FACET-BASED SURFACES FOR 3D MESH GENERATION

Steven J. Owen[1], David R. White[2], Timothy J. Tautges[3]

[1,2,3]*Sandia National Laboratories*, Albuquerque, NM U.S.A sjowen@sandia.gov*
[2]*Carnegie Mellon University, Pittsburgh, PA U.S.A drwhite@sandia.gov*
[3]*University of Wisconsin, Madison, WI U.S.A tjtautg@sandia.gov*

ABSTRACT

A synthesis of various algorithms and techniques used for modeling and evaluating facet-based surfaces for 3D surface mesh generation algorithms is presented, including implementation details of how these techniques are used within an existing mesh generation toolkit. An object oriented data representation for facet-based surfaces is described. Numerical techniques for describing G^1 continuous curves and surfaces from triangles forming quartic Bézier triangle patches is presented. A combination of spatial searching and minimization techniques to solve the smooth surface projection problem are described. Examples and performance of the proposed methods are put in context with relevant surface mesh generation problems.

Keywords: facetted models, 3D surface mesh generation, closest point, Walton patch

1. INTRODUCTION

Computational simulations of physical processes modeled using finite element analysis frequently employ complex automatic mesh generation techniques. Robust geometric representations of the physical domain are generally required in order for mesh generation algorithms to be reliable. Solid models or boundary representation (b-rep) models are most frequently employed, typically provided through a commercial computer aided design (CAD) package or third party library.

Since automatic mesh generation algorithms must *mesh* an underlying solid model, a geometry kernel is frequently provided via an application programmers interface (API). A small set of geometry and topology queries are sufficient to provide automatic mesh generation algorithms with enough information to generate a mesh for computational simulation.

Since mesh generation algorithms usually access the geometry representation via an API, any underlying representation of the geometry may be used, provided it meets the requirements of the prescribed API. In most cases the underlying API is a b-rep model where curves and surfaces are described by analytic expressions or non-uniforms rational b-splines (NURBS). B-rep models based on NURBS have become commonplace in the computer aided engineering (CAE) community. Popular CAD modeling packages use NURBS as the basis of their geometry kernels, frequently integrating automatic mesh generation algorithms as an added feature.

In recent years, facetted models have become more important as an alternative geometry representation from NURBS representations. Complete 3D geometric models can be represented as a simply connected set of triangles. In many cases, facetted models may be preferred or may be the only representation available. For example, where natural processes are simulated such as in bio-medical or geo-technical applications, NURBS representations can be difficult or impossible to fit to the prescribed data. Even if an initial NURBS representation is used to model a machined part, subsequent large deformations computed from an FEA analysis may make the NURBS data irrelevant. A facetted model may be the only alternative for remeshing or refinement on a deformed model. Additionally, facets in the form of STL (stereo lithography) or graphics files, may be a convenient or even last-ditch form of geometry transfer when other representations are unavailable or inadequate. Furthermore, in some cases a legacy FEA mesh may be the only geometric representation available of the model. Extracting the surface faces from

*Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000

the elements to form facets is one alternative to provide a geometric model for subsequent design studies.

In the ideal case, mesh generation software has been designed to work through a generic API, not relying on any specific geometry representation. When this is the case, a facetted representation may conveniently replace or serve as an alternative geometry kernel with little or no change to the mesh generation algorithms themselves. The objective of the current work is to propose such a system. Significant prior work has gone into developing a mesh generation toolkit [1], consisting of a wide variety of algorithms [2,3,4,5,6] that will access a generic geometry kernel through a common geometry module [7]. Although the commercial third party library, ACIS [8], has been successfully used as a NURBS-based geometry kernel, an alternative facet-based geometry representation is proposed. It does not necessarily replace the ACIS representation, but serves to enhance the existing capability of the mesh generation toolkit.

This work is a synthesis of a variety of algorithms and techniques used to successfully model surfaces for purposes of mesh generation. This work does not present or promote any specific meshing algorithm, but instead proposes a geometry system that can be utilized by a generic 3D meshing algorithm.

We first present some background to the problem in order to put the subsequent discussion into perspective. We will then describe the data representation used followed by a detailed description of the numerical formulation of the surfaces. Details of how the facetted surface representation is used within the context of 3D surface meshing will then be described along with relevant performance data. Finally, several examples will be presented, showing typical use of the facetted representation for 3D mesh generation.

2. BACKGROUND

2.1 Previous Work

Several authors have presented similar facet-based systems for use with specific mesh generation algorithms. For example, Rypl and Bittnar [9] propose a method for evaluating facet-based surfaces based upon interpolation subdivision. This method utilizes the hierarchal recursive refinement of the facets. Control points for a triangular Bézier patch are positioned by computing a weighted average of neighboring nodes resulting in a C^1 surface. Their facet-based surface representation is employed with a 3D advancing front triangle meshing algorithm. Frey and George [10] also provide details on the use of triangle Bézier patches for modeling surfaces of prescribed continuity for application to finite element mesh generation. Additionally, Frey [11], and Laug and Borouchaki [12] address issues related to parametric space mesh generation on facet-based surfaces. The current effort, in contrast addresses integration of a facet-based system as a geometry kernel for 3D surface mesh generation.

The current work extends the previous work of the authors [13] where a method for decomposing a finite element model to generate a complete geometric model for mesh generation was proposed. Sets of facets were generated and grouped together to form surfaces based upon user defined feature angle criteria and boundary condition data. The current work assumes that facets have been adequately grouped into topological surfaces as described in [13].

2.2 Surface Meshing

Surface meshing algorithms may be characterized as either parametric or 3D. A parametric surface-meshing algorithm creates the mesh in a two-dimensional space, mapping the final nodal coordinates to three dimensions only as a final step. Many examples of parametric surface meshing exist in the literature [14,15,16]. Fundamental to parametric meshing is a global 2D parametric (u,v) representation for the entire surface. NURBS representations provided by CAD modelers conveniently provide this information. For facetted models, however, surface parameterizations are not easy to come by and are a topic of active research [17,18].

