which states the probability of picking 3 sites before j others. Thus the total number of Delaunay triangles appearing is

$$\tau = \sum_{j=0}^{N-3} \frac{3!j!}{(j+3)!} T_j$$

which is shown in [GuiKS93] to be $O(N)$. In fact for the Green-Sibson algorithm the total number of Delaunay triangles that arise, τ, is

$$\tau \leq 6N - 15H_N + \frac{21}{2} \tag{3.3.0}$$

where $H_N = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{N}$.

The randomization of site insertion in the Green-Sibson algorithm also permits the construction of an efficient randomized structure for point location. This is needed to find the triangle in which the new site falls inside. We will give an algorithm which contributes an overall complexity of $N \log N$ to the insertion algorithm. This complexity is known to be optimal for planar triangulations. The central idea is to keep a history of the construction of the triangulation. Guibas makes an analogy of the history to that of a stack of paper triangles glued on top of one another. Thus we think of storing triangulations one atop another.

Assume we wish to insert a new site Q into the triangulation and that we know that the triangle T contains site Q at some step in the construction of the triangulation. If T is present in the final triangulation then the search is complete. More generally, T will have been previously split in one of two ways: either a new site of the triangulation was added which divided T into three children or one of the edges of T was swapped, in which case T will be split into two children. In both cases the appropriate children are found in constant time.

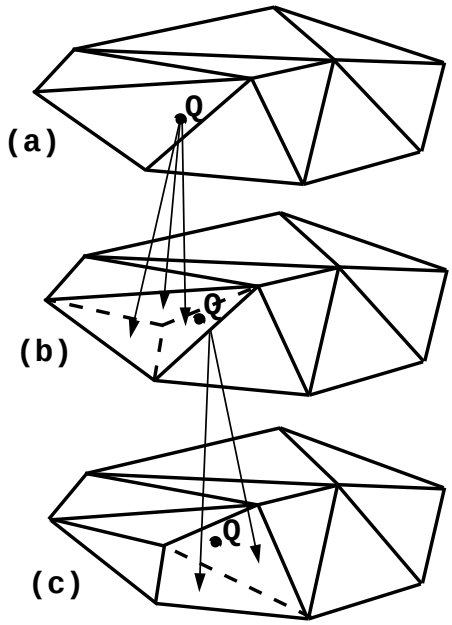

Figure 3.3.9 Data tree for site location. (a) Original triangulation containing Q. (b) Triangle is split into 3 from site insertion. (c) Triangle is split into 2 from edge swap.

The process would then continue until the search is complete. In this way we can think of the data structure as a tree with node branching factors of either two or three. In [GuiKS92] an amortized analysis reveals $O(\log N)$ depth per search yielding the overall $O(N \log N)$ complexity for the entire triangulation.

3.3d Delaunay Triangulation Via Global Edge Swapping

This algorithm due to Lawson [Law77] assumes that a triangulation exists (not Delaunay) then makes it Delaunay through application of edge swapping such that the equiangularity of the triangulation increases. The equiangularity of a triangulation, $A(T)$, is defined as the ordering of angles $A(T) = [\alpha_1, \alpha_2, \alpha_3, ..., \alpha_{3n(c)_3}]$ such that $\alpha_i \leq \alpha_j$ if $i < j$. We write $A(T^*) < A(T)$ if $\alpha_j^* \leq \alpha_j$ and $\alpha_i^* = \alpha_i$ for $1 \leq i < j$. A triangulation T is globally equiangular if $A(T^*) \leq A(T)$ for all triangulations T^* of the point set. Lawson's algorithm examines all interior edges of the mesh. Each of these edges represents the diagonal of the quadrilateral formed by the union of the two adjacent triangles. In general one must first check if the quadrilateral is convex so that a potential diagonal swapping can place without edge crossing. If the quadrilateral is convex then the diagonal position is chosen which optimizes a local criterion (in this case the local equiangularity). This amounts to maximizing the minimum angle of the two adjacent triangles. Lawson's algorithm continues until the mesh is locally optimized and locally equiangular everywhere. It is easily shown that the condition of local equiangularity is equivalent to satisfaction of the incircle test described earlier. Therefore a mesh which is locally equiangular everywhere is a Delaunay triangulation. Note that each new edge swapping (triangulation T^*) insures that the global equiangularity increases $A(T^*) > A(T)$. Because the triangulation is of finite dimension, this guarantees that the process will terminate in a finite number of steps. A more refined proof exploits the fact that the Delaunay triangulation problem in R^d is equivalent to a lower convex hull problem for a point set lifted onto unit paraboloid in R^{d+1}. In two dimensions the lower convex hull characterization reveals that once an edge in the triangulation is removed in Lawson's algorithm it can never return. Since the number of edges is $\binom{N}{2} = O(N^2)$, Lawson's algorithm must terminate after at most $O(N^2)$ swaps.

Iterative Algorithm: Triangulation via Lawson's Algorithm
swapedge = true
While(swapedge)do
  swapedge = false
  Do (*all interior edges*)
    If (*adjacent $\triangle$s form convex quad*)then
      Swap diagonal to form T^*.
      If (*optimization criteria satisfied*)then
        $T = T^*$
        swapedge = true
      EndIf
    EndIf
  EndDo
EndWhile

When Lawson's algorithm is used for constructing Delaunay triangulations, the test for quadrilateral convexity is not needed. It can be shown that nonconvex quadrilaterals formed from triangle pairs *never* violate the circumcircle test. When more general optimization criteria is used (discussed later), the convexity check must be performed.

3.4 Other 2-D Triangulation Algorithms

In this section, other algorithms which do not necessarily produce Delaunay triangulations are explored.

The MinMax Triangulation

As Babuška and Aziz [BabuA76] point out, from the point of view of finite element approximations, the MaxMin (Delaunay) triangulation is not essential. What is essential is that no angle be too close to 180°. In other words, triangulations which minimize the maximum angle are more desirable. These triangulations are referred to as MinMax triangulations. One way to generate a 2-D MinMax triangulations is via Lawson's edge swapping algorithm. In this case, the diagonal position for convex pairs of triangles is chosen which minimizes the maximum interior angle for both triangles. The algorithm is guaranteed to converge in a finite number of steps using arguments similar to Delaunay triangulation. Figures 3.4.0 and 3.4.1 present a Delaunay (MaxMin) and MinMax triangulation for 100 random points.

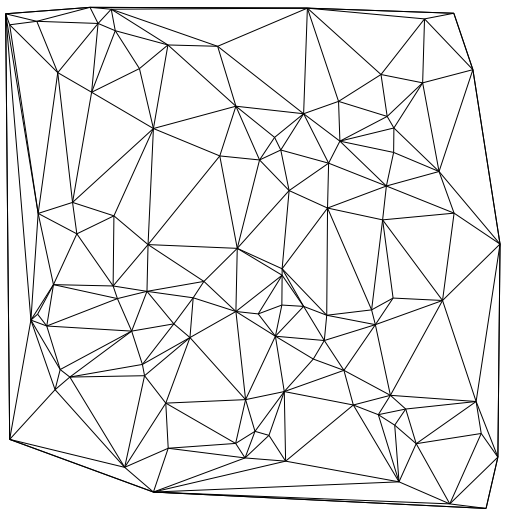

Figure 3.4.0 Delaunay Triangulation.

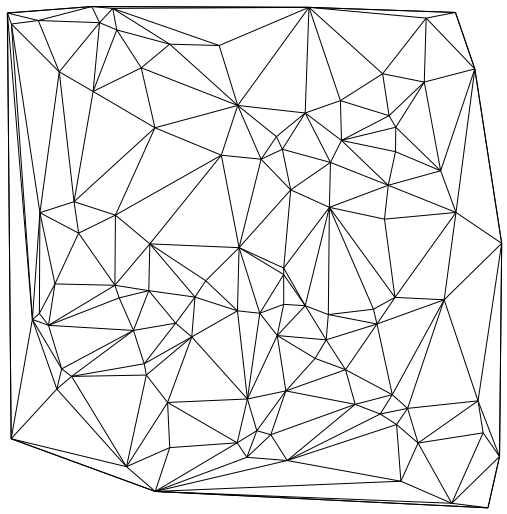

Figure 3.4.1 MinMax Triangulation.

Note that application of local MinMax optimization via Lawson's algorithm may only result in a mesh which is *locally* optimal and not necessarily at a global minimum. Attaining a globally optimal MinMax triangulation is a much more difficult task. The best algorithm to present date (Edelsbrunner, Tan, and Waupotitsch [Edel90]) has a high complexity of $O(n^2 \log n)$. The Green-Sibson algorithm can be modified to produce locally optimal MinMax triangulations using incremental insertion and local edge swapping. Again the basic idead is to replace the incircle test with another local optimization predicate. This added flexibility make the Green-Sibson algorithm attractive. The algorithm is implemented using recursive programming with complete forward and backward propagation (contrast figures 3.4.2 and 3.3.7). This is a necessary step to produce locally optimized meshes.

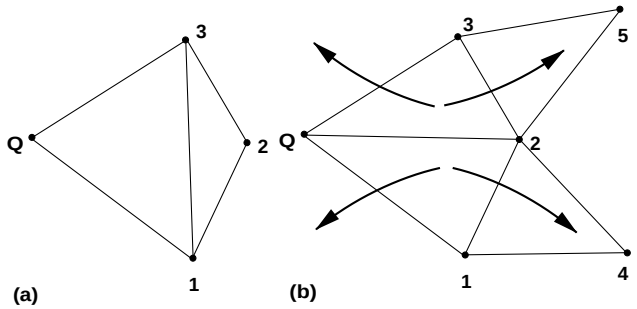

Figure 3.4.2 Edge swapping with forward and backward propagation.

