Surface Meshing Using Vertex Insertion

H. L. de Cougny (decougny@scorec.rpi.edu) and M. S. Shephard (shephard@scorec.rpi.edu)
Scientific Computation Research Center
Rensselaer Polytechnic Institute
Troy, NY 12180–3590

Abstract. This paper presents the requirements for a surface mesh to be valid. Aside from validity, it also discusses conditions for geometric similarity, which quantitatively describes how the surface mesh "matches" the model geometry. The surface mesh generation method presented here uses the parameter space of a model face to generate a coarse triangulation. In order to avoid the mapping problem, the coarse triangulation is refined in the real space to obtain proper mesh entity sizing. The Delaunay criterion is used to generate the coarse triangulation in the parameter space and a quasi-Delaunay criterion is used in the real space to complete the triangulation. The surface mesh is post-processed for self-intersection to guarantee its validity and for optimization. Problems coming from the possible periodicity and degeneracy of parameter spaces are addressed.

1. Introduction

An a priori simple solution to surface mesh generation is to consider the two-dimensional parameter space associated with a model face [1] [2] [3]. This reduction in dimensionality actually presents some difficulties since one has to account for the mapping from parameter to real space. One can explicitly account for that mapping when creating triangles in the parameter space [3]. One can also use a generation space that maps the parameter space into a secondary space suitable for meshing [1]. Consideration of parameter spaces can be completely (or partially) bypassed using a constrained "three-dimensional" advancing front method [4] or the Finite Octree method which decomposes the surface into octant-size surface loops [5].

At least three basic issues need to be addressed when meshing surfaces: (i) mesh entity size control, (ii) surface mesh validity, and (iii) surface mesh geometric similarity to the model [6]. Assuming the model is curved, it is often desirable to have mesh entities discretize the model boundary within some geometric discretization error. It is therefore important to have a surface meshing procedure that can precisely control mesh entity sizes. The main requirement for surface mesh validity is that mesh entities should not intersect. Because mesh entities on different model faces can intersect, the surface mesh validity check is a post-processing step. Note that it requires a localization structure (for example, an octree). Surface mesh validity is most critical since a three-dimensional mesh generator cannot accept an invalid surface mesh for input. Although not a validity requirement, geometric similarity is also important since it deals with how well the surface mesh "matches" the geometry of the surface.

The parameter spaces provided by the modeler should be used by the surface mesh generator since they are two-dimensional representations of three-dimensional objects, therefore providing a reduction in dimensionality. The surface meshing procedure presented here uses the parameter space to: (i) control mesh entity sizes, (ii) generate a coarse mesh of the surface, (iii) decide where to insert interior vertices, and (iv) control (to some extent) geometric similarity. Although locations for interior vertices are found in the parameter space, they are inserted using a quasi-Delaunay criterion in real space. Surface meshing in the parameter space using an insertion method requires the domain to be bounded by one outer loop and any number of inner loops. When a parameter space is periodic and possibly degenerate, the domain's boundary is not well defined and can not be used (as is) for surface meshing. This is due to the fact that the associated mapping is not one-to-one and is therefore not valid. Measures have to be taken to transform such a mapping into a valid one.

The second section describes how element sizes can be controlled in the parameter space. The third section discusses the requirements for surface mesh validity. The fourth section considers the concept of geometric similarity and the difficulties associated with checking for geometric similarity in the parameter space (provided by the modeler). The fifth section briefly describes model edge meshing. The sixth section describes the complete model face meshing procedure. The seventh section presents the check for self-intersection. The eighth section briefly describes a triangulation optimization procedure. Finally, some results are presented and concluding remarks are given.

2. Control of Element Sizes in Parameter Space

The tools to control element size and gradation are described here. It is assumed that element sizes can be controlled by imposing a maximum geometric discretization error ε for specified model entities (edges and/or faces). Here, the geometric discretization error is defined as the ratio of the maximum distance d from a mesh entity to the model entity (it is classified on) to the length of the arc Δs on the model entity. Fig. 1 illustrates graphically what the discretization error is in relation with the radius of curvature R. Note that this geometric discretization error is purely geometric and has nothing to do with the discretization error generally associated with finite element solutions. Control of the geometric discretization error helps in making sure that the surface mesh will "match" the geometry of the model and be geometrically similar. It should be stressed, however, that even if the geometric discretization error is controlled, geometric similarity may be violated.

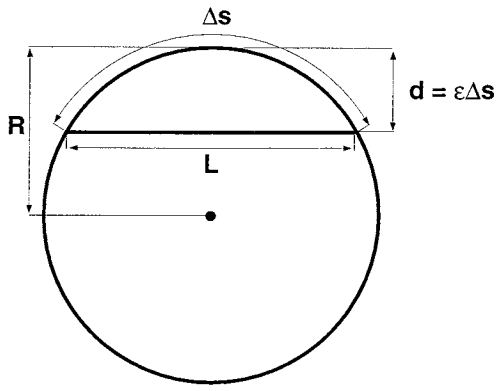

Figure 1 Geometric discretization error

Control of the geometric discretization error for a model entity (edge or face) can be accomplished by considering the curvature of the underlying geometric entity (curve or surface, respectively). To control the discretization error, one has to be able to: (i) given a segment in the parameter space (coming from the mapping of a mesh edge), get the corresponding arc length in the 3–d space and (ii) given the curvature at a point and some discretization error ε, get the corresponding maximum arc length in the 3–d space. The following shows how to obtain both arc lengths for model edges and model faces. Details about differential geometry can be found in numerous textbooks such as references [7] [8].

Consider a point on a model edge, it is defined by its curvilinear parameter s and some other parameter u in the associated parameter space. An infinitesimal parameter length du in the parameter space translates to an infinitesimal arc length ds such that $ds^2 = Edu^2$ where $E = S_u.S_u$ (S_u is the tangent vector). E can be thought of as the "stretch" factor. From the infinitesimal arc length, the arc length corresponding to a segment in the parameter space can be obtained by integrating along that segment. If the curvature at that point is κ (radius is $R = 1/\kappa$) and the desired discretization error is ε, the maximum allowable arc length is approximated by considering the osculating circle at the point. Note that the osculating circle is a second-order approximation. From Fig. 1, the maximum arc length is $\Delta s = 2R \arcsin(L\kappa/2)$. In practice, ε should be small enough to capture the complexity of the model. Then, the maximum allowable arc length can be approximated by the chord length $L = 8R\varepsilon/(1 + 4\varepsilon^2)$.