3D meshing algorithms, on the other hand, do not require an underlying parameterization. Rather than mapping between 2D and 3D spaces, the meshing algorithm is performed completely within the 3D space. The current work restricts its application to 3D meshing algorithms, leaving parametric meshing to subsequent research. Paving [2] and 3D advancing front triangle meshing [19] are examples of 3D meshing algorithms. These algorithms begin with an outer loop of mesh edges and begin placing elements on the surface marching systematically from the boundary towards the interior. Fundamental to these algorithms is the evaluation of the underlying surface. The *closest point* or *projection* routine combined with surface normal evaluations at a point are the main requirements these algorithms demand of the geometry kernel. While in most cases these functions are standard API calls available from a NURBS-based geometry system, they must be defined for a facet-based system. This work proposes several techniques for defining these routines for facet-based representations.

2.4 G^0 Surfaces

The facet representation can be characterized by the geometric continuity for which the composite set of triangles comprising a surface will maintain.

For purposes of this work, we will not consider parametric continuity (ie. C^0, C^1, …), as 3D surface meshing schemes will be employed. Instead G^0 and G^1 continuous representations will be considered. Farin [20] details the requirements of continuity across surface patches composed of Bézier patches. Kashyap [21] addresses higher order continuity between triangle patches.

A surface maintaining G^0 continuity guarantees only that that the facet edges meet edge-to-edge, without gaps. In this case, characteristic facets and ridges between triangles may be evident. In many cases, G^0 is sufficient to adequately characterize the surface for mesh generation. Planar surfaces, surfaces with large curvature or surfaces

with sufficiently dense triangles to represent curvature, may not require more than what a G^0 facet representation will provide. Since higher order representations will inevitably be more expensive, the option to provide for G^0 representations should be supplied in a facet-based geometry kernel for mesh generation.

Triangles may be conveniently parameterized using Barycentric coordinates. Instead of the standard *u,v* parameterization of tensor product surfaces, three parameters, *u,v,w*, are used, where $u+v+w=1$ and $u,v,w \geq 0$. Figure 1 shows the characteristic parametric domain for any triangle.

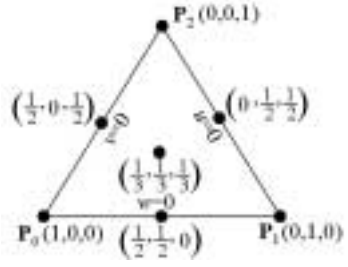


Figure 1. Barycentric coordinate system defined on a triangle.

Using Barycentric coordinates (*u,v,w*) any point **X** in the domain may be defined as:

$$\mathbf{X}(u,v,w) = u\mathbf{P}_0 + v\mathbf{P}_1 + w\mathbf{P}_2 \qquad (1)$$

where $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$ are the vertices of the triangle. In a similar manner, the normal at point **X** can be approximated as:

$$\mathbf{N}_X(u,v,w) = u\mathbf{N}_0 + v\mathbf{N}_1 + w\mathbf{N}_2 \qquad (2)$$

where $\mathbf{N}_0$, $\mathbf{N}_1$, $\mathbf{N}_2$ are the approximated normals at points $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$ respectively.

2.2 G^1 Surfaces

It may be necessary to provide a smooth approximation to the surface. To do so, Bézier triangular patches may be defined. A degree *n* triangular Bézier patch may be described with the following equation:

$$\mathbf{X}(u,v,w) = \sum_{i+j+k=n} B^n_{i,j,k}(u,v,w)\mathbf{P}_{i,j,k} \qquad (3)$$

$$B^n_{i,j,k} = \frac{n!}{i!\,j!\,k!}u^i v^j w^k \qquad (4)$$

$\mathbf{P}_{i,j,k}$ are the control points of the triangle and $B_{i,j,k}(u,v,w)$ can be thought of as the *weights* at polar location *i,j,k*, where $i+j+k=n$. Note also that equation (1) can be thought of as a special case of equation (3) where $n=1$. In this case $B_{ijk}=1$ for all *i,j,k*.

Triangular polar values are a convenient way of generalizing the control point indices for an arbitrary triangle patch. For example, Figure 2 shows the polar coordinate locations for triangle patches $n=3$ and $n=4$.

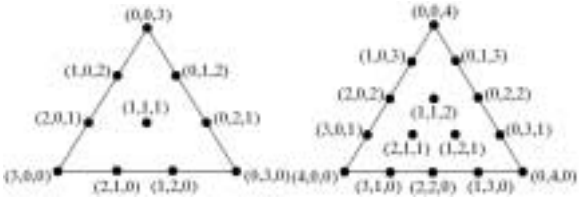


Figure 2. Polar values for triangles patches of *n*=3 and *n*=4

It has been established that the minimum order of a triangle Bézier patch necessary to model a G^1 surface is $n=4$ [22]. For this case, equation (3) can be written as:

$$\begin{aligned}\mathbf{X}(u,v,w) = {} & \mathbf{P}_{0,0,4}w^4 + \\ & 4\mathbf{P}_{1,0,3}uw^3 + 4\mathbf{P}_{0,1,3}vw^3 + \\ & 6\mathbf{P}_{2,0,2}u^2w^2 + 12\mathbf{P}_{1,1,2}uvw^3 + 6\mathbf{P}_{0,2,2}v^2w^2 + \\ & 4\mathbf{P}_{3,0,1}u^3w + 12\mathbf{P}_{2,1,1}u^2vw + 12\mathbf{P}_{1,2,1}uv^2w + 4\mathbf{P}_{0,3,1}v^3w + \\ & \mathbf{P}_{4,0,0}u^4 + 4\mathbf{P}_{3,1,0}u^3v + 6\mathbf{P}_{2,2,0}u^2v^2 + 4\mathbf{P}_{1,3,0}u^3v + \mathbf{P}_{0,4,0}v^4\end{aligned} \qquad (5)$$

Walton [23] describes a procedure for defining $\mathbf{P}_{i,j,k}$ as a function of the triangle vertex locations $\mathbf{P}_i$ {i=0..3} and normals $\mathbf{N}_i$ {i=0..3}. The Walton patch is central to the techniques proposed in the present work. How it is used within the context of a practical geometry kernel for mesh generation is the additional contribution proposed by the current research and development effort. Definition and implementation of the Walton patch is outlined in section 5.

