

- Mark Adams : Finite Element Methods
  1. FEM Intro
  2. Approximating the Weak Form with Subspaces
  3. Picking Good Subspaces
  4. Linear System Solvers
  5. Multigrid
  6. Unstructured Multigrid Algorithms
  7. Parallel Multigrid Solver for FE
- Distance Metrics for Voronoi Diagrams and Delaunay Triangulations
- Surface Modeling Using Triangulation
  1. Terrain Modeling
     (a) Methods for Representing Terrains
     (b) Benefits of Triangulation
     (c) Algorithm

1 Triangulation in Finite Element Methods (Mark Adams)

We begin with a look at the finite element method (FEM) and a parallel algorithm for solving FEM problems which makes use of the Delaunay triangulation.

1.1 FEM Intro

We will examine the basic FEM approach by applying it to the heat equation,

$$k\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2}\right) + Q = \rho c \frac{\partial T}{\partial t}$$

defined on a domain Ω with boundary $\partial\Omega$ where T is temperature, and Q is the heat generated in the domain. Take T = 0 on the boundary of Ω, and for simplicity assume that we are looking at the steady state case in 1D, so

$$\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial y^2} = \frac{\partial^2 T}{\partial z^2} = 0.$$

Thus, we have

$$k\frac{\partial^2 T}{\partial x^2} + Q = 0.$$

This is referred to as the strong form of the equation. In order to apply the finite element method, we must develop a computable form of this equation : the weak form. We start by writing a residual for the strong form as

$$R_\Omega = k\frac{\partial^2 T}{\partial x^2} + Q - 0.$$

Using this equation, the so-called weighted residual form is

$$\int_\Omega W \cdot R_\Omega \partial\Omega = 0$$

where W is some arbitrary function of position with the property that $\forall W\ s.t.\ W = 0$ on $\partial\Omega$. This assumption of W = 0 on the boundary, called a Dirichlet boundary condition, gives us a particular weak form of the original equation called the Galerkin form. We have

$$\int_\Omega \left(W \cdot k\frac{\partial^2 T}{\partial x^2} + W \cdot Q \right) \partial\Omega = 0,$$

and by applying integration by parts, we arrive at the weak form.

$$\int_\Omega \left(\frac{\partial W}{\partial x} k \frac{\partial T}{\partial x} - W \cdot Q \right) \partial\Omega = 0$$

1.2 Approximating the Weak Form with Subspaces

We now apply the finite element approximation by discretizing the domain of the problem into nodes and introducing functions which interpolate the solution over these nodes. In other words, we construct basis sets ϕ and φ such that

$$\tilde{T} \in span(\phi_1, \phi_2, ...\phi_n) \qquad \tilde{T} = \sum_{i=1}^{n} u_i \cdot \phi_i$$

$$\tilde{W} \in span(\varphi_1, \varphi_2, ...\varphi_m) \qquad \tilde{W} = \sum_{i=1}^{n} w_i \cdot \varphi_i$$

where the u_i form the solution we seek, and the w_i are the weights introduced earlier. The sets ϕ and φ are often taken to be the same. This produces a particular sort of approximation known as a Bubnov-Galerkin approximation.

Applying these subspace definitions to the weak form of the heat equation, we get

$$\sum_{i=1}^{n} w_i \int_\Omega \left(\sum_{j=1}^{n} u_j \frac{\partial \varphi_i}{\partial x} k \frac{\partial \phi_j}{\partial x} - \varphi_i \cdot Q \right) \partial\Omega = 0 \qquad \forall w_i.$$

Thus, for $i = 1 \ldots n$

$$\sum_{j=1}^{n} u_j \int_\Omega \left(\frac{\partial \varphi_i}{\partial x} k \frac{\partial \phi_j}{\partial x} \right) \partial\Omega = \int_\Omega \varphi_i \cdot Q \partial\Omega.$$

