A General Three Dimensional All Quadrilateral Surface Mesh Generation Algorithm

Roger Cass
Brigham Young University - Provo
A GENERALIZED THREE DIMENSIONAL ALL QUADRILATERAL SURFACE MESH GENERATION ALGORITHM

ROGER CASS
A Generalized Three Dimensional
All Quadrilateral Surface Mesh
Generation Algorithm

A Thesis
Presented to the
Department of Civil Engineering
Brigham Young University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by
Roger Cass
April 1992
This thesis by Roger Cass is accepted in its present form by the Department of Civil Engineering of Brigham Young University as satisfying the thesis requirement for the degree of Master of Science.

Steven E. Benzley, Committee Chairman

Richard J. Balling, Committee Member

July 13, 1992
Date

Lavere B. Merritt, Department Chairman
# Table of Contents

1 Table of Contents

2 List of Figures

3 Introduction

4 Overview of Research

5 Research Findings

6 Example Meshes

7 Conclusions

5.1 Future Enhancements

---

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table of Contents</td>
<td>iii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>v</td>
</tr>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 The Finite Element Technique</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Mesh Generation Techniques</td>
<td>4</td>
</tr>
<tr>
<td>Overview of Research</td>
<td>10</td>
</tr>
<tr>
<td>2.1 The Paving Algorithm</td>
<td>10</td>
</tr>
<tr>
<td>2.2 The Original Paving Algorithm</td>
<td>12</td>
</tr>
<tr>
<td>2.3 New Capabilities</td>
<td>12</td>
</tr>
<tr>
<td>Research Findings</td>
<td>13</td>
</tr>
<tr>
<td>3.1 Support of Arbitrary 3D Surfaces</td>
<td>13</td>
</tr>
<tr>
<td>3.2 Improving Logic and Robustness</td>
<td>22</td>
</tr>
<tr>
<td>3.3 Enhancing Modularity</td>
<td>23</td>
</tr>
<tr>
<td>Example Meshes</td>
<td>25</td>
</tr>
<tr>
<td>4.1 A Sphere with Multiple Holes</td>
<td>25</td>
</tr>
<tr>
<td>4.2 Two Intersecting Cylinders</td>
<td>27</td>
</tr>
<tr>
<td>4.3 The Piston</td>
<td>28</td>
</tr>
<tr>
<td>4.4 The Block and Torus</td>
<td>30</td>
</tr>
<tr>
<td>Conclusions</td>
<td>33</td>
</tr>
<tr>
<td>5.1 Future Enhancements</td>
<td>34</td>
</tr>
<tr>
<td>Appendix A</td>
<td>Implementation</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
</tr>
<tr>
<td>A.1 Definition</td>
<td>38</td>
</tr>
<tr>
<td>A.2 Boundary Preparation</td>
<td>41</td>
</tr>
<tr>
<td>A.3 Functional Decomposition</td>
<td>43</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Appendix B</th>
<th>Object Oriented Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.1 Programming Environment</td>
<td>57</td>
</tr>
<tr>
<td>B.2 Object Oriented Programs</td>
<td>57</td>
</tr>
<tr>
<td>B.3 Entities</td>
<td>58</td>
</tr>
<tr>
<td>B.4 Tools</td>
<td>58</td>
</tr>
<tr>
<td>B.5 The CUBIT Structure</td>
<td>59</td>
</tr>
<tr>
<td>B.6 Engineering Applications</td>
<td>60</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1.1. Grid Overlay Technique 5
Figure 1.2. Nodes-First Technique 6
Figure 1.3. Parametric Mapping Technique 7
Figure 1.4. Progressive-Front Technique 8
Figure 3.1. Projection of a Node 15
Figure 3.2. Closest Distance Between Two Lines 18
Figure 3.3. Closest Distance Between Two Line Segments 20
Figure 4.1. Side View of Spherical Surface 25
Figure 4.2. A Sphere with Multiple Holes 26
Figure 4.3. Two Intersecting Cylinders 28
Figure 4.4. The Piston, Showing Boundaries 29
Figure 4.5. The Piston, Showing Transitions 30
Figure 4.6. The Block and Torus 31
Figure 4.7. The Block and Torus, Top View 32
Figure A.1. Functional Decomposition 39
Figure A.2. Classification of Nodes 40
Figure A.3. Limiting Angles for Node Types 40
Figure A.4. Exterior and Interior Boundaries 41
Figure A.5. Closure Cases 46
Figure A.6. Types of Projections 47
Figure A.7. Expanding and Contracting Boundaries 49
Figure A.8. Seam on a Boundary 50
Figure A.9. Inserting a Wedge Before Seaming 50
Figure A.10. Propagating a Seam 51
Figure A.11. Qualifying Cases for Face Deletion 54
Figure A.12. Making Three Better from Two Poor Faces 55
Figure B.1. Object Tree for a Simple Paver Tool Application. 59
Engineers have long been striving to increase the percentage of their time spent improving designs and discovering new solutions. The general acceptance, and then overwhelming integration, of computers into engineering analysis has provided the foundation for a whole new set of tools. For the structural engineer, the foremost among these tools is the finite element technique.

Finite element technology has introduced its own set of obstacles. One that is particularly time consuming for engineering analysts is the construction of a finite element mesh. This thesis presents an algorithm for automatic finite element mesh generation using quadrilaterals on arbitrary three dimensional surfaces.

1.1 The Finite Element Technique

The finite element technique has found wide spread application in engineering and design since the late 1960's. The advent of large and reliable computers made the enormous number of calculations feasible. The finite element technique has become the keystone to understanding complex mechanisms and processes without the aid of expensive prototypes.
1.1.1 Finite Element Meshes

The fundamental concept of the finite element technique is the approximation of the shape of a complex object with a grid or collection of simpler shapes, known as elements. The mathematical models of the elements allow an analyst to compute an engineering parameter, such as stress or temperature, throughout the entire object as approximated by the value of the parameter in each element.

The set of elements used to approximate a complex object is known as a finite element mesh. The task of constructing these elements is known as meshing. When an object is analyzed using the finite element technique, the entire mesh is used to transform a continuous differential equation to a set of simultaneous algebraic equations. These equations approximate the actual behavior of the object when subjected to different types of boundary conditions.

Approximations inherent in the technique arise from many sources. One source is the simplification of the equations of behavior of the particular element type. Another source arises from the fact that each finite element is mathematically ideal when it is a perfect shape (i.e. being a cube and not being distorted in any way). In practical cases, many elements must be distorted to fit into the complex geometry of the object being modeled.

1.1.2 Accuracy and Cost of a Mesh

A key to a successful and accurate finite element analysis is a mesh that maximizes accuracy while minimizing costs. There are many ways to achieve these objectives, each suited to a particular application of the finite element technique. The research of this thesis focuses on structural applications.
There are three basic objectives in creating individual elements directly linked to the accuracy and cost of structural meshes:

- **Number** -- the more elements, the more equations must be solved
- **Size** -- the larger the element relative to the geometry, the larger the probability for error, but the fewer needed to create a mesh
- **Shape** -- the closer to the ideal shape (cube, triangle), the smaller the error of the approximation

### 1.1.3 Desired Mesh Qualities

The basic type of element has a great influence on the accuracy of results. The two basic types of surface elements are triangles and quadrilaterals. The two basic types of volume elements are tetrahedrons and hexahedrons. Triangles and tetrahedrons are much easier to generate in arbitrary geometric objects, but the resulting mesh often exhibits artificial stiffness in the analysis. Thus, in structural applications, the quadrilateral and hexahedron are desired.

