Journal of Computational Analysis and Applications, Vol. 5, No. 1, January 2003 (© 2003)

Algorithm for Optimal Triangulations in Scattered Data Representation and Implementation

Bradley W. Dyer[1] and Don Hong[1,2]

Scattered data collected at sample points may be used to determine simple functions to best fit the data. An ideal choice for these simple functions is bivariate splines. Triangulation of the sample points creates partitions over which the bivariate splines may be defined. But the optimality of the approximation is dependent on the choice of triangulation. An algorithm, referred to as an Edge Swapping Algorithm, has been developed to transform an arbitrary triangulation of the sample points into an optimal triangulation for representation of the scattered data. A Matlab package has been completed that implements this algorithm for any triangulation on a given set of sample points.

KEY WORDS: Bivariate splines; optimal triangulations; scattered data representation.

1. INTRODUCTION

In many applications, it is desirable to approximate a given surface with a high degree of accuracy. Scattered data on the surface may be collected by recording the distance from sample points in a fixed plane to the surface. Once the scattered data has been collected, it is necessary to determine simple functions to interpolate, or best fit, the data. An ideal choice for these simple functions is bivariate splines. Since a bivariate spline is piecewise-defined over its planar domain, it is necessary to create a partition of the sample points in the plane. One of the most applicable partitions in this case is triangulation.

A *triangulation* of a finite set of n sample points $\mathbf{v}_i = (x_i, y_i)$, $i = 1, \ldots, n$ in a plane is defined as a collection Δ of triangles satisfying (1) the vertices of the triangles are precisely the sample points $\mathbf{v}_i$, (2) the union of the triangles in Δ is a connected set, and (3) the intersection of any two adjacent triangles

[1] Department of Mathematics, East Tennessee State University, Johnson City, Tennessee 37614-0663.

[2] To whom correpondence should be addressed. e-mail: hong@etsu.edu

in Δ is either a common vertex or a common edge. The vertex set of the triangulation Δ will be denoted as V.

There are clearly many distinct triangulations on a sufficiently large finite set of vertices. Since the optimality of the approximation using bivariate splines is affected by the choice of triangulation, it is necessary to determine the optimal triangulation for representation of the scattered data.

Naturally, the idea of an optimal approximation of scattered data varies depending on the desirable properties of the approximation problem. In this paper, the optimal order of approximation is determined with respect to the order r of smoothness and degree k of the bivariate splines. The approximation order is defined over the triangulation's spline space $S_k^r(\Delta)$, the subspace of $C^r(\Delta)$ of all splines with total degree $\leq k$ and with grid lines given by the edges of the triangulation Δ. Note that $C^r(\Delta)$ is the space of functions that are differentiable and continuous on the ith derivative for $0 \leq i \leq r$.

Given the spline space $S := S_k^r(\Delta)$, the *approximation order* of S is the largest integer ρ such that $\text{dist}(f, S) \leq C|\Delta|^\rho$ for all sufficiently smooth functions f in S and an approximation constant C dependent only on f and the smallest angle of Δ. Here, $|\Delta|$ represents the maximum diameter of any triangle in Δ (see [1]).

It can be shown that the approximation order of $S_k^r(\Delta)$ cannot be higher than $k+1$, and is trivially $k+1$ in the case where $r = 0$. A triangulation Δ will be called an *optimal triangulation* if $S_k^r(\Delta)$ achieves the optimal approximation order of $k+1$ for a fixed pair of integers (k, r) (see [6]).

A considerable amount of research has been accomplished on determining the conditions under which $S_k^r(\Delta)$ achieves the optimal approximation order. It was proven in 1970 by Ženišek in [9] that $S_k^r(\Delta)$ achieves the optimal approximation order for $k \geq 4r+1$. This means that any triangulation is optimal for $S_k^r(\Delta)$ when $k \geq 4r+1$. By 1987, de Boor and Höllig proved theoretically in [1] that the optimal approximation order for $S_k^r(\Delta)$ is actually obtained as soon as $k \geq 3r+2$. Later, Chui, Hong, and Jia were able to constructively prove that $S_k^r(\Delta)$ attain the optimal approximation order of $k+1$ when $k \geq 3r+2$, as appeared in [5] in 1995. Consequently, any triangulation Δ is optimal for $S_k^r(\Delta)$ if $k \geq 3r+2$.

Since lower degree spline spaces are preferable for application purposes, it is beneficial to determine optimal triangulations for $S_k^r(\Delta)$ when $k \leq 3r+1$. This paper will specifically focus on spline spaces of C^1 quartic splines, $S_4^1(\Delta)$. A convenient triangulation of uniformly spaced lattice points is the three-directional mesh, denoted as $\Delta^{(1)}$ and illustrated in Fig. 1(a). However, de Boor and Jia proved in 1993 in [2] that the bivariate spline space $S_k^r(\Delta^{(1)})$ attains an approximation order of at most k for $k \leq 3r+1$. So $\Delta^{(1)}$ is not an optimal triangulation for the spline space S_k^r when

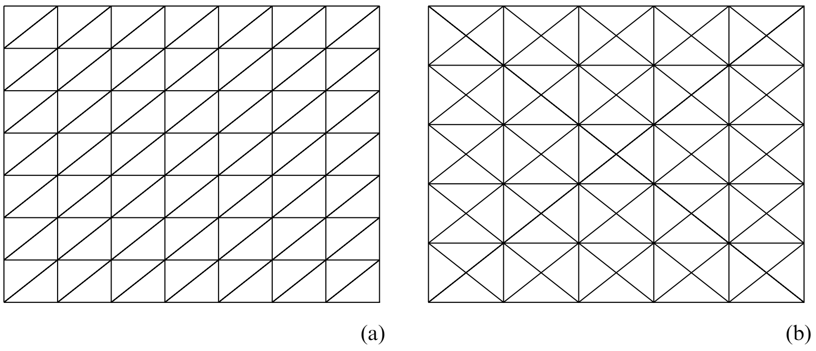

Fig. 1. Triangulations $\Delta^{(1)}$ and $\Delta^{(2)}$.