We now have a linear system $Au = f$ which we can solve for u. In this case,

$$A_{ij} = \int_\Omega \left(\frac{\partial \varphi_i}{\partial x} k \frac{\partial \phi_j}{\partial x} \right) \partial \Omega$$

and

$$f_i = \int_\Omega \Big(\varphi_i \cdot Q \Big) \partial \Omega.$$

1.3 Picking Good Subspaces

In order to achieve accurate results, we must take care in selecting the subspaces ϕ and φ. Methods which make use of sine transforms and wavelets, referred to as spectral methods and element free FEM, respectively, are occasionally used, primarily in research. However, the majority of FE work uses piecewise linear (or quadratic etc.) polynomials. Figure 79 illustrates the use of 1D piecewise linear polynomials for the subspaces. Here we have chosen $n = 5$, and we have applied the Bubnov-Galerkin approximation of $\phi = \varphi$. As can be seen in the figure, the domain Ω has been discretized into nodes, represented by empty circles. The hat functions interpolate between solutions at nodal points. Note that we assume that we have solution values at the boundaries of the domain. Figure 80 depicts a similar situation but this time in 2D with 2D piecewise linear polynomials. The dotted lines show the hat function over one node in the mesh.

The nodes which discretize the domain define the elements which give the method its name. In 2D these elements are usually triangles or quadrilaterals. In 3D they are normally tetrahedra or hexahedra. Hexahedral elements have a part of a higher order space than tetrahedral elements, so one can use larger hexahedral elements and less of them to get a solution which is as accurate as a solution generated from a mesh with a larger number of smaller tetrahedral elements. This can cut down on computation time. However, it is much easier to automate the process of generating triangular and tetrahedral meshes, so these elements are particularly appropriate for large, complex models.

1.4 Linear System Solvers

We now turn our attention to methods for solving the linear systems generated by the FEM. This is normally the most costly step in a FEM solution. Recall that we want to solve for u in $Au = f$. In this equation, A is sparse and there are roughly a constant number of non-zeros per row. In addition, A is symmetric in structure (*i.e.*, nonzeros vs zeros are symmetric), and can have symmetric or non-symmetric coefficients. Finally, we normally have a fairly large system with $10^4 - 10^{7+}$ equations. We can solve this system using either direct or iterative methods which we briefly examine below.

1.4.1 Direct Methods (*e.g.*, Gauss Elimination)

Start by factoring $A \Rightarrow LU$ $s.t.$ $L^T, U \in$Upper Triangular. The cost of this step $\approx O(n^2)$.

Now, solve $Ly = f$ for y and $Uu = y$ for u. The cost $\approx O(n^{3/2})$.