Consider a point on a model face defined by the two parameters (u, v) in the associated parameter space. An infinitesimal segment in the parameter space defined by (du, dv) translates to an infinitesimal arc length ds such that:

$$ds^2 = Edu^2 + 2Fdudv + Gdv^2$$

where $E = S_u.S_u$, $F = S_u.S_v$, and $G = S_v.S_v$ (S_u is the tangent vector in the u direction and S_v is the tangent vector in the v direction). This is known as the first fundamental form. E can be thought of as the "stretch" factor in the u direction, F as the "shear" factor, and G as the "stretch" factor in the v direction. From the infinitesimal arc length, the arc length corresponding to a segment in the parameter space can be obtained by integrating along

that segment. If the maximum curvature at that point is κ, the maximum allowable arc length is obtained from the formula used for model edges.

If the arc length of the segment is too large when compared to the maximum allowable length (coming from curvature), the mesh edge is not discretizing the geometry properly and should be refined. It is of importance to consider the arc length and not the actual length of the mesh edge for comparison purposes in this context. In Fig. 2, the mesh edge cannot be refined unless the arc length is considered since the actual length of the mesh edge is smaller than the maximum allowable length (imposed by the geometric discretization error).

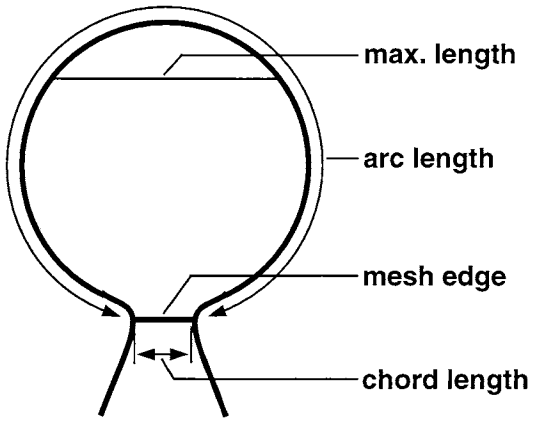

Figure 2 Arc length versus chord length

For each model edge, a linear grid, constructed in the parameter space of the model edge, is associated with the model edge. For each cell in the grid, κ and E are evaluated at the middle. The maximum allowable length due to curvature and E are stored in the cell. For each model face, a square grid, constructed in the parameter space of the model face, is associated with the model face. For each cell in the grid, κ, E, F, and G are evaluated at the middle. The maximum allowable length due to curvature, E, F, and G are stored in the cell.

These grids are constructed in order to reduce the total number of queries (to the geometric modeler) for mapping derivatives and/or curvature evaluation as mesh edges are checked for size during the meshing process. The grids should be fine enough to capture the geometric complexity of the model entities they are built upon. The geometric complexity of a model entity is not known beforehand, which makes missing local geometric complexities possible. A local geometric complexity can be a strong change in curvature and/or a strong change in mapping derivatives. For example, one can have a surface with little change in curvature but with strong changes in first order mapping derivatives. The strong change in first order derivatives is a local geometric complexity (due to the mapping) which can be troublesome when evaluating arc lengths. Techniques to estimate how fine the grids should be are currently under investigation.

Given a model entity (edge or face), the associated grid contains information which enables the creation of mesh entities of the correct size on that model entity. These mesh entities, although of correct sizes for that particular model entity, may be too large when compared to mesh entities created on neighboring model entities. In other words, a model entity (edge or face) needs to know the sizes of mesh entities around it in order to guarantee a smooth gradation between mesh entities. An octree can provide a tool to control gradation. For each grid cell, the maximum allowable length is transformed into a tree level which is inserted into the tree. By forcing a maximum 2:1 level of difference between (octant) edge neighbors, the tree becomes smoothly graded. Once the tree is adjusted, grid cells for all model entities (edge or face) are revisited. For each grid cell, the level coming from the tree is transformed into a length which replaces the value already stored in the cell if it is smaller. This octree will also be used later on to check for self-intersection of the mesh.

3. Surface Mesh Validity

A surface mesh is a valid representation of the model if it satisfies the three following requirements [6]:

1. classification, that is, each mesh entity is uniquely associated with a model entity,
2. topological compatibility, that is, given a model face, any mesh edge on a bounding model edge is connected to exactly one mesh face on the model face and any mesh edge interior to the model face is connected to exactly two mesh faces on the model face, and
3. no self-intersection, that is, any two mesh entities which do not share bounding mesh entities are disjoint (do not intersect).

Assuming the geometry associated with mesh entities is linear for mesh edges and planar for mesh faces, it is sufficient to perform the self-intersection test on (edge, face) pairs. The presented concept of validity comes from the need to be able to generate a three-dimensional mesh from a surface mesh. If the surface mesh satisfies the three above requirements, the outside and inside(s) of the model have been separated, which is all that is needed for a three-dimensional region mesh generator. The potential problem is that one can have a perfectly valid surface mesh which is not geometrically similar, that is, it does not actually "match" the model geometry. Geometric similarity is discussed next.