$k \leq 3r + 1$. In particular, $S_4^1(\Delta^{(1)})$ attains an approximation order of 4, but not the optimal approximation order of 5. So $S_4^1(\Delta^{(1)})$ is not optimal for C^1 quartic splines.

A couple of techniques have been implemented in recent years to determine optimal triangulations for C^1 quartic splines. In 1996, Chui and Hong developed in [3] a scheme known as a *Local Refinement Scheme* to transform an arbitrary triangulation of data points into an optimal triangulation for C^1 quartic splines. The disadvantage of this scheme is that it requires the inclusion of additional data points and often in applications no scattered data is available for additional data sites. Later, Hong and Mohapatra developed in [7] a *mixed three-directional mesh* which is an optimal triangulation for C^1 quartic splines on the existing data points. However, the disadvantage of this technique is that the data points cannot be arbitrary.

A more versatile triangulation scheme has been introduced in [4], known as a *type-O triangulation*, which is an optimal triangulation for C^1 quartic splines on the existing arbitrary data points. For this scheme, the concept of a *type-O vertex* of a triangulation is developed. A vertex u will be called a type-O vertex of a triangulation Δ if u satisfies at least one of the following conditions.

(i) u is a boundary vertex of Δ;
(ii) u is an interior vertex of Δ of degree 4;
(iii) u is an interior vertex of Δ of odd degree;
(iv) u is an interior vertex and there exists a vertex v of Δ satisfying either v is an interior vertex of odd degree or degree 4 or v is a boundary vertex such that $[u, v]$ is a nondegenerate edge of Δ with respect to u.

It should be noted from the definition that the degree of a vertex v refers to the number of edges in the triangulation incident to v. Also, given three consecutive edges e_{j-1}, e_j, and e_{j+1} incident to a common vertex v, the edge e_j is called degenerate with respect to v if the two edges e_{j-1} and e_{j+1} are colinear. The set of all type-O vertices in V will be denoted as V_O.

Now a type-O triangulation is defined as a triangulation of V with only type-O vertices. The following theorem reveals that a type-O triangulation is an optimal triangulation for C^1 quartic splines. See [4].

Theorem 1. Any type-O triangulation Δ admits the 5th order of approximation from $S_4^1(\Delta)$.

It is a direct result of this theorem that any odd-degree triangulation is a type-O triangulation. Also, the four-directional mesh, denoted as $\Delta^{(2)}$ and illustrated in Fig. 1(b), is a type-O triangulation. Thus, the following corollary is evident.

Corollary 2. (a) If a triangulation Δ contains only odd-degree interior vertices, then there exists an interpolation scheme from $S_4^1(\Delta)$ that attains the 5th order of approximation.

(b) There exists an interpolation scheme from $S_4^1(\Delta^{(2)})$ that yields the 5th order of approximation.

2. AN EDGE SWAPPING ALGORITHM FOR OPTIMAL TRIANGULATIONS

It has been shown that given any vertex set V, a type-O triangulation of V admits the optimal 5th order of approximation for C^1 quartic splines. In this chapter, an algorithm will be developed to transform any arbitrary triangulation Δ of V into a type-O triangulation $\hat{\Delta}$ by an edge swapping process.

Given an interior edge e of a triangulation Δ, Q_e will be used to denote the quadrilateral formed by the two triangles sharing e as a common edge. Let the four vertices of Q_e be labeled $v_1, \ldots, v_4$ in the clockwise direction so that the edge e is $[v_1, v_3]$; i.e., the endpoints of e are v_1 and v_3. Then an *edge swap* may be performed by removing e from Δ and adding a new edge $[v_2, v_4]$. As proposed in [8], an edge e is considered a *swappable* edge only if Q_e is convex and no three of its vertices are colinear. This condition ensures that the new partition formed by the edge swap is a triangulation.

Two vertices of Δ are referred to as neighbors of each other if they are endpoints of the same edge in Δ. Hence, while v_1 and v_3 were neighbors in Δ

in the above example, v_2 and v_4 became neighbors in the new triangulation after the edge e was swapped.

Given any set V of vertices, it is clear that there exists a triangulation Δ with vertices exactly those of V, provided that the vertices are not all colinear. Denote the set of vertices in V that are not type-O vertices of Δ as $\tilde{V}$. Then it can be seen from the definition of a type-O vertex that if a vertex $u \in \tilde{V}$, then u and its neighbors with nondegenerate edges with respect to u must be even-degree vertices of degree greater than or equal to 6.

So for any vertex $u \in \tilde{V}$, an appropriate edge swap with an edge incident to u will reduce the degree of u by one, resulting in an odd-degree vertex which is type-O. The remaining question is whether it is always possible given any triangulation to perform an appropriate edge swap for a vertex in $\tilde{V}$. The desired results are obtained from the following lemma, with E_u denoting the set of all edges incident to u in V.

Lemma 3. For every interior vertex u with $\deg(u) \geq 5$, there is a swappable edge $e \in E_u$.