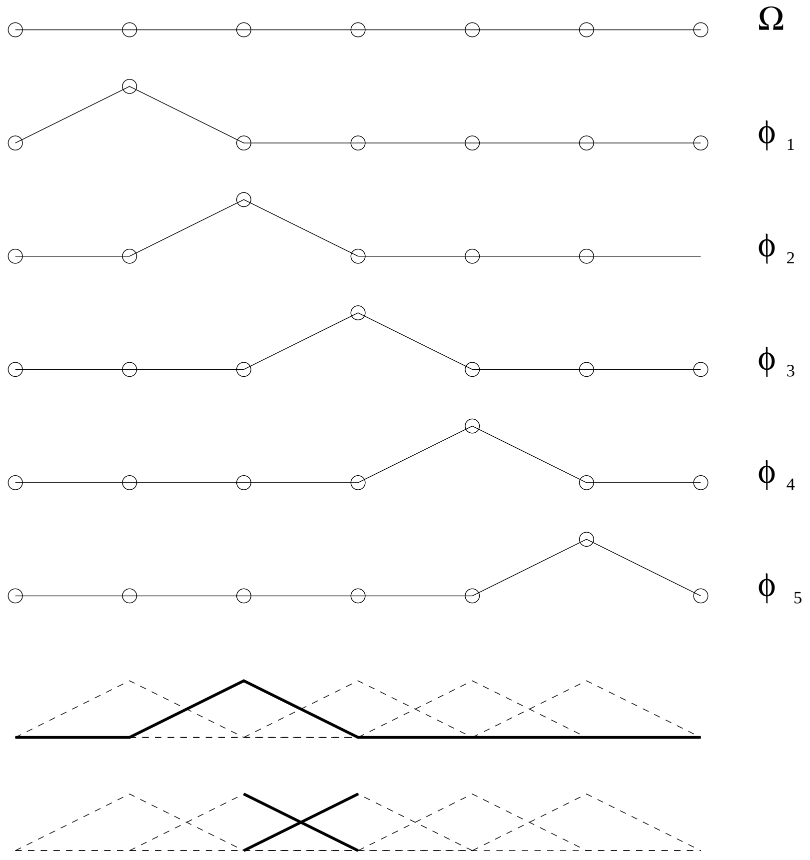


Figure 79: 1D Piecewise Linear FE Shape Functions

The cost of this method is sensitive to vertex ordering, so we have a graph problem. This ordering problem has been studied extensively, and is NP complete. Successful solutions include nested dissection, a divide and conquer algorithm, and minimum degree, a greedy algorithm.

1.4.2 Iterative Methods

The total cost of any of these methods is (number of iterations) $\cdot O(n)$, since for FE discretizations a matrix vector multiply has cost $= O(n)$.

Matrix Splitting

Take $A = (D - U)$ where, for example, $D = diagonal \Rightarrow \quad Jacobi$

Substituting into the equation $Au = f$ and rearranging, we have $u = D^{-1}(f + Uu) \Rightarrow$, so the iterative form is $u_{k+1} = D^{-1}(f + Uu_k)$.

The number of iterations $\approx O(n)$

Conjugate Gradients (CG)

Look for the "best" solution in $span\{f, Af, A^2f, A^3f, ...\}$

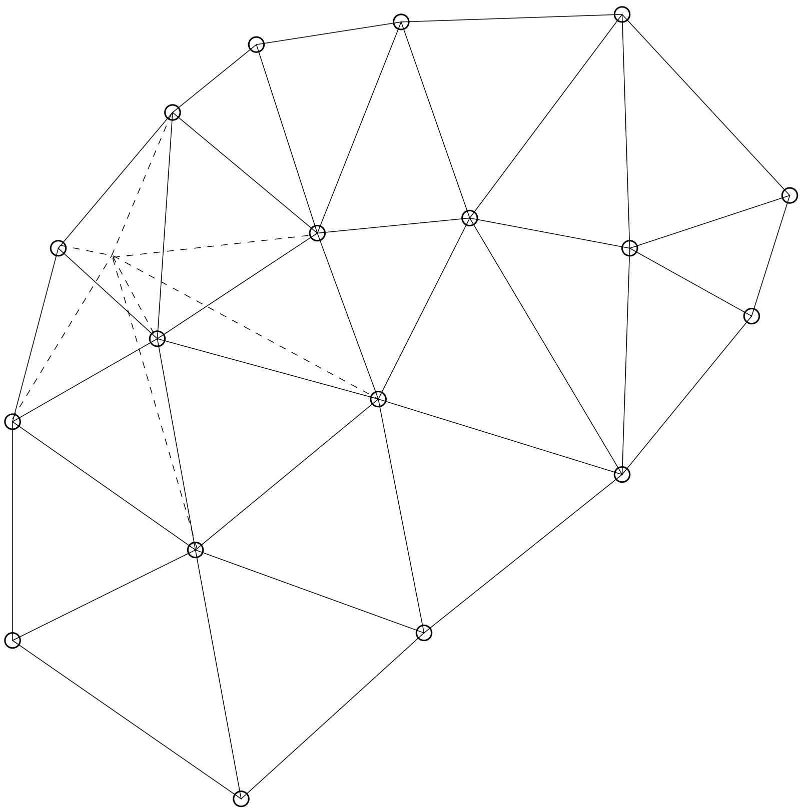


Figure 80: 2D FE Mesh

The number of iterations $\approx O(n^{1/2})$

Domain Decomposition

"Divide and conquer" the spatial domain.