The MinMax triangulation has proven to be very useful in CFD. Figure 3.4.3 shows the

Delaunay triangulation near the trailing edge region of an airfoil with extreme point clustering.

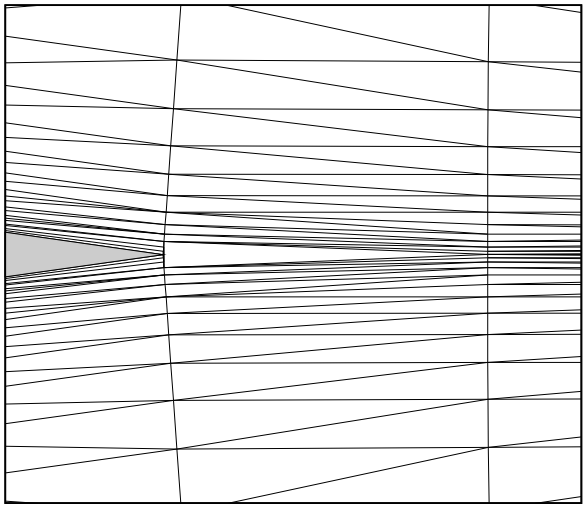

Figure 3.4.3 Delaunay triangulation near trailing edge of airfoil.

Upon first inspection, the mesh appears flawed near the trailing edge of the airfoil. Further inspection and extreme magnification near the trail edge of the airfoil (figure 3.4.4) indicates that the grid is a mathematically correct Delaunay triangulation.

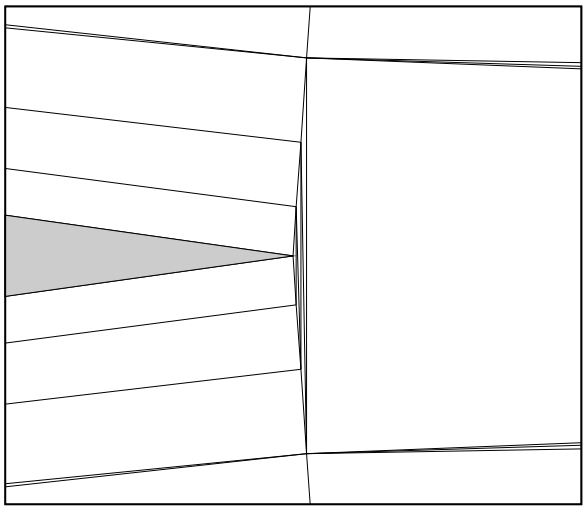

Figure 3.4.4 Extreme closeup of Delaunay triangulation near trailing edge of airfoil.

Because the Delaunay triangulation does not control the maximum angle, the cells near the trailing edge have angles approaching 180°. The presence of nearly collapsed triangles leaves considerable doubt as to the accuracy of any numerical solutions computed in the trailing edge region. Edge swapping based on the MinMax criteria via Lawson's algorithm

or incremental insertion using the modified Green-Sibson algorithm produces the desired result as shown in figure 3.4.5.

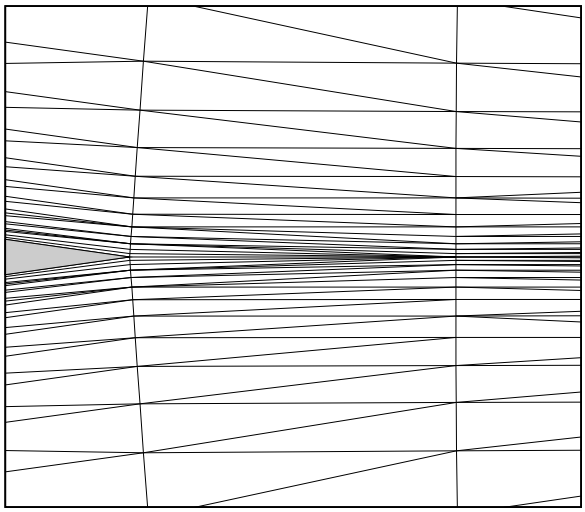

Figure 3.4.5 MinMax triangulation near trailing edge of airfoil.

The Greedy Triangulation

A greedy method is one that never undoes what it did earlier. The greedy triangulation continually adds edges compatible with the current triangulation (edge crossing not allowed) until the triangulation is complete, i.e. Euler's formula is satisfied. One objective of a triangulation might be to choose a set of edges with shortest total length. The best that the greedy algorithm can do is adopt a local criterion whereby only the shortest edge available at that moment is considered for addition to the current triangulation. (This does not lead to a triangulation with shortest total length.) Note that greedy triangulation easily accommodates constrained triangulations containing interior boundaries and a nonconvex outer boundary. In this case the boundary edges are simply listed first in the ordering of candidate edges. The entire algorithm is outlined below.

Algorithm: Greedy Triangulation

Step 1. Initialize triangulation T as empty.

Step 2. Compute $\binom{n}{2}$ candidate edges.

Step 3. Order pool of candidate edges.

Step 4. Remove current edge e_s from ordered pool.

Step 5. If(e_s *does not intersect edges of* T) add e_s to T

Step 6. If(*Euler's formula not satisfied*) go to Step 4.

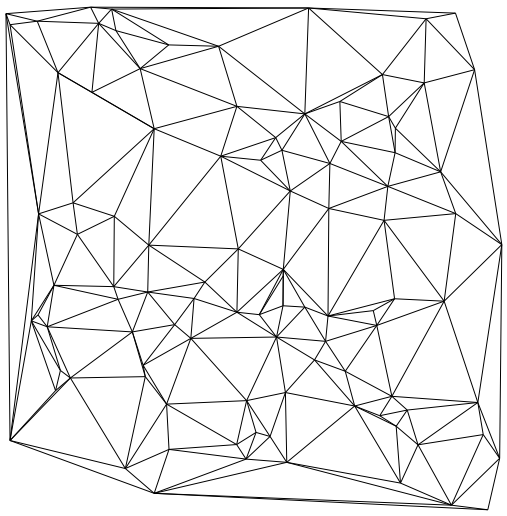

Figure 3.4.6 Greedy Triangulation.

Figures 3.4.0 and 3.4.6 contrast the Delaunay and greedy algorithm. The lack of angle control is easily seen in the greedy triangulation. The greedy algorithm suffers from both high running time as well as storage. In fact a naive implementation of Step 5. leads to an algorithm with $O(N^3)$ complexity. Efficient implementation techniques are given in Gilbert [Gil79] with the result that the complexity can be reduced to $O(N^2 \log N)$ with $O(N^2)$ storage.

Data Dependent Triangulation

Unlike mesh adaptation, a data dependent triangulation assumes that the number and position of vertices is fixed and unchanging. Of all possible triangulations of these vertices, the task is to find the best triangulation under data dependent constraints. In Nira, Levin, and Rippa [Nira90a], they consider several data dependent constraints together with piecewise linear interpolation. In order to determine if a new mesh is "better" than a previous one, a local cost function is defined for each interior edge. Two choices which prove to be particularly effective are the JND (Jump in Normal Derivatives) and the ABN (Angle Between Normals). Using their notation, consider an interior edge with adjacent triangles T_1 and T_2. Let $P(x,y)_1$ and $P(x,y)_2$ be the linear interpolation polynomials in T_1 and T_2 respectively:

$$P_1(x,y) = a_1 x + b_1 y + c_1$$

$$P_2(x,y) = a_2 x + b_2 y + c_2$$

The JND cost function measures the jump in normal derivatives of P_1 and P_2 across a common edge with normal components n_x and n_y.

$$s(f_T, e) = |n_x(a_1 - a_2) + n_y(b_1 - b_2)|, \qquad \text{(JND cost function)}$$

The ABN measures the acute angle between the two normals formed from the two planes P_1 and P_2. Using the notation of [Nira90a]:

$$s(f_T, e) = \theta = \cos^{-1}(A)$$

$$A = \frac{a_1 a_2 + b_1 b_2 + 1}{\sqrt{(a_1^2 + b_1^2 + 1)(a_2^2 + b_2^2 + 1)}}, \qquad \text{(ABN cost function)}$$

The next step is to construct a global measure of these cost functions. This measure is required to decrease for each legal edge swapping. This insures that the edge swapping process terminates. The simplest measures are the l_1 and l_2 norms:

$$R_1(f_T) = \sum_{edges} |s(f_T, e)|$$

$$R_2(f_T) = \sum_{edges} s(f_T, e)^2$$

Recall that a Delaunay triangulation would result if the cost function is chosen which maximizes the minimum angle between adjacent triangles (Lawson's algorithm). Although it would be desirable to obtain a global optimum for all cost functions, this could be very costly in many cases. An alternate strategy is to abandon the pursuit of a globally optimal triangulation in favor of a locally optimal triangulation. Once again Lawson's algorithm is used. Note that in using Lawson's algorithm, we require that the global measure decrease at each edge swap. This is not as simple as before since each edge swap can have an effect on other surrounding edge cost functions. Nevertheless, this domain of influence is very small and easily found.

Iterative Algorithm: Data Dependent Triangulation via Modified Lawson's Algorithm
swapedge = true
While(swapedge)do
  swapedge = false
  Do (*all interior edges*)
    If (*adjacent triangles*
      form convex quadrilateral)then
      Swap diagonal to form T^*.
      If ($R(f_{T^*}) < R(f_T)$)then
        $T = T^*$
        swapedge = true
      EndIf
    EndIf
  EndDo
EndWhile

Edge swapping only occurs when $R(f_{T^*}) < R(f_T)$ which guarantees that the method terminates in a finite number of steps. Figures 3.4.0 and 3.4.7 plot the Delaunay triangulation of

100 random vertices in a unit square and piecewise linear contours of $(1+\tanh(9y-9x))/9$ on this mesh. The exact solution consists of straight line contours with unit slope.