Similarly, smooth normals for the quartic patch at $\mathbf{X}(u,v,w)$ may be computed by evaluating a cubic Bézier patch.

$$\begin{aligned}\mathbf{N}_x(u,v,w) = {} & \mathbf{N}_{0,0,3}w^3 + \\ & 3\mathbf{N}_{1,0,2}uw^2 + 3\mathbf{N}_{0,1,2}vw^2 + \\ & 3\mathbf{N}_{2,0,1}u^2w + 6\mathbf{N}_{1,1,1}uvw + 3\mathbf{N}_{0,2,1}v^2w + \\ & \mathbf{N}_{3,0,0}u^3 + 3\mathbf{N}_{2,1,0}u^2v + 3\mathbf{N}_{1,2,0}uv^2 + \mathbf{N}_{0,3,0}v^3\end{aligned} \qquad (6)$$

Assuming the normals at $\mathbf{N}_{3,0,0}$, $\mathbf{N}_{0,3,0}$, and $\mathbf{N}_{0,0,3}$ are the computed normals at the triangle vertices $\mathbf{N}_0$, $\mathbf{N}_1$ and $\mathbf{N}_2$ respectively, a procedure for computing the remaining $\mathbf{N}_{i,j,k}$ by extending the work of Walton is introduced in section 5.

3. DATA REPRESENTATION

As stated in the Introduction, facet-based surface construction is employed in a variety of contexts. These types of surfaces can be used to support mesh generation on facet-based models. The method used to store and evaluate the facet data can vary according to the source of those data. For example, in the context of large-deformation FEA, it is important to maintain links between the original mesh and the corresponding facets used to model the surface. Also, for these analyses, duplication and updating of facet data can be problematic because of memory limitations commonly found in parallel computing

environments and because position and topology of facets may be changing. Facets originating from STL or graphics files may not have those limitations. In contrast, the functions needed to support mesh generation do not depend on the source of the facets being evaluated, and therefore should look the same regardless of the facet representation. We accomplish both these goals utilizing inherited classes in C++ for our facet data structure.

We define a generic FacetEntity class, from which generic Point, FacetEdge and Facet classes are derived. Contained in these classes is the normal and control point data necessary to describe the G^1 surface approximation. A simplified representation of the data contained in these classes is illustrated in Figure 3.

```
class FacetEntity {}

class Facet : public FacetEntity {
    Vector3D *controlPoints;
    Vector3D normal;
    Box boundingBox;
    virtual get_points() = 0;
}

class FacetEdge : public FacetEntity {
    Vector3D *controlPoints;
    Boolean isFeature;
    virtual get_adjacent_facets() = 0;
}

class Point : public FacetEntity {
    List<PointNormal *> normalList;
    virtual get_location() = 0;
    virtual get_adjacent_facets() = 0;
}

class PointNormal (
    Vector3D normal;
    int surfaceID;
}
```

Figure 3. Simplified C++ classes used for defining facet representation

These classes also support typical functionality like point location and point-edge-facet adjacency. These functions are defined as virtual functions to allow representation-specific versions to be implemented. An example inheritance tree for the facet representation is illustrated in Figure 4.

The actual implementation of the functions described above is done in classes derived from Facet, FacetEdge and Point. We describe here two specific implementations: one that stores the facet data in the actual derived classes, and one that references those data from an analysis application. In the first child class, PointData, the point data (i.e. coordinates and adjacency information) are stored in the class and are used directly to implement the required functions. This is illustrated in Figure 4 in the box marked A. The second implementation example, in the FEAPoint class, does not store the point data in the class; those data are retrieved from the mesh objects in the FEA code, to which the FEAPoint object keeps a pointer. Box B in Figure 4 illustrates this case. Implementations of the FacetEdge and Facet classes are similar to those for Point.

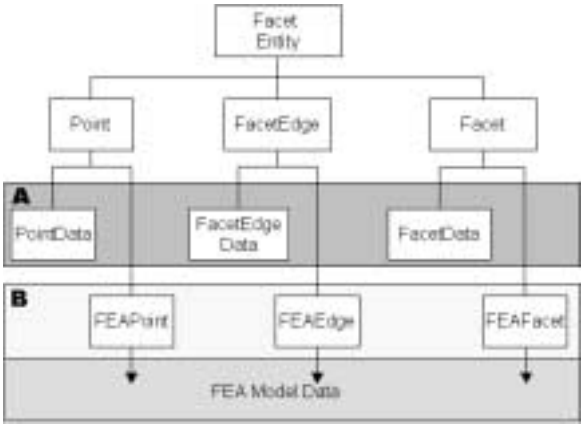


Figure 4. Inheritance tree for object oriented facet representation

The proposed object oriented implementation also leaves open the possibility for additional code reuse. Other applications need only supply the required topology information for the facets in order to take advantage of the proposed surface representation and evaluation.

