A Two Dimensional
Mesh Verification Algorithm

By

R.B. Simpson

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

CS-80-37

August, 1980

# Faculty

# of

# Mathematics

University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

# A Two Dimensional Mesh Verification Algorithm

R.B. Simpson*

Abstract

A finite element mesh is usually represented in a program by lists of data, i.e. vertex coordinates, element incidences, boundary data. This paper is concerned with conditions on the list data which ensure that the lists describe a 'tiling' of some planar region without overlap or gaps. For a particular format of lists, a set of such conditions is given which is proven to be sufficient to guarantee such a 'tiling'. These conditions have been chosen so as to be verifiable by the algorithm referred to in the title, which is described in detail and is claimed to be of reasonable efficiency.

Keywords - mesh, finite element, triangulation

* This research was carried out at the Brunel Institute of Computational Mathematics, Uxbridge, U.K., while the author was on sabbatical leave from the University of Waterloo, and was supported by a grant from the Natural Science and Engineering Research Council, Canada. The author, and hopefully the manuscript, have benefitted from referee's suggestions.

# A Two Dimensional Mesh Verification Algorithm

## § 1 Introduction

A mesh on a region of the plane generally appears to the reader of a text book or research paper as a diagram showing a partition of the region into finite elements of simple geometric shapes, usually triangles or quadrilaterals. Intuitively, the partition can be thought of as a tiling of the region up to its boundaries by the finite elements as tiles. On the other hand, it appears to the user of the method in the source code of his programs as lists of numbers of specific types, e.g. positive integers less than M, real numbers, etc. In general, however, if the lists are filled with arbitrary data of the correct type, they only represent some collection of elements, which may overlap each other, or leave gaps in the region's interior. Whether the collection represents a proper tiling or not is data dependent. The purpose of this paper is to make an explicit statement of this dependence in the form of a set of four conditions that the data must satisfy, these conditions being specific to two dimensional meshes. Although the conditions are geometrically simple, their verification for the lists of a particular mesh involve non trivial computations which have been organized in this paper into an algorithm which is referred to as the mesh verification algorithm. One consideration in the choice of these conditions, then, has been that the mesh verification algorithm be sufficiently efficient to be practically viable.

Implemented in a program, the algorithm can be used to check a mesh produced for a particular computation. It is a common practice in employing the finite element method to prepare a mesh using a mesh generation program, possibly store it in a file, and plot it to examine its correct-

ness and suitability to the region and the problem before proceeding to the subsequent stages of the method. Often this procedure is repeated a number of times because the mesh is shown to be incorrect, typically due to error in input data to the mesh generation program, or weaknesses in its algorithm or 'bugs' in its programming. An algorithmic verification of the output lists based on the criteria in this paper is viewed as being a check on the mesh which is complementary to a graphical examination in the sense of being faster and not dependent on graphic facilities, but not providing the positive evidence of the suitability of the mesh that a visual inspection gives.

A second motivation for these conditions is to give a mathematically rigorous definition of a finite element mesh that can serve in the study and development of mesh generation programs, allowing explicit specifications for algorithms and providing a debugging tool. A number of methods for mesh generation have been discussed in the published literature, e.g. [3], [6], [11], [15], which have been implemented in programs giving extensive satisfactory use. There is no doubt that mesh generation programs can be written which are pragmatically successful without a mathematical definition of a finite element mesh. However, to say whether such a program is correct or not, or to state under what conditions it fails, or to compare two such programs is difficult because of the vagueness about the specifications for output as well as input even at the algorithmic level. The programs produce meshes in their list form, but as we have mentioned above, not all lists correspond to legitimate meshes. It is a generally held tenet of software engineering that one of the sources of debugging and reprogramming efforts, and lack of reliability in modular systems,

is the imprecise specifications of the inputs to and functions of the modules. If this is the case, then a more precise statement of the function of a mesh generating module might be expected to contribute to the more efficient production of finite element packages. In particular, it is believed that the four conditions of the definition can be used as verification conditions for proving mesh handling programs correct (see Van Emden [18], for a pragmatic discussion of the use of verification conditions in programming). While the question of whether, or in what measure, program correctness procedures will aid program synthesis is an area of speculation; there seems little doubt that a better understanding of the mathematics of an algorithm and its data generally leads to programs which are better in a variety of senses.

In §2, the list representation that we will use for a collection of elements is introduced, and with it, four conditions on the collection are stated geometrically which form the proposed definition of a planar mesh. These conditions are then justified in §3, on a mathematical level by proving that a set of lists which meet the conditions describe a tiling of some region of the plane. The definition is restricted to apply to meshes for regular planar regions (possibly multi-connected, or disconnected); in particular, it does not extend to meshes for describing regions with cracks as used in some finite element applications. Readers who are not interested in the formal justification of these conditions may proceed to the development of the mesh verification algorithm in §4 and §5. In these sections, the conditions stated for the collection of elements represented by mesh lists are expressed as computational checks on the list data.

In these computations, information about the element sharing a common edge with a given element (i.e. an element's neighbour) is required. This information is used in a variety of other contexts in the finite element method, e.g. averaging stresses over neighbouring elements ([17], page 168), improving the triangulation of a region [11], or performing local mesh refinements [13], [15]. The process of obtaining and verifying this information involves a list inversion of one of the mesh lists, and is of some independent interest, so it is discussed separately in §4. The other aspects of checking the conditions are dealt with in §5. At this stage, the generality of the elements' geometry becomes a significant factor, so the discussion is specialized to triangular meshes to avoid unwarranted complexity.

In composing the algorithms of §4 and §5, a compromise between simplicity and optimality has been sought. Some comments on the performance of a FORTRAN implementation are given in §6. An inspection of the components of the verification algorithm indicates that it should run in times linear in the number of triangles in the collection being verified, subject to some restrictions on the mesh topology that are quite natural for finite element meshes.

While the author is unaware of other algorithms for verifying a mesh in this sense, several algorithms for verifying other geometric 'objects' have appeared. There is a substantial literature on algorithms for verifying graph planarity (e.g. Hopcroft and Tarjan, [9]). Recently, a linear time algorithm for verifying the planarity of a 2 complex has been published by Gross and Rosen [8], and in [14], Shamos and Hoey give an algorithm for verifying when a planar polygon is simple. In these

references, the algorithms tend to be described at a high level, with the primary emphasis placed on the analysis of the complexity of the process.

## §2 List Definitions and the Conditions for a Mesh

The list representation for a finite element mesh which we will assume here consists of two basic lists:

- the vertex coordinate list of length $N_v$
- the element incidence list of length $N_e$

and an auxiliary list

- the boundary reference table of length $N_b$.

The basic lists contain independent data, but the boundary reference table's information about the mesh can be obtained from the other two. The $k^{th}$ entry of the vertex coordinate list is the coordinates $(x_k, y_k)$ of the $k^{th}$ vertex of the mesh, also denoted $P(k)$ in the sequel. The term 'vertex' here refers to the points which determine the geometric shape of the element as a region (e.g. the 3 vertices of a triangle), as opposed to the term 'node' which is commonly used for points associated with degrees of freedom of the element shape function. The entries in the vertex coordinate list are required to be unique, i.e. if $j \neq k$, then $P(j) \neq P(k)$. The $j^{th}$ entry in the element incidence list is itself a sublist of the indices in the vertex coordinate list of the vertices of the $j^{th}$ element. $E(j)$ will be used to denote the $j^{th}$ element and if it has $I(j) \geq 3$ vertices, then the $j^{th}$ entry of the element incidence list consists of integers $v(1,j), v(2,j), \ldots, v(I(j),j)$ with $0 < v(i,j) \leq N_v$, and $v(i,j) \neq v(k,j)$ if $i \neq k$. The $i^{th}$ vertex of $E(j)$ is $P(v(i,j))$ and $i$ will be referred to as the local vertex number of $P(v(i,j))$. The $i^{th}$ edge of $E(j)$ is the directed line segment running from $P(v(i,j))$ to $P(v(\overline{i+1},j))$, where

$$\overline{i} \equiv i \bmod I(j) \tag{2.1}$$

is a notation used in the sequel for indexing the 'next' vertex around $E(j)$, and $i$ will be referred to as the local side number of this side.