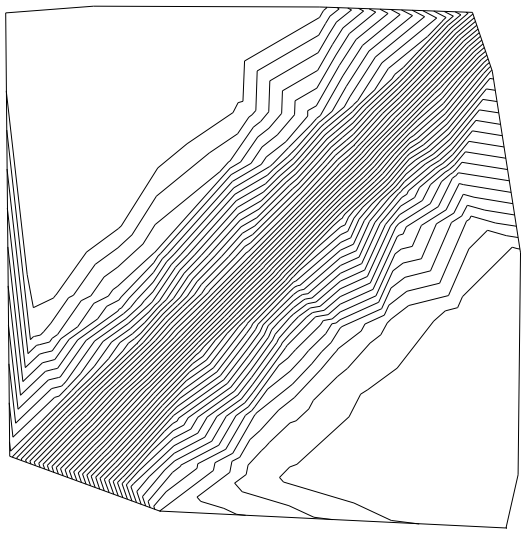

Figure 3.4.7 Piecewise Linear Interpolation of $(1+\tanh(9y-9x))/9$.

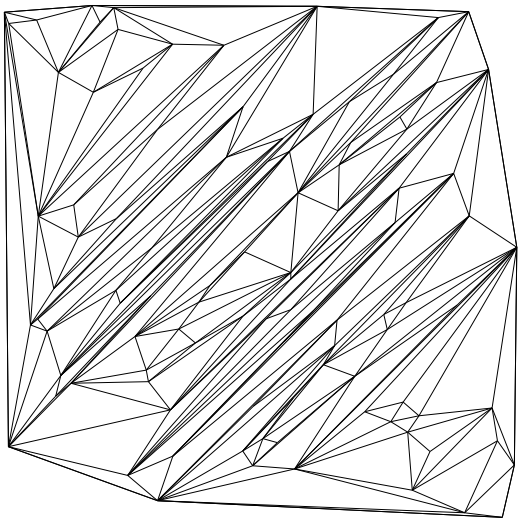

Figure 3.4.8. Data Dependent Triangulation.

In figures 3.4.8 and 3.4.9 the data dependent triangulation and solution contours using the JND criteria and l_1 measure are plotted.

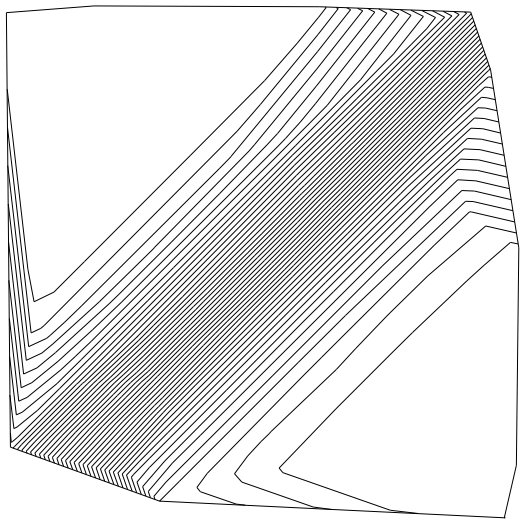

Figure 3.4.9 Piecewise Linear Interpolation of $(1 + \tanh(9y - 9x))/9$.

Note that the triangulations obtained from this method are not globally optimal and highly dependent on the order in which edges are accessed. Several possible ordering strategies are mentioned in [Nira90b].

3.5 2-D Steiner Triangulations

Definition: A Steiner triangulation is any triangulation that adds additional sites to an existing triangulation to improve some measure of grid quality [BernE92].

The insertion algorithms described earlier also provide a simple mechanism for generating Steiner triangulations. In the following paragraphs we consider different strategies for the computation of Steiner triangulations in 2-D. The first strategy produces nearly equilateral triangles. These meshes are suitable for the finite-element and finite-volume computation of inviscid fluid flows. The second strategy gives preference to the generation of right triangles, thus allowing the element shape to attain a high aspect ratio (the ratio of circumcircle to incircle). These meshes are suitable for the calculation of viscous fluid flow at high Reynolds numbers.

Circumcenter Refinement

A number of researchers have independently discovered the feasibility of inserting sites at circumcenters of Delaunay triangles into an existing 2-D triangulation to improve measures of grid quality, [HolS88], [Chew90], and [Rupp92]. This strategy has the desired effect of placing the new site in a position that guarantees that no other site in the triangulation can lie closer than the radius of the circumcircle, see figure 3.5.0. In a loose sense, the new site is placed as far away from other nearby sites as conservatively possible.

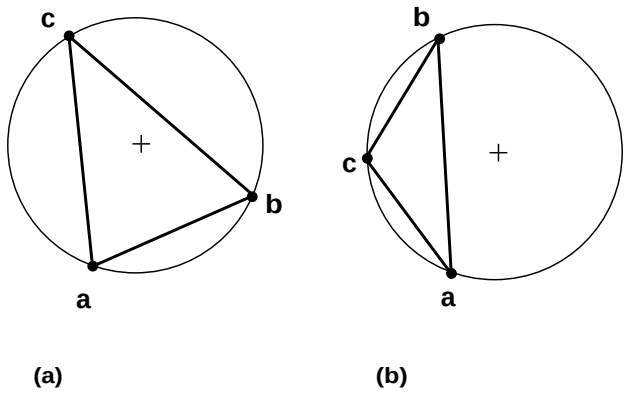

Figure 3.5.0 Inserting site at circumcenter of acute and obtuse triangles.

Most algorithms then follow a procedure similar to that proposed by Chew:

Algorithm: Steiner Triangulation, [Chew90]

Step 1. Construct a constrained Delaunay triangulation of the boundary points and edges.
Step 2. Compute a measure of shape and size for each element. A triangle passes only if: (1) it is well-shaped, i.e. the smallest angle is greater than 30°, (2) it is well-sized, i.e. the triangle passes measure of size (area, containment circle size, etc.). Any size measure can be specified as long as it can be achieved by making the triangle smaller.
Step 3. If all triangles pass then halt. Otherwise choose the largest triangle, $\triangle$, which fails and determine its circumcenter, c.
Step 4. Traverse from $\triangle$ toward c until either a constraining boundary edge is encountered or the triangle containing c is found.
Step 5. If a triangle is found containing c then insert c into the triangulation and procede to *Step 2.*
Step 6. If a boundary edge is encountered during the traversal then split the boundary edge into halves and update the triangulation. Let l be the length of the new edges and consider the new vertex located on the boundary. Delete each interior vertex of the triangulation which is closer than l to this boundary site. Procede to *Step 2.*

Using this algorithm it is proven in [Chew90], [Chew93] that guaranteed-quality meshes are obtained:

(1) All angles in the triangulation lie between 30° and 120°. (Smaller angles are required if boundary angles less that 30° are allowed).

(2) All triangles will pass the user specified measure.

(3) All boundary edge constraints will be preserved by the final triangulation.

Figure 3.5.1 shows Step 1 of the algorithm. The triangulation is constrained to the boundary edge set using local transformations, see [BernE90] or [George91]. Note that in

Step 2 it is possible to maintain a *dynamic heap* data structure of the quality measure. (Heap structures are a very efficient way of keeping a sorted list of entries with insertion and query time $O(\log N)$ for N entries.) The triangle with the largest value of the specified measure will be located at the top of the heap at all times during the triangulation. In Step 5 sites are inserted using the Green-Sibson algorithm with boundary refinement as discussed in Step 6. The resulting triangulation in shown in fig. 3.5.2.

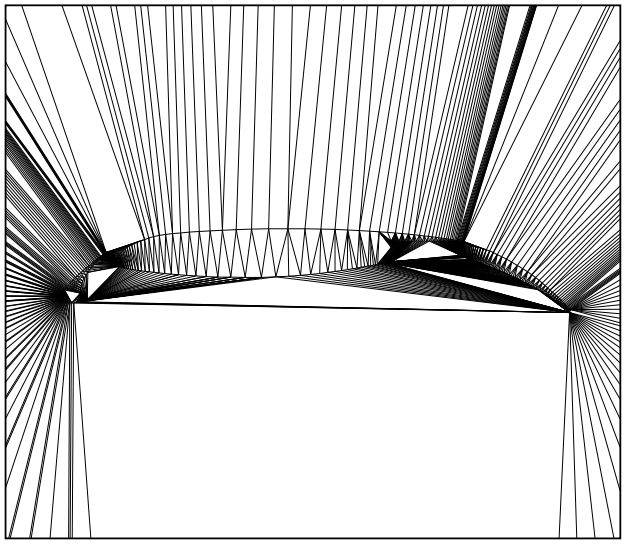

Figure 3.5.1 Initial triangulation of boundary points.

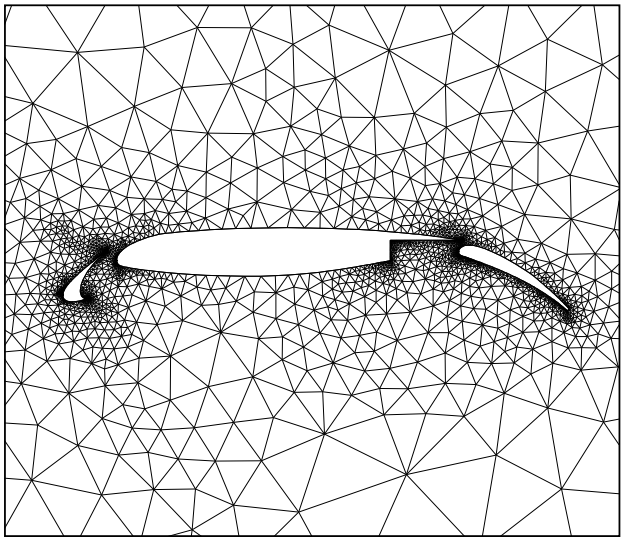

Figure 3.5.2 Steiner triangulation with sites inserted at circumcenters to reduce maximum cell aspect ratio.