4. FACET INITIALIZATION AND EVALUATION

The following procedure outlines the proposed process for initializing facet-based surfaces for use as a geometry kernel for 3D mesh generation. Figure 3 illustrates the basic data needed to sufficiently describe the smooth surface. Control point and normal data necessary to compute equations (5) and (6) are necessary. Depending on the speed vs memory requirements of the application, this data may be precomputed and stored with the facet entities or may be computed only as evaluations are required. For the mesh generation applications used with this work, it was found that pre-computing the normal and control point data was more efficient.

It should be noted that curves and surfaces with zero curvature can be accurately modeled with a G^0 surface representation. Given that many surfaces used to model machined parts have flat surfaces, it is worthwhile to check for this condition and only initialize those with any curvature. Furthermore, it may be worthwhile to perform an initial decimation of the facets comprising the surface to reduce memory requirements and increase the performance of the closest_point operation discussed later.

The initialization procedure begins first with the approximation of the normals at the Points. The normals are then used as input to compute the control points on the FacetEdges. Finally, the control point data on the Facets is computed using the FacetEdge control points. Details for initializing each of these data objects is now described.

4.1 Points

The definition of Bézier triangle facets requires normals to be provided or approximated at each of the facet vertices. If normals are available from an external source, such as those provided through an STL format, these should ideally

be used. If they are not provided, it is necessary to approximate them. Reference [13] describes the method used in the current work, defining the normal, $\mathbf{N}_P$ at a point as the weighted average of the adjacent facets, $\mathbf{N}_j$, where the weight, w_j is the normalized facet angle at the point. Equations (7), (8) and Figure 5 describe this procedure.

$$\mathbf{N}_P = \sum_{j=1}^{n} \mathbf{N}_j w_j \tag{7}$$

$$w_i = \frac{\alpha_i}{\sum_{i=1}^{n} \alpha_i} \tag{8}$$

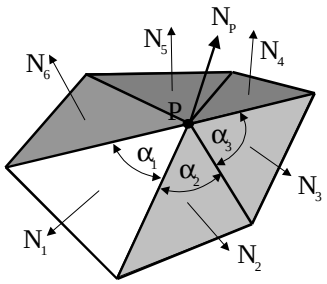


Figure 5. Definition of a normal at point P

At feature breaks, it is necessary to compute multiple normals at a point, where the normal is the weighted average of only the facet normals defined on a given surface. Figure 3 describes the Point class as containing a list of PointNormal objects. The PointNormal object contains both the computed normal and a reference to the surface to which it applies. It should be emphasized that this extra information is only required at breaks or surface boundaries to distinguish surface features and is only allocated at the Points as needed to ensure minimal additional memory overhead.

4.2 Facet Edges

To completely represent the quartic Bézier facet edge, five control points are needed. In addition to serving as the control points for the boundary curves, The control points on the FacetEdges are utilized in the definition of the triangle Bézier patches (discussed later) that are immediately adjacent the FacetEdge. Adjacency information provided by the data representation described in the previous section is necessary to keep track of this data and utilize it as necessary.

Since we may utilize the Point locations for the ends of the Bézier curve, the remaining three control points, $\mathbf{P}_{i,\ j}$ {j=1..3} may be stored with the FacetEdge. In practice, the required input to compute $\mathbf{P}_{i,j}$ (Figure 6) are the FacetEdge endpoints $\mathbf{P}_i$, $\mathbf{P}_{i+1}$, the surface normals, $\mathbf{N}_i$, $\mathbf{N}_{i+1}$ and the curve tangent vectors, $\mathbf{T}_i$, $\mathbf{T}_{i+1}$ at the same points.

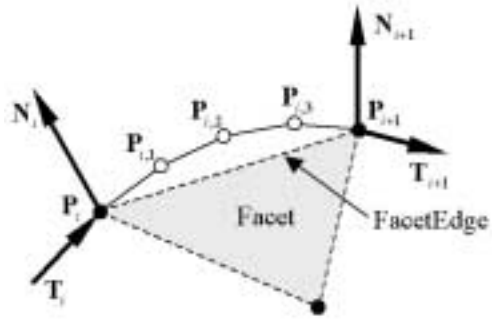


Figure 6. FacetEdge required input to compute control points $\mathbf{P}_{i,j}$

It is assumed that point $\mathbf{P}_i$ and $\mathbf{P}_{i+1}$ are provided. The previous section addresses approximation of normals $\mathbf{N}_i$, $\mathbf{N}_{i+1}$. In order to maintain G^0 continuity between adjacent facet-based surfaces, it is necessary to approximate the curve tangents, $\mathbf{T}_i$, $\mathbf{T}_{i+1}$. For edges on the interior of the surface where G^1 continuity is to be maintained, it is sufficient to define both $\mathbf{T}_i$ and $\mathbf{T}_{i+1}$ as the direction of the FacetEdge

$$\mathbf{T}_i = \mathbf{T}_{i+1} = \frac{\mathbf{P}_{i+1} - \mathbf{P}_i}{\left\|\mathbf{P}_{i+1} - \mathbf{P}_i\right\|} \tag{9}$$