The $k^{th}$ entry of the boundary reference table consists of a pair of integers $(b(1,k), b(2,k))$ which describe the $k^{th}$ boundary edge of the mesh by giving the index of the element to which it belongs, $0 < b(1,k) \leq N_e$, and the local side number, $0 < b(2,k) \leq I(b(1,k))$. This list is ordered first by $b(1,k)$ and within entries having the same value for $b(1,k)$ by $b(2,k)$ i.e.,

$$\begin{aligned} k_1 < k_2 \Rightarrow & \; b(1,k_1) \leq b(1,k_2) \\ & \text{and if } b(1,k_1) = b(1,k_2) \\ & \text{then } b(2,k_1) < b(2,k_2) \end{aligned} \tag{2.2}$$

This list is the least standard of the three in the literature on finite element programming. It was proposed by J.A. George in his thesis, [7], and doubtless has been used independently by other implements of the finite element method. As mentioned above, the mesh information contained in it is redundant, as it is contained implicitly in the element incidence list information. The algorithms that we discuss below could either build the boundary reference table from the element incidence list, or check that it is consistent. In view of the use of meshes in the finite element method, we have chosen to describe the latter. Typically, the table contains additional problem specific data concerning boundary conditions and is generated with this data at the time the mesh is generated (i.e. the other two lists are constructed.) The ordering of the table is done to facilitate synchronizing a scan of the element incidence list with a scan through the table. An example of this occurs in (4.7) of §4;

however, the primary instance of this, for the finite element method, occurs in the scan of the elements to generate local stiffness matrices. In some implementations, these boundary data are added as extra fields in the entries of the element incidence sublists either directly or through pointers. A discussion of some alternative representations for meshes, and issues associated with them may be found in [16]. While the definition given here clearly does not conform directly to a variety of implemented data structures for mesh representation, it is expected that a fairly simple identification or translation can be made between the representation used here, and the data structures of most implementations.

We now specify the four conditions on the collection of elements described by the mesh lists that qualify this collection to be a mesh.

C1 The directed polygonal curve formed by traversing the element sides in local side number order forms a simple closed curve with bounded interior (i.e. bounded region on the left of the curve).

The finite element $E(j)$ is defined to be the closure of the interior of this curve.

C2 The $i^{th}$ edge of $E(j)$ is either the only edge joining its end points or there is one other element, $E(\ell)$, having an edge joining these vertices. In the latter case, the line segment joining these vertices must have the opposite direction as an edge of $E(j)$ to its direction as an edge of $E(\ell)$.

In the first case of C2, the $i^{th}$ edge of $E(j)$ is a boundary edge and $E(j)$ is its boundary element. To be consistent, the boundary reference table must have an entry with $b(1,k) = j$, $b(2,k) = i$. In the second

case of C2, the $i^{th}$ edge of $E(j)$ is an interior edge with $E(j)$ and $E(\ell)$ as neighbours across this edge. The requirement concerning the directions of the line segment as an edge of $E(\ell)$ and $E(j)$ will be referred to as edge consistency and if it holds then the vertex of index $v(\overline{i+1},j)$ will be the $m^{th}$ vertex of $E(\ell)$ for some value of $m$ which is referred to as the complementary local edge number in §4.

C3 No interval of a boundary edge intersects an element other than its boundary element

C4 A vertex can have at most one boundary edge directed away from it.

Vertices with one such edge will be referred to as boundary vertices in this paper.

The first requirement is really a definition of the finite element as a point set in the plane. The generality of the element's shape in this definition is intended to avoid introducing geometric considerations extraneous to mesh definition rather than to imply that a broad range of element shapes is useful or desirable. While meshes of elements with mixed shapes have been reported (e.g. [12], quadrilaterals and triangles), the programming complexity which arises from implementing variable length records for the element incidence list or element stiffness matrices is generally not justified and hence the common practice of using elements of a fixed number of edges. Further requirements on the element's geometric shape arise from the need to define shape functions on these element domains, and from the approximation properties required of these functions (e.g. [2] or [13] §3.3).

In §3, we want to establish that these requirements are sufficient to ensure that the lists describe sets of elements which 'tile' a region of the plane. However, first we will discuss some examples in which C2 - C4 are violated to illustrate the senses in which they are necessary. The requirement in C1 that edges have the direction that puts the element on the left coupled with the edge consistency restriction of C2 ensures that neighbours lie on opposite sides of their common edge, i.e. that a short line segment which intersects an interior edge passes from the interior of one neighbour into that of the other. Without these requirements, for example, the configuration of Figure 2.1 would qualify as a mesh with the list of vertices $P(1) = (1,1)$; $P(2) = (1,-1)$; $P(3) = (-1,-1)$; $P(4) = (-1,1)$; element incidences $v(i,j)$ given by

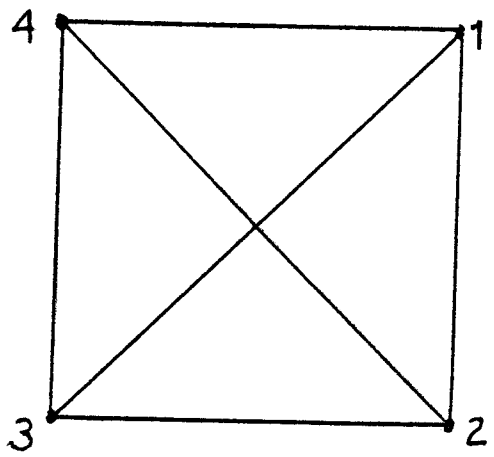



Figure 2.1

| j \ i | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 3 | 2 | 1 |
| 2 | 4 | 2 | 1 |
| 3 | 3 | 1 | 4 |
| 4 | 3 | 2 | 4 |

Table 2.1 - v(i,j) for Figure 2.1

and an empty boundary reference table.

In Figure 2.2, the vertical and horizontal lines are intended to be an interior section of a mesh of squares, described by lists meeting the definition. In addition, one 'extra' quadrilateral element is

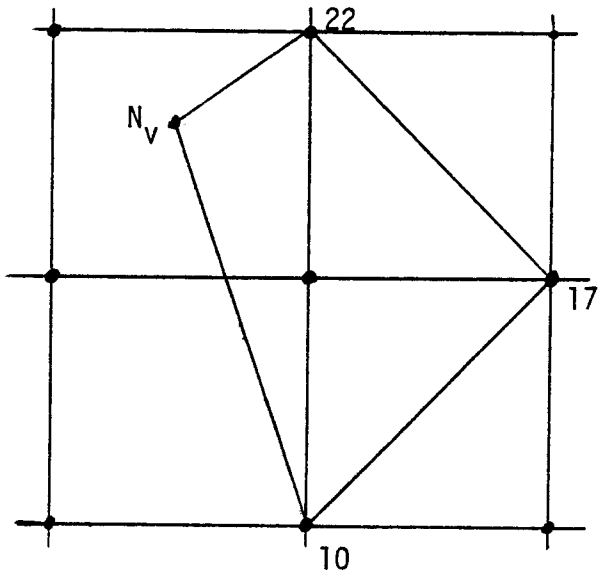



Figure 2.2

included in the lists, by extending the vertex list to include the vertex not on the square mesh, to be indexed as $N_v$. The 'extra' element can be given index $N_e$, say, and has incidence list $v(1,N_e) = 17$, $v(2,N_e) = 22$, $v(3,N_e) = N_v$, $v(4,N_e) = 10$. If each one of the edges of the extraneous $N_e^{th}$ element is declared as a boundary edge in four entries of the boundary

reference table, then the resulting lists meet all the requirements of the definition but C3. The necessity of this requirement can also be seen in a somewhat different sense from the obvious overlap in Figure 2.3, referred to as mesh 'overspill' in [6].

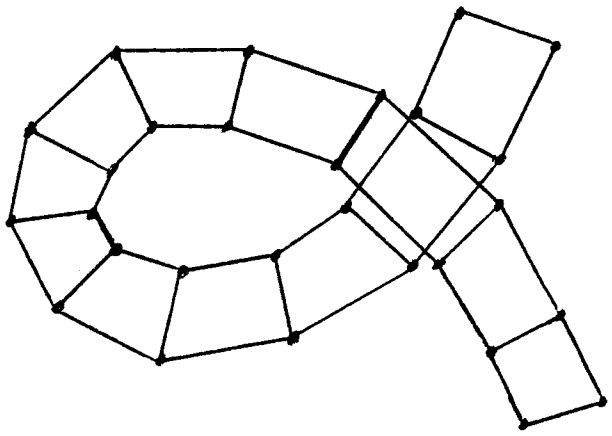


Figure 2.3

In Figure 2.4, an example in which the region covered by the elements of the mesh has both overlap and a 'gap' is shown. The elements are the eight outer squares plus the four triangles, and the boundary reference table records the outer edges of the squares, and the oblique sides of the triangles as boundary edges. The triangles share their horizontal or vertical edges with a square, but are not strictly neighbours because the edge consistency along their common edge cannot hold, or counter

clockwise listing of vertices fails, i.e. C1 and C2 cannot both hold.