This triangulation has proven to be very flexible. For instance, figure 3.5.3 shows a Steiner triangulation of the Texas coast and Gulf of Mexico.

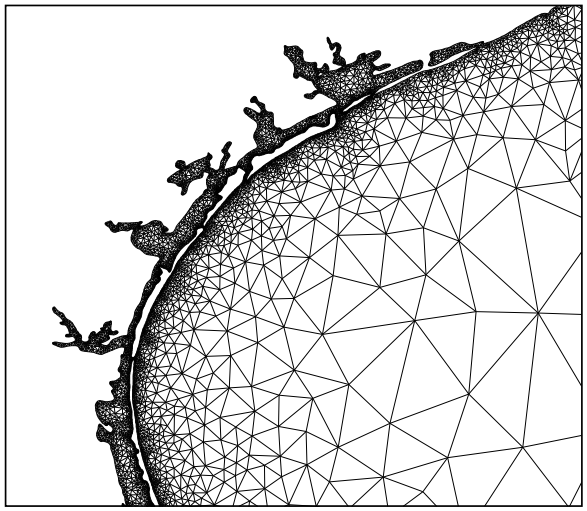

Figure 3.5.3 Steiner triangulation of Texas coast and the Gulf of Mexico.

Stretched Triangle Refinement

In this approach we consider a related strategy for the generation of stretched triangulations. The stretching is based on a parameterization of a scalar function representing the minimum Euclidean distance from interior vertices to boundary segments. The basic Steiner point insertion algorithm follows closely that discussed in the previous section with two notable differences:

(1) Determination of candidate Steiner point locations.

(2) Proximity filtering of candidate Steiner point locations.

The user specifies a one-dimensional function of desired mesh spacings (shown in fig. 3.5.4) which is parameterized by the minimum Euclidean distance function, S, so that $\Delta r = \Delta r(S)$.

Candidate Steiner point locations for each triangle are determined in the following way. For each interior vertex v of the existing triangulation we first find a point v_b on the boundary which achieves minimum Euclidean distance from v_b to v. This minimum distance, S, is stored for all vertices. In addition we compute and store a unit vector from v_b to v, $\widehat{s}$. Thus at each vertex of a triangle we have the distance function S, the direction vector $\widehat{s}$, and the desired mesh spacing Δr. In Step 2 of Chew's algorithm, a triangle measure is chosen which is a ratio of the maximum dimension of the triangle projected along the vector $\widehat{s}$ to the desired mesh spacing Δr.

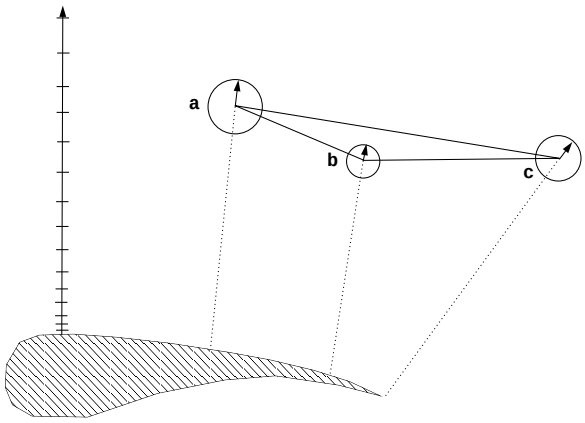

Figure 3.5.4 Typical triangle abc showing the determination of distance to solid boundaries $S(a), S(b), S(c)$ and the associated direction vectors, $\widehat{s}$, at a, b, and c.

In Step 3 candidate Steiner point locations are obtained from values of $\widehat{s}$ and Δr at the three vertices of a triangle. The candidate Steiner points are placed a distance Δr from the vertices of the triangle along the direction vectors $\widehat{s}$, see fig. 3.5.4. Candidate Steiner point locations are eliminated (filtered) if *any* other vertex in the existing triangulation lies within radius Δr from the candidate point. This proximity query is performed using a dynamic tree structure of the current triangulation vertex set. Of the remaining candidates, the optimal candidate is determined which produces the smallest maximum angle when inserted into the triangulation.

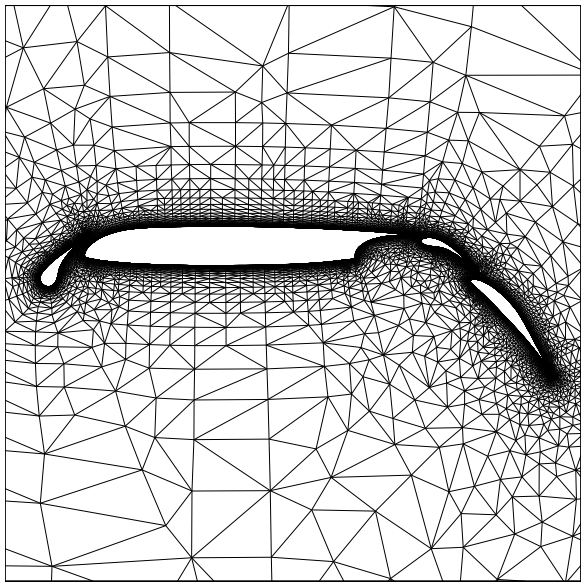

Figure 3.5.5 Stretched Triangulation Computed About Multi-element Airfoil.

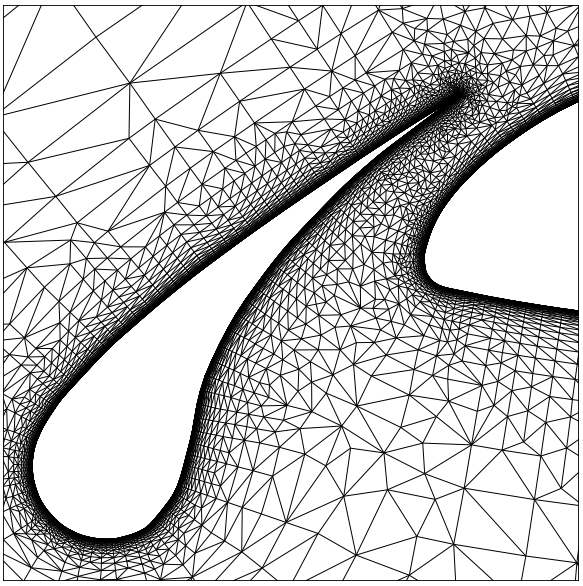

Figure 3.5.6 Closeup of Triangulation Computed About Multi-element Airfoil.

Figures 3.5.5 and 3.5.6 show a typical mesh generated about a multi-element airfoil using this strategy. The minimum wall spacing was prespecified at .0001 chord units. The maximum angle generated by the algorithm is approximately 135°. Figure 3.5.7 shows an extreme closeup near the leading edge of the main element showing the highly stretched elements.

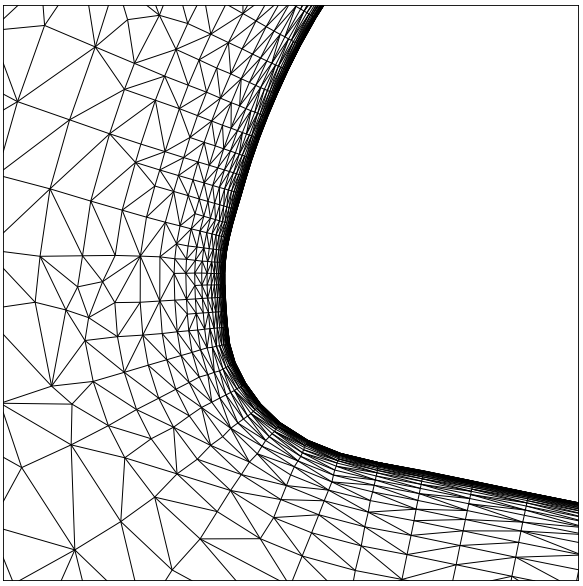

Figure 3.5.7 Extreme Closeup of Triangulation Computed About Multi-element Airfoil.

Mesh Adaptation

One clear advantage of triangulations is the ability to locally refine the mesh. The incremental triangulation algorithms earlier all extend naturally to include mesh adaptation. (This is the good news.) One strategy is to modify Step 2 of the Chew algorithm to include

solution dependent measures.

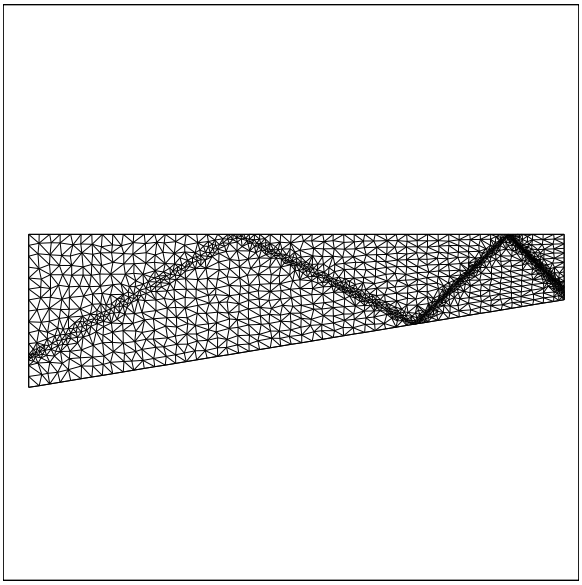

Figure 3.5.8 Adapted mesh obtained from supersonic channel flow computation showing reflecting shock pattern.