For edges marked as Features, using equation (9) may result in gaps between the facets as illustrated in Figure 7. In this example a cylinder has been represented with a coarse set of facets. Facet A and Facet B are on different surfaces, but are required to meet at edge $\mathbf{P}_i\mathbf{P}_{i+1}$. Using only the facet normals and the tangents of equation (9), gaps between the facets would result in different $\mathbf{P}_{i,j}$ for the same edge depending upon which facet is used for the computation. This is illustrated in Figure 7 by $(\mathbf{P}_{i,j})_A$ and $(\mathbf{P}_{i,j})_B$ computed from Facet A and Facet B respectively. Instead, the following logic is employed:

$$\mathbf{T}_i = \begin{cases} \mathbf{L}_i \cdot \mathbf{L}_{i-1} < \cos(\theta), & \dfrac{\mathbf{L}_i + \mathbf{L}_{i-1}}{\left\|\mathbf{L}_i + \mathbf{L}_{i-1}\right\|} \\ \textit{otherwise}, & \mathbf{L}_i \end{cases} \tag{10}$$

$$\mathbf{T}_{i+1} = \begin{cases} \mathbf{L}_i \cdot \mathbf{L}_{i+1} < \cos(\theta), & \dfrac{\mathbf{L}_i + \mathbf{L}_{i+1}}{\left\|\mathbf{L}_i + \mathbf{L}_{i+1}\right\|} \\ \textit{otherwise}, & \mathbf{L}_i \end{cases} \tag{11}$$

where θ is the user defined feature angle, and vectors $\mathbf{L}_i, \mathbf{L}_{i-1}$ and $\mathbf{L}_{i+1}$ are defined as:

$$\mathbf{L}_i = \frac{\mathbf{P}_{i+1} - \mathbf{P}_i}{\left\|\mathbf{P}_{i+1} - \mathbf{P}_i\right\|} \tag{12}$$

$$\mathbf{L}_{i-1} = \frac{\mathbf{P}_i - (\mathbf{P}_i)_{prev}}{\left\|\mathbf{P}_i - (\mathbf{P}_i)_{prev}\right\|} \tag{13}$$

$$\mathbf{L}_{i+1} = \frac{(\mathbf{P}_{i+1})_{next} - \mathbf{P}_{i+1}}{\left\|(\mathbf{P}_{i+1})_{next} - \mathbf{P}_{i+1}\right\|} \tag{14}$$

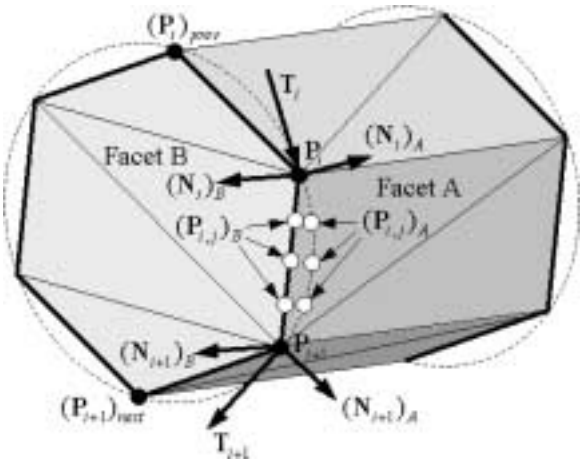


Figure 7. Cylinder represented by coarse facets. Tangent vectors required to maintain G^0 continuity between surfaces.

Figure 7 illustrates the location of point $(\mathbf{P}_i)_{prev}$ and $(\mathbf{P}_{i+1})_{next}$. These are the *previous* and *next* vertices on adjacent FacetEdges to $\mathbf{P}_i$ and $\mathbf{P}_{i+1}$ respectively that are on a feature edge. Note that multiple feature edges may meet at a single point **P**. In this case, the appropriate selection of $(\mathbf{P}_i)_{prev}$ and $(\mathbf{P}_{i+1})_{next}$ must be determined from local topology by traversing FacetEdges in order at $\mathbf{P}_i$ and $\mathbf{P}_{i+1}$ to locate the next FacetEdge marked as a feature.

Armed with the six vectors described in Figure 6, we are now prepared to compute the three control points $\mathbf{P}_{i,j}$ {j=0..3}. The procedure used is defined in Walton and Meeks [23], but is illustrated here for clarity. Note the modification to the Walton patch to accommodate the tangent vectors defined in equations (10) and (11) at feature edges.

$$d_i = \left\| \mathbf{P}_{i+1} - \mathbf{P}_i \right\| \tag{15}$$

$$a_i = \mathbf{N}_i \cdot \mathbf{N}_{i+1} \tag{16}$$

$$a_{i,0} = \mathbf{N}_i \cdot \mathbf{T}_i \tag{17}$$

$$a_{i,1} = \mathbf{N}_{i+1} \cdot \mathbf{T}_{i+1} \tag{18}$$

$$\rho_i = \frac{6(2a_{i,0} + a_i a_{i,1})}{4 - a_i^2} \tag{19}$$

$$\sigma_i = \frac{6(2a_{i,1} + a_i a_{i,0})}{4 - a_i^2} \tag{20}$$

$$\mathbf{V}_{i,1} = \mathbf{P}_i + \frac{d_i(6\mathbf{T}_i - 2\rho\mathbf{N}_i + \sigma\mathbf{N}_{i+1})}{18} \tag{21}$$

$$\mathbf{V}_{i,2} = \mathbf{P}_{i+1} - \frac{d_i(6\mathbf{T}_{i+1} + \rho\mathbf{N}_i + 2\sigma\mathbf{N}_{i+1})}{18} \tag{22}$$

$$\mathbf{P}_{i,1} = \frac{1}{4}\mathbf{P}_i + \frac{3}{4}\mathbf{V}_{i,1} \tag{23}$$

$$\mathbf{P}_{i,2} = \frac{1}{2}\mathbf{V}_{i,1} + \frac{1}{2}\mathbf{V}_{i,2} \tag{24}$$

$$\mathbf{P}_{i,3} = \frac{3}{4}\mathbf{V}_{i,2} + \frac{1}{4}\mathbf{P}_{i+1} \tag{25}$$

In the current work, $\mathbf{P}_{i,j}$ from equations (23) to (25) are stored with the FacetEdge. To further ensure continuity between facets on adjoining surfaces, it is necessary to average the final locations $\mathbf{P}_{i,j}$ computed with respect to each adjoining facet.