Proof. Let $n := \deg(u)$ and label the neighbors of $u\mathbf{v}_i$, $i = 1, \ldots, n$ in the clockwise direction. Note that a set of consecutive angles with vertex at u may be represented as

$$\bigcup_{i=1}^{n} [v_i, u, v_{i+1}],$$

where $v_{n+1} := v_1$. If $\alpha_i := \angle v_{i-1} v_i v_{i+1}$, then $\sum_{i=1}^{n} \alpha_i$ becomes the summation of the interior angles of an n-sided polygon. Thus,

$$\sum_{i=1}^{n} \alpha_i = (n-2)\pi.$$

Suppose that $\alpha_i < \pi$ for at most two values of i. Then there are at least $n-2$ angles α_i such that $\alpha_i \geq \pi$. So the sum of these angles is $\sum_{j=1}^{n-2} \alpha_j \geq (n-2)\pi$, which implies the contradiction that the remaining two angles are of degree 0. Thus, at least three of the α_i's are less than π.

Let $\theta_i := \angle v_i u v_{i+1}$. Clearly, $\sum_{i=1}^{n} \theta_i = 2\pi$ since this is the sum of the consecutive angles with vertex at u. So $\sum_{i=1}^{n} (\theta_i + \theta_{i+1}) = 4\pi$ since each of these angles are counted twice.

Suppose that $\theta_i + \theta_{i+1}$ exceeds π for at least three values of $i = 1, \ldots, n-1$. Then $\sum_{i=1}^{k} (\theta_i + \theta_{i+1})$ for these k values of i is minimized at $k = 3$, where these values are chosen so that four angles are considered and two angles are counted twice. Without loss of generality, these angles can be relabeled as $a_1, a_2 = a_3, a_4 = a_5, a_6$, where $a_i + a_{i+1} > \pi$ for three values of $i = 1, \ldots, 5$. But regardless of which three values of i are chosen, the sum $a_1 + a_2 + a_4 + a_6 > 2\pi$, which is a contradiction. Thus, $\theta_i + \theta_{i+1}$ exceeds π for at most two values of i.

Since at least three of the α_i's are less than π, there must be at least one vertex v_i such that both $\alpha_i = \angle v_{i-1}v_iv_{i+1}$ and $\angle v_{i-1}uv_{i+1} = \theta_{i-1} + \theta_i$ are less than π. Therefore, the quadrilateral $Q := [v_{i-1}, v_i, v_{i+1}, u]$ is convex, and consequently, the edge $[u, v_i]$ is swappable. □

Now that it is known that for any vertex u in $\tilde{V}$ of a triangulation there exists an edge swap changing u to a type-O vertex, it is clear that any triangulation of a finite set V may be transformed to a type-O triangulation by a finite number of edge swaps. An algorithm, referred to as a *Swapping Algorithm*, has been developed in [4] to transform any such triangulation Δ into a type-O triangulation $\hat{\Delta}$.

Swapping Algorithm

```
Do while ($\tilde{V} \neq \emptyset$)
Pick any vertex $u$ in $\tilde{V}$ and consider its neighbors.
Pick any neighbor $v$ of $u$ so that the edge $[u, v]$ is swappable.
Swap $[u, v]$, yielding a new edge $[u', v']$.
Form a subset of $\tilde{V}$ by deleting from $\tilde{V}$ all the neighbors $w$ of $w' :=$
    $u, v, u'$, or $v'$, with $[w, w']$ being a nondegenerate edge with respect to $w$.
Call this subset $\tilde{V}$.
Enddo
```

The new triangulation generated by applying this Swapping Algorithm to a triangulation Δ will now be denoted as $\hat{\Delta}$. Since $\hat{\Delta}$ is a type-O triangulation, the following theorem is obtained.

Theorem 4. Every finite set V of sample points admits a triangulation $\hat{\Delta}$ so that $S_4^1(\hat{\Delta})$ attains the 5th order of approximation.

3. MATLAB IMPLEMENTATION OF THE EDGE SWAPPING ALGORITHM

A Matlab package has been completed which applies the Edge Swapping Algorithm to any triangulation on a finite set of vertices to construct a type-O triangulation of the sample points. The package includes a main function `swap.m` as well as subfunctions `consecv.m`, `delrow.m`, `findrow.m`, `findtri.m`, `nbors.m` and `trimesh2.m`, a modification of the Matlab 5.0 function `trimesh.m`. See Appendix for the source codes of these functions.

The input variables for the main function *swap.m* are two $n \times 1$ vectors x and y, an integer n, and an $r \times 3$ matrix `tri` where r is an integer. The vectors x and y are defined so that the n sample points are $(x(i), y(i))$ for $i = 1, \dots, n$. The `tri` matrix indexes into the x and y vectors so that each row of `tri` is a triangle where each entry t in the row is a vertex $(x(t), y(t))$ of the triangle.

The function first defines a vector z of length n as a vector of zeros, since only planar triangulations will be considered. A matrix V is then defined so that the ith row of V is the point $(x(i), y(i), z(i))$. This results in the convention that each entry of the `tri` matrix is equivalent to the row number in V of the respective point. Note that in Matlab, $V(i, :)$ represents the ith row of V.

A new matrix V_n is set equal to V. Throughout the execution of the program, this matrix will represent the set of points that have not yet been characterized as type-O vertices of the triangulation. The outermost loop determines for each point $V(i, :)$ in V whether it is a type-O vertex and, if so, performs an edge-swap followed by the removal of appropriate vertices from V_n.