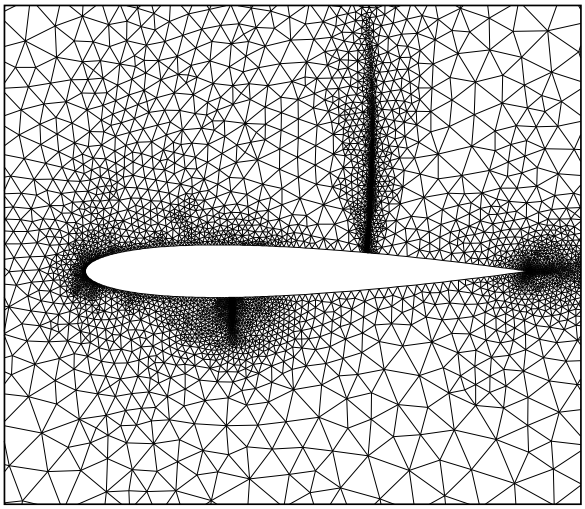

Figure 3.5.9 Adapted mesh obtained from transonic airfoil computation showing shock and slip lines.

In Step 3 of the Chew algorithm circumcenter location are chosen for site insertion. Note that although the theory of solution adaptive meshing is well developed for elliptic equations, the bad news is that a general theory suitable for nonlinear hyperbolic systems of equations is still under development. Several subtle issues surround the development of a general theory. A discussion of mesh adaptation theory and error estimation is well beyond the scope of these notes and will not be pursued. Even so, solution dependent measures

based on heuristic arguments can often yield impressive results. Figures 3.5.8 and 3.5.9 show density gradient-adapted triangulations obtained from two fluid flow computations. These calculations are presented in detail in section 6.0.

3.6 Three-Dimensional Triangulations

The Delaunay triangulation extends naturally into three dimensions as the geometric dual of the 3-D Voronoi diagram. The Delaunay triangulation in 3-D can be characterized as the unique triangulation such that the circumsphere passing through the four vertices of any tetrahedron must not contain any other point in the triangulation. As in the 2-D case, the 3-D Delaunay triangulation has the property that it minimizes the maximum containment sphere. In two dimensions, it can be shown that a mesh entirely comprised of acute triangles is automatically Delaunay. To prove this, consider an adjacent triangle pair forming a quadrilateral. By swapping the position of the diagonal it is easily shown that the minimum angle always increases. Rajan [Raj91] shows the natural extension of this idea to three or more space dimensions. He defines a "self-centered" simplex in $\mathbf{R}^d$ to be a simplex which has the circumcenter of its circumsphere interior to the simplex. In two dimensions, acute triangles are self-centered and obtuse triangles are not. Rajan shows that a triangulation entirely composed of self-centered simplices in $\mathbf{R}^d$ is automatically Delaunay.

3.6a 3-D Bowyer and Watson Algorithms

The algorithms of Bowyer [Bow81] and Watson [Wat81] extend naturally to three dimensions with estimated complexities of $O(N^{5/3})$ and $O(N^{4/3})$ for N randomly distributed vertices. They do not give worst case estimates. It should be noted that in three dimensions, Klee [Klee80] shows that the maximum number of tetrahedra which can be generated from N vertices is $O(N^2)$. Thus an optimal worst case complexity would be a least $O(N^2)$. Under normal conditions this worst case scenario is rarely encountered.

3.6b 3-D Edge Swapping Algorithms

Until most recently, the algorithm of Green and Sibson based on edge swapping was thought not to be extendable to three dimensions because it was unclear how to generalize the concept of edge swapping to three or more dimensions. In 1986, Lawson [Law86] published a paper in which he proved the fundamental combinatorial result:

Theorem: (Lawson, 1986) The convex hull of $d+2$ points in $\mathbf{R}^d$ can be triangulated in at most 2 ways.

Joe [Joe89,91], and Rajan [Raj91] have constructed algorithms based on this theorem. In joint work with A. Gandhi [GanB93], we independently constructed an incremental Delaunay triangulation algorithm based on Lawson's theorem. The remainder of this section will review our algorithm and the basic ideas behind 3-D edge swapping algorithms.

It is useful to develop a taxonomy of possible configurations addressed by Lawson's theorem. Figure 3.6.0 shows configurations in 3-D of five points which can be triangulated in only one way and hence no change is possible. We call these arrangements "nontransformable". Figure 3.6.1 shows configurations which allow two ways of triangulation. It is possible to flip between the two possible triangulations and we call these arrangements

"transformable". These figures reveal an important difference between the two and three-dimensional algorithms. *The number of tetrahedrons involved in the swapping operation need not be constant.*

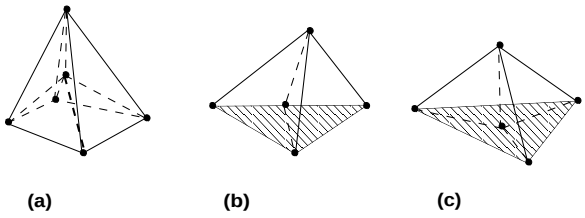

Figure 3.6.0 Generic nonswappable configurations of 5 points. Shaded region denotes planar surface.

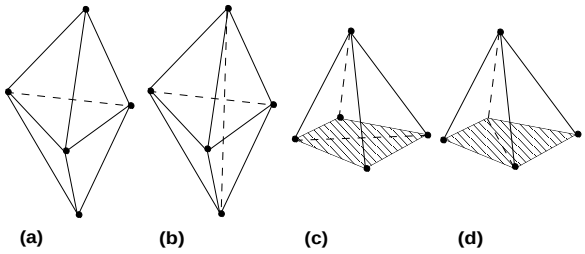

Figure 3.6.1 Generic swappable configurations of 5 points. Shaded region denotes planar surface.

There are two arrangements that allow two triangulations. Figures 3.6.1(a) and 3.6.1(b) show the subclass of companion triangulations that can be transformed from one type to another thereby changing the number of tetrahedra from 2 to 3 or vice-versa. Figures 3.6.1(c) and 3.6.1(d) show the other subclass of configurations that can be transformed from one type to another while keeping constant the number of tetrahedra (2). It can be shown that all nontransformable configurations satisfy the circumsphere characterization of the Delaunay triangulation. Among the class of transformable configurations if the particular configuration fails the insphere test it must necessarily pass in the other configuration.

Incremental Delaunay Triangulation

The local transformation described above provide the essential machinery for extending the Green-Sibson algorithm into R^d.

Algorithm: Incremental Delaunay Triangulation Via Local Transformations

Step 1. Locate existing tetrahedron enclosing point Q.

Step 2. Insert site and connect to 4 or 5 surrounding vertices.

Step 3. Identify suspect faces.

Step 4. Perform edge swapping of all suspect faces failing the insphere test.

Step 5. Identify new suspect faces.

Step 6. If new suspect faces have been created, go to step 3.

The correctness of this algorithm has been proven by Joe [Joe89]. The above algorithm can be implemented using recursive programming techniques with forward propagation. One distinct advantage of the present technique is that it allows other criteria for edge swapping to be used. The only additional complexity involves the inclusion of back propagation in the recursion process. Given the experience gained in two dimensions concerning stretched triangulations, this added flexibility appears highly advantageous.

3.6c 3-D Surface Triangulation

The modified Green-Sibson algorithm has been extended to include the triangulation of surface patches. Although the concept of Dirichlet tessellation is well defined on a smooth manifolds using the concept of geodesic distance, in practice this is too expensive. Finding geodesic distance is a variational problem that is not easily solved. We have implemented a simpler procedure in which surface grids in 3-D are constructed from rectangular surface patches (assumed at least C^0 smooth) using a generalization of the 2-D Steiner triangulation scheme.

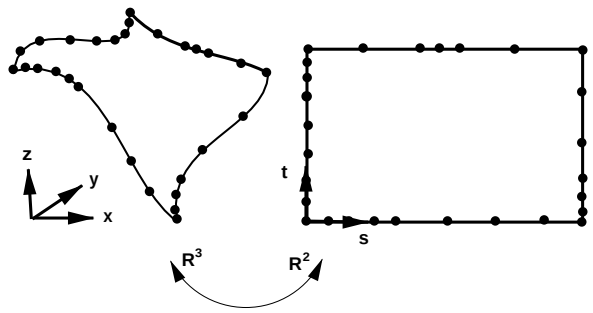

Figure 3.6.2 Mapping of rectangular patches on (s,t) plane.

Points are first placed on the perimeter of each patch using an adaptive refinement strategy based on absolute error and curvature measures. The surface patches are projected onto the plane, see figure 3.6.2. Simple stretching of the rectangular patches permits the user to produce preferentially stretched meshes. (This is useful near the leading edge of a wing.)

The triangulation takes place in the two dimensional (s,t) plane. The triangulation is adaptively refined using Steiner point insertion to minimize the maximum user specified absolute error and curvature tolerance on each patch. The absolute error is approximated by the perpendicular distance from the triangle centroid (projected back to 3-space) to the true surface as depicted in figure 3.6.3.

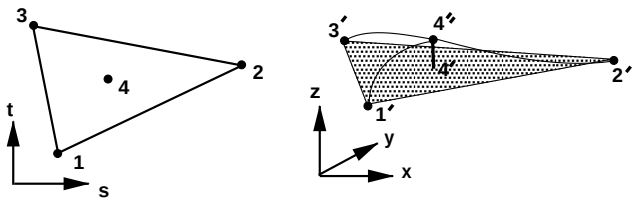

Figure 3.6.3 Calculation of triangulation absolute error by measurement of distance from face centroid to true surface.

The user can further refine based on triangle aspect ratio in the (s,t) plane if desired. Figure 3.6.4 shows a typical adaptive surface grid generated using the Steiner triangulation method.