Since the objective of this exercise is to provide a means to evaluate equation (5), the following correspondence exists between the points $\mathbf{P}_{i,j,k}$ defined with polar indices, $\mathbf{P}_i$ and $\mathbf{P}_{i,j}$.

$$\begin{array}{llll} \mathbf{P}_{4,0,0} = \mathbf{P}_0 & \mathbf{P}_{0,3,1} = \mathbf{P}_{0,1} & \mathbf{P}_{1,0,3} = \mathbf{P}_{1,1} & \mathbf{P}_{3,1,0} = \mathbf{P}_{2,1} \\ \mathbf{P}_{0,4,0} = \mathbf{P}_1 & \mathbf{P}_{0,2,2} = \mathbf{P}_{0,2} & \mathbf{P}_{2,0,2} = \mathbf{P}_{1,2} & \mathbf{P}_{2,2,0} = \mathbf{P}_{2,2} \\ \mathbf{P}_{0,0,4} = \mathbf{P}_2 & \mathbf{P}_{0,1,3} = \mathbf{P}_{0,3} & \mathbf{P}_{3,0,1} = \mathbf{P}_{1,3} & \mathbf{P}_{1,3,0} = \mathbf{P}_{2,3} \end{array} \tag{26}$$

4.3 Facets

Making use of the control points on the FacetEdges, the quartic Bézier patch, control points $\mathbf{P}_{1,1,2}$, $\mathbf{P}_{2,1,1}$ and $\mathbf{P}_{1,2,1}$ on the interior of the facet can now be defined. The locations of these points are critical to describing G^1 continuity between adjoining patches. Walton and Meeks propose a method, whereby two candidate locations for interior control points are defined for each FacetEdge, resulting in six vectors $\mathbf{G}_{i,j}$ {i=0,1,2; j=0,1}. Final locations for interior control points $\mathbf{P}_{1,1,2}$, $\mathbf{P}_{2,1,1}$ and $\mathbf{P}_{1,2,1}$ are a function of the evaluated (u,v,w) parameters and $\mathbf{G}_{i,j}$. Once again the procedure outlined by Walton and Meeks is described here for clarity.

For each FacetEdge, *i*, in the Facet, compute the following:

$$\mathbf{W}_{i,0} = \mathbf{V}_{i,1} - \mathbf{P} \tag{27}$$

$$\mathbf{W}_{i,1} = \mathbf{V}_{i,2} - \mathbf{V}_{i,1} \tag{28}$$

$$\mathbf{W}_{i,2} = \mathbf{P}_{i+1} - \mathbf{V}_{i,2} \tag{29}$$

$$\mathbf{D}_{i,0} = \mathbf{P}_{i,3} - \frac{1}{2}(\mathbf{P}_{i,1} + \mathbf{P}_i) \tag{30}$$

$$\mathbf{D}_{i,1} = \mathbf{P}_{i,1} - \frac{1}{2}(\mathbf{P}_{i+1} + \mathbf{P}_{i,3}) \tag{31}$$

$$\mathbf{A}_{i,0} = \frac{\mathbf{N}_i \times \mathbf{W}_{i,0}}{\left\| \mathbf{W}_{i,0} \right\|} \tag{32}$$

$$\mathbf{A}_{i,2} = \frac{\mathbf{N}_{i+1} \times \mathbf{W}_{i,2}}{\left\| \mathbf{W}_{i,2} \right\|} \tag{33}$$

$$\mathbf{A}_{i,1} = \frac{\mathbf{A}_{i,0} + \mathbf{A}_{i,2}}{\left\| \mathbf{A}_{i,0} + \mathbf{A}_{i,2} \right\|} \tag{34}$$

$$\lambda_{i,0} = \frac{\mathbf{D}_{i,0} \cdot \mathbf{W}_{i,0}}{\mathbf{W}_{i,0} \cdot \mathbf{W}_{i,0}} \tag{35}$$

$$\lambda_{i,1} = \frac{\mathbf{D}_{i,1} \cdot \mathbf{W}_{i,2}}{\mathbf{W}_{i,2} \cdot \mathbf{W}_{i,2}} \tag{36}$$

$$\mu_{i,0} = \mathbf{D}_{i,0} \cdot \mathbf{A}_{i,0} \tag{37}$$

$$\mu_{i,1} = \mathbf{D}_{i,1} \cdot \mathbf{A}_{i,2} \tag{38}$$

The two control points $\mathbf{G}_{i,0}$, $\mathbf{G}_{i,1}$ with respect to FacetEdge i may now be defined as:

$$\begin{aligned}\mathbf{G}_{i,0} = &\frac{1}{2}(\mathbf{P}_{i,1} + \mathbf{P}_{i,2}) \\ &+\frac{2}{3}\lambda_{i,0}\mathbf{W}_{i,1} + \frac{1}{3}\lambda_{i,1}\mathbf{W}_{i,0} \\ &+\frac{2}{3}\mu_{i,0}\mathbf{A}_{i,1} + \frac{1}{3}\mu_{i,1}\mathbf{A}_{i,0}\end{aligned} \tag{39}$$

$$\begin{aligned}\mathbf{G}_{i,1} = &\frac{1}{2}(\mathbf{P}_{i,2} + \mathbf{P}_{i,3}) \\ &+\frac{1}{3}\lambda_{i,0}\mathbf{W}_{i,2} + \frac{2}{3}\lambda_{i,1}\mathbf{W}_{i,1} \\ &+\frac{1}{3}\mu_{i,0}\mathbf{A}_{i,2} + \frac{2}{3}\mu_{i,1}\mathbf{A}_{i,1}\end{aligned} \tag{40}$$