Edges and boundaries, where the analysis functions have their greatest values and highest gradients, are often the areas of most importance in a structure. It is therefore of interest to put the best shaped elements nearest the boundary to make it sensitive to the analysis functions.

Additionally, in order to ensure the best shaped elements, a mesh should have a majority of regular nodes, where a regular node is defined as having exactly four attached elements. More or less than this number will force a distortion into one or more of the elements at that node.

---

1. Four for surfaces. Eight for volumes.
Finally, the mesh should be orientation independent; the geometry should be able to be transformed in space without a change in the resulting mesh. This is important when considering multiple analyses on the same object as it deforms.

In summary, the desired qualities are:

- Quadrilateral/hexahedral mesh
- Boundary sensitivity
- Maximized regular nodes
- Orientation independence

1.2 Mesh Generation Techniques

This thesis is concerned only with mesh generation on surfaces using quadrilaterals. Mesh generation has historically been performed by hand by skilled analysts. Such methods are tedious and prone to error. Automation of mesh generation has been greatly desired by analysts to free their time for analysis instead of mesh generation. Many techniques are available and can be grouped into four major categories[1]:

- Grid overlay
- Nodes-first
- Parametric mapping
- Progressive front

All techniques discussed here can be extended to three-dimensional surfaces using the node relaxation technique. This technique generates a mesh as though the surface were two-dimensional, then transforms the mesh into space and moves individual nodes to the actual surface from the transformed plane. This technique fails when the actual surface is highly curved or does not project accurately onto a plane.
For a technique to be generalized to three-dimensions, it must be able to generate a mesh in three-dimensional space and must put the nodes directly on the actual surface.

1.2.1 Grid Overlay

This technique superimposes a grid of a desired density, which can vary, on the object to be meshed. The boundary elements are then developed by selecting the squares where some percent (typically 50%) of the square lies inside the boundary, or by forming quadrilaterals along the boundary which follow the edges of the boundary. Figure 1.1 shows this technique applied to an object.

![Grid Overlay Technique](image)

Figure 1.1. Grid Overlay Technique

The technique can be successfully used on objects with interior holes and irregular boundaries, and all nodes are regular, except at the boundary. The drawbacks of this technique are that the mesh is not orientation independent, the worst elements
are put along the boundaries, and the technique cannot be extended to three-dimensional surfaces.

### 1.2.2 Nodes-First

This technique generates a regular grid of nodes along the boundaries of the object and in the interior without regard to forming elements. After all nodes have been generated, a triangulation technique is used to generate an all-triangle mesh. Quadrilaterals are then formed by removing interior lines, as shown in Figure 1.2.

![Figure 1.2. Nodes-First Technique](image)

Nodes-first has the desirable qualities of orientation independence, extension to three-dimensions, and generalization to any surface and boundaries. It does not provide boundary sensitivity or maximum regular nodes.
1.2.3 Parametric Mapping

The parametric mapping technique uses a parametric map of the surface, or regions thereof, to generate a regular grid of quadrilaterals. A parametric map uniquely defines any point in the region by a pair of coordinates \((u, v)\), that vary from 0.0 to 1.0. Parametric space has the severe limitations that there must be two pairs of opposing edges (the \(u\) and \(v\) axes), and there cannot be any interior holes in a region. Figure 1.3 shows an example of this technique. Notice that the original object must be broken down into regions to accommodate the hole.

![Meshing regions](image)

**Figure 1.3. Parametric Mapping Technique**

Parametric mapping is a powerful technique because it gives both a regular grid of elements with no irregular nodes and provides boundary sensitivity. It is limited when no suitable region can be formed without forcing a degenerate edge,\(^1\) such as.

---

1. A degenerate edge has a zero length. It has degenerated to a point.
when meshing a triangle. In this case, all of the elements are quadrilaterals except the ones forming the tip at the degenerate edge, which become triangular elements. In order to ensure all quadrilaterals, the object must be carefully subdivided, and automatic techniques exist for this purpose. The parametric mapping technique does not provide well-shaped elements if the region is not well-shaped.

1.2.4 Progressive Front

The progressive front technique uses successive layers of elements placed one after the other starting at the boundaries. As the mesh grows inward, the “meshing front” will eventually collide with the front moving from another boundary, and the fronts will be carefully connected. The fronts continue inward until the entire object has been meshed. This technique is illustrated in Figure 1.4.

![Progressive-Front Technique](image)

**Figure 1.4. Progressive-Front Technique**

Progressive-front successfully generates all quadrilateral meshes on arbitrary objects. It can accommodate any number of interior holes, is boundary sensitive, and
can be adapted to three-dimensional surfaces. The latter issue is the subject of this thesis. Progressive-front gains the advantage of better shaped elements, especially on the boundaries, but at the cost of additional irregular nodes, especially in the interior.

This thesis presents an algorithm for generating quadrilateral meshes on generalized three-dimensional surfaces using progressive-front technology.
Chapter 2

Overview of Research

This chapter introduces a progressive front algorithm called paving. Paving's origins are reviewed, and the starting point for the original research contained in this thesis is identified.

2.1 The Paving Algorithm

The paving algorithm is a progressive-front technique with many enhancements to control the mesh creation logic. This technique iteratively places rows of elements on each front (hence the name “paving”), until the entire object is meshed. In particular, the algorithm places a row of elements on each edge of a front until the a complete layer is formed, then moves to the next front and continues. As the meshing fronts collide, they are carefully connected to produce the most acceptable front for the continuation of the process.

While the mesh is being constructed, it is constantly adjusted in an effort to maintain ideal elements. This smoothing\(^1\) process, consists of iterative node adjustments intended to converge to a well-formed mesh. After the mesh has been completed, the entire mesh is once again smoothed, then checked for special correction cases.

---

1. See section A.1
The algorithm often inserts unnecessary elements or creates patterns during initial construction that can be replaced by simpler patterns with better shaped elements.

2.1.1 Input to the Paving Algorithm

The input to the algorithm consists of one or more ordered closed loops of nodes. These nodes are fixed boundaries which may not move during the mesh generation process. They are originally generated by a node generation scheme for curves.

The bounding loops of nodes must follow a “left-hand-rule” for the order of connection. This “left-hand-rule” states that if one is standing on a surface on the outward side (the direction of the normal), the left side of the loop will be meshed. This is equivalent on a plane to counter-clockwise exterior boundaries and clockwise interior boundaries.

2.1.2 Overview of the Paving Algorithm Steps

The essential steps of the paving scheme can be summarized, in algorithm form, as follows:

Do until all loops are closed
    Select the best loop to work on
    Do until a full layer has been added to the loop
        Check for cases in which loop can be closed with a preset pattern
        Otherwise, select the best row to construct
        Add row
        Adjust row
        Adjust loop
    End do
End do
Optimize the completed mesh

Each of these steps will be explained in detail later in this thesis.
2.2 The Original Paving Algorithm

Ted Blacker[2, 3] developed an automatic meshing program for two-dimensional problems named FASTQ. His efforts were motivated by the desire of stress analysts to spend less time on meshing and more on analysis. The successful implementation of this method has since become a widely used tool in numerous applications.