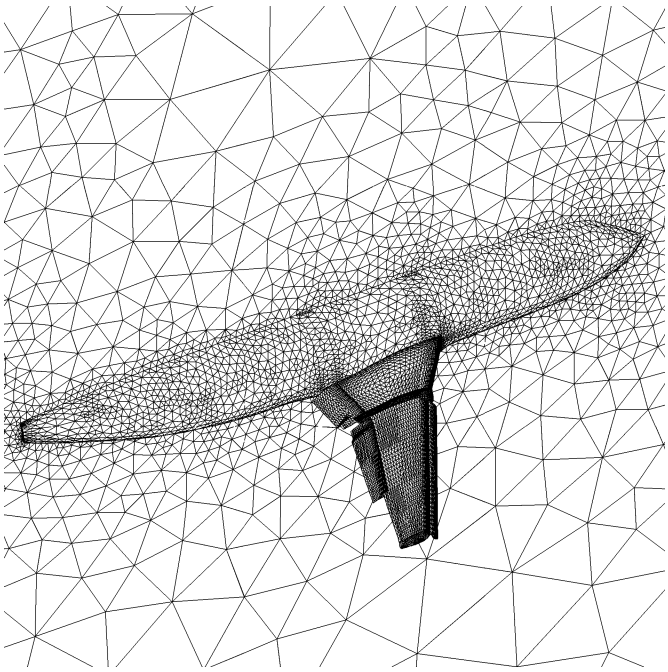

Figure 3.6.4 Adaptive Steiner triangulation of surface geometry about Boeing 737 with flaps deployed.

3.7 Computational Aspects of Triangulation

In practical application, it is important to understand the impact of finite-precision arithmetic in the construction of two and three-dimensional triangulations using the Green-Sibson and Joe algorithms. For purposes of this discussion we assume that the sites to be inserted are *prespecified* and the task at hand is to insert them (correctly) into the existing triangulation.

Convexity Test:

The convexity test determines whether the shape formed by the tetrahedron $\mathcal{T}_1$ and $\mathcal{T}_2$ is convex. Let the vertices of the tetrahedra be numbered $\mathcal{T}_1 = (1,2,3,4)$ and $\mathcal{T}_2 = (1,2,3,5)$ i.e., $(1,2,3)$ is the face shared by $\mathcal{T}_1$ and $\mathcal{T}_2$ and nodes 4 and 5 are at the two ends of $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively. We make use of the notion of barycentric coordinates to perform the convexity test. The $b_{1,2,3,4}$ satisfying

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} 1 \\ x_5 \\ y_5 \\ z_5 \end{bmatrix}$$

are called the barycentric coordinates of node 5. They indicate the position of 5 in relation to the nodes of tetrahedron $\mathcal{T}_1$. For each s, the sign of b_s indicates the position of 5 relative to the plane H_s passing through the triangular face opposite node s. Thus $b_s = 0$ when 5 is in H_s, $b_s > 0$ when 5 is on the same side of H_s as node s, and $b_s < 0$ when 5 is on the opposite side of H_s from node s. Clearly, $b_{1,2,3,4} > 0$ if 5 lies inside tetrahedron $(1,2,3,4)$. If we imagine a cone formed by planes $H_{1,2,3}$ of $\mathcal{T}_1$, then $\mathcal{T}_1$ and $\mathcal{T}_2$ would form a convex shape if and only if node 5 lies in the cone on the side opposite from node 4 (figure 3.7.0).

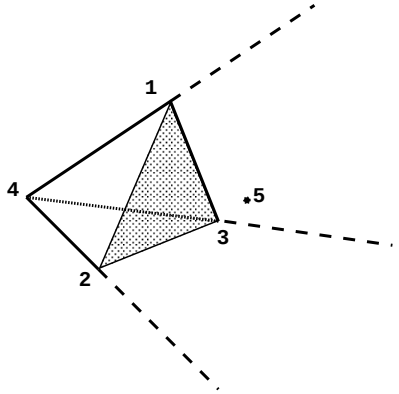

Figure 3.7.0 Convexity cone for node 4.

This requires that $b_4 < 0$ and $b_{1,2,3} > 0$. Thus we observe that convexity only requires knowledge of the *sign* of each b_i. This simplifies the task considerably and reduces the problem to the following:

$$b_1 = sign \begin{vmatrix} 1 & 1 & 1 & 1 \\ x_5 & x_2 & x_3 & x_4 \\ y_5 & y_2 & y_3 & y_4 \\ z_5 & z_2 & z_3 & z_4 \end{vmatrix}, \quad b_2 = sign \begin{vmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_5 & x_3 & x_4 \\ y_1 & y_5 & y_3 & y_4 \\ z_1 & z_5 & z_3 & z_4 \end{vmatrix} \qquad (3.7.0a-b)$$

$$b_3 = sign \begin{vmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_5 & x_4 \\ y_1 & y_2 & y_5 & y_4 \\ z_1 & z_2 & z_5 & z_4 \end{vmatrix}, \quad b_4 = sign \begin{vmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_5 \\ y_1 & y_2 & y_3 & y_5 \\ z_1 & z_2 & z_3 & z_5 \end{vmatrix} \qquad (3.7.0c-d)$$

These computations effectively determine the orientation of the tetrahedrons $(5,2,3,4)$, $(1,5,3,4)$, $(1,2,5,4)$, $(1,2,3,5)$ formed from the configuration.

Circumsphere Test:

The 3-D Delaunay triangulation is defined as the unique triangulation such that the circumsphere of any tetrahedron contains no other point in the mesh. To determine where point e lies in relation to the circumsphere of tetrahedron (a, b, c, d), denoted by$(\bigcirc\ abcd)$, we use the *InSphere* primitive :

$$InSphere(abcde) = \begin{cases} 1 & \text{if } e \text{ is inside } \bigcirc\ abcd \\ 0 & \text{if } e \text{ is on } \bigcirc\ abcd \\ -1 & \text{if } e \text{ is outside } \bigcirc\ abcd \end{cases}$$

where *InSphere* is computed from the following determinant:

$$InSphere(abcde) = sign \begin{vmatrix} 1 & 1 & 1 & 1 & 1 \\ x_a & x_b & x_c & x_d & x_e \\ y_a & y_b & y_c & y_d & y_e \\ z_a & z_b & z_c & z_d & z_e \\ w_a^2 & w_b^2 & w_c^2 & w_d^2 & w_e^2 \end{vmatrix}$$

and $w_p^2 = x_p^2 + y_p^2 + z_p^2$

This test is motivated by the observation in 3-D that the intersection of a cylinder and a unit paraboloid is an ellipse lying in a plane (figure 3.6.3). So, any four co-circular points in 2-D will project to four co-planar points and the volume of the tetrahedra made from these four co-planar points will be zero. The paraboloid is a surface that is convex upward and the points interior to a circle get projected to the paraboloid below the intersection plane and the points exterior to it get projected above the the intersection plane. If a point lies outside the circumcircle of three other points, the tetrahedron made from these four points will have positive volume provided the three points were ordered in a counter-clockwise fashion. The volume will be negative if the point lies inside the circumcircle.

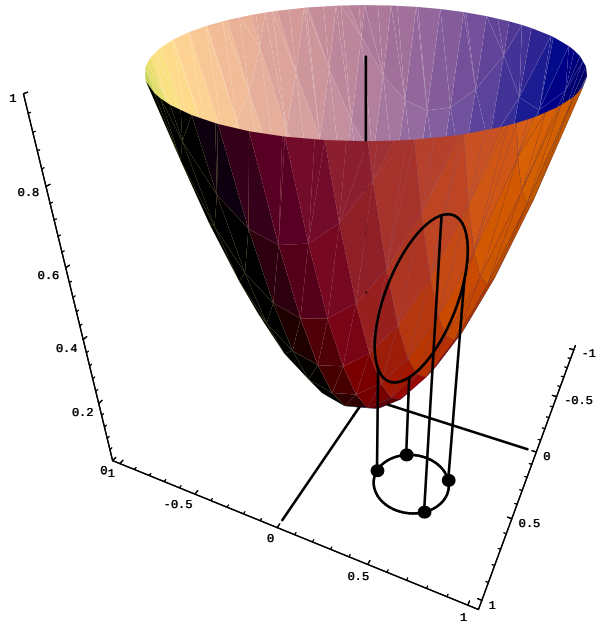

Figure 3.6.3 Projection of cocircular points onto unit paraboloid.

Robust Triangulation:

In this section we assume that the (predetermined) sites to be inserted can be represented exactly by b bit mantissas. IEEE single precision floating point representations have a 24 bit mantissa length while double precision floating point representations have a 53 bit mantissa length. In practice the accuracy to which geometry information can be obtained is far less that 53 bits. In typical 3-D applications we usually insist by least significant bit truncation that all site coordinates be exactly represented by 36 bits. This simplifies the task described below.

An important observation is that floating point arithmetic is only required for the calculation of the ternary decision predicates for convexity and the circumsphere test given $d+2$ sites in R^d. Consequently, these predicates can be evaluated exactly *since they only require the operations of addition, subtraction, and multiplication. Notably absent is the operation of division.*

To see why this is true, recall from the previous sections that the determination of convexity and the circumsphere test can both be reduced to the problem of determinant evaluation. Moreover, the determinant evaluation is equivalent to the problem of polynomial evaluation which only requires the operations of addition, subtraction, and multiplication. Also note that the decision predicates are *ternary*, that is to say that the determinants can be positive, negative, or zero. The latter response indicates a degeneracy situation which may call for a "tie-breaking" decision. A rather elaborate and elegant theory for handling these situations has been developed by Knuth [Knu92]. In the context of the Green-Sibson or Joe algorithms, choosing one possibility over another never leads to a topologically incorrect triangulation but can produce a final triangulation which deviates from a true Delaunay triangulation.