The `findrow` subfunction is first used to determine if the point $V(i, :) \in V_n$. Its input arguments are three real numbers xx, yy, zz and a $3 \times c$ matrix V_n, c an integer. V_n is searched for a row $V_n(j, :) = [xx, yy, zz]$. If such a row is found then the output argument *row* is set equal to j. Else, *row* is set to 0. If $V(i, :) \notin V_n$, then the next value of i is considered since $V(i, :)$ is already a type-O vertex.

The `nbors` subfunction is next applied to the point $V(i, :)$. Its input arguments are the $3 \times n$ matrix V, an integer i representing the row number of the point in V, the `tri` matrix which indexes into V, and the number r of row s in `tri`. The output arguments are a $k \times 3$ matrix N whose rows are the neighbors of $V(i, :)$, the number k of neighbors, and the number t of triangles incident to $V(i, :)$. The neighbors of $V(i, :)$ are determined by first searching the `tri` matrix with `findrow` for any row with an entry i. This row would represent a triangle with a vertex at $V(i, :)$. The output argument t represents the number of such rows found in `tri`. For each row found with an i entry, the points indexed by the other two entries are added to the N matrix if no such row already exists.

The process of determining if $V(i, :)$ is a type-O vertex begins with initializing a variable $det1$ to zero, which will be set equal to one if $V(i, :)$ is found to be type-O. The following lemma will be used to identify the boundary vertices of the triangulation.

Lemma 5. A vertex v of a triangulation Δ is a boundary vertex of Δ iff the number of neighbors of v does not equal the number of triangles in Δ sharing a vertex at v.

Consequently, a vertex $V(i,:)$ is found to be a boundary vertex if $k \neq t$ after applicaton of the `nbors` subfunction. Since the degree of a vertex is equal to the quantity of its neighbors, $V(i,:)$ is of degree 4 if $k = 4$. Also, $V(i,:)$ is of odd degree if $k(mod2) = 1$.

In order to check condition 4 of the type-O criteria, it is necessary to consider the neighbors of each neighbor of $V(i,:)$. For each point $N(j,:)$ in N, the corresponding row is located in V using `findrow`, and the `nbors` subfunction is applied to this point in V. The new parameters are a matrix NN containing the neighbors of $N(j,:)$, the number kk of neighbors, and tt of triangles incident to $N(j,:)$. Condition $4(a)$ is satisfied for an interior neighbor $N(j,:)$ of $V(i,:)$ if $kk = 4$ or $kk(mod2) = 1$. Condition $4(b)$ is satisfied for a boundary neighbor $N(j,:)$ of $V(i,:)$ if $[V(i,:), N(j,:)]$ is a nondegenerate edge of the triangulation with respect to $V(i,:)$.

The subfunction `consecv` is used to help determine if $[V(i,:), N(j,:)]$ is nondegenerate with respect to $V(i,:)$. Its input arguments are a 1×3 vector v representing a point, a 1×3 vector v_1 representing a neighbor of v, a $k \times 3$ matrix N of the neighbors of v, an integer k, the vertex set V, and the `tri` matrix. The output arguments are the two 1×3 vectors v_0 and v_2 needed to form the quadrilateral Q_e, where $e = [v, v_1]$. The subfunction `consecv` first removes v_1 from N, which is then relabeled as M. This is accomplished by using the subfunction `delrow`, which accepts a matrix A and an integer i and modifies A by deleting the ith row. The subfunction `findrow` was previously used to locate the row number of v_1 in N. Next, a vector A is set equal to the vector from v to v_1, and for each neighbor $M(i,:)$ of v in M, a vector B is assigned to the vector from v to $M(i,:)$. The angle between A and B is calculated each time until a neighbor v_0 is found that minimizes this angle. The vector from v to v_0 is then labeled as BB, and `delrow` is used to create a new matrix L by removing v_0 from M.

For each neighbor $L(i,:)$ of v in L, a vector C is assigned to the vector from v to $L(i,:)$. It is not sufficient to determine v_2 simply by locating a remaining neighbor that minimizes the angle between A and C, since the edges determined by BB and C cannot be on the same *side* of the edge determined by A. So v_2 is found to be the neighbor in L so that the angle between A and C is the minimal one that does not equal the sum of the angles between A and B and B and C.

Once the vertices v_0 and v_2 that form the quadrilateral associated with the edge $[V(i,:), N(j,:)]$ have been located, it is possible to determine if $[V(i,:), N(j,:)]$ is nondegenerate with respect to $V(i,:)$. This is true if

$[V(i,:), v_0]$ and $[V(i,:), v_2]$ are not colinear, which is checked by calculating the respective slopes.

If the vertex $V(i,:)$ is found to be type-O, then $V(i,:)$ is removed from the matrix V_n using `findrow` and `delrow`, and the next value of i is considered. Else, the program will seek to perform an edge swap with an edge incidental to $V(i,:)$. Recall that an edge e is swappable if the quadrilateral Q_e is convex and no three of its vertices are colinear.

Each neighbor $N(h,:)$ of $V(i,:)$ is considered until a swappable edge is found. The latter condition is satisfied if the slope of $[V(i,:), v_0]$ does not equal the slope of $[V(i,:), v_2]$ and the slope of $[N(h,:), v_0]$ does not equal the slope of $[N(h,:), v_2]$. In order to determine if Q_e is convex, where $e = [V(i,:), N(h,:)]$, the barycentric coordinates of $V(i,:)$ are calculated. For an arbitrary triangle $\tau = [v_0, v_1, v_2]$ in a quadrilateral $[v_0, v_1, v_2, v_3]$, the barycentric coordinates, C_i, $i = 0\ldots 2$ of the remaining vertex x are defined to be A_i/A, where A is the ordered area of τ and A_i is the ordered area of the triangle formed by replacing v_i with x in τ. The convexity of the quadrilateral may then be tested by the following lemma.