4. Geometric Similarity on Model Face

Qualitatively, geometric similarity [9] requires that two different mesh faces discretize disjoint portions of the model face, in other words, there cannot be overlapping in the discretization. The concept of geometric similarity can be defined exactly if a valid (in a meshing context) mapping to a two-dimensional parameter space is available. Given a surface, a mapping to a parameter space is valid if it is (i) one-to-one, that is, any point on the surface has a unique parametric representation and (ii) continuous. Qualitatively, the mapping for the whole surface is continuous if it is continuous for any curve on the surface, which means that the image of any continuous curve on the surface is a continuous curve in the parameter space. Such a mapping is in general provided by the geometric modeler from where the surface originates. Given a model face and a valid mapping to a parameter space, geometric similarity is guaranteed if, for any mesh edge interior to the model face, the two connected mesh faces do not intersect when mapped to the parameter space [9]. The question of how to map a mesh face onto the parameter space is not straightforward. Consider first the problem of mapping a mesh edge onto the parameter space. Intuitively, one may assume a mesh edge on a model face is supposed to discretize the shortest path on the surface between its two end mesh vertices (points on the surface). The shortest path between two surface points is a geodesic curve [8]. Due to the continuity condition placed on the mapping, the image of a (continuous) geodesic curve is a (continuous) curve in the parameter space. Since there is no reason to believe that this curve is straight in the parameter space, such a mesh edge on a model face should appear curved in the parameter space. Then, the image (in the parameter space) of a mesh face on a model face should be a triangle (assuming the mesh face has three sides) with curved sides. Fig. 3 shows the mapping of a mesh face, shown with straight sides in real space, from model face to parameter space. The thin curves represent the iso-parametric lines.

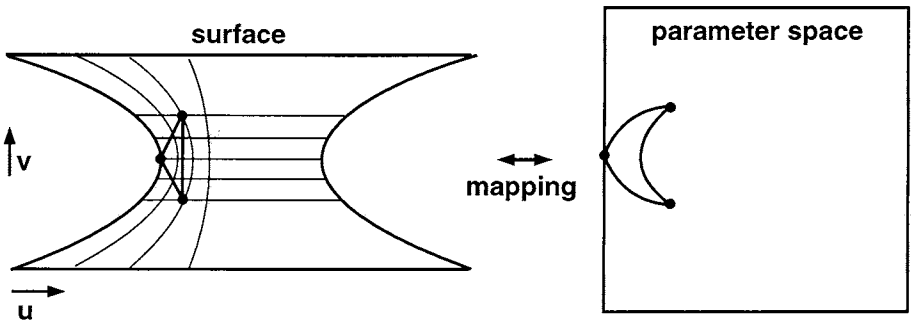

Figure 3 Mapping of a mesh face onto parameter space (curved sides in parameter space)

Geodesics are governed by the following coupled set of two second-order non-linear ordinary differential equations:

$$\frac{d^2u}{ds^2} + \Gamma_{11}^1 \left(\frac{du}{ds}\right)^2 + 2\Gamma_{12}^1 \left(\frac{du}{ds}\right)\left(\frac{dv}{ds}\right) + \Gamma_{22}^1 \left(\frac{dv}{ds}\right)^2 = 0$$
$$\frac{d^2v}{ds^2} + \Gamma_{11}^2 \left(\frac{du}{ds}\right)^2 + 2\Gamma_{12}^2 \left(\frac{du}{ds}\right)\left(\frac{dv}{ds}\right) + \Gamma_{22}^2 \left(\frac{dv}{ds}\right)^2 = 0$$

where s is the arc length (of the geodesic curve) and the Γ's are the Christoffel symbols of the second kind (they depend on the first and second mapping derivatives) [8] [10]. Given a starting point (in the parameter space) defined by (u, v) and a starting direction (in the parameter space) defined by $(\frac{du}{ds}, \frac{dv}{ds})$, the above equations define a unique geodesic curve. This initial value problem can be solved numerically using, for example, a fourth-order Runge-Kutta scheme [10] [11]. Finding the geodesic between two points is a boundary value problem which can be solved by, for example, using the initial value problem solver with a fixed starting point but variable "shooting" direction [11]. The problem can then be reformulated as finding the direction for which the initial value problem yields a geodesic curve passing through the second point, which is a one-variable optimization problem. This can be a costly process since one has to iterate on an already relatively expensive initial value problem. In most cases, the shortest path between two points is unique. In some specific cases, it is not. Fig. 4 shows the two shortest paths for bypassing a "hill" on a surface.

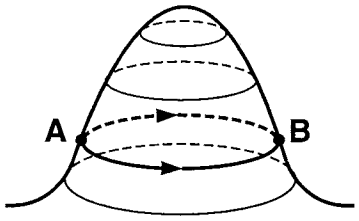

Figure 4 Example for non-unique shortest paths between two points on a surface

It is natural to think that the mappings provided by the geometric modeler should be used to mesh surfaces, and in particular, to check for geometric similarity, since they are available for "free". However, if geodesics (associated with mesh edges) curve strongly in the parameter space and mesh edges are not curved in the parameter space, they cannot be used to check for geometric similarity. If the mesh edges are small enough for the associated geodesics to be considered straight in the parameter space, the parameter space can be used to check for geometric similarity without difficulty. In this paper, no assumptions are made concerning the mapping. It is therefore assumed that geodesics may curve strongly in the parameter space. The question to be raised is whether or not trying to curve mesh edges in the parameter space is worth the computational cost of finding geodesics. Geodesics are possibly worth the computational cost when mesh entities need to be curved to exactly match the geometry [12] [13] and/or extremely coarse meshes need to be generated. In general, it is believed that the computational cost of finding geodesics is too high.

If mesh edges are represented by straight segments in the parameter space, an implicit assumption must be made concerning how mesh edges discretize the surface. The assumption is that any mesh edge on model face discretizes the arc on the surface obtained from a straight segment in the parameter space. Fig. 5 describes graphically this implicit assumption. With this assumption, the parameter space provided by the modeler can not be used to check for geometric similarity (unless mesh edges are small enough). The presented surface mesh generator redefines geometric similarity to make it independent of the parameter space. A surface mesh is considered to be geometrically similar if any mesh face has a normal that "agrees" (within 90 degrees) with the surface normal. How this geometric similarity check is performed and how dissimilarities are resolved will be discussed in the model face meshing section.