In actual computer implementation, the exact computation of these predicates can be carried out using conventional arithmetic on *redundant expressions*. One example of a redundant expression is the following formula:

$$a2^{28} + b2^{14} + c, \quad |a|, |b|, |c| < 2^{29}.$$

The use of redundant expressions is illustrated in the following computer code for the calculation the arithmetic dot product $sign(x_1y_1+x_2y_2+x_3y_3)$ assuming 28 bit coordinates.

Example [Knu93]: Exact Redundant Expression Calculation.

Let $s_i = sign(x_i * y_i)$, if all $s_i \geq 0$ or $s_i \leq 0$ then the answer is clear. Otherwise reorder and possibly negate the data so that $s_1 = s_2 \geq 0$, $s_3 < 0$ and all terms $x_i * y_i \geq 0$.

```
{ register long lx,rx,ly,ry;
  lx = x1/#4000;
  rx = x1 % #4000;
  ly = y1/#4000;
  ry = y1 % #4000;
  a = lx*ly; b = lx*ry + ly*rx;
  c = rx*ry;
```

```
  lx = x2/#4000; rx = x2 % #4000;
  ly = y2/#4000; ry = y2 % #4000;
  a += lx*ly; b += lx*ry + ly*rx;
  c += rx*ry;
  lx = x3/#4000; rx = x3 % #4000;
  ly = y3/#4000; ry = y3 % #4000;
  a -= lx*ly; b -= lx*ry + ly*rx;
  c -= rx*ry;
  if(a≡0) goto ez;
  if(a < 0) a= -a,b=-b,c=-c,s3=-s3;
  while(c < 0) {;
   a--; c+= #10000000;
   if(a≡0) goto ez;
  }
  if(b ≥ 0) return -s3;
  b = -b;
  a-=b/#4000;
  if(a > 0)return -s3;
  if(a ≤ -2) return s3;
  return -s3*((a*#4000
      - b%#4000)*#4000 + c);
 ez:if(b ≥ #8000) return -s3;
  if(b ≤ -#8000)return s3;
  return -s3*(b*#4000 + c);
}
```

where $\#4000 = 2^{14}, \#10000000 = 2^{28}$.

Fortune and Van Wyk [FortW93] have also examined this problem and have proposed procedures which use interval estimates of the error in finite precision calculations so that exact evaluation is rarely needed.

4.0 Maximum Principle Analysis

One of the best known tools employed in the study of differential equations is the maximum principle. Any function $f(x)$ which satisfies the inequality $f'' > 0$ on the interval $[a, b]$ attains its maximum value at one of the endpoints of the interval. Solutions of the inequality $f'' > 0$ are said to satisfy a maximum principle. Functions which satisfy a differential inequality in a domain Ω and because of the form of the differential equation achieve a maximum value on the boundary $\partial\Omega$ are said to possess a maximum principle. Recall the maximum principle for Laplace's equation. Let $\Delta u \equiv u_{xx} + u_{yy}$ denote the Laplace operator. If a function u satisfies the strict inequality

$$\Delta u > 0 \tag{4.0.0}$$

at each point in Ω, then u cannot attain its maximum at any interior point of Ω. The strict inequality can be weakened

$$\Delta u \geq 0 \tag{4.0.1}$$

so that if a maximum value M is attained in the interior of Ω then the entire function must be a constant with value M. Without any change in the above argument, if u satisfies the inequality

$$\Delta u + c_1 u_x + c_2 u_y > 0 \tag{4.0.2}$$

in Ω, then u cannot attain its maximum at an interior point.

The second model equation of interest is the nonlinear conservation law equation:

$$u_t + (f(u))_x = 0, \quad \frac{df}{du} = a(u) \tag{4.0.3}$$

In the simplest setting the initial value problem is considered in which the solution is specified along the x-axis, $u(x,0) = u_0(x)$ in a periodic or compact supported fashion. The solution can be depicted in the $x-t$ plane by a series of converging and diverging characteristic straight lines. From the solution of (4.0.3) Lax provides the following observation: *the total increasing and decreasing variations of a differentiable solution between any pairs of characteristics are conserved.*

$$\mathcal{I}(t+t_0) = I(t_0), \quad \mathcal{I}(t) = \int_{-\infty}^{+\infty} \left| \frac{\partial u(x,t)}{\partial x} \right| d\,x$$

Moreover in the presence of entropy satisfying discontinuites the total variation decreases (information is destroyed) in time.

$$\mathcal{I}(t+t_0) \leq I(t_0) \tag{4.0.4}$$

An equally important consequence of Lax's observation comes from considering a monotonic solution between two nonintersecting characteristics: *between pairs of characteristics, monotonic solutions remain monotonic*, no new extrema are created. Also from (4.0.4) we have that

(1) Local maxima are nonincreasing
(2) Local minima are nondecreasing

These properties of the differential equations serve as basic design principles for numerical schemes which approximate them. In the next section we review the fundamental theory surrounding discrete matrix operators equipped with maximum principles.

4.1 Discrete Maximum Principles for Elliptic Equations

Laplace's Equation on Structured Meshes

Consider Laplace's equation with Dirichlet data

$$\begin{gathered} \mathcal{L}u = 0, x, y \in \Omega \\ u = g, x, y \in \partial\Omega \\ \mathcal{L} = \Delta \end{gathered} \tag{4.1.0}$$

From the maximum principle property we have that

$$\sup_{x\in\Omega} |u(x,y)| \leq \sup_{x\in\partial\Omega} |u(x,y)|$$

For simplicity consider the unit square domain

$$\Omega = \{(x,y) \in R^2 : 0 \leq x, y \leq 1\}$$

with spatial grid $x_j = j\Delta x$, $y_k = k\Delta x$, and $J\Delta x = 1$. Let $U_{j,k}$ denote the numerical approximation to $u(x_j, y_k)$. It is well known that the standard second order accurate approximation

$$\mathcal{L}_\Delta U = \frac{1}{\Delta x^2}[U_{j+1,k} + U_{j-1,k} + U_{j,k+1} + U_{j,k-1} - 4U_{j,k}] \tag{4.1.1}$$

exhibits a discrete maximum principle. To see this simply solve for the value at (j,k).

$$U_{j,k} = \frac{1}{4}[U_{j+1,k} + U_{j-1,k} + U_{j,k+1} + U_{j,k-1}]$$

If $U_{j,k}$ achieves a maximum value M in the interior then

$$M = \frac{1}{4}[U_{j+1,k} + U_{j-1,k} + U_{j,k+1} + U_{j,k-1}]$$

which implies that

$$M = U_{j+1,k} = U_{j-1,k} = U_{j,k+1} = U_{j,k-1}$$

Repeated application of this argument for the four neighboring points yields the desired result.

Monotone Operators

The discrete Laplacian operator $-\mathcal{L}_\Delta U$ obtained from (4.1.1) is one example of a *monotone* operator.

Definition: The discrete matrix operator $\mathcal{M}$ is a monotone operator if and only if $\mathcal{M}^{-1} \geq 0$ (all entries are nonnegative).

Lemma 4.1.0: A sufficient but not necessary condition for $\mathcal{M}$ monotone is that the operator be M-type. M-type matrix operators have the sign pattern $\mathcal{M}_{ii} > 0$ for each i, $\mathcal{M}_{ij} \leq 0$ whenever $i \neq j$. In addition $\mathcal{M}$ must either be strictly diagonally dominant

$$\mathcal{M}_{ii} > \sum_{j=1, j\neq i}^{n} |\mathcal{M}_{ij}|, \quad i = 1, 2, ..., n \qquad \text{(strict diagonal dominance)}$$

or else $\mathcal{M}$ must be irreducible and

$$\mathcal{M}_{ii} \geq \sum_{j=1, j\neq i}^{n} |M_{ij}|, \quad i = 1, 2, ..., n \qquad \text{(diagonal dominance)}$$

with strict inequality for at least one i.

Proof: The proof for strictly diagonally dominant M is straightforward. Rewrite the matrix operator in the following form

$$\begin{aligned}\mathcal{M} =& D - N, \qquad D > 0, N \geq 0 \\ =& [I - ND^{-1}]D \qquad D^{-1} > 0 \\ =& [I - P]D \qquad P \geq 0\end{aligned}$$

From the strict diagonal dominance of $\mathcal{M}$ we have that eigenvalues of $P = ND^{-1}$ are less than unity, so that the Neumann series for $[I - P]^{-1}$ is convergent. This yields the desired result:

$$\mathcal{M}^{-1} = D^{-1}[I + P + P^2 + P^3 + ...] \geq 0 \tag{4.1.2}$$

When $\mathcal{M}$ is not strictly diagonally dominant then $\mathcal{M}$ must be irreducible so that no permutation $\mathcal{P}$ exists such that

$$\mathcal{P}^T \mathcal{M} \mathcal{P} = \begin{bmatrix} \mathcal{M}_{11} & \mathcal{M}_{12} \\ 0 & \mathcal{M}_{22} \end{bmatrix} \quad \text{(reducibility)}.$$

This insures that eigenvalues of P are less than unity [Taussky48]. Once again we have that the Neumann series is convergent and the final result follows immediately.

■

Example: Maximum Principles and Uniform Convergence.