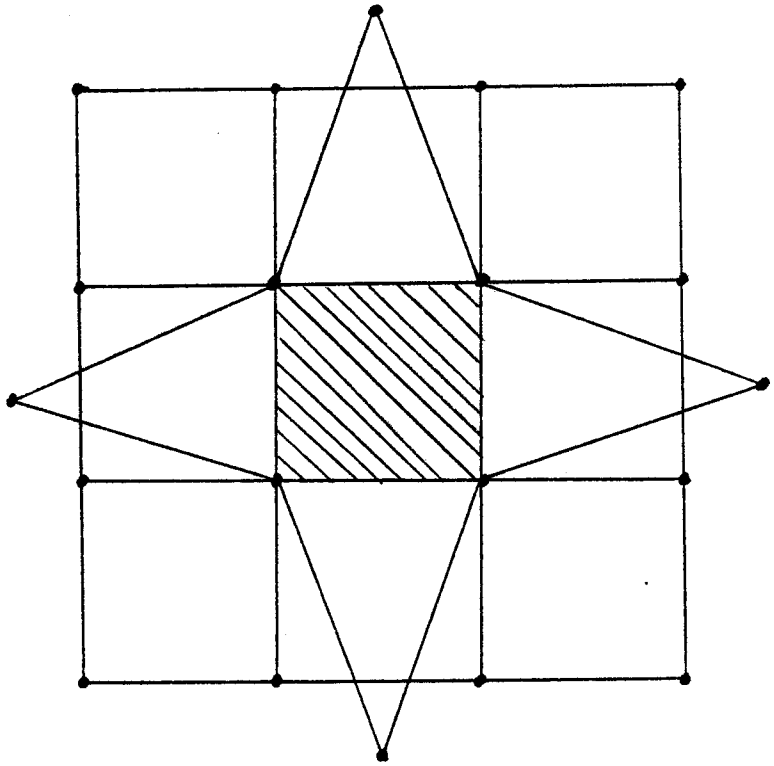


Figure 2.4

The central square of the figure is not an element of the mesh, but is a region of the plane not separated from the elements of the mesh by a boundary edge. It is a form of 'gap' in the mesh. C3 and one of C1 or C2 are violated in this case.

The role of C4 seems less clear; certainly it precludes undesirable anomolies such as a mesh with every edge a boundary edge, like the black squares of a checkerboard. However, it appears that its primary implication is that the boundary curves form simple closed curves as in Theorem 1, and no example has been constructed in which C2 and C3 are satisfied, C4

is violated, and an overlap of element interiors occurs.

## §3 Proof of Sufficiency

We turn now to establishing that the requirements C1 - C4 are sufficient to ensure that the lists of (2.1) correspond in some rigorous way to the intuitive idea of the elements tiling a region of the plane. First it is shown that the set of boundary edges of the mesh form a set of disjoint simple closed curves in Theorem 1. We then show that these curves are oriented in a consistent manner, so that their interiors define bounded regions of the plane for which they are the boundaries in Lemma 3. It is then shown in Theorem 2 that this region is covered without gaps by the elements of the mesh and in Theorem 3 that it is covered without overlap. To avoid repetition of 'polygonal' in this section, we shall assume henceforth that all arcs or curves are polygonal. For a helpful, if elementary, reference on curves and regions in the plane, see [1].

The following lemma concerning a type of connectivity of the mesh will be quite useful in the sequel.

Lemma 1 Let $P$ be a point lying in the interior of $m$ elements of the mesh for $m \geq 0$ and let $P$ be joined to a point $Q$ by an arc which passes through no mesh vertex and intersects no boundary edge. If $Q$ does not lie on an element edge, then $Q$ lies in the interior of $m$ elements of the mesh.

Proof As a point moves along the arc from $P$ towards $Q$, it can only leave an element through an interior edge, at which point it enters its neighbour (C2).

Lemma 2 A vertex is a boundary vertex if and only if it has a boundary edge directed towards it. The incoming boundary edge of a boundary vertex is unique.

Proof Let P be a vertex and let $E$ be the set of edges having P as an endpoint. Assign $e \in E$ a value 1 if it is directed towards P; -1 if it is directed away from P. Since each element for which P is a vertex contributes two edges to $E$, one incoming and one outgoing, it is clear that the sum of the values of edges in $E$ is zero. By requirement C2, the contribution to this sum from all interior edges in $E$ is zero, and by C4 the contribution from all outgoing boundary edges is 0 or 1. Hence the contribution from all incoming boundary edges at P must be 0 or -1, from which the lemma follows.

Theorem 1 The boundary edges of the mesh form a set of simple closed oriented curves, $C_1$, $C_2$, ..., $C_M$, which do not intersect each other.

Proof Let P be a boundary vertex and consider the curve traced out by following the unique outgoing boundary edge from each vertex to the boundary vertex to which it is directed (Lemma 2) starting from P. Since this process can be continued indefinitely, and there are only a finite number of boundary vertices, one must be traversed twice by this curve. If the first one to occur twice is not P, then there are two distinct boundary edges leading to it, in violation of Lemma 2. Hence the curve formed by carrying out this process until P is encountered a second time is a closed oriented curve which can be labelled $C_1$. To see that this curve is simple, we note that from C3 it cannot intersect itself on the line segments between vertices, and as no vertex other than P is

visited twice in a circuit starting at P , it cannot intersect itself at a vertex. If not all the boundary vertices lie on $C_1$, then the argument can be applied to another boundary vertex to establish a simple closed oriented curve $C_2$ and so on. Two of these curves cannot intersect at the interior of a line segment by C3 nor at a vertex by Lemma 2, hence they must be disjoint.

The region on the left of a simple closed oriented curve, C, is conventionally referred to as its interior while the region on the right is referred to as its exterior. Such a curve divides the plane into a bounded region and an unbounded one, and whether the bounded region is the curve's interior or not depends on the curve's orientation, of course. Although we can expect from C1 and C4 that the elements of the mesh should lie in the interiors of these curves in some sense, Theorem 1 does not give any information about the relative orientations of these boundary curves. E.g. it is conceivable that both $C_1$ and $C_2$ have bounded interiors and that $C_2$ lies in the interior of $C_1$. If no curve lies between $C_1$ and $C_2$, then there is no region for which $C_1$ and $C_2$ are oriented boundary curves, as shown in Figure 3.1A, with the interiors shaded. However, if $C_3$ lies between $C_1$ and $C_2$ as in Figure 3.1B, and is oriented so as to have an unbounded interior, then the shaded region is a (disconnected) region with $C_1$, $C_2$ and $C_3$ as its oriented boundaries.

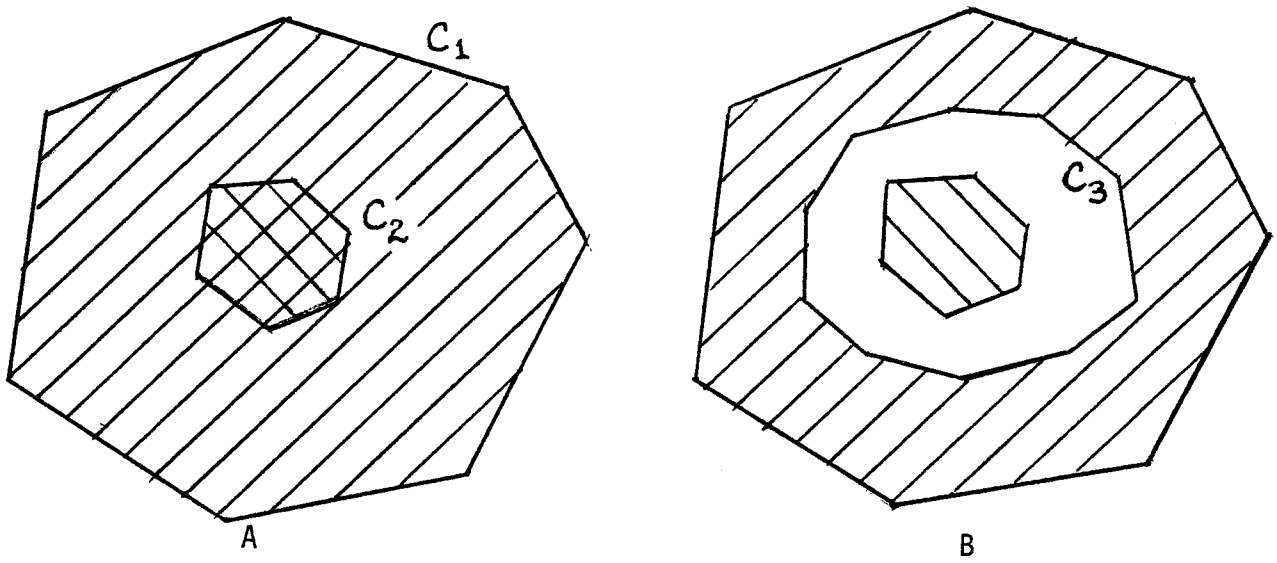



Figure 3.1

Our aim now is to show that the boundary curves established in Theorem 1 are oriented in a consistent way, so that each curve of bounded interior defines a connected region of possibly multiple connectivity, the boundaries of the 'holes' in this region coinciding with boundary curves of unbounded interior. A curve $C_i$ will be said to be maximal in $C_j$ if $C_i$ lies in the bounded domain determined by $C_j$, but not in the bounded domain of any other curve lying in the bounded domain of $C_j$, (e.g. $C_3$ is maximal in $C_1$ in Figure 3.1b, while $C_2$ is not).*