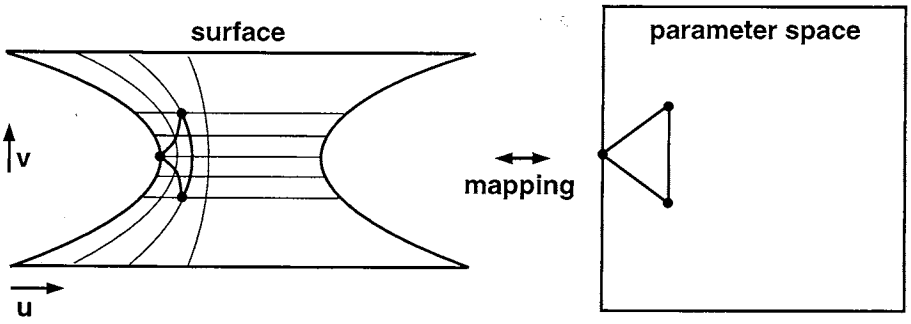

Figure 5 Mapping of a mesh face onto parameter space (straight sides in parameter space)

The next two sections describe the presented surface mesh generator. It is a hierarchical procedure, that is, model vertices are meshed first, edges second, and faces third. The meshing of model vertices is straightforward and is therefore not discussed in this paper.

5. Model Edge Meshing

Consider a model edge to be meshed. Initially, the model edge is discretized with a single mesh edge. If the model edge has two end model vertices, the mesh edge is created so that it connects the two mesh vertices classified on the two model vertices. If the model edge is periodic with one bounding model vertex, the two end vertices of the mesh edge are identical. If the model edge is periodic with no bounding model vertex, a mesh vertex classified on the model edge is created, and as in the previous case, the two end vertices of the mesh edge are identical.

For each edge classified on the model edge, a decision must be made concerning whether or not it appropriately discretizes (locally) the model edge. Recall that one has to compare the arc length with the maximum allowable length (from curvature). The arc length is computed by numerically integrating $ds = \sqrt{E}du$ between the two end vertices. This numerical integration is simply a finite summation on $\Delta s = \sqrt{E}\Delta u$. For each Δs, the "stretch" factor E is assumed to be constant on Δu and is obtained from the grid attached to the model edge. The maximum allowable length is obtained by sampling the mesh edge and retaining the smallest value. If the arc length is at least twice as large as the maximum allowable length, the mesh edge is split in two by considering a point at the middle of the segment in the parameter space. The very same decision making will be used for model face meshing.

6. Model Face Meshing

Model face meshing relies on the Delaunay insertion method in the geometric modeler's parameter space and in real space. Meshing of a model face can be decomposed into seven phases:

1. determination of the domain's boundary,
2. initial triangulation,
3. insertion of boundary vertices,
4. boundary recovery,
5. deletion of outside mesh entities,
6. insertion of interior vertices, and
7. geometric similarity check.

The basic theoretical and implementation details for boundary recovery and Delaunay insertion can be found in references [14] [15].

6.1 Domain's Boundary

For each model face, one has to obtain a closed discretized boundary in order to be able to mesh it. This is due to the fact that an insertion type meshing procedure is used here. This closed boundary is made of at least one (closed) loop. The discretized boundary can be obtained by gathering the mesh vertices and edges classified on the model face's boundary. For better data access during the actual meshing process, these mesh entities are copied into a temporary mesh. In what follows, mesh entities are considered in that temporary mesh. The actual coordinates of mesh vertices (in the temporary mesh) are replaced by parameter values on the model face, which makes the problem completely two-dimensional until interior mesh vertices are inserted. Once done with meshing the model face, the temporary mesh is copied back into the global mesh and then deleted. In general, determining the domain's boundary is straightforward. However, the domain's boundary may not be well-defined when the parameter space is periodic and possibly degenerate. The next two sections discuss what needs to be addressed for periodic and possibly degenerate parameter spaces.

Periodic parameter spaces Model faces associated with parameter spaces periodic in u and/or v are problematic for a surface mesher that works in the parameter space, like the one presented here. A periodic parameter space violates the one-to-one condition for parameter space validity. If a surface is associated with a parameter space periodic in the u direction, with period α, and parameter range $[0, \alpha]$, any point on the surface at $u = 0$ has actually two u parameter values: $u = 0$ and $u = \alpha$.

Consider a mesh edge classified on some model edge bounding that model face. If the sub-curve discretized by the mesh edge crosses (only consider go-through intersections) the parametric curve at $u = 0$ an odd number of times, α must be added to the lowest of the two parameters (of the two end vertices) or subtracted to the highest of the two in order to have a correct parametric representation of the mesh edge. Such a mesh edge is referred to as a "bridge". In Fig. 6, the "bridge" edges are represented after having added the period to the parameter of one of the two end vertices. What is problematic about "bridge" edges is that they cannot be identified with complete certainty due to the one-to-one mapping violation. A simple solution to the problem is to split the model face along the constant parameter lines $u = 0$ and $u = \alpha/2$. This forces the mapping to become one-to-one. The other motivation for model face splitting will come from the parallel version of the presented surface mesher. Assuming that a model face is meshed by a single processor, scalability of the method may require splitting model faces (even if not periodic) in order to properly partition the problem.

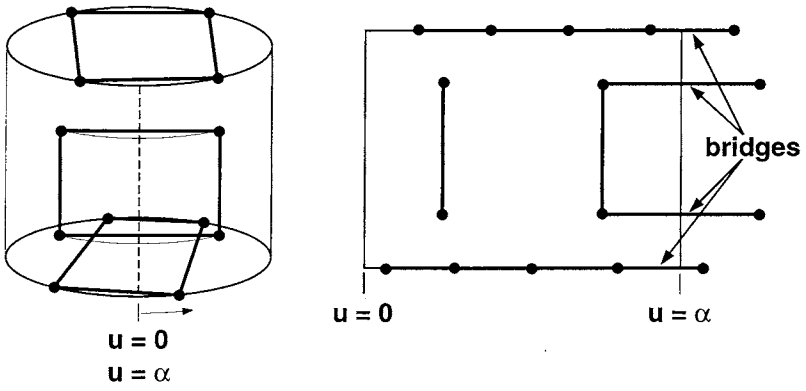

Figure 6 Bridge edges on a periodic model face

In general, splitting a model face results in the creation of:

1. sub-vertices resulting from intersection of the model face with the constant parameter line,
2. sub-edges (type edge) resulting from the split of model edges,
3. sub-edges (type face) resulting from intersection of the model face with the constant parameter line, and
4. sub-faces resulting from the split of the model face.