In practice, the six control points, $\mathbf{G}_{i,j}$ are stored with the Facet object. When the facet is to be evaluated, interior control points $\mathbf{P}_{1,1,2}$, $\mathbf{P}_{2,1,1}$ and $\mathbf{P}_{1,2,1}$ may be uniquely evaluated as:

$$\mathbf{P}_{1,1,2} = \frac{1}{u+v}(u\mathbf{G}_{2,2} + v\mathbf{G}_{0,1}) \tag{41}$$

$$\mathbf{P}_{1,2,1} = \frac{1}{w+u}(w\mathbf{G}_{0,2} + u\mathbf{G}_{1,1}) \tag{42}$$

$$\mathbf{P}_{2,1,1} = \frac{1}{v+w}(v\mathbf{G}_{1,2} + w\mathbf{G}_{2,1}) \tag{43}$$

Finally, in order to evaluate the Bézier facet at parameter (u,v,w), equation (5) may be employed by using the control points defined in equation (26) and equations (41) to (43). Care must be taken to extract the appropriate control points in the correct order from the FacetEdges and Points so that control points $\mathbf{P}_{i,j,k}$ are appropriately assigned.

4.4 Normals

Surface normals may be computed by evaluating the cubic Bézier of equation (6). Once again the normals computed at the facet vertices $\mathbf{N}_0$, $\mathbf{N}_1$, $\mathbf{N}_2$ can be used for the control points $\mathbf{N}_{3,0,0}$, $\mathbf{N}_{0,3,0}$ and $\mathbf{N}_{0,0,3}$ respectively. Left to be computed are the seven remaining control points: two per edge and one at the center of the triangle.

As with the calculation of the quartic control points, depending upon the speed vs. memory requirements of the application, the normals for the cubic patch of equation (6) may be precomputed and stored, or they may be computed only as needed. If they are precomputed, Two additional control points per FacetEdge, $\mathbf{N}_{i,0}$ and $\mathbf{N}_{i,1}$ would be computed and stored. In this case we may take advantage of the cross edge derivative information computed in equations (30) and (31).

$$\mathbf{N}_{i,0} = \frac{\mathbf{D}_{i,0} \times \mathbf{W}_{i,0}}{\left\|\mathbf{D}_{i,0} \times \mathbf{W}_{i,0}\right\|} \tag{44}$$

$$\mathbf{N}_{i,1} = \frac{\mathbf{D}_{i,1} \times \mathbf{W}_{i,2}}{\left\|\mathbf{D}_{i,1} \times \mathbf{W}_{i,2}\right\|} \tag{45}$$

The correspondence between the control points $\mathbf{N}_{i,j,k}$ in equation (6) and the $\mathbf{N}_{i,j}$ at the FacetEdges defined in equations (44) and (45) is as follows:

$$\begin{aligned}&\mathbf{N}_{3,0,0} = \mathbf{N}_0 \quad \mathbf{N}_{0,2,1} = \mathbf{N}_{0,0} \quad \mathbf{N}_{1,0,2} = \mathbf{N}_{1,0} \quad \mathbf{N}_{2,1,0} = \mathbf{N}_{2,0} \\ &\mathbf{N}_{0,3,0} = \mathbf{N}_1 \quad \mathbf{N}_{0,1,2} = \mathbf{N}_{0,1} \quad \mathbf{N}_{2,0,1} = \mathbf{N}_{1,1} \quad \mathbf{N}_{1,2,0} = \mathbf{N}_{2,1} \\ &\mathbf{N}_{0,0,3} = \mathbf{N}_2\end{aligned} \tag{46}$$

Remaining to be computed is the internal control point of the cubic Bézier. This may be computed when the facet is evaluated using the interior control points from the quartic Bézier as follows:

$$\mathbf{N}_{1,1,1} = \frac{\left(\mathbf{P}_{1,2,1} - \mathbf{P}_{2,1,1}\right) \times \left(\mathbf{P}_{1,1,2} - \mathbf{P}_{2,1,1}\right)}{\left\|\left(\mathbf{P}_{1,2,1} - \mathbf{P}_{2,1,1}\right) \times \left(\mathbf{P}_{1,1,2} - \mathbf{P}_{2,1,1}\right)\right\|} \tag{47}$$

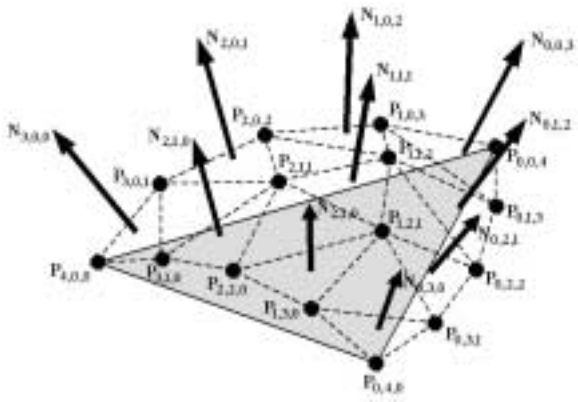


Figure 8. Quartic control point grid showing relationship to cubic grid of normals.

In practice, rather than storing $\mathbf{N}_{i,j}$ for each FacetEdge, $\mathbf{N}_{i,j}$ may be computed only as needed. Instead of regenerating the cross derivatives $\mathbf{D}_{i,j}$ for equations (44) and (45), the normals $\mathbf{N}_{i,j}$ can be derived directly from the control points of the quartic Bézier in a similar manner to equation (47). Figure 8 shows the relationship between the quartic control point grid and the normals required for the cubic Bézier patch. The normal of a local triangle in the quartic Bézier control grid should provide a sufficient approximation to the cubic Bézier control points. Therefore, the control points for the cubic Bézier can be determined as follows:

$$\mathbf{N}_{i,j,k} = \frac{\left(\mathbf{P}_{i,j+1,k} - \mathbf{P}_{i+1,j,k}\right) \times \left(\mathbf{P}_{i,j+1,k} - \mathbf{P}_{i+1,j,k}\right)}{\left\| \left(\mathbf{P}_{i,j+1,k} - \mathbf{P}_{i+1,j,k}\right) \times \left(\mathbf{P}_{i,j+1,k} - \mathbf{P}_{i+1,j,k}\right) \right\|} \quad (48)$$
$$\{i+j+k=3\}, \{i,j,k \geq 0\}$$