Lemma 3 i) Let $C_j$ be a boundary curve of bounded interior and let $C_i$ be any curve maximal in $C_j$. Then $C_i$ is oriented so that its interior is unbounded.

---

* This terminology is motivated by the fact that inclusion generates a partial ordering on the unoriented simple closed curves and $C_i$ is maximal in the set of $C_k < C_j$ in the sense of this ordering.

ii) Let $C_i$ be a boundary curve with unbounded interior, then there is a curve of bounded interior, $C_j$, such that $C_i$ is maximal in $C_j$.

Proof i) Let D be the domain obtained by removing from the bounded interior of $C_j$ the bounded regions determined by all curves maximal in $C_j$. If there are no such maximal curves, then there is nothing to prove, so let us assume there is at least one, and designate it $C_i$. Let E be a boundary element with a boundary edge on $C_j$ and let P be a point of the interior of E arbitrarily near $C_j$. Let Q be in D arbitrarily close to an edge of $C_i$ and not lying on an element edge. Then we can join P to Q by a polygonal arc in D, which does not cross any boundary edges and can be adjusted to avoid any vertices (since D is an open, connected set). By C2, P is in exactly one element and hence by Lemma 1, Q is in one element, E'. However, Q may be arbitrarily close to $C_i$ so E' has a boundary edge on $C_i$ and $C_i$ must be oriented so that D lies in its interior.

ii) Let $\overline{C}$ be a circle sufficiently large to contain all the elements. If the statement were not true, then $C_i$ would be maximal either in $\overline{C}$ or in $C_j$ where $C_j$ also has unbounded interior. We will use $\overline{C}$ to refer either to the circle or to $C_j$, and note that in either case, points of the bounded region of $\overline{C}$ near $\overline{C}$ do not lie in any element of the mesh. Then, as in part (i) let D be the domain formed by removing from the bounded region of $\overline{C}$ the bounded regions of all curves maximal in $\overline{C}$. Then we can pick P in D close to $\overline{C}$ so that P lies in no element and, since D is connected and open, join it to a point Q in a boundary element near $C_i$ by an arc in violation of Lemma 1. This contradiction establishes 3 (ii).

This lemma shows then that every mesh boundary curve of bounded interior defines a possibly multiply connected region with mesh boundary curves as the region's boundary, and that every mesh boundary curve is a part of the boundary of such a region. While finite element problems typically require meshes for one connected region, there does not seem to be a compelling reason to restrict the definition of a mesh so that its boundary curves determine only one. However, for ease of exposition, we shall now assume that there is only one boundary curve of bounded interior and label it $C_1$. Then, from Lemma 3, it follows that if $K > 1$ there are $K - 1$ curves $C_2, \ldots, C_K$ inside $C_1$ oriented to have unbounded interiors, so that the region

$$R = \bigcap_{k=1}^{K} (\text{interior of } C_k) \tag{3.2}$$

is a bounded region of connectivity $K - 1$, which will be defined as the covered region of the mesh. This terminology will be justified by showing that the elements of the mesh cover this region without gaps (in Theorem 2) and without overlap (in Theorem 3).

Theorem 2

$$\bigcup_{i=1}^{N_e} E(i) = R$$

Proof To show that $\bigcup_{i=1}^{N_e} E(i) \subset R$, suppose that for some $i$ and $j$, $E(i) \cap$ (the exterior of $C_j$) is not void. Then, since the exterior of $C_j$ is connected, we can join $P \in E(i)$ to $Q \in C_j$ by an arc in exterior of $C_j$ which passes through no mesh vertices. Since the exteriors of the $C_k$ are

disjoint, this arc will not pass through any boundary edges between P and Q. But near Q it lies outside every element, which contradicts Lemma 1. If there were a point $Q \in R - \bigcup_{i=1}^{N_e} E(i)$ then an arc from $C_1$ to Q could be similarly constructed which again violates Lemma 2, so $R = \bigcup_{i=1}^{N_e} E(i)$. To establish no overlap, we have

Theorem 3 Let P be a point of R which is neither a mesh vertex nor lies on an element edge. Then P lies in the interior of one element.

Proof If the (x,y) plane is identified as the complex plane in the usual way, then the winding number of P with respect to a simple closed oriented curve, D, can be defined as

$$W(D) = \oint_D (z-P)^{-1}\, dz/(2\pi i) \tag{3.3}$$

(see [1], e.g.), and its significance is that, if D has the orientation that gives it a bounded interior, then W(D) = 1 if P is in this interior, and W(D) = 0 if it is in the exterior of D.

If we denote the boundary of R by

$$C = \bigcup_{k=1}^{K} C_k \tag{3.4}$$

then for $P \in R$

$$W(C) = \sum_{k=1}^{K} W(C_k) = 1 \tag{3.5}$$

since

$$W(C_1) = 1, \quad W(C_k) = 0 \qquad k > 1 \quad . \tag{3.6}$$

The proof consists of showing that (3.4), (3.5) holds while R and C are consecutively modified by removing elements not containing P until R consists of one element containing P. Suppose that there is a boundary edge in C which is the edge of an element, E, not containing P. The edges of E can be separated into two subsets, $\partial E(b)$, consisting of boundary edges and $\partial E(i)$ consisting of interior edges. It may happen that $\partial E(b)$ is a simple arc on $C_i$ for some i; and that $\partial E(i)$ is a simple arc with no points in common with C other than its end points. In this case $C_i$ can be replaced by a new simple closed curve, $C_i'$ formed by replacing $\partial E(b)$ by $\partial E(i)$ with the directions of its edges reversed. By C3, no boundary segment of C other than E(b) is affected by this change and (3.4) - (3.6) are valid with $C_i'$ replacing $C_i$.

If $\partial E(b)$ is not a single arc, or if $\partial E(i)$ has interior vertices in common with C, the situation is more complex. Its discussion can be simplified somewhat by introducing, for each vertex, Q of each element a circle of radius $\varepsilon$ centred on Q, with $\varepsilon$ so small that only edges of the mesh incident on Q intersect the circle, and P does not lie in any of these circles. An element circuit will then be defined to be a simple closed curve resulting from following an edge of the element along its direction until it encounters a circular arc and then entering the element along the arc until it leaves the element and so on. If $G$ is a subset of elements of the mesh, then the algebraic sum of element circuits for elements in $G$ will be denoted $\gamma(G)$ and referred to as the circuit of $G$.

This curve will consist of the edges of $E \in G$ for which there is no neighbouring element in $G$, traversed between the circular arcs about the vertices of E exactly once, plus a section of such an arc, plus possibly a number of circles about vertices which are interior to $G$. While it can be seen, for arbitrary choice of $G$, that $\gamma(G)$ is a collection of closed curves, it is less obvious that they are simple closed curves and we do not use this for general choice of $G$, but establish it only for the particular choices required by the proof.

The proof proceeds inductively by constructing a finite sequence of subsets of elements, $G_1, G_2, \ldots, G_F$, such that

(3.7)

a) each piecewise linear segment of $\gamma(G_i)$ is the directed edge of exactly one element in $G_i$, which lies on its left.

b) $\gamma(G_i)$ is a set of simple closed curves, $C_k$, $k = 1, 2, \ldots, K_i$ with $C_k \subset$ interior of $C_1$ for $k = 2, \ldots, K_i$ if $K_i > 1$

c) if P lies in an element E of the mesh, then $E \subset G_i$

d) (3.4) - (3.6) hold for $\gamma(G_i)$.