FASTQ contains an implementation of the paving algorithm. This implementation is restricted to two-dimensional problems containing any number of interior holes. This original development of FASTQ was based on earlier research[4, 5], and has led to a platform for continuing research[6]. This effort generated a large set of FORTRAN subroutines which, although simple in theory, were difficult to extend. A major revision was deemed necessary to add desired capabilities and make the code compatible with other modules.

2.3 New Capabilities

The objectives of this work are to:

- Support arbitrary three-dimensional surfaces
- Improve the logic and robustness of the algorithm
- Enhance the modularity of the algorithm and provide a finished code

The above list is a concise summary of the independent research contained in this thesis. The software generated from these objectives is designed to greatly enhance the utility and application of the paving algorithm.

Each of the objectives will be discussed in detail in the next chapter.
The outline of desired capabilities mentioned in the previous chapter will be further subdivided into specific discoveries, and each of these will be explained in detail. In each section, the major problems confronted in this research and their solutions are presented.

3.1 Support of Arbitrary 3D Surfaces

The original paving algorithm did not support three dimensional non-planar surfaces. It was confined to simple two-coordinate space. Many of the techniques used in the original algorithm to advance the meshing front and create new elements did not prove satisfactory, or even sufficient, to provide well-formed elements on three-dimensional surfaces.

The following is a list of the major problems which needed to be solved to extend the paving algorithm to three-dimensional surfaces.

- Projection of nodes
- Determining the angle at a node

1. The completed mesh could be transformed to a plane in three dimensions.
• Detecting collisions between the advancing mesh fronts
• Support of periodic surfaces

3.1.1 Projection of Nodes

The paving algorithm relies on extending the current mesh front out into the un-meshed region of the surface. This step requires the projection of each edge on the front in an extrusion-like operation, forming a new element. The projection of an edge is simplified if the end points, or nodes, are projected instead. The root of projection for a new node will be called the root node.

A root node, which lies on the surface by definition, may be projected in one of three ways: along a curve scribed on the surface, along a tangent to the surface, or along a chord of the surface. All three methods depend on determining a desired projection distance and direction. Since direction vectors on a surface are only defined at a point, and are tangent to the surface, the easiest methods are tangent projection and chord projection.

The new edge connecting the root node to the new node will be a chord of the surface. Since the direction of the chord, which in general does not lie in the tangent plane, is not know beforehand, projection along the final chord is not possible. However, the node can be projected along a tangent vector in the desired direction to the desired distance and then moved back to the closest point on the surface. This approximates the chord, and is suitable when the curvature at the root node is not very high. Such curvature restrictions are typical for a well sized mesh.

The tangent projection technique is illustrated in Figure 3.1.

---

1. A periodic surface is one that wraps around in space an connects with itself (i.e. cylinder, sphere, torus, etc.).
In this figure, \( N \) represents the normal at the point of projection, \( V \). \( P \) is a reference vector that represents \( \theta = 0 \). \( Q \) is the desired chordal projection point. The first step is to create a local coordinate system with the \( x \) axis coinciding with the projection of \( P \) onto the tangent plane. This is accomplished by equations 1 and 2.

\[
\begin{align*}
R_y &= N \times P \\
R_x &= R_y \times N
\end{align*}
\]

This local coordinate system is translated to the origin, and then rotated such that the local \( x \) and \( y \) axes correspond to the global axes. In this way, vectors can be rotated on the tangent plane, which is now the \( xy \) plane, with simple trigonometric functions. The vector \( Q' \), which is the tangent projection vector, is found by rotating a unit vector through the angle \( \theta \), and scaling by the desired projection distance. \( Q' \) is
then subjected to the inverse of the original transformations to move it back into the global coordinate system.

The final step is to move the tip of the vector $Q'$, which still lies in the tangent plane, back to the surface to form $Q$, the chord projection.

This technique requires two special functions for the surface being meshed, namely, returning the normal at a point on the surface and moving a point to the surface. The implementation of these functions is discussed in section 3.3.1.

### 3.1.2 Determining the Angle at a Node

Finding the angle at a node is a key operation in the paving algorithm logic. The type of projection performed at a node is dependent upon the angle formed with the node at the vertex.$^1$

Finding an angle is very similar to projecting a node. The two vectors representing the sides of the angle (corresponding to the $P$ and $Q$ vectors in Figure 3.1) are projected onto the tangent plane at the root node. After transforming the tangent plane to the $xy$ plane, the angle between the projected vectors is calculated using trigonometric functions.

An important observation is that the calculated angle is guaranteed to be greater than the actual angle between the vectors, and that the sum of the calculated angles between all vectors attached to a node will still be $360^\circ$.

---

1. See section A.1.
3.1.3 Detecting Collisions Between Mesh Fronts

Detecting collisions in two dimensions is simply finding a pair of edges that intersect in the plane. In three dimensions, however, the edges have become chords of the surface, which almost never intersect. These chords could be projected onto the surface as scribed curves, and the curves compared for intersection; but such a procedure is prohibitively expensive.

The solution chosen is one that solves the intersection problem and also provides superior control of approaching fronts. Instead of testing for an intersection between two fronts, a heuristic equation for calculating a proximity factor is used. The procedure is based on several criteria for the value of a connection between the fronts.

Generally, lines in three dimensions are skewed, so there is one unique point on each line which is closest to the other and can be used for determining the proximity of two lines. It is difficult, however, to determine the relative value of connecting two particular segments on those lines in the context of generating a mesh.

The proximity factor, as derived for this work, is calculated as given in equation 3.

\[ F = f_{\text{dist}} \frac{d}{s} + f_{\text{angle}} (1 - u \cdot v) + f_{\text{case case}} \]

Here, \( F \) is the proximity factor, which, to be sufficiently close, must be less than 100. Three terms are used to compute the relative value of connecting a particular pair of segments: closest distance between the segments, the angle between the segments, and the case of the closest distance. These cases are, midpoint-to-midpoint, end-to-midpoint, and end-to-end, the first being the most valuable connection. \( f_{\text{dist}}, f_{\text{angle}}, \) and \( f_{\text{case case}} \) are weighting factors, empirically derived, which assign relative values to the different terms. These factors are currently equal to 100, 25, and 10, respectively.
The sticky space, $S$, is found by the equation $s = \frac{\alpha}{2} (l_1 + l_2)$, where $l_1$ and $l_2$ are the lengths of the line segments being compared. The factor $\alpha$ is set to 55% to ensure that two quadrilaterals approaching each other will be connected rather than forcing a third one into the space. The closest distance, $d$, is shown in Figure 3.2. Equations 4 through 12 show the calculation for the closest distance between two lines in three dimensions.

\[ \tilde{P} = P + u s \]
\[ \tilde{Q} = Q + v t \]

$\tilde{P}$ represents any point on the line $P + u s$, and $\tilde{Q}$ represents any point on the line $Q + v t$, where $\tilde{P}$ and $\tilde{Q}$ are points in space. The unit vectors $u$ and $v$ lie along the

\[ \text{Figure 3.2. Closest Distance Between Two Lines} \]
lines. The expression \( \mathbf{Q} - \mathbf{P} \) may be used to represent the segment joining any point on \( \mathbf{P} \) to any point on \( \mathbf{Q} \). For any segment \( \mathbf{Q} - \mathbf{P} \) to be perpendicular to both \( \mathbf{P} \) and \( \mathbf{Q} \),

\[
\begin{align*}
(\mathbf{Q} - \mathbf{P}) \cdot \mathbf{u} &= 0 \\
(\mathbf{Q} - \mathbf{P}) \cdot \mathbf{v} &= 0
\end{align*}
\]