Lemma 7. A quadrilateral $Q = [v_0, v_1, v_2, v_3]$ is convex if a vertex v_i of Q has a negative barycentric coordinate.

The barycentric coordinates of $V(i,:)$ are determined using determinates to calculate the ordered areas. If $V(i,:)$ is found to have a negative barycentric coordinate, then Q_e is convex and $[V(i,:), N(h,:)]$ is a swappable edge of the `tri` triangulation. After a swappable edge is found, the row numbers of $N(h,:)$, v_0, and v_2 in V are labeled as rv, rv_0, and rv_2, respectively.

Now the desired edge swap is to swap the edge $e = [V(i,:), V(rv,:)]$ of Q_e with a new edge $e' = [V(rv_0,:), V(rv_2,:)]$. This may be accomplished by

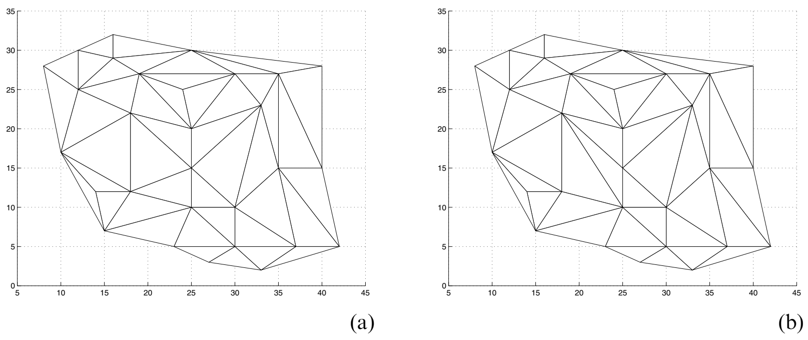

Fig. 2. Triangulation and optimal triangulation for a given set of data sites.

modifying the `tri` matrix so that the two rows representing triangles with edge e in common are replaced by two rows with edge e' in common. The first task is to locate the two triangles in `tri` which have e in common. The subfunction `findtri` is used for this purpose. Its input arguments are three integers a, b, and c representing the row numbers in V of three vertices and the `tri` matrix. The subfunction `findrow` is used repeatedly to search for any row of `tri` containing a permutation of the integers a, b, and c. If such a row is found, then the output argument t is assigned to this row number. Else, t is set equal to zero.

The edge swap is accomplished by first finding the row number t_1 of the triangle in `tri` whose vertices index into rows i, rv, and rv_0 of V as well as the row number t_2 of the triangle with indices i, rv, and rv_2 using `findtri`. Next, rows t_1 and t_2 of `tri` are replaced by two new rows, one with indices i, rv_0, and rv_2, and one with indices rv, rv_0, and rv_2. This completes the edge swap, and $V(i,:)$ is now a type-O vertex.

The next step in implementing the Edge Swapping Algorithm is to form a subset of V_n by deleting from V_n all the neighbors w of $w' := V(i,:)$, $V(rv,:)$, $V(rv_0,:)$, or $V(rv_2,:)$, with $[w, w']$ being a nondegenerate edge with respect to w. The deletion of these vertices from V_n is possible since the edge swap results in all of these neighbors satisfying the type-O criteria. For each neighbor $N(a,:)$ of $V(i,:)$, `consecv` is used to determine the other vertices v_0 and v_2 forming Q_e for $e = [V(i,:), N(a,:)]$. Then the appropriate slopes are compared to determine if e is a nondegenerate edge with respect to $V(i,:)$. If so, `findrow` and `delrow` are used to remove $N(a,:)$ from V_n. This same procedure is repeated for the other three vertices $V(rv,:)$, $V(rv_0,:)$, and $V(rv_2)$ of w'.

After the outermost loop has been executed for all the vertices in V, V_n will be an empty matrix and the `tri` matrix will represent a type-O triangulation of the vertices in V. The remaining step is to display the type-O triangulation of the vertex set. This is achieved by the `trimesh2` subfunction, which is a modification of the Matlab 5.0 subfunction `trimesh` to output triangulations on a planar grid. Its input arguments are the `tri` matrix and the x and y vectors, and it returns the triangulation as a figure.

The `swap` program may be used to effectively implement the Edge Swapping Algorithm on any initial triangulation of sample points for which a triangulation admitting an optimal approximation with C^1 quartic splines is desired. Fig. 2(a) shows a triangulation of some scattered sample points which has been defined in Matlab using the x and y vectors and the `tri` matrix. This triangulation was transformed by `swap` to the type-O triangulation in Fig. 2(b) with a single edge swap. The first non-type-O vertex encountered by `swap` was located at (25, 15). As the neighbors of this vertex were considered, the neighbor at (18, 22) was the first one found to form a

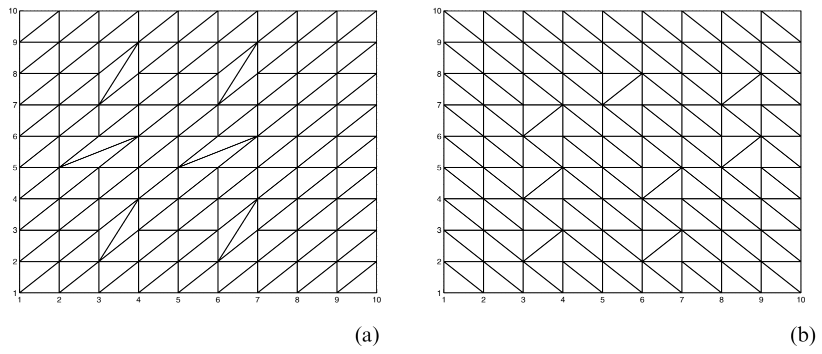

Fig. 3. Optimal triangulations from $\Delta^{(1)}$.

swappable edge. The resulting edge swap was sufficient to create the type-O triangulation in the latter figure.