The sequence is constructed by starting with $G_1$ = the entire mesh, which clearly satisfies (3.7a)-d)), and by consecutively removing elements with edges in $\gamma(G_{i-1})$ which do not contain P. It terminates when there are no such elements left, with $G_F$ a subset of elements such that each piecewise linear arc of $\gamma(G_F)$ is an edge of an element containing P. It is then shown that there can only be one element in $G_F$, which completes the proof.

Suppose we have constructed subsets $G_1$, $G_2$, ..., $G_i$ and let $E \in G_i$ with $\partial E \cap \gamma(G_i) \equiv \partial E(b)$ be non empty and $P \notin E$. Let $\partial E(i) = \partial E - \partial E(b)$; in order to establish a) for $G_{i+1}$, we show that an edge of $\partial E(i)$ does not intersect any other edge of the mesh. Suppose, to the contrary, that there were an element, H, for which an edge of H intersects $\partial E(i)$. Then the intersecting edge of H cannot be a boundary edge, by C3, and the intersection must take place outside the circles of radius $\varepsilon$ about each vertex. Consider part of an element circuit for E which starts at a point $Q_1$ on $\partial E(b)$ and terminates inside $H \cap E$ at $Q_2$. If necessary, we can move $Q_2$ from the edge of E slightly inside E, in which case, by the induction hypothesis (3.7a)) $Q_1$ lies in one element while $Q_2$ lies in two elements and they are joined by an arc passing through no vertex or boundary edge of the mesh, violating Lemma 1. Clearly, then, in the circuit of $G_i$ - E, at no point can a linear segment intersect another linear segment and so by the construction of the circular segments, the closed curves of $\gamma(G_i\text{-}E)$ are simple.

However, these curves may have disjoint interiors. If we denote by $B_1$, $B_2$, ..., $B_M$ for $M \geq 1$, the curves of $\gamma(G_i\text{-}E)$ of bounded interior then one of them contains the connected component of the interior of $\gamma(G_i\text{-}E)$ containing P, which we relabel $B_1$. Then $G_{i+1}$ is defined to be the set of elements intersecting the interior of $B_1$, and it follows that (3.7a)) through d) follows for $G_{i+1}$.

Eventually this induction process leads to a subset of elements, $G_F$, for which every element with an edge in $\gamma(G_F)$ contains P. If $G_F$ contains more than one element, then $\gamma(G_F)$ has edges from more than one element, so

suppose this latter to be the case. A point, Q, on an edge in $\gamma(G_F)$ belonging to element $E \in G_F$ can be joined to P by an arc lying in E. At Q, this arc lies in exactly one element, whereas at P it lies in more than one, violating Lemma 1. Hence $G_F$ consists of one element and the theorem is proven.

## §4 Verifying C2

In the process of verifying C2, we build two lists, a list of elements incident on a given vertex, and a list of the neighbours of each element. As mentioned in §1, these lists are useful in a variety of contexts in the finite element method that require geometric adjacency information that is not directly available from the basic mesh lists. A convenient method for building these lists is described by Lewis and Robinson in [11], and the basic ideas have doubtless been used in one form or another by many implementors of finite element programs. By adding some refinements to the basic idea (for which no published algorithmic description seems to be available) we obtain an algorithm which serves both the purposes of building these lists and checking C2 of the mesh definition. This algorithm is developed in this section.

The method involves first constructing a vertex incidence list, i.e. a list whose kth entry is the sublist e(i,k), i = 1, 2, ..., K(k), where e(i,k) is the index of an element which has vertex k as one of its vertices. K(k) is then the number of elements incident at vertex k, which, in general, varies with k. The vertex incidence list is the 'inverse' list to the element incidence list in the sense that for the $j^{th}$ element and its $i^{th}$

vertex, j is in the sublist e(s, v(i,j)), s = 1, 2, ..., K(v(i,j)).* In the following algorithm to construct e(i,k), K(k) is used as an array of pointers to the last entry made in list e(i,k) for each k.

```
for k = 1      to  N_v
  K(k) ← 0                                        (4.1)
for j = 1      to  N_e
    for i = 1 to  I(j)
      k ← v(i,j)
      ℓ ← K(k) + 1
      e(ℓ,k) ← j
      K(k) ← ℓ
```

When operating on an element incidence list as defined in §2, this algorithm produces a vertex incidence sublist for vertex k, which is sorted into increasing order,

$$e(1,k) < e(2,k) < \ldots < e(K(k),k), \tag{4.2}$$

which can be used to advantage in searching these sublists. Note that the success of this algorithm does not depend on the order of the vertex indices in the element incidence sublists. In particular, then, its success does not depend on the vertex orientation requirements of C1 nor the edge consistency requirements of C2.

From this list, the algorithm constructs an element neighbour's list, which is a list of $N_e$ entries in which the $j^{th}$ entry is a sublist, n(i,j), i = 1, 2, ..., I(j), with n(i,j) being the index of the neighbour of element j on its $i^{th}$ edge, or zero if this edge is a boundary edge. In outline, the algorithm for constructing this list consists of a synchronized scan over the element incidence list and the boundary reference table

---

* Such inverted lists are discussed, usually rather briefly, in texts on list processing, see e.g. Knuth, Sorting and Searching, for remarks.

examining the edges of elements in order. If, on the current edge of the current element, it is determined that there is a neighbour of higher index, then this fact is recorded in the neighbour's sublists of both the current element and its neighbour. However, prior to recording this, a check is made to be sure that for the current edge of the current element the presence of a neighbour of lower index has not already been recorded, which would be a violation of C2. If neighbours of neither higher nor lower indices are present, the edge is presumed to be a boundary edge and the boundary reference table is checked for consistency. The remainder of this section describes these steps in further detail and need not be read prior to §5.

If element j has a neighbour on its $i^{th}$ edge, of index $\ell > j$ then this can be determined by searching the vertex incidence sublists of vertices $v(i,j)$ and $v(\overline{i+1},j)$, the end points of the $i^{th}$ edge, for a common entry (recall (2.1), $\overline{i} \equiv i \bmod I(j)$). This search is described in the following algorithm, which is labelled as a procedure returning the common entry's value in a variable named $\ell$ so that it can be referred to in the subsequent part of §4. The variables of our discussion are regarded as global to the procedure, and comments are enclosed in face brackets.

```
procedure  find common entry [ℓ]
v1 ← v(i,j)
v2 ← v(ī+1,j)
{p1 and p2 are pointers into the vertex incidence sublists
 for v1 and v2}
p1 ← 1
p2 ← 1
while (e(p1,v1) ≤ j)   p1 ← p1+1
while (e(p2,v2) ≤ j)   p2 ← p2+1
{look for common entry in these sublists above j in value}

while e(p1,v1)  ≠ e(p2,v2)
  if  e(p1,v1) < e(p2,v2)
    then    p1 ← p1+1
    else    p2 ← p2+1
ℓ ← e(p1,v1)
```

(4.3)

After locating $\ell$, n(i,j) can be set to $\ell$ and it is known that element j is the neighbour of element $\ell$ for some local edge number, m. More precisely, it is known that the vertex indices v(i,j) and $v(\bar{i}+1,j)$ appear in the element incidence sublist for $E(\ell)$, but it is not assured that $v(\bar{i}+1,j)$ immediately precedes v(i,j) as is necessary to meet the edge consistency requirements of C2. The following procedure returns m if the edges of E(j) and $E(\ell)$ are consistent and -m otherwise.

```
procedure  find complementary local edge number [m]

m ← 1

ī ← i mod I(j)
while  (v(m,ℓ) ≠ v(ī+1,j))   m ← m+1
m̄ ← m mod I(ℓ)
if  v(m̄+1,ℓ) ≠ v(i,j)  then  m = -m
```

(4.4)

If no such $\ell$ exists, then (4.3) will fail by p1 or p2 becoming an invalid pointer into the corresponding sublist. This failure can be avoided by extending the element incidence list by a 'guard' entry at the end of each sublist, i.e.

```
for k = 1 to Nv                                    (4.5)
    e(K(k)+1,k) ← Ne + 1
```

which maintains the order of the sublists (4.2). Then 'find common entry' (4.3) ends either with $\ell$ in the range $j < \ell \leq N_e$ indicating the presence of a neighbour of higher index, or $\ell = N_e + 1$ indicating none.