Consider the model problem (4.1.0) on the unit square.

$$\mathcal{L}_\Delta U_{j,k} = 0$$

Inserting the exact solution into the discrete operator yields the truncation error

$$\mathcal{L}_\Delta u(x_j, y_k) = T_{j,k} \tag{4.1.3}$$

Denote the error $e_{j,k} = U_{j,k} - u(x_j, y_k)$ so that the problem becomes

$$\mathcal{L}_\Delta e_{j,k} = -T_{j,k}, \quad (j,k) \in \Omega \tag{4.1.4}$$

with $e_{j,k} = 0, \ \forall \ (j,k) \in \partial\Omega$. Next let

$$\|e\|_\infty = \max_{(j,k)\in\Omega} |e_{j,k}|$$

and

$$\|T\|_\infty = \max_{(j,k)\in\Omega} |T_{j,k}|$$

so that

$$\|e\|_\infty = \|\mathcal{L}^{-1}T\|_\infty \leq \|\mathcal{L}^{-1}\|_\infty \|T\|_\infty \tag{4.1.5}$$

From consistency of the approximation we have from Taylor series analysis that $\exists\ C > 0$ independent of Δx such that

$$\max_{(j,k)\in\Omega_\Delta} |T_{j,k}| = \|T\|_\infty \leq C\Delta x^2. \tag{4.1.6}$$

From (4.1.2) we have that

$$-\mathcal{L}^{-1} = \frac{\Delta x^2}{4}[I + P + P^2 + P^3 + ...] \quad P \geq 0$$

Next define the "summation" vector

$$s = [1, 1, ..., 1]^T$$

so that $\|P\|_\infty = \|Ps\|_\infty$ since all entries of P are nonnegative. Close inspection of the terms appearing in the Neumann series reveals that terms in the sum eventually geometrically vanish (and can be summed) so that when combined with the fact that $J\Delta x = 1$ we have a bound on the inverse operator of $\frac{1}{8}$.

$$\begin{aligned}\|\mathcal{L}^{-1}\|_\infty =& \frac{\Delta x^2}{4}\|I + P + P^2 + P^3 + ...\|_\infty \\ =& \frac{\Delta x^2}{4}\|s + Ps + P^2 s + P^3 s + ...\|_\infty \\ \leq& \frac{1}{8}\end{aligned} \tag{4.1.7}$$

When the stability estimate (4.1.7) is combined with consistency result (4.1.6), convergence is proven

$$\|e\|_\infty \leq \|\mathcal{L}^{-1}\|_\infty \|T\|_\infty \leq \frac{C}{8}\Delta x^2 \tag{4.1.8}$$

Laplace's Equation on Unstructured Meshes

Consider solving the Laplace equation problem (4.1.0) on a planar triangulation using a Galerkin finite element approximation with linear elemental shape functions. (Results using a finite volume method are identical but are not considered here.) In the finite element method we consider the variational form of the Laplace operator which is obtained by integration by parts.

$$\int_\Omega w\Delta u \ d\Omega = -\int_\Omega (\nabla w \cdot \nabla u)\ d\Omega + \int_{\partial\Omega} w(\nabla u \cdot \mathbf{n}) d\Gamma \tag{4.1.9}$$

This equation is effectively discretized by assuming a solution and test space which are piecewise linear interpolations of pointwise function values. For example, given function values f_j at vertices of the triangulation, a global piecewise linear interpolant can be constructed

$$f(x, y) = \sum_{vertices} N_j(x, y) f_j$$

where the shape functions $N_j(x,y)$ are piecewise linear functions which distance one support on the triangulation so that $N_j(x_j,y_j)=1$ and $N_j(x_i,y_i)=0,\ i\neq j$, see fig. 4.1.1.

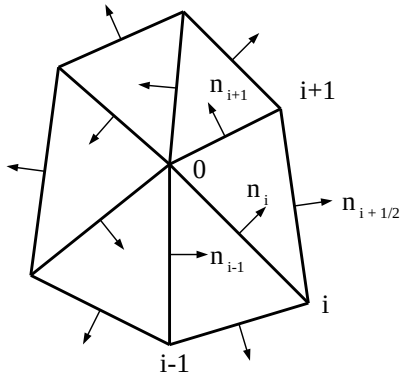

Figure 4.1.0 Local mesh configuration and geometry about vertex v_0.

Since the shape functions serve as a complete basis for representing any piecewise linear function on the triangulation, we need only consider w equal to a single shape function to extract the discretized form of the Laplace operator. This implies that the discretization will involve only distance one neighbors in the triangulation as shown in fig. 4.1.0. Notationally, we define the following sets:

$$\mathcal{T}_0 = \{\triangle's\ incident\ to\ v_0\}$$

$$\mathcal{N}_0 = \{vertices\ distance\ 1\ from\ v_0\}$$

and the convention that $T_{i+\frac{1}{2}} \equiv \triangle(v_0, v_i, v_{i+1})$ with area $A_{i+\frac{1}{2}}$. The domain of interest is now $\Omega = \sum_{\mathcal{T}_0} \cup T_i$ with $w=0$ on $\partial\Omega$.

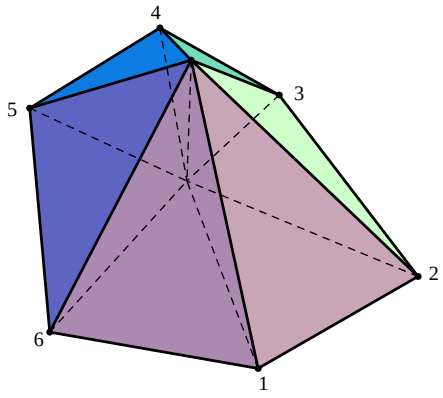

Figure 4.1.1 Shape function $N_0(x,y)$ about central vertex v_0.

For piecewise linear u and w, both ∇u and ∇w are constant in each triangle so that

$$\int_\Omega w\Delta u\ d\,\Omega = -\int_\Omega (\nabla w\cdot\nabla u)\ d\,\Omega + 0 = -\sum_{i\in\mathcal{N}_0} (\nabla w\cdot\nabla u)_{i+\frac{1}{2}} A_{i+\frac{1}{2}}. \qquad (4.1.10)$$

Let $\vec{\mathbf{n}}$ be a normal vector *scaled by the length of the edge* as shown in fig. 4.1.0. Some algebra reveals that

$$\nabla w_{i+\frac{1}{2}} = \frac{-1}{2A_{i+\frac{1}{2}}}\vec{\mathbf{n}}_{i+\frac{1}{2}}$$

and

$$\nabla u_{i+\frac{1}{2}} = \frac{-1}{2A_{i+\frac{1}{2}}}(\vec{\mathbf{n}}_{i+1}(U_i - U_0) - \vec{\mathbf{n}}_i(U_{i+1} - U_0)).$$

Inserting these expressions into (4.1.10) with some rearrangement yields

$$\mathcal{L}_\Delta U_0 = \int_\Omega w\Delta u \; d\,\Omega = \sum_{i\in\mathcal{N}_0} W_{0i}(U_i - U_0) \tag{4.1.11}$$

with

$$W_{0i} = \frac{-1}{4}\left(\frac{\vec{\mathbf{n}}_{i+\frac{1}{2}}\cdot\vec{\mathbf{n}}_{i+1}}{A_{i+\frac{1}{2}}} - \frac{\vec{\mathbf{n}}_{i-\frac{1}{2}}\cdot\vec{\mathbf{n}}_{i-1}}{A_{i-\frac{1}{2}}}\right).$$

This result can be further simplified. Note that

$$\frac{\vec{\mathbf{n}}_{i+\frac{1}{2}}\cdot\vec{\mathbf{n}}_{i+1}}{A_{i+\frac{1}{2}}} = 2\;\frac{\vec{\mathbf{n}}_{i+\frac{1}{2}}\cdot\vec{\mathbf{n}}_{i+1}}{|\vec{\mathbf{n}}_{i+\frac{1}{2}}\times\vec{\mathbf{n}}_{i+1}|} = -2\frac{\cos(\alpha_{0i})}{\sin(\alpha_{0i})} = -2\mathrm{cotan}(\alpha_{0i})$$

and

$$-\frac{\vec{\mathbf{n}}_{i-\frac{1}{2}}\cdot\vec{\mathbf{n}}_{i-1}}{A_{i-\frac{1}{2}}} = -2\frac{\vec{\mathbf{n}}_{i+\frac{1}{2}}\cdot\vec{\mathbf{n}}_{i+1}}{|\vec{\mathbf{n}}_{i-\frac{1}{2}}\times\vec{\mathbf{n}}_{i-1}|} = -2\frac{\cos(\beta_{0i})}{\sin(\beta_{0i})} = -2\mathrm{cotan}(\beta_{0i})$$

where α_{0i} and β_{0i} are the two angles subtending the edge $e(v_0, v_i)$, see fig. 4.1.2. Using the subtending angles, the discretized Laplacian weights assume a particularly simple form:

$$W_{0i} = \frac{1}{2}[\mathrm{cotan}(\alpha_{0i}) + \mathrm{cotan}(\beta_{0i})] \tag{4.1.12}$$

From (4.1.11) we see that Lemma 4.1.0 does not hold unless all $W_{0i} \geq 0$. Fortunately, we have the following result for planar triangulations.

Lemma 4.1.1: A necessary and sufficient for the weights appearing in (4.1.12) to be nonnegative ($-\mathcal{L}_\Delta$ monotone) is that the triangulation be a Delaunay triangulation.

Proof: Rearrangement of the weights appearing in (4.1.12) yields

$$\begin{aligned} W_{0i} =& \frac{1}{2}\left[\mathrm{cotan}(\alpha_{0i}) + \mathrm{cotan}(\beta_{0i})\right] \\ =& \frac{1}{2}\left[\frac{\cos(\alpha_{0i})}{\sin(\alpha_{0i})} + \frac{\cos(\beta_{0i})}{\sin(\beta_{0i})}\right] \\ =& \frac{1}{2}\left[\frac{\sin(\alpha_{0i}+\beta_{0i})}{\sin(\alpha_{0i})\sin(\beta_{0i})}\right] \end{aligned} \tag{4.1.13}$$