Recall that the three-directional mesh $\Delta^{(1)}$ is a convenient triangulation for a collection of uniformly spaced lattice points, but it is not optimal for C^1 quartic splines. Figure 3(a) depicts a type-O triangulation resulting from an application of `swap` to the sample $\Delta^{(1)}$ in Fig. 1. Since `swap` considers the vertices of the initial triangulation in sequential order, the type-O triangulation returned by `swap` may be dependent on the order in which the vertices are defined in the x and y vectors. The result in Fig. 3(a) was achieved by ordering the vertices from the bottom to the top of each column, beginning with the leftmost column. Figure 3(b) depicts a quite different type-O triangulation of this vertex set, where only the direction of the diagonals in the initial triangulation was changed. This illustrates how the output of `swap` on a particular vertex set may be changed, when desirable, by reordering the vertices or altering the initial triangulation.

ACKNOWLEDGMENTS

This research was supported in part by a Research Development Grant #00-007/m from East Tennessee State University. The authors are also very grateful to the anonymous referees for their valuable comments and suggestions which helped improve the writing of this paper.

APPENDIX. MATLAB 5.0 SOURCE CODES

SWAP.M

```
function X = swap(x, y, n, tri)
z = zeros(1, n);
[r c] = size(tri);
for i = 1:n
V(i, :) = [x(i) y(i) z(i)];
end
Vn = V;
rec = 0;
for i = 1:n
if findrow(V(i, 1), V(i, 2), V(i, 3), Vn)  = 0
[N, k, t] = nbors(V, i, tri, r);
det1 = 0;
if k  = t,
det1 = 1;
elseif k == 4,
det1 = 1;
elseif mod(k, 2) == 1,
det1 = 1;
end
for j = 1:k
row = findrow(N(j, 1), N(j, 2), N(j, 3), V);
[NN, kk, tt] = nbors(V, row, tri, r);
if kk == tt
if kk == 4 | mod(kk, 2) == 1
det1 = 1;
end
elseif det1  = 1
[v0, v2] = consecv(V(i, :), N(j, :), N, k, V, tri);
if (v0(1) - V(i, 1))  = 0 & (v2(1) - V(i, 1))  = 0
slope1 = (v0(2) - V(i, 2))/(v0(1) - V(i, 1));
slope2 = (v2(2) - V(i, 2))/(v2(1) - V(i, 1));
if slope1  = slope2
det1 = 1;
end
elseif (v0(1) - V(i, 1))  = (v2(1) - V(i, 1))
det1 = 1;
end
end
end
```

```
if det1 == 1
row = findrow(V(i, 1), V(i, 2), V(i, 3), Vn);
if row  = 0
Vn = delrow(Vn, row);
end
elseif findrow(V(i, 1), V(i, 2), V(i, 3), Vn)  = 0
for h = 1:k+1
if h == k+1
det1 = -1, break
end
if findrow(N(h, 1), N(h, 2), N(h, 3), Vn)  = 0
[v0, v2] = consecv(V(i, :), N(h, :), N, k, V, tri);
if (v0(1)-V(i, 1)) = 0 &(v2(1)-V(i, 1)) = 0 &(v0(1)-N(h, 1)) =0
&(v2(1)-N(h, 1)) =0
slope1 = (v0(2) - V(i, 2))/(v0(1) - V(i, 1));
slope2 = (v2(2) - V(i, 2))/(v2(1) - V(i, 1));
slope3 = (v0(2) - N(h, 2))/(v0(1) - N(h, 1));
slope4 = (v2(2) - N(h, 2))/(v2(1) - N(h, 1));
if slope1  = slope2 & slope3  = slope4
rv0 = findrow(v0(1), v0(2), v0(3), V);
rv2 = findrow(v2(1), v2(2), v2(3), V);
rv = findrow(N(h, 1), N(h, 2), N(h, 3), V);
A = det([1 V(rv0,1) V(rv0,2); 1 V(rv2,1) V(rv2,2),
1 V(rv,1) V(rv,2)]);
c0 = det([1 V(i,1) V(i,2); 1 V(rv2,1) V(rv2, 2),
1 V(rv,1) V(rv,2)])/A;
c1 = det([1 V(rv0,1) V(rv0,2); 1 V(i,1) V(i,2),
1 V(rv,1) V(rv,2)])/A;
c2 = det([1 V(rv0,1) V(rv0,2); 1 V(rv2,1) V(rv2,2),
1 V(i,1) V(i,2)])/A;
if c0<0 | c1<0 | c2<0
break
end
end
elseif (v0(1)-V(i, 1)) =(v2(1)-V(i, 1)) &(v0(1)-N(h, 1)) =(v2(1)-
N(h, 1))
rv0 = findrow(v0(1), v0(2), v0(3), V);