Multigrid

"Divide and conquer" the frequency domain.
number of iterations $\approx O(1)$

Preconditional Conjugate Gradient (PCG)

The mathematical idea is to premultiply both sides of the equation by the matrix $M \approx A^{-1}$. Thus, $MAu = Mf$. Note that we don't explicitly compute $MAu = Mf$ in practice. We just want a preconditioner for which solving $Mz = y$ is cheap in comparison to its effectiveness.

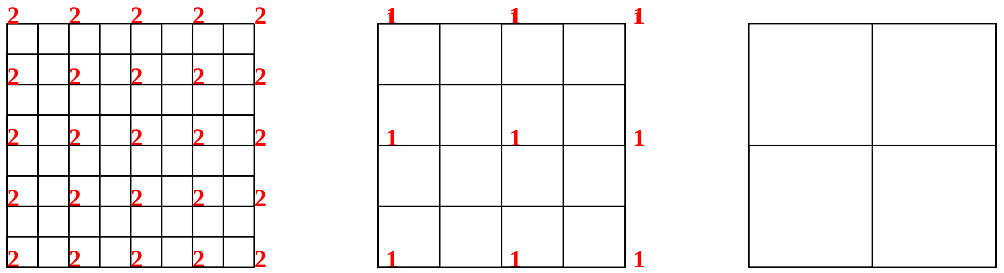
One can choose from a variety of different preconditioners which vary greatly in their cost and overall effectiveness. Note that by effectiveness we refer to how fast the iterative method converges to a solution. Examples of preconditioners include :

- Matrix splitting, in which we use the inverse of a diagonal matrix D. This method is cheap but ineffective.

- Direct method, in which we directly calculate $M = A^{-1}$ in 1 iteration. This is extremely effective, but also very expensive.
- Incomplete factorization, in which we take $A \approx LU$. This method is not all that cheap but is fairly effective.
- Domain decomposition, in which we use a block diagonal matrix instead of a diagonal matrix as in the case of matrix splitting. This has cost and effectiveness similar to that of incomplete factorization.
- Multigrid. This method is rather expensive, but it is very effective. We will use this method in the algorithm discussed below.

1.5 Multigrid

Simple iterative methods are good for large problems and reduce "high frequency" error, *i.e.*, they smooth the residual. Hence, we will use cheap iterative methods at multiple scales of resolution so that the iterative methods are used in cases for which they work best. The basic idea is that we want to generate coarser meshes from an initial fine mesh by reducing the number of vertices in a finer mesh and then remeshing. We then solve the problems on the coarser mesh and interpolate back to the finer mesh. Figure 81 shows a simple progression of coarser meshes.



$P^{(3)}$: 9 by 9 grid of points
7 by 7 grid of unknowns
Points labeled 2 are part of next coarser grid

$P^{(2)}$: 5 by 5 grid of points
3 by 3 grid of unknowns
Points labeled 1 are part of next coarser grid

$P^{(1)}$: 3 by 3 grid of points
1 by 1 grid of unknowns

Figure 81: Classic Multigrid Sequence of Grids

1.6 Unstructured Multigrid Algorithms

We apply the following algorithm recursively :

- Evenly coarsen vertex set $\rightarrow$ smaller vertex set
- Mesh this vertex set
- Create interpolation operators R_i and matrices A_i, for coarse grids

We coarsen a set of mesh nodes by finding the maximal independent set of these nodes and using this as the set of nodes for the new coarse set. This new set is meshed using Delaunay tessellation. We use the finite element shape functions (the ϕ's) to generate interpolation operators (R_i), and we calculate the coarse grid matrices using $A_{i+1} = R_i A_i R_i^T$. Figures 82 and 83 show examples of the grid coarsening process.