The verification of C2 requires a check that each edge of each element is either an internal edge or a boundary edge. This information can then be used to verify that the boundary reference table contains references to boundary edges exclusively and exhaustively. Initially, the neighbours list is set to zero, i.e.

```
for j = 1 to Ne
  for i = 1 to I(j)                                (4.6)
    n(i,j) ← 0
```

The verification is then carried out as in algorithm (4.7) by a scan over the mesh edges in element order, and for each element, in local edge number order; the current edge of the scan will be the $i^{th}$ edge of the $j^{th}$ element. The first step of the check is to establish the presence or absence of a neighbour on this edge. Procedure (4.3), find common entry, is used to determine the presence of a neighbour of index $\ell > j$. If one is found, then a check is made to ensure that a second neighbour of index less than

j has not already been recorded in the neighbours list. If not, $\ell$ is assigned to n(i,j); the edge consistency between these neighbours is checked using (4.4), and the presence of element j as a neighbour of element $\ell$ is recorded. This latter assignment is made regardless of edge consistency so that the $m^{th}$ edge of element $\ell$ will not be interpreted as a boundary edge later in the scan.

When the status of a neighbour for the $i^{th}$ edge of the $j^{th}$ element has been determined, the consistency of the boundary reference table can be assessed. A scan of this table is synchronized with the scan of element edges by maintaining a pointer, k, into the table. If the current edge of the scan coincides with the table entry pointed to by k, then the table is consistent if the current edge is a boundary edge, and inconsistent if it is not a boundary edge. In this case, when the consistency has been checked, the pointer k must be incremented by one to maintain synchronization. Conversely, if the current edge is not referenced by the table entry pointed to by k, then the table is consistent if the edge is an internal edge, but inconsistent if it is a boundary edge. The discovery of a violation of C2, or the inconsistency of the boundary reference table are marked in the algorithm by comments, which in an implementation would be replaced by an error recording and/or reporting mechanism. If no violation of C2 is encountered, a valid element neighbours list is constructed. If the boundary reference table were not constructed a priori, then the section of (4.7) which validates it could be replaced by a section which constructs it.

```
for j = 1 to Ne
  for   i   to   I(j)
    {first - establish presence of a neighbour}
    find common entry [ℓ]    {via (4.3)}
    if  ℓ ≤ Ne
      then
        if   n(i,j) ≠ 0
          then  {violation of C2, multiple neighbours}
          else  {one neighbour of index  ℓ > j}
            n(i,j) ← ℓ
            find complementary local edge number [m] {via (4.4)}
            if   m > 0
              then  n(m,ℓ) ← j
              else  {violation of C2, edge inconsistency}        (4.7)
                    n(-m,ℓ) ← j
    {second - check consistency of boundary reference table}
    if  b(1,k) = j     and     b(2,k) = i
      then
        if   n(i,j) ≠ 0
          then   {brt inconsistent - contains reference to internal edge}
        if   k < Nb
          then     k ← k+1
      else
        if   n(i,j) = 0
          then    {brt inconsistent, reference to boundary edge omitted}
```

When, as is usual in practice, the elements have a fixed number of edges, the implementation of the element incidence and element neighbours lists are simplified considerably because their sublists have fixed length and hence the lists are two dimensional arrays. This simplication does not extend to the vertex incidence list so readily, as a typical case of a modestly irregular triangular mesh shows. However, the extent of variation

in the number of elements meeting at a vertex (= the length of the sublist for that vertex) is usually fairly small; even for triangular meshes these lengths could be expected to range between 4 and 11 (allowing for the terminating guard entry of (4.5)] for most meshes. Hence, particularly if the vertex incidence list is to be created temporarily using working space storage as in this algorithm, the simplicity to be gained by using a two dimensional array of maximal sublist length to implement it can be expected to usually justify the resulting inefficient use of memory space. In our FORTRAN implementation of the algorithm this is done, and since the array K(k) of sublist lengths is only needed during (4.1), the first column of n(i,j) is used for this purpose.

As a rough guide to the expected running time characteristics of (4.7), it can be seen from the looping structure that if the elements have a fixed, or bounded, number of sides, and a bounded number of them can be incident on any one vertex, then the running times can be expected to be linear with respect to $N_e$ and $N_b$, with dependence on $N_e$ being the dominant effect.

## §5 Verification of C3 and C4

Algorithm (4.7), which checks condition C2 of the mesh definition and the consistency of the boundary reference table, is the first of three major steps in verifying the mesh. The second involves a pairwise comparison of entries in the boundary reference table to check C4 and part of condition C3 of the definition. In this step, the number of independent boundary curves is determined for the third major step, which verifies condition C1 and the remaining part of C3 in a scan over the element list.

The algorithms for these steps are designed so that they can each be run independently of the outcome of the previous ones to provide as much diagnostic information as possible about invalid meshes.

Several comments are in order before we embark on a more detailed discussion of these steps. In the preceding sections, the geometric shape of the finite elements of the mesh played little role so there was no benefit to restricting this shape beyond its being a general polygonal domain. However, such generality adds considerable complexity to the discussion of this section, which seems unwarranted since most of the meshes in common use consist of triangle and quadrilaterals. We shall discuss triangular elements, pointing out that the algorithm extends directly to elements which can be triangulated conveniently. For example, if E(j) is convex, with I(j) vertices, then by introducing the I(j)-2 traingles $T_i(j)$ with vertices v(i,j), v(i+1,j) and v(i+2,j), we have the triangulation for E(j) of

$$E(j) = \bigcup_{i=1}^{I(j)-2} T_i(j) \ . \qquad (5.1)$$

We have assumed that the lists to be verified do represent a valid collection of traingles, e.g. that the values of the entries in the incidence list and boundary reference table fall in the correct ranges, or that the entries in the vertex coordinate list are unique. Such a check would be a useful part of a mesh verification program, but conditions that the lists represent triangles are fairly obvious and can be checked by simple inspection algorithms. Another significant consideration in implementing a mesh verification program is the fact that it depends on real number arithmetic and comparisons. Despite its importance to a

successful implementation, we shall not comment further on this because the techniques for dealing with these difficulties are not peculiar to these calculations and are well documented in texts on numerical computation (e.g. [5]).

The third condition of the definition, C3, requires that no interval of a boundary edge intersect any mesh element other than the one of which it is an edge. It can be seen from the proof of Theorem 1 in §3 that the boundary edges of a collection of triangles satisfying conditions C2 and C4 of the definition form a set of simple oriented closed polygonal curves which do not cross at the vertices of the mesh. Suppose that one vertex $P_1$ of such a curve is known to lie outside each element except those on which it is incident. We shall refer to $P_1$ as the starting vertex for this boundary curve, and we shall refer to the order in which the edges appear when the boundary curve is traversed from $P_1$ in the direction of its orientation as curve order. If any edge in the curve intersects an element of the collection of triangles in the manner forbidden by C3, then there must be a first such edge in the curve order of edges, and this first edge must intersect some boundary edge. The intersection may occur at the endpoint (=vertex) of one of the two edges involved, but not both, or a violation of C4 occurs. Hence condition C3 may be divided into two sub-criteria

C3 i) Check one point of each boundary curve to ensure that it lies outside every element on which it is not incident

C3 ii) Check that no two boundary edges intersect unless they are consecutive edges of a boundary curve. (5.2)

The second major step of the mesh verification algorithm described below at (5.6) checks criterion C3 ii) and condition C4. The outline of an algorithm for determining when two or more of a set of $N_b$ line segments intersect which runs in times $O(N_b \log (N_b))$ has been described by Shamos and Hoey in their interesting paper, [14], which also shows that this behaviour is asymptotically optimal for this task. We describe here a much simpler algorithm which can be expected to run in times $O(N_b^2)$. If execution efficiency considerations justified the additional complexity of implementing an optimal algorithm for this step, then the approach of [14] could probably be substituted for the algorithm described here. However, since the first step of the verification process is generally $O(N_e)$, and for most region shapes $O(N_b^2) = O(N_e)$, no improvement in the total asymptotic running time behaviour for most meshes would be expected.

An edge will be referred to as checked when criterion C3 ii) has been examined for it, which involves comparing the edge to every other unchecked boundary edge of the collection of triangles, except its predecessor and successor in curve order. The check of a boundary edge from the vertex of index p to that of index q uses the function

$$f_{p,q}(x,y) = (y(q)-y(p))(x-x(p)) - (x(q)-x(p))(y-y(p)) \qquad (5.3)$$

which vanishes only for (x,y) on the line through the vertices and takes values of opposite sign for points on opposite sides of this line. If r and s are the indices of two other vertices, then the line segment from r to s intersects that from p to q only if

$$f_{p,q}(x(r),y(r))\ f_{p,q}(x(s),y(s)) \le 0$$

and (5.4)

$$f_{r,s}(x(p),y(p))\ f_{r,s}(x(q),y(q)) \le 0 \quad .$$

The boundary edges are checked along each boundary curve in curve order. For the scanning of a curve, three pointer variables into the boundary reference table are used; a pointer named start which marks the starting vertex of the first edge of a curve to be checked, a pointer, k, into this table is maintained to reference the edge currently being checked, and a pointer, next, is maintained to the successor of edge k. When next coincides with start, the curve has been scanned. If unchecked boundary edges remain, it is necessary to reinitialize start on a new boundary curve.