rv2 = findrow(v2(1), v2(2), v2(3), V);
rv = findrow(N(h, 1), N(h, 2), N(h, 3), V);
A = det([1 V(rv0,1) V(rv0,2); 1 V(rv2,1) V(rv2,2),
1 V(rv,1) V(rv,2)]);
c0 = det([1 V(i,1) V(i,2); 1 V(rv2,1) V(rv2, 2),
1 V(rv,1) V(rv,2)])/A;
c1 = det([1 V(rv0,1) V(rv0,2); 1 V(i,1) V(i,2),
1 V(rv,1) V(rv,2)])/A;
 c2 = det([1 V(rv0,1) V(rv0,2); 1 V(rv2,1) V(rv2,2),
1 V(i,1) V(i,2)])/A;
if c0<0 | c1<0 | c2<0
break
end
end
end
end
if det1  = -1
t1 = findtri(i, rv, rv0, tri);
t2 = findtri(i, rv, rv2, tri);
tri(t1, :) = [i rv0 rv2];
tri(t2, :) = [rv rv0 rv2];
rec = rec + 1;
for a = 1:k
[v0, v2] = consecv(V(i, :), N(a, :), N, k, V, tri);
if (v0(1) - V(i, 1))  = 0 & (v2(1) - V(i, 1))  = 0
slope1 = (v0(2) - V(i, 2))/(v0(1) - V(i, 1));
slope2 = (v2(2) - V(i, 2))/(v2(1) - V(i, 1));
if slope1  = slope2
row = findrow(N(a, 1), N(a, 2), N(a, 3), Vn);
if row  = 0
Vn = delrow(Vn, row);
end
end
elseif (v0(1) - V(i, 1))  = (v2(1) - V(i, 1))
row = findrow(N(a, 1), N(a, 2), N(a, 3), Vn);
if row  = 0
Vn = delrow(Vn, row);
end
end
end
[Nrv, kk, tt] = nbors(V, rv, tri, r);
for a = 1:kk
[v0, v2] = consecv(V(rv, :), Nrv(a, :), Nrv, kk, V, tri);
if (v0(1) - V(rv, 1))  = 0 & (v2(1) - V(rv, 1))  = 0
slope1 = (v0(2) - V(rv, 2))/(v0(1) - V(rv, 1));
slope2 = (v2(2) - V(rv, 2))/(v2(1) - V(rv, 1));
if slope1  = slope2
```

```
row = findrow(Nrv(a, 1), Nrv(a, 2), Nrv(a, 3), Vn);
if row  = 0
Vn = delrow(Vn, row);
end
end
elseif (v0(1) - V(rv, 1))  = (v2(1) - V(rv, 1))
row = findrow(Nrv(a, 1), Nrv(a, 2), Nrv(a, 3), Vn);
if row  = 0
Vn = delrow(Vn, row);
end
end
end
[Nrv0, kk, tt] = nbors(V, rv0, tri, r);
for a = 1:kk
[v0, v2] = consecv(V(rv0, :), Nrv0(a, :), Nrv0, kk, V, tri);
if (v0(1) - V(rv0, 1))  = 0 & (v2(1) - V(rv0, 1))  = 0
slope1 = (v0(2) - V(rv0, 2))/(v0(1) - V(rv0, 1));
slope2 = (v2(2) - V(rv0, 2))/(v2(1) - V(rv0, 1));
if slope1  = slope2
row = findrow(Nrv0(a, 1), Nrv0(a, 2), Nrv0(a, 3), Vn);
if row  = 0
Vn = delrow(Vn, row);
end
end
elseif (v0(1) - V(rv0, 1))  = (v2(1) - V(rv0, 1))
row = findrow(Nrv0(a, 1), Nrv0(a, 2), Nrv0(a, 3), Vn);
if row  = 0
Vn = delrow(Vn, row);
end
end
end
[Nrv2, kk, tt] = nbors(V, rv2, tri, r);
for a = 1:k
[v0, v2] = consecv(V(rv2, :), Nrv2(a, :), Nrv2, kk, V, tri);
if (v0(1) - V(rv2, 1))  = 0 & (v2(1) - V(rv2, 1))  = 0
slope1 = (v0(2) - V(rv2, 2))/(v0(1) - V(rv2, 1));
slope2 = (v2(2) - V(rv2, 2))/(v2(1) - V(rv2, 1));
if slope1  = slope2
row = findrow(Nrv2(a, 1), Nrv2(a, 2), Nrv2(a, 3), Vn);
if row  = 0
Vn = delrow(Vn, row);
end
```

```
end
elseif (v0(1) - V(rv2, 1))  = (v2(1) - V(rv2, 1))
row = findrow(Nrv2(a, 1), Nrv2(a, 2), Nrv2(a, 3), Vn);
if row  = 0
Vn = delrow(Vn, row);
end
end
end
end
end
end
end
rec
trimesh2(tri, x, y);
```

CONSECV.M

```
function [v0, v2, vargout] = consecv(v, v1, N, k, V, tri)
row = findrow(v1(1), v1(2), v1(3), N);
M = delrow(N, row);
A = v1 - v;
theta = 1000;
for i = 1:k-1
B = M(i, :) - v;
dotp = sum(A.*B);
theta2 = acos(dotp/(norm(A)*norm(B)));
if theta2 < theta
v0 = M(i, :);
theta = theta2;
BB = B;
end
end
row = findrow(v0(1), v0(2), v0(3), M);
L = delrow(M, row);
theta1 = 1000;
for i = 1:k-2
C = L(i, :) - v;
dotp = sum(A.*C);
theta2 = acos(dotp/(norm(A)*norm(C)));
dotp = sum(BB.*C);
theta3 = acos(dotp/(norm(BB)*norm(C)));
if theta2 < theta1 & theta3 + theta  = theta2
v2 = L(i, :);
theta1 = theta2;
```

```
end
end
```