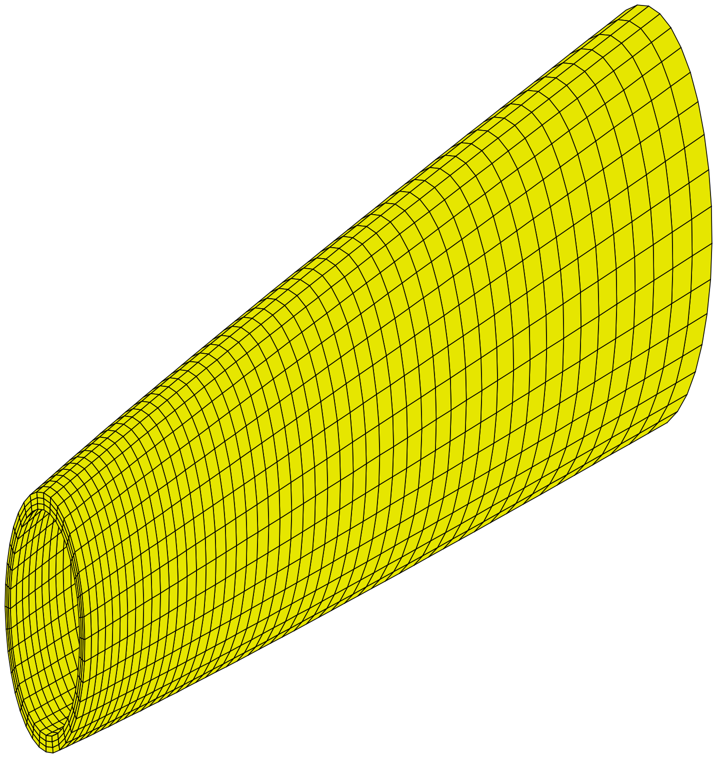


Figure 82: Sample Input Grid

1.7 Parallel Multigrid Solver for FE

Mark Adams has implemented a parallel version of the algorithm discussed above by making use of and developing a variety of software components. The system uses MPI for the message passing implementation, and a package called ParMetis to partition a mesh into sections which can then be assigned to separate processors. The R_i and A_i are determined by Adam's MGFeap program. The matrix A_0 is generated using a serial FE code called FEAP. A program called PETSc handles the parallel preconditional conjugate gradient calculations and the multigrid infrastructure and provides numerical primitives with a multigrid preconditioner.