To keep track of which edges have been checked, algorithm (5.6) uses a temporary boolean array of length $N_b$ named 'checked' with 'checked (k) = true' when the edge described by the $k^{th}$ entry of the boundary reference table has been checked. This step of the algorithm also determines the number of distinct boundary curves, and the indices of one vertex on each curve, which is used by the third step of the verification algorithm to

check criterion C3 i). This information is stored in algorithm variable ' ncurves' and array 'bv(i), i = 1 to ncurves' respectively.

The Algorithm (5.6) requires a valid boundary reference table, as provided from algorithm (4.7) of the preceding section. As long as C2 and C4 are not violated, then each boundary edge has a unique successor (Theorem 1 of §3) so that the algorithm can proceed to check all edges. However, if a violation of these conditions has occurred this is no longer guaranteed, which could result in an infinite loop developing during the scanning of a boundary curves edges. Hence, the outer control structure of algorithm (5.6) is a while loop with two exit criteria, (a) nchecked = $N_b$, regarded as the 'normal' exit, although it may occur with invalid as well as valid meshes, and (b) next = k indicating that no successor was found for the edge referenced by the $k^{th}$ entry of the boundary reference table, which can only occur for an invalid mesh. Algorithm (5.6) requires the following initializations of variables

```
ncurves  ← 0                                        (5.5)
nchecked ← 0
k        ← 0
next     ← 1
start    ← 1
for  m = 1  to  N_b
    checked (m) ← false
```

```
while nchecked < N_b      and     next ≠ k
  {check if current boundary curve is completed}
  if   next = start
    then  {initialize next to start of new boundary curve}
      next ← 1
      while  checked (next)       next ← next + 1
      start ← next
      ncurves ← ncurves + 1
      bv(ncurves) ← v(b(2,start), b(1,start))
  k ← next
  p ← v(b(2,k), b(1,k))
  q ← v(b(2,k) + 1, b(1,k))
  for  m = 1   to   N_b                                          (5.6)
    if  m ≠ k   {check m^th boundary edge against k^th}
      then
        r ← v(b(2,m), b(1,m))
        s ← v(b(2,m)+1, b(1,m))
        if  p = r
          then  {violation of C4}
        if  q = r
          then  {m^th edge is successor of k^th edge}
            next ← m
          else
            if  not checked (m)
              then
                if  edges intersect {using (5.4)}
                  then  {violation of C3ii}
  nchecked ← nchecked + 1
  checked (k) ← true
  {end of while loop}
```

The third major step of the algorithm involves a scan over the elements to verify C1, i.e. that their vertices are listed in the element incidence list in counter clockwise order, and C3 i) of (5.2), i.e. the starting

vertices of the boundary curves do not lie in elements other than those of which they are incident as vertices. These checks can be conveniently implemented for triangular elements using the familiar transformation to the area coordinates of an element. For completeness, we review the formulae for this transformation, which can be found, e.g. in [19]. Let the coordinates of the $j^{th}$ element be

$$x_k = x(v(k,j)),\ y_k = y(v(k,j))\ k = 1,2,3 \tag{5.7}$$

and let

$$D = \det \begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix} \tag{5.8}$$

$$A_1 = (y_2-y_3)/D;\ B_1 = (x_3-x_2)/D;\ C_1 = (x_2y_3-x_3y_2)/D$$

$$A_2 = (y_3-y_1)/D;\ B_2 = (x_1-x_3)/D;\ C_2 = (x_3y_1-x_1y_3)/D$$

$$A_3 = (y_1-y_2)/D;\ B_3 = (x_2-x_1)/D;\ C_3 = (x_1y_2-x_2y_1)/D\ .$$

Then the area coordinates relative to the triangle E(j) of a point of the plane having Cartesian coordinates (x,y) are the three numbers

$$L_i(x,y) = A_ix + B_iy + C_i\ i = 1,2,3 \tag{5.9}$$

Their significance to our discussion is that (x,y) lies in E(j) if and only if $0 \le L_i(x,y) \le 1$ for i = 1,2,3. Moreover, the vertices of E(j) form a triangle and are specified in counter clockwise order if and only if $D > 0$. The condition $0 \le L_i(x,y) \le 1$ means, geometrically, that (x,y) lies

in the strip between the dashed lines of Figure 5.1

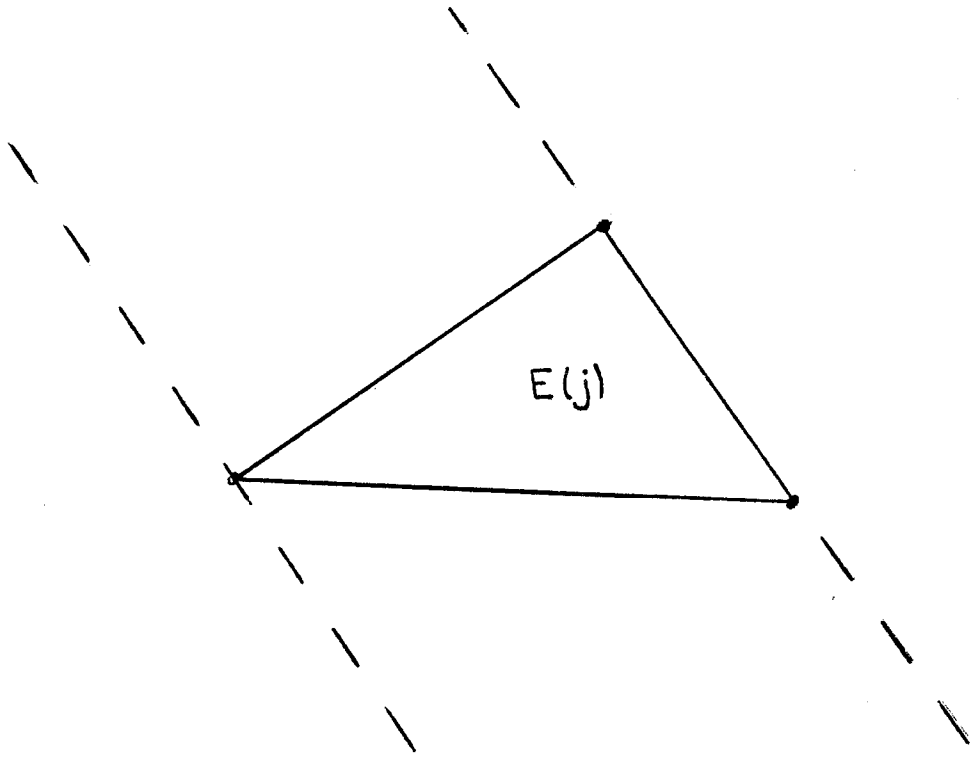



Figure 5.1

Even if there are several boundary curves, it can be expected that the starting vertex of any of them will fall in this strip only for relatively few elements. Hence, in algorithm (5.10) we compute and check the $A_i$, $B_i$, $C_i$ and $L_i(x,y)$ sequentially, expecting that $L_1$ will not lie between 0 and 1 and we can avoid computing them for $i = 2$ and 3 for most of the elements. An alternative test has been used by C.L. Lawson in [10] for determing whether a point lies in an element or not.