6

7

Substituting equations 4 and 5 into 6 and 7 yields,

\[
\begin{align*}
(Q \cdot v) + (v \cdot v t') - (P \cdot v) - (u \cdot v s') &= 0 \\
(Q \cdot u) + (u \cdot v t') - (P \cdot u) - (u \cdot u s') &= 0
\end{align*}
\]

8

9

The parameters \( s' \) and \( t' \) define the segment that is perpendicular to both \( \mathbf{P} \) and \( \mathbf{Q} \) simultaneously, which is the closest distance between \( \mathbf{P} \) and \( \mathbf{Q} \). Equations 8 and 9 can be written in matrix form as,

\[
\begin{bmatrix}
-(u \cdot v) & 1 \\
-1 & u \cdot v
\end{bmatrix}
\begin{bmatrix}
s' \\
t'
\end{bmatrix} =
\begin{bmatrix}
(P - Q) \cdot v \\
(P - Q) \cdot u
\end{bmatrix}
\]

10

Solving for \( s' \) and \( t' \) yields,

\[
s' = \frac{1}{u \cdot v (Q - P) \cdot u} \begin{vmatrix}
1 & (Q - P) \cdot v \\
-(u \cdot v) & 1 \\
-1 & u \cdot v \\
\end{vmatrix}
\]

11

\[
t' = \frac{1}{u \cdot v (Q - P) \cdot u} \begin{vmatrix}
1 & (Q - P) \cdot v \\
-(u \cdot v) & 1 \\
-1 & u \cdot v \\
\end{vmatrix}
\]

12

Equations 13 through 18 show the calculations involved in finding the additional constraints on two line segments, as shown in Figure 3.3.
The two segments are defined from \( P_1 \) to \( P_2 \) (\( \tilde{P} \)) and \( P_3 \) to \( P_4 \) (\( \tilde{Q} \)). The unit vectors \( u \) and \( v \) lie along the segments. If \( P_n \) is any end point, then the parameter value, \( s_n \), of the point on \( \tilde{P} \) closest to \( P_n \) (and likewise for \( \tilde{Q} \)) can be found by,

\[
\begin{align*}
    s_n &= (P_n - P_1) \cdot u \\
    t_n &= (P_n - P_3) \cdot v
\end{align*}
\]

Equations 11 and 12 can be simplified to,

\[
s' = \frac{((P_3 - P_1) \cdot u) - ((P_3 - P_1) \cdot v) (u \cdot v)}{1 - (u \cdot v)^2}
\]
\[
t' = \frac{(P_3 - P_1) \cdot u}{1 - (u \cdot v)^2} (u \cdot v) - ((P_3 - P_1) \cdot v)
\]

The closest distance between the two lines, \(d\), is given by,

\[
d = \| (P_1 + s'u) - (P_3 + t'v) \|
\]

For the case of parallel lines, \(1 - (u \cdot v)^2 \Rightarrow 0\), so,

\[
d = \| (P_3 - P_1) \times v \|
\]

The parameters \(s'\) and \(t'\) may fall outside the range of \(s_1\) to \(s_2\) and \(t_1\) to \(t_2\), in which case, one of the end points is the closest point that is also on the segment.

The result of the proximity equation is that, in the worst case, a segment can be \(100 - 25 - 10 = 65\) percent out into the sticky space of the other segment, or about \(\frac{1}{3}\) of the average segment length, and still qualify for connection to the other segment.

### 3.1.4 Support of Periodic Surfaces

The paving algorithm, and the progressive front technique in general, has no special requirements to be able to mesh periodic surfaces.\(^1\) The boundaries are treated just the same, and fronts that wrap around the surface as they progress until they collide with themselves can be easily connected.

There is, however, one unique problem on periodic surfaces. If, for example, a cylinder of finite length is being meshed, then each mesh front originating at the ends will progress towards the other and finally overlap. In most cases, there will be no in-

---

1. A periodic surface is one that wraps around in space and re-connects with itself to form a closed loop. Examples of periodic surfaces include the cylinder, sphere, and torus.
tersection of the two fronts because they will overlap completely in one step. This is due to the paving algorithm convention of completing an entire row of elements before looking for any suitable connections.

As mentioned in section 3.1.3, the proximity factor has an additional benefit. It will connect two mesh fronts that are almost touching, before or after they overlap. This process guarantees that the two cylinder fronts will be connected.

### 3.2 Improving Logic and Robustness

The implementation of the paving algorithm determines the logic used. The basic pattern for the algorithm was not significantly altered from Blacker's scheme, but the underlying decision process was clarified.

The fundamental factors controlling the creation of the mesh were identified and grouped categorically. When a particular process was identified, the possible cases which might occur were identified, and generalized solutions derived. The cases were then placed in decision-making modules to centrally locate all controls.

This type of categorization approaches what is known as an expert system. Such a system is a break down and organization of all possibilities for action and the factors controlling the selection of the correct action. In a given situation, the pertinent factors are evaluated and compared to the alternatives for action. The best action for the situation is then chosen.

The research conducted enhanced the organization and hierarchy of decision making, and identified more robust cases for evaluation. Since this thesis is not meant to be an explanation of the implementation of the paving algorithm, this structure will not be explained here. Appendix A does provide, however, a simplified decomposition of the actual code used.
3.3 Enhancing Modularity

One of the most important aspects of software design is the capability for reuse and extension. These factors were determined to be of utmost importance in the development of the improved paving algorithm. The modularity introduced by this research can be categorized as algorithm modularity and code modularity.

3.3.1 Algorithm Modularity

In order to make the algorithm as general as possible, no reference to the type of surface being meshed is required. It assumed that the surface is acceptable for meshing (i.e. it does not intersect itself, has finite boundaries, etc.). The algorithm contains ports\(^1\) to a generic surface\(^2\) that allow it to gather needed localized information about the surface.

In a typical application using the paving algorithm, the ports would be provided by a solid modeling package into which the algorithm was built. Following are the only required ports.

- Return the surface normal at a given point on or near the surface
- Return the closest point on the surface to a given point

3.3.2 Code Modularity

A new finite element mesh generation package, named CUBIT, that includes both three dimensional surface meshing using this research, and all-hexahedral meshing using an extension of the paving algorithm to hexahedrons\([8, 9]\), is currently

---

1. In this context, ports are references to sections of algorithm or code that must be supplied outside of the body of the algorithm.
2. A generic surface simply means a surface whose definition is unknown and unneeded by the algorithm.
under development at Sandia National Laboratories. The package uses object-oriented concepts to facilitate code modularity.\(^1\) In addition, many prepackaged libraries have been included to reduce development time.

CUBIT is written in C++\(^{[10]}\), which is an object oriented language based on ANSI-C. It is nearly as efficient as ANSI-C, but provides significant enhancement in the development of large codes due to the additional object-oriented capabilities.