It should be noted that the split of a model edge on the boundary of the model face that is being split requires the update of the connected model faces (in terms of bounding model edges). In order not to modify the original model, sub-entities are created and stored in a model database outside of the geometric modeler. Sub-entities maintain links to the model entities they emanate from in order for the surface mesher to still be able to get information from the geometric modeler. The following paragraph explains how a model face is split at $u = const$.

Consider a model face bounded by model edges (may include sub-edges of both types) and model vertices (may include sub-vertices). The first step is to intersect all model edges bounding the model face with the constant parameter line. For each intersection location (associated with a model edge), a sub-vertex is created pointing to the originating model edge. The model edge is split into two sub-edges (type edge) unless it has no end vertices (periodic model edge). Next, intersection vertices are sorted with respect to increasing parameter v value. For each two consecutive intersection vertices, a sub-edge (type face) is created if it lies on the model face. Any new sub-edge (type face) points to the originating model face and stores the fact that it results from a split at $u = const$. Fig. 7 shows an example of sub-vertices and sub-edges of type edge and face. It should be noted that E which links together arc lengths in the real space and lengths in the parameter space for the sub-edge is equal to G for the originating model face. The split at $u = const$ has separated the parametric space of the model face into two well-defined sub-spaces: (i) $u \leq const$ and (ii) $u \geq const$. Considering the $u \leq const$ sub-space (the same is true for the other sub-space), all model edges and vertices that lie in that sub-space are gathered. The basic idea is to construct one loop of model edges and create one sub-face corresponding to that loop. Note that the sub-face may be multiply connected. The newly created sub-face points to the originating model face and store its parametric range. Loop building is such that: (i) a model edge can be used only once (for simplification of explanation, assume a two-manifold setting) and (ii) some edges (original edges or sub-edges of type edge) have imposed directions (coming from how the original model edges they derive from were used by the original model face).

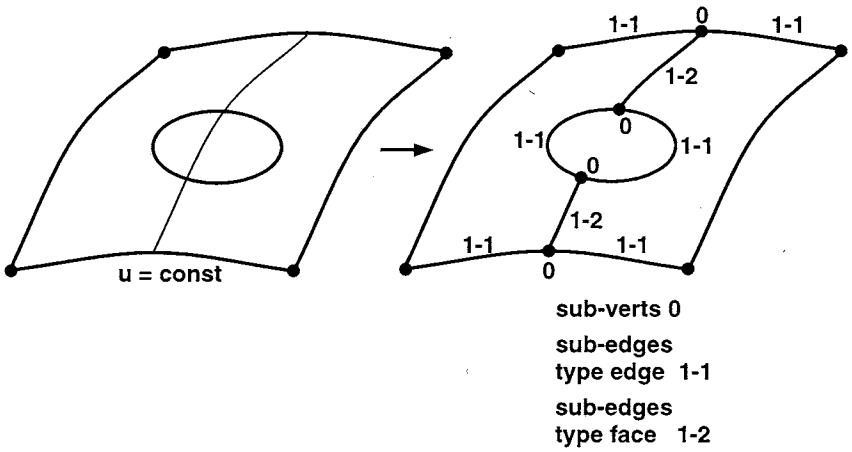

Figure 7 Sub-vertices and sub-edges resulting from a model face split

Once a periodic model face is split, say, into two sub-faces, there is no ambiguity concerning the parameter values of mesh edges on the boundary of the sub-faces. Given a mesh vertex lying on the $u = 0$ line, its parameter value is 0 if one considers the sub-face with range $[0, \alpha/2]$ and α if one considers the sub-face with range $[\alpha/2, \alpha]$. The mapping is one-to-one and therefore valid. Then, the determination of the domain's boundary becomes straightforward. Other difficulties in determining the domain's boundary may arise when the parameter space is periodic and degenerate. This is discussed next.

Degenerate periodic parameter spaces Assuming the parameter space is periodic in the u direction, there is a degeneracy at $v = const$ if the corresponding parameter line represents a single location in real space (e.g., tip of a cone or poles of a sphere for classical mappings). If the degeneracy is not actually on the model face, it can be ignored. Because periodic model faces are split, a mesh vertex should exist at the location of the degeneracy. The problem with degenerate spaces is that, without modifications, the edges connected to the degenerate vertex are

"pulled" to it. This creates an artificial distortion in the parameter space. The first step in resolving this distortion is to record the minimum and maximum u parameter values for the opposite vertices (to the degenerate vertex) on the connected mesh edges. What then follows can be decomposed into four steps:

1. the degenerate mesh vertex is moved from the $(u = 0, v = const)$ to the $(u = min(u), v = const)$ location,
2. a mesh vertex is created at the $(u = max(u), v = const)$ location (duplicate vertex of the degenerate vertex),
3. a mesh edge is created connecting the two mesh vertices (degenerate edge), and
4. the degenerate vertex is replaced by the newly created vertex (becomes a duplicate edge) for the mesh edge corresponding to the maximum u parameter.

Fig. 8 shows the initial (at center) and modified (at right) domain's boundary in the parameter space.

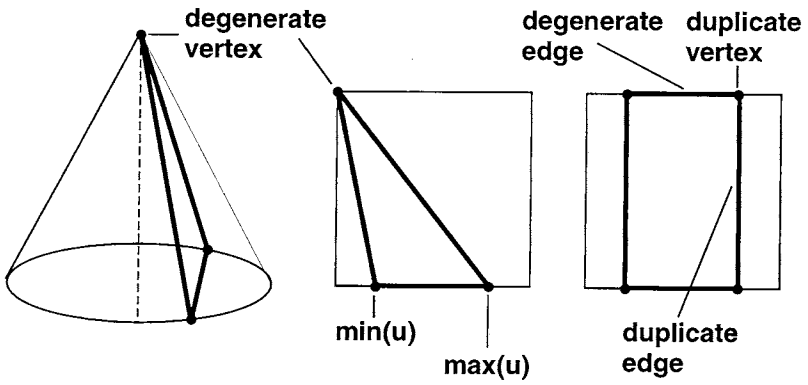

Figure 8 Degeneracy

6.2 Initial Triangulation

By considering the locations in the parameter space of the mesh vertices, a bounding box is constructed which is slightly enlarged to make sure no vertex lies on the boundary of the box. The box is triangulated with two triangles, which yields the initial triangulation.