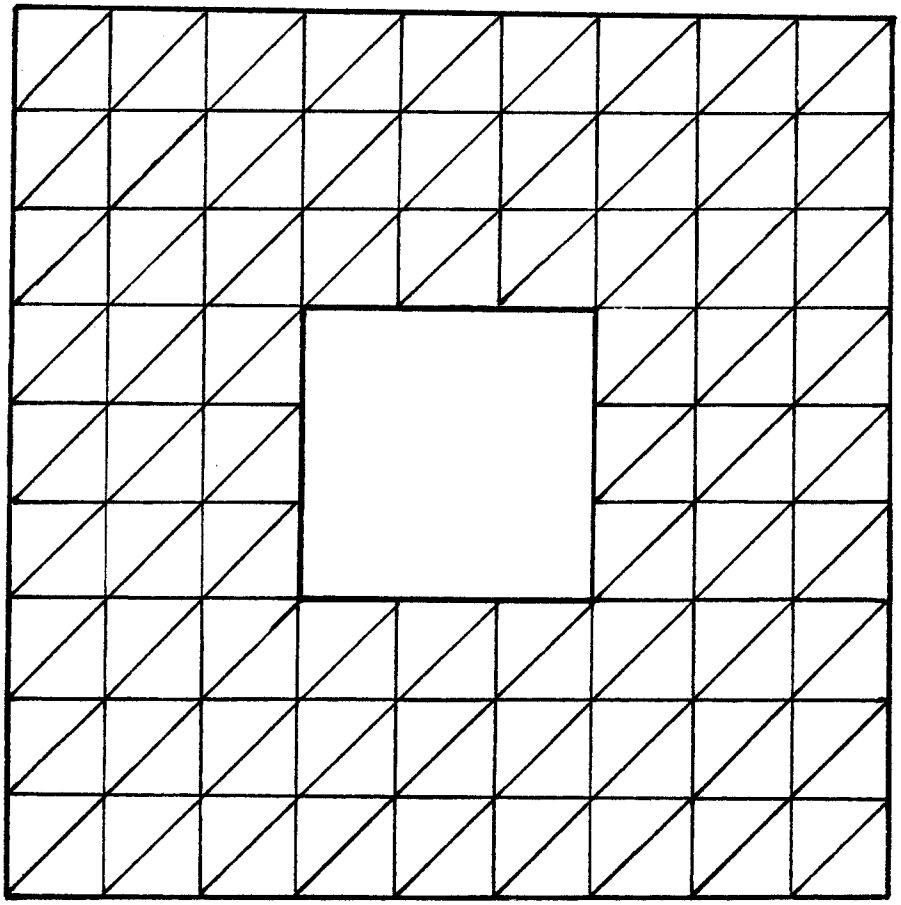
The scan for verifying C1 and C3 i) can be described by the following algorithm, using data concerning the number of boundary curves and a starting vertex on each, ncurves and bv(i), i = 1 to ncurves from (5.6) and the vertex incidence list, e(k,n) , k = 1 to K(n) , n = 1 to $N_v$ from (4.7).

```
for  j = 1   to   Ne
  compute D  {via (5.8)}
  if   D ≤ 0
    then    {violation of C1}
  for   n = 1   to    ncurves
      {check if element j is incident on vertex of index bv(n)}
    for   k = 1 to  K(bv(n))
      if  j = e(k,bv(n))
        then   incident ← true
    if not incident
      then
        x ← x(bv(n))
        y ← y(bv(n))
        compute A1, B1, C1, L1(x,y)   {via (5.8)}
        if   0 ≤ L1 ≤ 1
          then
            compute A2, B2, C2, L2(x,y)  {via (5.8)}
            if   0 ≤ L2 ≤ 1
              then
                compute A3, B3, C3, L3(x,y) {via (5.8)}
                if   0 ≤ L3 ≤ 1
                  then  {violation of C3.i)}
```

(5.10)

## §6. Performance of a FORTRAN Implementation

A FORTRAN implementation of the mesh verification algorithm has been prepared and some tests run on regular meshes to provide some evidence of run time performance which we discuss here, as well as some tests to check out the various mesh errors detected, which are not reported here. The meshes for the performance tests comprise two families each characterized by a parameter N which is proportional to the number of elements in the mesh. One family is a collection of valid meshes on a hollow square, as shown in Figure 6.1,



and having $3\sqrt{N}$ triangle edges along each outer side of the square, and $\sqrt{N}$ triangle edges along the inner sides of the 'hollow'. The second family is a collection of invalid 'meshes' created by folding the meshes of the first family along the line $y = x/3$, i.e. by applying the transformation

$$(\bar{x}, \bar{y}) = (x, y) \quad \text{if } y \geq x/3$$
$$= \frac{1}{5}(4x + 3y, 3x - 4y) \quad \text{if } y < x/3$$

The meshes are quite regular in the sense that 6 triangles meet at each interior vertex, and less than 6 meet at each boundary vertex; hence one would expect the algorithm's running time to be proportional to N. The FORTRAN implementation used for the test was compiled using the IBM optimizing compiler, FORTRAN H extended, and executed on the University of Waterloo's IBM 3031. For the tests involving the invalid meshes, the details of the mesh errors were determined in each case, but the detailed output of these details was not performed.

The running time for the algorithm to check meshes 1600 to 4096 triangles are shown in Tables 6.1 and 6.2, along with slopes of successive line segments of the graph of running times versus N. The algorithm seems clearly to be behaving as linear in the number of triangles in the mesh, at least over this range of meshes. It can be seen that there is a slight premium to be paid for the determination of the nature of the errors in the invalid meshes.

| i | $N_i$ | No. of Triangles | time = $t_i$ (seconds) | Slope = $(t_i-t_{i-1})/(N_i-N_{i-1})$ |
|---|---|---|---|---|
| 1 | 100 | 1600 | 3.7 | |
| 2 | 144 | 2304 | 5.3 | .036 |
| 3 | 196 | 3136 | 7.3 | .038 |
| 4 | 256 | 4096 | 9.5 | .037 |

Test Timings for Valid Mesh
on Hollow Square

Table 6.1

| i | $N_i$ | No. of Triangles | time = $t_i$ (seconds) | Slope = $(t_i-t_{i-1})/(N_i-N_{i-1})$ | No. of errors reported |
|---|---|---|---|---|---|
| 1 | 100 | 1600 | 4.4 | | 254 |
| 2 | 144 | 2304 | 6.3 | .043 | 375 |
| 3 | 196 | 3136 | 0.5 | .042 | 522 |
| 4 | 256 | 4096 | 11.2 | .045 | 692 |

Test Timings for Invalid
(folded) Mesh on Hollow Square

Table 6.2

References

1. Ahlfors, L.V., Complex Analysis, McGraw Hill, 1966.

2. Babuska, I. and Aziz, A.K., On the angle condition in the finite element method. SIAM J. of Num. Anal. 13, 214-226, 1976.

3. Bykat, A., Automatic generation of triangular grids, I-Subdivision of a general polygon into convex subregions, II-Triangulation of convex polygons. Int. J. of Num. Meth. Eng. 10, 1329-1342, 1976.

4. Cavendish, J.A., Automatic triangulation of arbitrary planar domains for the finite element method. Int. J. of Num. Meth. Eng. 8, 679-696, 1974.

5. Forsythe, G., Malcolm, M.A., and Moler, C.B., Computer Methods for Mathematical Computations. Prentice Hall, 1977.

6. Gorden, W.J. and Hall, C.A., Construction of curvilinear coordinate systems and applications to mesh generation. Int. J. Num. Meth. Eng. 7, 461-477, 1973.

7. George, J.A., Computer implementation of the finite element method, Tech. Rep. STAN-CS-208, Stanford Univ., Stanford, Calif., 1971.

8. Gross, J.L. and Rosen, R.H., A linear time planarity algorithm for 2-complexes, J. ACM 26, 611-617, 1979.

9. Hopcroft, J. and Tarjan, R., Efficient planarity testing, J. ACM 21, 549-568, 1974.

10. Lawson, C.L., Software for $C^1$ surface interpolation, Mathematical Software III, ed. J.R. Rice, 1977. Academic Press.

11. Lewis, B.A. and Robinson, J.S., Triangulation of planar regions with applications. Comp. J. 21, 324-332, 1978.

12. Mitchell, A.R. and Wait, R., The Finite Element Method in Partial Differential Equations. J. Wiley, 1977.

13. Sewell, E.G., An adaptive computer program for the solution of Div(p(x,y) grad u) = f(x,y,u) on a polygonal region. The Mathematics of Finite Elements and Applications II, ed. J.R. Whiteman, MAFELAP 1975, Academic Press, 1976.

References - Cont'd.

14. Shamos, M.I. and Hoey, D., Geometric Intersection Problems, 17th Annual Symp. on Found. of Comp. Sc. 208-215, IEEE, 1976.

15. Simpson, R.B., Automatic local refinement for irregular rectangular meshes, Int. J. for Num. Meth. Eng. 14, 1665-1678, 1979.

16. Simpson, R.B., A survey of two dimensional mesh generation Proc. Ninth Manitoba conference on Numerical Mathematics and Computing, 1980.

17. Strang, G. and Fix, G.J., An Analysis of the Finite Element Method. Prentice Hall, 1973.

18. Van Emden, M.H., Programming with verification conditions, IEEE Trans. Soft. Eng. 5, 148-159, 1979.

19. Zienkiewicz, O.C., The Finite Element Method in Engineering Science, McGraw Hill, 1971.