C++ is a relatively new to the field of engineering applications. There are only a few major commercially available applications written in C++. As mentioned above, CUBIT employs a number of pre-packaged libraries. The solid modeling package, ACIS\(^{[11]}\), is also written in C++. The interface between the CUBIT application and the solid modeler was greatly enhanced by the use of C++.

Without an explanation of the concepts of object-oriented programming,\(^2\) the following are offered as fundamental types of objects in an engineering application.

- **Entities**: abstract units that typically represent physical or easily identifiable parts of a process (e.g. Node, Element, Boundary, Mesh) which can interact with each other.
- **Tools**: collections of functions that entities can use to perform application specific duties, such as drawing or archiving to a file. An application will typically need only one of each tool for all the entities to use.
- **Drivers**: collections of functions that interact with a particular hardware platform to provide graphics, user-interface, printer, or other device specific functionality.

An application will diversify these basic definitions to suit its particular needs, but the basic categories can remain intact.

---

1. See Appendix B
2. Ibid.

Research Findings 24
This chapter presents examples of meshes generated using the three dimensional paving technology. The examples attempt to include representations of the important features of the algorithm.

4.1 A Sphere with Multiple Holes

This example illustrates the ability of the paving algorithm to generate a mesh with multiple interior boundaries. The mesh is also being generated on a doubly curved spherical surface, which is a difficult task for methods which project a two-dimensional mesh onto a three-dimensional surface. Figure 4.1 shows the curvature of the sphere, and Figure 4.2 shows the completed mesh. Notice that the poorly shaped

Figure 4.1. Side View of Spherical Surface

curved spherical surface, which is a difficult task for methods which project a two-dimensional mesh onto a three-dimensional surface. Figure 4.1 shows the curvature of the sphere, and Figure 4.2 shows the completed mesh. Notice that the poorly shaped
elements and irregular nodes have generally been pushed away from the boundaries. This procedure is important because it promotes regular meshes on the boundaries.

This example shows the power of the paving algorithm in automatic meshing. No effort was made to subdivide the original geometry into easily meshed regions, as would be required for mapping, nor was any direction given in how to perform transi-
tions between large and small elements. The only input was the number of intervals desired on each original boundary.

4.2 Two Intersecting Cylinders

This example illustrates two important aspects of the paving algorithm. First, generation of a mesh on a circle, and second, generation of a mesh between two boundaries that approach closely.

As seen in Figure 4.3, the circular boundary is meshed from the boundary to the middle, in non-concentric patterns. A typical mapped mesh generation scheme would tend to follow radial projections. Again, this process has ensured that the best elements are nearest to the boundary.

The two cylinders are slightly different sizes. This creates a special meshing problem. Near the top and middle of the larger cylinder, the intersection boundaries form a narrow region. This type of problem is very difficult to solve in the parametric space of a surface because that space has become highly distorted. This is no problem for the paving algorithm because it simply connects the two opposing boundaries and forms two new boundaries, one on each side of the connection.

Another significant solution to a problem illustrated here is meshing of periodic surfaces. An ordinary mapped meshing scheme requires an artificial seam to be scribed from one end cap of the cylinder to the other, so that there are four bounding lines\(^1\) instead of just two. The paving algorithm does not have this requirement. It will begin at both ends of a cylinder and propagate the mesh boundaries until they can be connected.

---

1. Two of the lines will lie on top of each other along the seam.
4.3 The Piston

This example shows a complicated boundary on a periodic surface, and a transition in limited space. The significance of periodic surfaces has already been discussed in section 4.2. The difficulty with complicated boundaries is twofold. First, there is often a need to perform a transition within a small distance of the original boundary in order to accommodate a second nearby boundary with a different element size. Second, an ordinary mapped meshing scheme requires that the boundary be divided into many small regions that can be easily mapped.
The paving algorithm has no problem in connecting dissimilar element sizes, as long as the transition is reasonable.\textsuperscript{1} It also does not require the meshing region to be subdivided in any way, thus simplifying the job of the analyst.

Figure 4.4 shows the piston's complicated boundaries. Careful observation will reveal how the progressing boundaries met and were attached using transition wedges. The especially difficult transition can be seen in Figure 4.5.

![Figure 4.4. The Piston, Showing Boundaries](image)

\textsuperscript{1} Transitions with element size differentials of up to 6:1 have been successful.
4.4 The Block and Torus

This example shows a highly curved surface with difficult boundaries. It was formed by subtracting a torus from a flat block. Figure 4.6 shows the block from a vantage point which gives a view of the mesh inside the torus. The torus is also a good ex-
ample of a doubly curved surface. Here again, the paving algorithm simply proceeds from the boundary without regard to the shape or complexity of the surface.

Figure 4.6. The Block and Torus

The front face is an illustration of a transition mesh. The wedges inserted at various locations provide the means of increasing the number of edges on a meshing boundary to make the transition between large and small elements. As shown in Figure 4.7, the narrow regions on the top face are also satisfactorily handled.
Figure 4.7. The Block and Torus, Top View
Paving technology has proven to be a useful tool for generating all-quadrilateral meshes on arbitrary surfaces. The capability to incorporate holes and highly irregular boundaries is indispensable in automatic mesh generation. The power of the paving algorithm stems from its layer by layer mesh propagation, and the continuous smoothing\(^1\) during generation.

In this thesis, an extension of the paving technology to three-dimensional surfaces has been explained. By far, the most difficult problem solved was the calculation of the relative value of the connection between two segments. The technique used is an empirical combination of three simple calculations, closest distance between the segments, angle between the segments, and a case number for the closest distance.

Other important techniques introduced were the projection of nodes and determination of non-planar angles. The paving algorithm projects on the tangent plane at the point of projection. This means distances are all tangent and not arc or chord lengths. After many successful meshes, the conclusion is drawn that for elements properly sized for the curvature of the surface, the difference between projection along a tangent projector and the final chord is negligible. Any errors introduced in the ini-

\(^{1}\) See section A.1.
tial projection will be corrected by the subsequent smoothing operations. Conversely, if the original projection is a chord-length technique, the subsequent smoothing will move the node as well. A by-product of tangent projection is a method to determine the approximate interior angles of a non-planar quadrilateral.

5.1 Future Enhancements

The paving module is expandable by design. Thus it is easy for future research to broaden its applications. The following are some possible avenues of expansion.

5.1.1 Curvature Based Element Sizing

At present, the size of an interior element is determined by propagating the original size of elements at the boundary. This type of size definition becomes inadequate when the curvature of the surface changes away from the boundary. This condition occurs with any surface that does not have a constant curvature, such as a second-order or spline surface.

A possible correction for this condition would allow the user to set a minimum and maximum allowable element size. The paving module would optionally be allowed to determine the initial boundary interval size. Then, the element size would automatically be adjusted to accommodate changes in curvature with a simple conversion from curvature to desired element size.

Element size can be calculated by specifying a maximum deviation of the chord defined by an edge from the surface. This may be accomplished by evaluating the curvature in a given direction, or by taking the maximum curvature in any direction. All of these techniques are non-linear, and it remains to be seen if convergence can be guaranteed.
5.1.2 Scalar or Vector Sizing and Orientation

A problem similar to curvature based sizing is adaptive re-meshing based on the results of a previous analysis. Such an algorithm could allow localized adjustments to the existing mesh by comparing the scalar function to the existing element size, and locally inserting or deleting elements as needed. The changes could then be smoothed to form the completed mesh. Alternatively, the entire mesh could be regenerated with the element sizes, including the boundaries, being determined by the scalar field.