DELROW.M

```
function [A, varargout] = delrow(A, i)
t = A;
[r c] = size(A);
A = zeros(r -1, c);
for j = 1:i-1
A(j, :)=t(j, :);
end
for j = i+1:r
A(j-1, :)=t(j, :);
end
```

FINDROW.M

```
function [row, varargout] = findrow(xx,yy,zz,Vn)
[r c] = size(Vn);
for i = 1:r+1
if i == r+1
row = 0; break, end
if Vn(i, 1)==xx & Vn(i, 2)== yy & Vn(i, 3)==zz
row = i; break, end
end
```

FINDTRI.M

```
function [t, varargout] = findtri(a, b, c, tri)
if findrow(a, b, c, tri)  = 0
t = findrow(a, b, c, tri);
elseif findrow(a, c, b, tri)  = 0
t = findrow(a, c, b, tri);
elseif findrow(b, a, c, tri)  = 0
t = findrow(b, a, c, tri);
elseif findrow(b, c, a, tri)  = 0
t = findrow(b, c, a, tri);
elseif findrow(c, a, b, tri)  = 0
t = findrow(c, a, b, tri);
elseif findrow(c, b, a, tri)  = 0
t = findrow(c, b, a, tri);
else
t = 0;
end
```

NBORS.M

```
function [N, k, t] = nbors(V, i, tri, r)
k = 0; t = 0;
```

```
N = zeros(1, 3);
for j = 1:r
if tri(j, 1) == i,
t = t + 1;
if findrow(V(tri(j, 2),1), V(tri(j, 2),2), V(tri(j, 2),3), N) == 0,
k = k + 1;
N(k, :) = [V(tri(j, 2), :)];
end
if findrow(V(tri(j, 3),1), V(tri(j, 3),2), V(tri(j, 3), 3),N) == 0,
k = k + 1;
N(k, :) = [V(tri(j, 3), :)];
end
elseif tri(j, 2) == i,
t = t + 1;
if findrow(V(tri(j, 1),1), V(tri(j, 1),2), V(tri(j, 1),3), N) == 0,
k = k + 1;
N(k, :) = [V(tri(j, 1), :)];
end
if findrow(V(tri(j, 3),1), V(tri(j, 3),2), V(tri(j, 3),3), N) == 0,
k = k + 1;
N(k, :) = [V(tri(j, 3), :)];
end
elseif tri(j, 3) == i,
t = t + 1;
if findrow(V(tri(j, 1),1), V(tri(j, 1),2), V(tri(j, 1),3), N) == 0,
k = k + 1;
N(k, :) = [V(tri(j, 1), :)];
end
if findrow(V(tri(j, 2),1), V(tri(j, 2),2), V(tri(j, 2),3), N) == 0,
k = k + 1;
N(k, :) = [V(tri(j, 2), :)];
end
end
end
```

TRIMESH2.M

```
function hh = trimesh(tri,x,y,varargin)
ax = newplot;
start = 1;
if nargin>3 & rem(nargin-3,2)==1,
c = varargin1;
start = 2;
elseif nargin<3
```

```
error('Not enough input arguments');
else
for k = 1:length(x)
c(k) = 1;
end
end
if isstr(get(ax,'color')),
fc = get(gcf,'Color');
else
fc = get(ax,'color');
end
h = patch('faces',tri,'vertices',[x(:) y(:)],
'facevertexcdata',c(:),...
'facecolor',fc,'edgecolor',get(ax,'defaultsurfacefacecolor'),...
'facelighting', 'none', 'edgelighting', 'flat',...
vararginstart:end);
if  ishold, view(2), grid on, end
if nargout==1, hh = h; end
```

REFERENCES

1. C. de Boor and K. Höllig, Approximation power of smooth bivariate *pp* functions, *J. Math. Z.* 197, 343–363 (1988).
2. C. de Boor and R. Q. Jia, A sharp upper bound on the approximation order of smooth bivariate *pp* functions, *Approx. Theory* 72, 24–33 (1993).
3. C. K. Chui and D. Hong, Construction of local C^1 quartic spline elements for optimal-order approximation, *Math. Comp.* 65, 85–98 (1996). MR 96d:65023.
4. C. K. Chui and D. Hong, Swapping edges of arbitrary triangulations to achieve the optimal order of approximation. *SIAM J. Numer. Anal.* 34, 1472–1482 (1997).
5. C. K. Chui, D. Hong, and R. Q. Jia, Stability of optimal-order approximation by splines over arbitrary triangulations, *Trans. of Amer. Math. Soc.* 374, 3301–3318 (1995). MR 96d:41012.
6. D. Hong, Optimal triangulations using edge swappings, in *Approximation Theory VIII, Vol. 1*, (C. K. Chui and L. L. Schumaker, eds.), World Scientific Publishing Co., Inc., 1995, pp. 249–256.
7. D. Hong and R. N. Mohapatra, Optimal-order approximation by mixed three-directional spline elements, *J. Comp. Math. Appl.* 40, 127–135 (2000).
8. L. L. Schumaker, Computing optimal triangulations using simulated annealing, *J. Comp. Aided Geometric Design* 10, 329–345 (1993).
9. A. Ženišek, Interpolation polynomials on the triangle, *Numer. Math.* 15, 283–296 (1970).