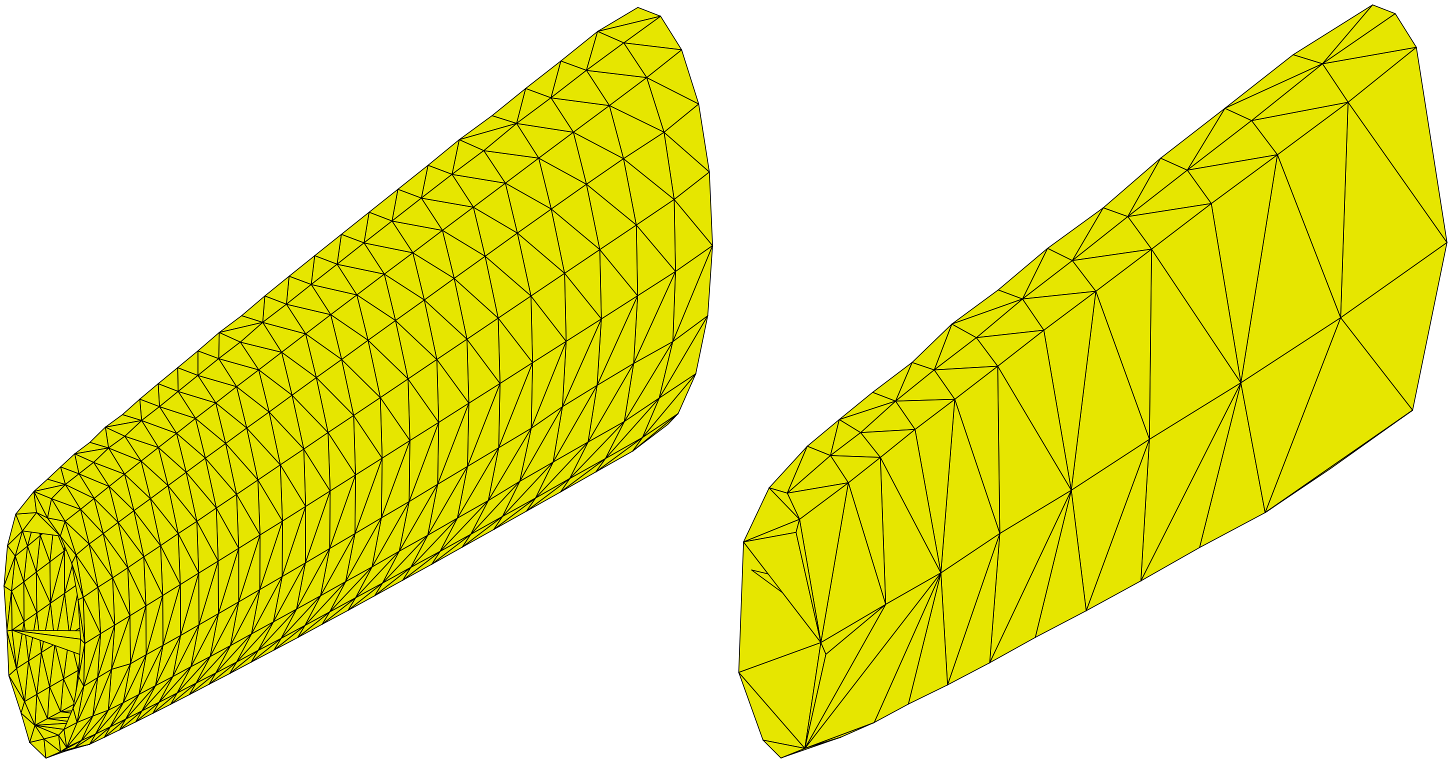


Figure 83: Sample Coarse Grids

2 Distance Metrics for Voronoi Diagrams and Delaunay Triangulations

We are not restricted to using only Euclidean distances when computing a Voronoi diagram or its dual, the Delaunay triangulation. Other possible distance metrics include :

- distances measured on spheres or conics
- Manhattan distance in which only the paths along coordinate axis directions are considered, *i.e.*, $d = \Delta x_1 + \Delta x_2$
- Supremum distance for which $d = max(\Delta x_1, \Delta x_2)$
- Karlsruhe distance which is a sort of polar distance for which

$$d = \begin{cases} |r_1 - r_2| + |\phi_1 - \phi_2| & if|\phi_1 - \phi_2| < \pi/2 \\ |r_1 + r_2| & otherwise \end{cases}$$

- weighted distance for which $d(p, p_i) = 1/w_i|x - x_i|$ for all points p where p_i are the seeds
- shortest path on a graph, for example the shortest path between two points on a city map if the path must stay on streets

3 Surface Modeling Using Triangulation

We will examine the use of triangulations in modeling and interpolating surfaces, particularly with respect to terrain modeling. In the first case, we wish to reduce the time and space required to represent and manipulate surfaces. For the second, we want to find interpolating surfaces between data points. Techniques to achieve these goals can be applied to problems in fields such as terrain modeling, computer graphics, medical imaging in which surface models are built by processing consecutive 2D slices of a volume of interest, and geophysical modeling used, for example, in oil exploration.

3.1 Terrain Modeling

Terrain modeling has been used for decades in military vehicle simulators and map displays in order to accurately represent landscape features. The dirt moving application discussed in an earlier lecture also makes use of terrain modeling.