An additional feature could orient the elements according to an in-plane vector field. This scheme may have limited applicability because paving, by nature, forces square elements, thus the orientation would have little bearing on the final analysis. As the next section discusses, however, meshes for computational fluid mechanics have a great need for high aspect-ratio elements oriented with the direction of low gradient.

5.1.3 Fluid Oriented Mesh Generation

Long, slender boundary elements are often required in fluid analyses to define a boundary layer. However, such elements pose a formidable challenge to an algorithm that prefers square elements. A potential procedure to accommodate this situation would allow control of the placement of the high aspect ratio boundary elements, then make a smooth transition to square elements, using the paving technology to solve problems in corners and at flow separation/reattachment areas.

5.1.4 Automatic and User-Defined Refinements

The final mesh created by the paver algorithm can generally be refined to improve the overall element quality. Identification of additional cases for refinement is an area of further research. Often a user may require additional refinements after the
mesh is completed. After refinements are made, the paving module can check the mesh again to optimize its quality.

The original objective of paving is automatic meshing, thus allowing a user to adjust the mesh should be a final step. If the paving module is part of an automatic adaptive analysis, there will be no opportunity for user interaction, so the automatic refinement must be made sufficient.

5.1.5 Iterative Volume Meshing

The technology for generating general hexahedral meshes lags behind surface mesh generation[8, 9]. It is likely that any extension of the paving method to volume meshes would initially require all bounding surfaces to be meshed. It is probable that there will be cases generated by the surface meshing module that will render a volume mesh impossible, thus requiring surface re-meshing. Interaction between a volume mesh generator and a surface mesh generator will be important. Determination of the controlling factors of volume meshing will allow additional constraints to be placed on the surface mesh when initially generated.


In this appendix, the generalized paving algorithm is detailed as implemented for this research. It is hoped that this chapter will serve as a guideline for additional implementations in the future. It is not intended to be an exact representation of computer code, but uses C-like syntax to explain the implementation.

A diagram of the functional decomposition is shown in Figure A.1. Each function will be briefly described, giving its primary purpose and interaction with other functions.

### A.1 Definitions

The following terms are defined as they apply to generalized three-dimensional mesh generation.

- **Surface**: a geometric surface of any type in three dimensions on which a mesh is generated.
- **Node**: a point on the surface.
- **Edge**: a line connecting two nodes.
- **Face**: a quadrilateral on the surface.
**Figure A.1. Functional Decomposition**

*Loop*  
a closed series of nodes which define the current meshing front.

*Boundary*  
a loop on a surface defining a permanent boundary for the mesh.

*Smooth*  
an attempt to improve the quality of the mesh by adjusting each node until an iterative optimum is found.

*Irregular Node*  
a node with more than four connecting edges

Figure A.2 shows the different types of nodes in a typical mesh. These nodes are identified as side, end, corner, and reversal. The types of nodes are classified by the angle between its two boundary edges. Figure A.3 shows how nodes are classified according to the interior angle formed with the node at the vertex. The mesh is sensitive to these angles, and the angle definitions have been selected after numerical experimentation.
Figure A.2. Classification of Nodes

Figure A.3. Limiting Angles for Node Types
A.2 Boundary Preparation

The first step in creating a mesh is to define the boundary of nodes that lie on the edges defining the surface. The algorithm assumes that these nodes have been created previously. There can be both exterior and interior loops, as defined in the next section.

A.2.1 Bounding and Interior Loops

A bounding loop’s direction is defined as counterclockwise around the surface normal, while an interior loop’s direction is clockwise, as shown in Figure A.4. By analogy, if one were to walk on the surface along the loop with one’s head on the “outside”
of the surface, the mesh would always be generated on one's left (commonly called the "left-hand rule").

A.2.2 Open and Closed Surfaces

An open surface, such as a plane or hyperbolic paraboloid, extends infinitely in space and requires one, and only one, exterior boundary. A semi-closed surface, such as a truncated cone, requires either a single exterior boundary, or two boundaries which truncate the meshing region. A closed surface, such as a sphere or torus, needs no exterior boundary although one can be defined. For example, a sphere may be meshed by defining one hole and paving the surface area outside of the hole.

A.2.3 Requirements

Boundaries must satisfy certain requirements in order to ensure correct generation of the interior mesh. These requirements are:

1) The number of intervals on the boundary must be even.

2) The nodes must be in a series according to the definitions stated above for exterior and interior boundaries.

3) Severe size differentiation of nearby edges must be avoided to ensure well shaped faces. Gradual transitions between large and small edges are best, especially at corners. The technique has successfully meshed corners with aspect ratios as high as 6 to 1.

4) The boundaries must not cross each other or themselves.

5) Edge sizes must be reasonable for the topology to be meshed. Large edges on opposing sides only a small distance apart will cause difficult problems and poor meshes. A good guideline is for the maximum edge size to be approximately half the distance across the area to be meshed.
A.3 Functional Decomposition

This section details the steps in each function shown in Figure A.1. In this thesis, a function refers to any set of programming commands, such as a function in C, or a subroutine in FORTRAN. Often the functions may be grouped together, or they can be subdivided into convenient parts. The order of presentation of the functions generally follows the order in which they are called.

Each section that follows contains pseudo-code for the algorithm. This pseudo-code is only a suggested implementation of the necessary steps in the function, and is drastically simplified for presentation. In the pseudo code, bold-face type denotes other functions in the algorithm, and the number in parenthesis is the section where the referenced function may be found.

A.3.1 Create Mesh

This is the master function called by the modeling module to generate a mesh on a surface. Its main function is to:

- Make a local copy of all boundary nodes obtained from the solid modeler
- Process Mesh (A.3.2)
- Return created nodes, edges, and faces to the modeler

A.3.2 Process Mesh

This is the master internal function. It is called after all the data for the boundaries has been obtained and checked for validity. From this point on, the algorithm assumes the starting information is valid. The pseudo code is:
Do until all loops are closed
   Pick the best loop to work on
   Process Loop (A.3.3)
   Local Length Weighted Laplacian Smooth (LLWLS) (A.3.21)
   Clean Up Mesh
   LLWLS

The best loop is determined by the minimum level of any of its nodes, where the level refers to how many layers of mesh have been added onto the initial boundary. For example, if a loop, after connecting to an interior loop, had a few level 3 nodes while most were level 5, and another had all level 4 and above, the first loop would be chosen because of the level 3 nodes. This feature is primarily designed to ensure that all permanent boundaries are sufficiently covered with elements before intersection with other loops will occur. A difficult case to solve is a row of elements intersecting with a permanent boundary.

A.3.3 Process Loop

In this function, each loop is treated individually, one at a time. The loop is processed until a complete circuit of additional elements has been added.

Determine if another row needs to be added
Target layer = the deepest layer in the loop + 1
While loop is not closed and all nodes still are not as deep as the Target layer
   Close Loop (A.3.4)
   Add Row (A.3.6)
   Adjust Row (A.3.9)
   Adjust Loop (A.3.11)

A.3.4 Close Loop

This function checks the loop to see if it can be closed using a predetermined case. The only cases identified are for loops with 6, 4, or 2 nodes. Often the loop can be closed further by first seaming.
If loop is not closed and size is 6 or less
  If loop size is 2, Close 2 (remove one edge)
  If loop size is 4, Close 4 (insert one face)
  Try to Seam (A.3.13)
  If loop size is 4, Close 4
  If loop size is 6, Close 6 (A.3.5)

A.3.5 Close 6

Here, one of the predetermined closure cases is identified and used to close the loop.

