the shape with quadrilaterals and run without any user input, beyond providing the geometry to be meshed. The holy grail of meshing remains filling a 3-D shape with 3-D blocks.