6.3 Insertion of Boundary Vertices

Mesh vertices are inserted in turn into the current triangulation using Watson's algorithm [16] in the parameter space. In theory, the Delaunay criterion guarantees that the insertion polygon's bounding edges are fully visible from the point to be inserted (point convexity). In reality, it not necessarily true due to round off error. It is therefore necessary to explicitly check if any edge is fully visible from the point (dot product computation). If it is not visible, the face that connects to the problem edge is removed from the set (the new edges will be checked for visibility as well). At worst, Delaunay insertion ends up being an edge split (two faces are split into four).

6.4 Boundary Recovery

Since the Delaunay triangulation was not constrained, it is possible that mesh edges that were part of the domain's boundary are now missing. Given a missing mesh edge defined by its two end vertices, the mesh edges that are in the path of the missing edge are gathered into a set. Edges in the set are attempted to be removed using an edge swapping procedure. An edge swap is performed if (i) the new edge does not intersect the missing edge and (ii) the swap is considered valid, that is, the new edge does not already exist and it will not create a new mesh face with a negative area. In practice, this procedure solves most cases. However, if there are still edges in the set, a more complicated scheme that allows swapping, even if the new edge intersects the missing edge, is used.

6.5 Deletion of Outside

Any mesh face that is bounded by at least one of the four mesh vertices defining the initial triangulation is classified as being outside. For any mesh face that is classified outside, for any bounding edge that is part of the domain's boundary, the face on the other side of the edge is classified on the model face. From there, proper classification can be propagated through edge neighbors. Mesh entities that are outside are deleted (top-down deletion).

6.6 Insertion of Interior Vertices

So far, the Delaunay method has been used strictly in the parameter space. If the temporary mesh was to be considered in the 3–d space, the quality of the triangulation could be quite bad if the mapping between the parameter space and the real space has a good amount of stretch and shear. This is without even taking into account the geometric similarity issue. From this point on, a quasi-Delaunay method is used in the real space. Watson's algorithm is used on mesh faces instead of mesh regions. For a given mesh face, the circumsphere is defined as the sphere that is centered on the face's circumcircle and of the same radius. Fig. 9 shows the circumcircle and circumsphere of a mesh face. If the model face is planar, the resulting quasi-Delaunay triangulation is actually Delaunay.

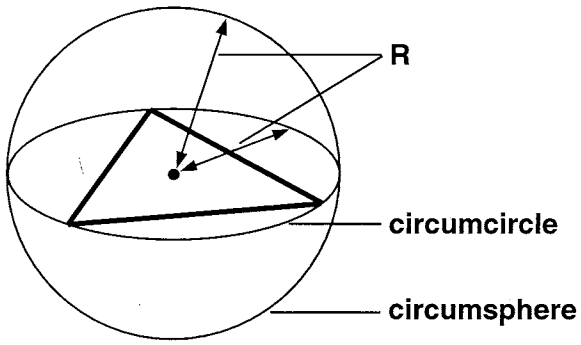

Figure 9 Quasi-Delaunay circumsphere used for surface meshing

Before inserting interior vertices, the mesh is transformed using edge swapping so that it conforms to the quasi-Delaunay criterion in 3–d space. Here, the fact that the Delaunay triangulation maximizes the minimum angle is used. For each edge connected to two mesh faces, the minimum angle among the two connected faces is computed. If an edge swap is valid and it increases the minimum angle, the swap is performed.

For each edge interior to the domain, decisions concerning whether or not the edge is of the appropriate size are made as in model edge meshing. If the edge is too big, a vertex is created at the middle of the segment in the parameter space and inserted into the quasi-Delaunay triangulation. The check for point visibility is performed in the parameter space as in the insertion of boundary vertices. It is important to perform this check since it will guarantee convergence of the geometric similarity check described next.

6.7 Geometric Similarity Check

Geometric similarity is assumed to be obtained if all mesh faces have face normals consistent (within 90 degrees) with surface normals. This is assuming that edge-connected mesh faces on a model face have consistent orientations, that is, they "use" the common edge in opposite directions.

Recall that it is assumed that a mesh edge is a straight segment in the parameter space and that it discretizes the arc corresponding to the mapping of that straight segment to the surface. The triangulation obtained after interior vertex insertion may not be geometrically similar if geodesics (associated with mesh edges) strongly curve in the parameter space. The following will explain how geometric dissimilarities occur and how they can be resolved

with the help of an example. For a given mesh edge judged too "big", interior vertex insertion considers a vertex at the middle of the (straight) mesh edge in the parameter space. In Fig. 10 (at left), the straight segments are mesh edges in real space and the dashed curved segments represent the arcs they are discretizing on the surface. Assume the common edge is judged too "big". Fig. 10 (at center) shows the mesh after insertion of a vertex at the middle of the associated arc. For simplification, insertion of the vertex was limited to a simple edge split. The new mesh is geometrically dissimilar since one new mesh face is "upside-down". Note that the new mesh is perfectly valid in the parameter space (assuming straight segments). In Fig. 10 (at right), the geometric dissimilarity is resolved by inserting an additional vertex on one of the sides of the "upside-down" face. This example helps in showing that the source of geometric dissimilarity comes from mesh edges which are relatively far from the arcs they are discretizing on the surface. Geometric dissimilarities usually occur when the surface is associated with a complex mapping, where geodesics strongly curve in the parameter space.