Count number of row ends and number of edges between each end
Determine which closure primitive applies and perform either
  Connect Across,
  Close One Center, or
  Close Two Center

The possible closure cases are shown in Figure A.5.

A.3.6 Add Row

In this function, a row is added to the current loop. Each time Add Row is called, it looks for the best starting node for a row based on several criteria and adds a complete row to the loop.

Find the best starting node
  Start row if necessary by using preferentially
    Start On Reversal,
    Start On Corner, or
    Start On Side
  While not at the end on the row
    Add a Face (A.3.7)

Row ends always start and end rows. Picking the best row end is a function of the smallest element size on the anticipated row and the lowest level on the anticipated row. The rows with the lowest levels are picked first, and then the row with the smallest element among those. If no row ends exist, they are generated by projecting
from, in order of preference, a reversal, a corner, or a side. The order assures that the case requiring the most elements is performed.

### A.3.7 Add a Face

This function is the basic principle behind propagating the mesh to the interior. A node is projected away from the current boundary to form a new node. A new element is formed using these nodes, and the meshing front propagates down the row. The details behind node projection are found in [7].
Add a face based on the next type of node encountered, and either
End Row,
Add On Side,
Add On Corner, or
Add On Reversal
All of the above projection functions use Project Node (A.3.8).

The current node in the loop will be a row end, whether at the beginning of a
row or just after adding a face. The type of projection to use is based on the type of node
next on the loop. If it is a side, one face is added; if it is a corner, two are added, and
for a reversal, three, as in Figure A.6.

One new face

Two new faces

Three new faces

Side

Corner

Reversal

Figure A.6. Types of Projections

A.3.8 Project Node

The details of this function are given in section 3.1.1.

A.3.9 Adjust Row

Called after each row is added, Adjust Row smooths and modifies the loop to
ensure a well formed and well placed mesh. Localized modification at frequent inter-
vals helps preserve the integrity of the mesh, and improves the current meshing
boundary for the next row.
Smooth Node List (A.3.22) formed from all the nodes added on the row
Move to start of row
Try to Tuck and Wedge (A.3.10) the row
If row was modified, Smooth Node List again
Move to the end of the row

A.3.10 Tuck and Wedge

Here, the loop is checked for special expansion and contraction fixes. As a mesh progresses away from a curvilinear boundary, the size of the edges along the boundary increase for convex boundaries and decrease for concave boundaries. After a few rows, the elements will become poorly shaped. The solution is to insert or delete elements as needed to equalize the lengths of the edges.

Move along row, finding groups of edges longer or shorter than the opposite edges
If a group is too long or too short
  Find how many tucks or wedges need to be performed and either
  Take Tuck, or
  Insert Wedge

For a given set of connected edges, if their total length is longer than the total length of their opposite edges by more than two edge lengths, a wedge could successfully be inserted. This brings the average length of the edges down. Similarly, with shorter lengths, an element could be removed by tucking. Figure A.6 shows examples of how these are performed. The apparent disparity in sizes and shapes is taken care of when the mesh is smoothed in that area.

A.3.11 Adjust Loop

After a complete circuit of the loop has been made, Adjust Loop checks the loop for small angles at corners (or overlap), and seams the loop if necessary. The loop is checked for crossing itself and other loops.
Figure A.7. Expanding and Contracting Boundaries

**Seam Loop (A.3.12)**

Until no change
- Check all loops for **Cross Self** (A.3.14)
- Check all loops for **Cross Another** (A.3.16)
- Add any new loops to loop list

**A.3.12 Seam Loop**

This function collapses small angles at corners so that the next row does not need to force another element into the small opening. Seams will propagate along the small crack until it is opened far enough to allow a reasonable element to be inserted there later.

- Check if node can be seamed at all
- Check if node should be seamed by interior angle and number of edges
- See if a wedge would help
- **Seam** (A.3.13)

Seaming is inappropriate if a node to be seamed, or either of its neighbors, lies on a boundary. This is because the algorithm should not move boundary nodes. The
preference in this case is to force another element into the seam, propagating the seam inward one row, as shown in Figure A.8.

![Figure A.8. Seam on a Boundary](image)

When the size of elements to be joined across a seam is decisively different, it is advantageous to insert a wedge on the large size to equalize the edge sizes. This procedure is shown in Figure A.9.

![Figure A.9. Inserting a Wedge Before Seaming](image)

The basis for determining if a node should be seamed is twofold. First if the angle at a node is smaller than the “must-seam” tolerance, as defined in Figure A.3, the
seam will be closed because it would force a poor element to be formed. However, there is a range between $30^\circ$ and $45^\circ$ where the seam could be performed. The second criterion for performing a seam in this situation is that an irregular node, one with more than four edges, will not be formed.

A.3.13 Seam

Seam connects edges opposite a small angle to eliminate the angle and propagate the seam forward. Often the seam will close a number of small angles, as shown in Figure A.10. This will tend to round out the remaining interior boundary, eliminating as many small angles as possible and allowing a better row to be placed in the next iteration.

Figure A.10. Propagating a Seam
A.3.14 Cross Self

This function compares approaching boundaries of a loop. As two opposing boundaries approach each other, they need to be checked for sufficient proximity to warrant connection.

Compare every pair of edges in the loop for sufficient Proximity()
If a pair is found to be close enough to connect
   Find the closest valid pair
   Split Loop (A.3.15)

Because a loop must have an even number of nodes, only every other pair of edges can be connected to form two even-numbered loops. If, however, and invalid pair is still close enough to connect, the best valid pair is selected, and the loop split.

A.3.15 Split Loop

This function takes a loop which crosses itself and splits it into two new loops.

Connect the two edges given and create a new loop
Seam (A.3.13) starting at the new connection on both loops
Often the seaming operation will completely close the new loop.

A.3.16 Cross Another

In this function, the current loop is checked against all the other loops. The many loops on a mesh will eventually be sufficiently close to connect them together.

Compare every edge in loop to every edge in every other loop for Proximity (A.3.18)
If a pair is close enough
   Connect Loops (A.3.17)
A.3.17 Connect Loops

Here, two loops which are sufficiently close are connected to form a single composite loop.

Connect edges on two loops to form one loop
Delete the unneeded loop from the loop list
Seam the loop starting at the connection

A.3.18 Proximity

The details of this function are described in section 3.1.3.

A.3.19 Clean Up Mesh

After the initial mesh has been created, Clean Up Mesh will check it carefully for possible improvements.

Scan all faces and try to Fix Face (A.3.20)
Iterate until no faces were fixed, watching for oscillating patterns

A.3.20 Fix Face

Fix Face examines a face to see if it can be deleted according to a variety of cases. The pseudo-code is simply:

Check for any qualifying reason to adjust the face
Either Delete Face or
Make Three From Two Faces

The main goal of the paving algorithm is to create as good a mesh as possible initially, and then go back and fix up any poorly formed areas. Fix Face looks for any of several reasons to delete a face. The cases checked by Fix Face are illustrated in Figure A.11.
Figure A.11. Qualifying Cases for Face Deletion

A unique case occurs when a large transition is forced near permanent boundaries. An example of this case, in which two poorly shaped faces are converted to three better ones, is shown in Figure A.12.