3.1.1 Methods for Representing Terrains

We could use a variety of different approaches to model terrains. For example, we could describe a terrain as a collection of line segments approximating the curves of a contour plot. However, it would be difficult to convert such a representation from one measurement system to another, *e.g.*, from feet to meters. In addition, it is unclear how one would determine elevation values for points near but not on contours. Another option is to use a height field to store elevation data, but this representation is discontinuous and uses large amounts of space to store regions with constant elevations which could be compressed.

We could also view terrain modeling as a lossy compression problem. From this point of view we need to select an appropriate set of basis functions for the compression. If we use triangles as the bases, then we can use large triangles in terrain areas in which the slope is not changing much and smaller ones only in areas of changing slope. Thus, we compress the regions of constant slope. In particular, the second derivative of the terrain surface gives a measure of local feature size, so we want to have a greater density of triangles in regions with large second derivative values. This is, in fact, the representation used in terrain modeling.

It is interesting to note that the use of TINS, or triangulated irregular networks, in terrain modeling dates from the 1970s, before efficient Delaunay algorithms were known.

3.1.2 Benefits of Triangulation

Using a triangulation to represent a terrain provides a number of important benefits. For example, as discussed above, such a representation does not require much space, since large triangles can be used to represent areas of constant elevation, and we only need dense concentrations of triangles in areas where the slope changes rapidly. In addition, most modern graphics systems are optimized for rendering triangles, so rendering the terrain is quite efficient. It is also straight forward to find the slope of a triangle, so finding the slope at a particular point in the terrain is easy. Furthermore, constant height lines in the terrain are straight lines on the approximating triangles, so reconstructing contour lines is fairly easy.

3.1.3 Algorithm

Our basic problem is to generate a triangulation based on a given mesh of height values. Once the triangulation has been calculated, we can find a height value for any point (x, y) by interpolating between the heights of the corners of the triangle containing (x, y). We can calculate an error metric for our triangulation by finding the difference between the height value in the original mesh for point (x, y) and the interpolated value from the triangulation. Thus, if $I_t(x, y)$ = interpolated value at (x, y) based on corners of triangle containing the point, $z(x, y)$ = value at x, y in the original mesh, and error at $x, y = e(x, y)$, then $e(x, y) = |z(x, y) - I_t(x, y)|$. We make use of this error metric in the following algorithm due to Garland and Heckbert to calculate a triangulation.

function Surface_Model(M, ϵ)
    // M is the input mesh of altitudes
    // ϵ is the bound on the maximum error
    // The Output P is a subset of the original mesh points and its triangulation T
  P = the 4 corners of M
  T = DelaunayTriangulation(P)
  repeat while $(\max(\{e(x, y) : (x, y) \in M\}) > \epsilon)$
    select point p with maximum error
    $P = P \cup \{p\}$
    T = DelaunayTriangulation(P)
  return P and T
end

3.1.4 Finding $\max(e(x, y))$

We use a priority queue to find the max error at each step. The queue contains one entry per triangle, and the entries are ordered on the maximum error for each triangle. For each new triangle, we calculate its maximum error and insert it into the queue. Note that for each iteration i, there are $4 + i$ points and $2i$ triangles.

3.1.5 Running Time

Assume there are n points in the original mesh and m points in the final solution $(m \leq n)$. We can use the incremental method to add a new point to the Delaunay triangulation on each step (we need not start from scratch). Then the times for the i^{th} iteration are

	"expected"	worst case
find maximum error :	$O(\log(i))$	$O(\log(i))$
modify Delaunay triangulation :	$O(1)$	$O(i)$
find errors for each point in new triangles :	$O(n/i)$	$O(n)$

and the total times are

Total (expected) :

$$\sum_{i=4}^{m} \log(i) + n/i = O((n+m)\log(m))$$

Total (worst case) :

$$\sum_{i=4}^{m} \log(i) + n = O(nm)$$

Note that for the "expected time" for finding the errors for each point in new triangles all of the triangles are assumed to be approximately the same size, so for n points and i triangles, each triangle will cover n/i of the points.