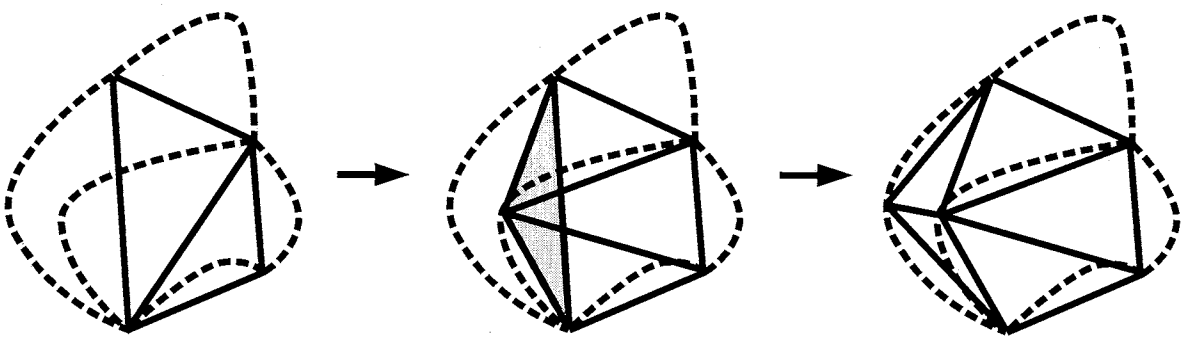

Figure 10 Geometric dissimilarity induced by vertex insertion and how to resolve it

The following presents the procedure (in simplified pseudo-code format) to check for and solve geometric dissimilarities on a given model face:

```
for each face on model face {
    compute face normal
    compute surface normal at centroid
    if the normals are within 90 degrees, continue
    find the bounding edge with maximum variation between edge length and arc length
    insert a vertex at middle of edge using interior insertion procedure
}
```

The mesh edge with maximum variation between edge length and arc length is the one that most likely causes the geometric dissimilarity. This is because the edge is too far from the arc it is discretizing. This procedure will converge since, at the limit, all mesh edges will "match" the arcs they are discretizing. At that point, geodesics (corresponding to the mesh edges) are nearly straight in the parameter space and geometric similarity checks can be performed in that space. Because the Delaunay insertion checks for point convexity in the parameter space, there cannot be geometric dissimilarity in the parameter space (in that case).

Although geometric similarity ensures that the mesh "matches" the geometry of the model, it does not guarantee that the mesh is valid. The next section describes the check for self-intersection, key for ensuring mesh validity.

7. Triangulation Self-Intersection Check

It is possible that mesh entities intersect each other, meaning the mesh is not valid. It is assumed here that mesh edges and faces are linear and planar, respectively, in the real space. Intersection needs only to be checked between mesh edges and mesh faces. Intersection can occur between mesh entities on the same model face or on different model faces. Because, at this point, all surface meshes are known to be geometrically similar, self-intersection usually comes from the mesh being too coarse (relative to the complexity of the model's geometry).

To be performed efficiently, the intersection checks require a localization structure. The octree built for the purpose of controlling mesh gradation can be used for localization if terminal octants keep track of the mesh faces within their volumes. If an intersection is detected between a mesh edge and a mesh face, they both are split. Splitting forces mesh entities to locally become closer to the model entities they are discretizing and it therefore reduces the geometric discretization error locally. The mesh is made locally finer, which will always, at the limit, solve any intersection problem. This guarantees that one can always generate valid surface meshes.

Splitting a mesh face is performed by first finding an appropriate split point. At this point in the meshing process, it is not desirable to use the parameter space to find the location of the split point since it can potentially lead to geometric dissimilarity. Instead, it is found directly in real space without considering the parameter space of the model face. An infinite line segment perpendicular to the mesh face and passing through its centroid is intersected with the surface associated with the model face. The split point is chosen as the point closest to the mesh face among all the intersection points for which the surface normal (at that location) differs by no more than 90 degrees with the mesh face normal. Once the appropriate split point has been found, the mesh face is split into three new mesh faces. Fig. 11 describes graphically this process in two dimensions. This example also shows that the method used here is preferable to a projection method which, in that particular case, would yield an undesirable split point (on the "wrong" side of the surface). It should be noted though that projection is acceptable, though costly, when the geometric discretization error is small enough. The determination of the split point for a mesh edge depends upon the classification of the mesh edge. If it is classified on a model edge, the split point is at the middle of the corresponding line segment in the geometric modeler's parameter space. If it is classified on a model face, the method used for mesh faces can be adapted to find the split point. The infinite line segment is defined by the average normals of the two connected mesh faces and passes through the middle of the mesh edge.

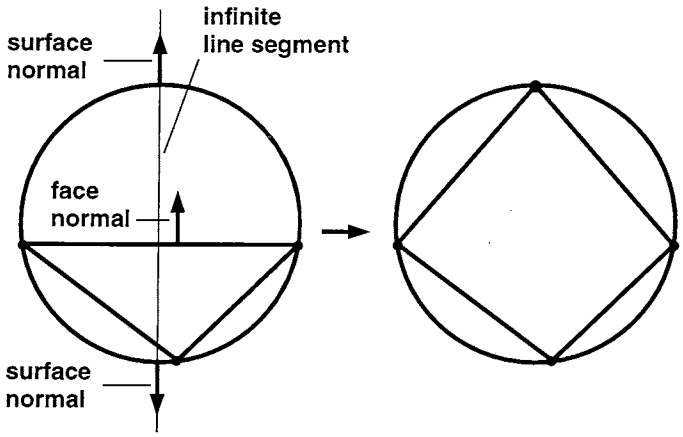

Figure 11 Finding the split point when splitting a mesh entity in real space

8. Triangulation Optimization

The obtained triangulation can be optimized in terms of quality. The quality of a mesh face is given here by its maximum face angle. The optimization procedure attempts to bring the maximum face angle among all mesh faces below a given angle threshold. It is based on local retriangulation tools. Any mesh face with a face angle above the threshold is attempted to be removed using:

1. edge collapsing,
2. edge swapping, or
3. edge or face splitting, or
4. combinations of the above.

A local retriangulation tool cannot be applied if it creates any geometric dissimilarity. Geometric similarity checks are performed using face and surface normal information. As for the triangulation self-intersection check, parameter spaces of model faces are not used.

9. Results

Fig. 12 shows the surface mesh of a nacelle. The triangulation has 2,206 triangles. The largest face angle is 137.9 degrees (optimization threshold was set at 130 degrees).