**A.3.21 Local Length Weighted Laplacian Smooth**

A local length weighted Laplacian smooth attempts to reduce the amount of computation time needed for smoothing a node by only looking at neighboring nodes up to three levels from the seed node. Nodes farther away than three levels are seldom affected much by a move of the seed node. Length weighted Laplacian smoothing has
the desirable properties of preserving the squareness of faces and equalizing their sizes.

Add surrounding 3 levels of nodes to smoothing list
Until nodes do not move
Perform Length Weighted Laplacian Smooth (A.3.24) on each node

A local smooth will be performed on every node in the mesh after its final completion.

A.3.22 Smooth Node List

This function smooths the nodes in the smoothing list, which is built by the other functions during their operations and contains only nodes on the loop. This continuous smoothing greatly improves the final mesh.

Make sure only loop nodes are present in list
Until loop nodes do not move
Perform Isoparametric Smooth (A.3.23) on each node
Add surrounding 3 levels of interior nodes to smoothing list
Until interior nodes do not move
Perform Length Weighted Laplacian Smooth (A.3.24) on each node
The Isoparametric smooth tends to even the nodes on the loop, and the Laplacian smooth tends to form well shaped faces. Thus a two-part smooth is employed on the loop after each row is added to provide the best starting point for the next iteration.

A.3.23 Isoparametric Smooth

This function adjusts the position of a node on the loop to improve the continuity of the loop boundary.

Calculate new position of node
If node is a corner node
   Adjust to prevent squeezing it out of shape

A complete explanation of isoparametric smoothing can be found in [7].

A.3.24 Length Weighted Laplacian Smooth

This function adjusts the position of a node to equalize the distance to its neighbors. A complete explanation of length weighted Laplacian smoothing can be found in [7].
Recently the software development trend has been to move to a highly abstract and human-oriented method of programming. The result is a programming architecture that more closely approximates a natural thought process. Objects act upon and manage one another in a hierarchal fashion. This type of problem decomposition lends itself to scientific modeling and mesh generation.

B.1 Programming Environment

There are many object oriented languages currently in use, each of which has its relative merits and weaknesses. This development is being conducted using the C++ language [10]. This thesis assumes the reader has a small familiarity with C++ and object oriented programming.

B.2 Object Oriented Programs

CUBIT is object oriented. An engineer can create a model using the ACIS [11] solid modeler, then simply tell the computer to mesh the model. The model understands how to mesh itself, for there are only a few details it must ever understand. These might be as simple as instructions to mesh each of its constituent parts. Each
of these parts, in turn, knows what kind of part it is and can mesh itself and its own sub-parts. This tree hierarchy continues until all subordinate branches have completed their tasks and the model reports completion.

### B.3 Entities

An entity is an independent object in the model. It has a definition, and can allow its definition to be inquired or changed. It may also know its relationship to other entities.

A node, for example, is simply a set of coordinates defining a point in space. It is connected to a number of edges, which in turn are connected to faces. A node knows which edges it is connected to, and can return this information, or allow changes to that list. A node can figure out which faces it belongs to and return that information, or it can have a flag to tell whether it is on an edge or in the interior of the mesh.

Defining entities in this way makes them actors in the meshing process instead of just data elements, and allows a better conceptualization of the role of each data element in the whole process.

### B.4 Tools

An entity need not know how to do everything for itself. For example, a quadrilateral may not know how to draw itself because it is a very application dependent process. Putting the ability to draw into every entity would require many changes should the graphics platform ever be changed, a serious modularity issue.

A tool provides the functionality for these kinds of tasks. The tool is designed for a specific application or set of objects, and provides the entities with specific abilities. For example, a drawing tool knows how to draw a quadrilateral using graphics.
commands. It also knows how to draw the whole model, display in hidden line mode, draw labels at the nodes, or any other graphically oriented task. The drawing tool, in turn relies on a graphics driver for each hardware platform used by the application.

In short, entities are application independent, tools are application dependent, and drivers are platform dependent.

**B.5 The CUBIT Structure**

The basic object tree for CUBIT is shown in figure Figure B.1. The tree shows the inheritance of the different classes of objects and how they are related. Much can be understood about the purpose of each class without seeing any of the code itself.

![Object Tree for a Simple Paver Tool Application](image)

Figure B.1. Object Tree for a Simple Paver Tool Application.
The heart of the meshing process is the MeshTool class and its sub-classes wherein the algorithms for meshing an entity reside. A geometric entity creates a tool, uses it to create a mesh, then stores the mesh in its local database. This way each entity only needs to know which tool to use (and it may have several to choose from) to generate a mesh.

B.6 Engineering Applications

Object-oriented programming is a style and a conceptual model more than a fixed syntax. It is embodied in a programming language that does its best to follow the philosophy of this model. C++ is just one of several such languages available, and was selected in this case because of its close relationship with ANSI-C. In the process of development, I have encountered a few drawbacks to C++ in general as it pertains to engineering software.

C++ dynamically creates and destroys instances of classes. This means that an instance may be created temporarily for some use, and then destroyed soon after. In the paving algorithm, there are many vector calculations, so a class called Vector was used that could perform basic vector operations.

Initially, the paving algorithm was significantly slower than would be acceptable to analysts. To determine the source of the sluggishness, a profile of function time was conducted. The results showed that nearly half of the CPU time was devoted to creating and destroying Vectors. To solve this problem, a simple ANSI-C based structure was created in the object-oriented style. This replaced all references to Vectors in the algorithm. The result was a four-fold increase in speed.

It can be concluded that the overall architecture of an application can be greatly enhanced by object-oriented concepts. However, for efficiency, standard and specialized structures may need to be implemented in key locations. The high-speed code can
still be created in a manner to mimic C++ syntax and provide continuity in the pro-
gram.

An additional problem deals with the dynamic allocation of memory. If a hybrid
of different languages is being combined into one application, the various methods
used by the different languages to allocated and free memory may not be compatible.
This is currently the case between the C function malloc and the C++ function new.

Overall, the use of object-oriented languages for engineering will increase the
development and distribution of new technology, and make possible new dimensions
of analysis and design.
A Generalized Three Dimensional
All Quadrilateral Surface Mesh
Generation Algorithm

Roger Cass
Department of Civil Engineering
M.S. Degree, April 1992

ABSTRACT

Among the available all-quadrilateral mesh generation algorithms progressive-front offers the best quality and robustness. The paving algorithm is a progressive-front technique that generates a two dimensional mesh of all quadrilaterals on planar surfaces. The technique is particularly suited to meshing irregular boundaries with interior holes. This thesis presents an extension of the paving algorithm to generalized three dimensional surfaces.

The major problems in three-dimensional surface mesh generation are projection of nodes, calculation of interior angles, and connection of mesh fronts. The solutions to these problems are presented.

This work is part of the ongoing CUBIT project at Sandia Labs. The application is written in object oriented style in C++. A description of object oriented programming is given, with the relative merits and drawbacks for engineering applications.

COMMITTEE APPROVAL:

Steven E. Benzley, Committee Chairman
Richard J. Balling, Committee Member
Lavere B. Merritt, Department Chairman