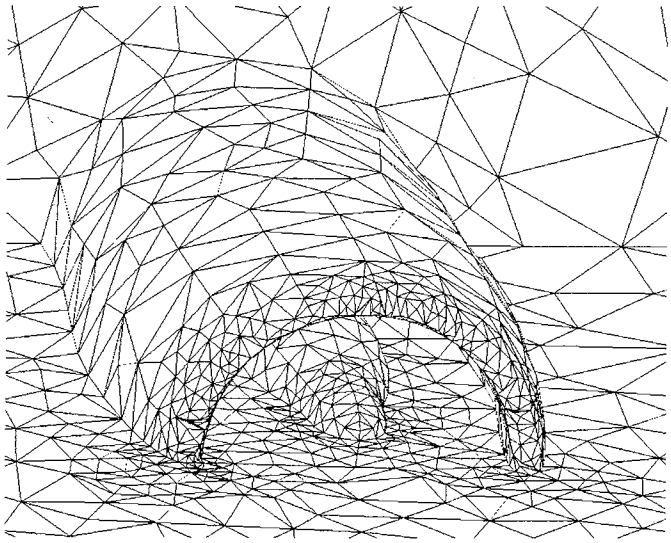

Figure 12 Surface mesh of a nacelle

Fig. 13 shows the surface mesh of a turbine blade. The triangulation has 3,380 triangles. The largest face angle is 152.3 degrees (optimization threshold was set at 130 degrees).

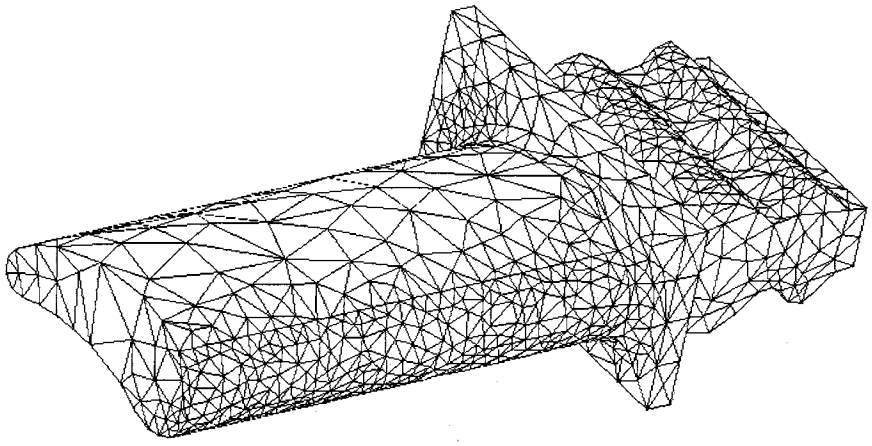

Figure 13 Surface mesh of a turbine blade

10. Concluding Remarks

This paper has presented a surface mesh generator based on Delaunay vertex insertion. Insertion is key for speed. The Delaunay criterion is key to (i) get a good distribution of interior vertices and (ii) generate relatively good quality surface triangles. Focus has been given to (i) surface mesh validity requirements and (ii) geometric similarity issues. Concerning geometric similarity, it has been pointed out that parameter spaces provided by geometric modelers may not be appropriate if mesh edges are represented as straight line segments in the parameter space. Since curving mesh edges in the geometric modeler's parameter space, solely for meshing purposes, is

not practical, geometric similarity should be redefined using, for example, a geometric criterion based on surface normals. In practice, geometric similarity becomes a subjective concept since its definition is not unique. It is nevertheless a useful concept since it guarantees that the surface mesh "matches" the geometry of the model.

Bibliography

[1] R. Kreiner and B. Kröplin. Unstructured quadrilateral mesh generation on surfaces from cad. In *Proc. of 4th Inter. Conf. on Numer. Grid Generation in Comp. Fluid Dyn. and Related Fields*, pages 211–221. Cromwell Press, 1994.

[2] X. Sheng and B. E. Hirsch. Triangulation of trimmed surfaces in parametric space. *Computer-Aided Design*, 24(8):437–444, 1992.

[3] D. Rypl and P. Krysl. Triangulation of 3-d surfaces. Technical report, Czech Tech. Univ. - Prague, Czech Rep., 1994.

[4] K. Nakahashi and D. Sharov. Direct surface triangulation using the advancing front method. *American Inst. of Aero. and Astro.*, 1686:442–451, 1995.

[5] M. S. Shephard and M. K. Georges. Automatic three-dimensional mesh generation by the Finite Octree technique. *Int. J. Numer. Meth. Engng.*, 32(4):709–749, 1991.

[6] M. S. Shephard and M. K. Georges. Reliability of automatic 3-D mesh generation. *Comp. Meth. Appl. Mech. Engng.*, 101:443–462, 1992.

[7] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press Inc., 1990.

[8] E. Kreyszig. *Differential Geometry*. Univ. of Toronto Press, 1959.

[9] W. J. Schroeder and M. S. Shephard. On rigorous conditions for automatically generated finite element meshes. In J. Turner, J. Pegna, and M. Wozny, editors, *Product Modeling for Computer-Aided Design and Manufacturing*, pages 267–281. North Holland, 1991.

[10] J. M. Beck, R. T. Farouki, and J. K. Hinds. Surface analysis methods. *IEEE Comp. Graph. and Appl.*, 6(12):18–36, 1986.

[11] U. M. Ascher, R. M. M. Mattheij, and R. D. Russell. *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*. Prentice Hall Inc., 1988.

[12] M. S. Shephard, S. Dey, and M. K. Georges. Automatic meshing of curved three-dimensional domains: Curving finite elements and curvature-based refinement. In I. Babuska, J. E. Flaherty, J. E. Hopcroft, W. D. Henshaw, J. E. Oliger, and T. Tezduyar, editors, *Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations*, volume 75, pages 67–96. Springer-Verlag, 1995. IMA Volumes in Mathematics and its Applications.

[13] S. Dey and Mark S. Shephard. Geometry based framework for h-p adaptive fem. Technical Report SCOREC # 1-1996, Scientific Computation Research Center, Rensselaer Polytechnic Institute, Troy, NY 12180-3590, 1996.

[14] P. L. George, F. Hecht, and E. Saltel. Automatic mesh generator with specified boundary. *Comp. Meth. Appl. Mech. Engng.*, 92:269–288, 1991.

[15] P. L. George and F. Hermeline. Delaunay's mesh of a convex polyhedron in dimension d. application to arbitrary polyhedra. *Int. J. Numer. Meth. Engng.*, 33:975–995, 1992.

[16] D. F. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *The Computer J.*, 24(2):167–172, 1981.