4.5 FacetEdge Evaluation

3D mesh generation algorithms frequently require the curves to be evaluated. Since we have defined our curves to be a set of FacetEdges, curve evaluation can utilize the control point information stored with the FacetEdge. One approach is to utilize equation (5), setting one of the parameters (u,v,w) to zero. This method however requires computing unnecessary control point information on the facet. Instead, the standard Bézier equation may be used:

$$\mathbf{P}(t) = \sum_{i=0}^{n} B_i^n(t)\mathbf{P}_i \quad (49)$$

$$B_i^n(t) = \frac{n!}{i!(n-i)!}(1-t)^{n-i}t^i \quad (50)$$

For the quartic case, evaluation of the FacetEdge, i, Bézier curve at parameter t $\{0 \leq t \leq 1\}$ using the facet vertices $\mathbf{P}_0$, $\mathbf{P}_1$ and edge control points $\mathbf{P}_{i,j}$ (equations (23) to (25)) may be defined as:

$$\mathbf{X}(t) = \mathbf{P}_i(1-t)^4 + 4\mathbf{P}_{i,1}(1-t)^3 t + 6\mathbf{P}_{i,2}(1-t)^2 t^2 + 4\mathbf{P}_{i,3}(1-t)t^3 + \mathbf{P}_{i+1}t^4 \quad (51)$$

The tangent vector along the edge may also be computed by evaluating a cubic Bézier.

$$\mathbf{T}(t) = \mathbf{T}_i(1-t)^3 + 3\mathbf{T}_{i,1}(1-t)^2 t + 3\mathbf{T}_{i,2}(1-t)t^2 + \mathbf{T}_{i+1}t^3 \quad (52)$$

where $\mathbf{T}_i$ and $\mathbf{T}_{i+1}$ are defined by equations (9) to (11) or more generally $\mathbf{T}_{i,j}$ may be defined as:

$$\mathbf{T}_{i,j} = \mathbf{P}_{i,j+1} - \mathbf{P}_{i,j} \{j = 0...3\} \quad (53)$$

5. CLOSEST POINT

The most common API request for 3D mesh generation algorithms is the closest point, or surface projection: given an arbitrary point in space, **X**, find the closest point on the surface $\mathbf{X}_s$. Since this procedure may be called thousands of times to mesh a single surface, an efficient method is sought to perform this operation.

The closest point calculation involves two main steps. The first task is selection of a small list of facets close to **X** from which to evaluate. Once this is determined a minimization procedure is employed for finding the closest point on the facet and the distance from **X** to each facet in the list is computed. The facet whose computed distance is smallest is selected as $\mathbf{X}_s$.

5.1 Closest Facet

An obvious solution to the closest facet problem is to project point **X** to each of the facets in the surface and select the point that is the least distance from **X**. Since this would be very inefficient, the objective of the closest facet procedure is to minimize the number of facets that must eventually be evaluated. Various spatial searching techniques are available in the literature [24]. The proposed method involves a combination of two methods.

5.1.1 Bounding Box Elimination Method

The first method is a simple bounding box elimination method. In this case, the axis-aligned bounding box of each facet, $\mathbf{B}_f$ is evaluated and stored with the Facet object (see Figure 3). This may be done once for each facet upon initialization of the facet surface. Care should be taken to utilize the Bézier control points in the bounding box calculation. Upon projection of point **X**, a trial axis-aligned bounding box, $\mathbf{B}_X$ is defined around **X** such that $(\mathbf{B}_X)_{min}$ = **X**-δ and $(\mathbf{B}_X)_{max}$ = **X**+δ. δ is a small scalar distance representative of the model size. In practice, the diagonal of the model bounding box scaled by 10^{-3} is used as the initial value of δ.

Facets selected to evaluate are only those where the bounding box $\mathbf{B}_f$ overlaps with $\mathbf{B}_X$. For this method, every facet on the surface must be checked and their bounding boxes verified for overlap. If after checking every facet, no bounding box $\mathbf{B}_f$ is found to overlap with $\mathbf{B}_X$, δ is scaled by 2, $\mathbf{B}_X$ is recomputed and the facets checked again. This procedure continues, scaling δ each time, until at least one $\mathbf{B}_f$ is found to overlap with $\mathbf{B}_X$. For 3D mesh generation algorithms, the point to be projected is generally close to the surface. As a result, in most cases, not more than a single pass through the facets is required. Care should be taken in choosing an initial δ, as choosing δ too small could result in multiple iterations through the facet list before encountering any overlap. Choosing δ too large, on the other hand, may result in too many facets being selected to evaluate. This procedure can become expensive as the number of facets increase. For this reason an alternate spatial search algorithm is utilized.

5.1.2 R-Tree Spatial Search

The dynamic index structure, *R-Tree* [25], is used to find only the closest facets to evaluate. The R-Tree structure was selected because of the multi-dimensional search structure of the algorithm. Most spatial search algorithms such as octrees and kd-trees [26], are typically only useful for storing and retrieving k dimensional data. For instance, problems, which involve finding the closest 3D point, can be readily solved by octrees. Facet data, however, if transformed into a single point would require nine dimensional octrees (3 dimensions for each x-y-z location of each point). Additionally, determining intersections in nine dimensional spaces are often error prone. The R-Tree defined by Guttman is specifically designed to handle mutli-dimensional search spaces efficiently. It is a height-balanced tree with its leaf nodes containing pointers to Facet objects. The structure is designed so that only a small number of leaf nodes are visited for any search operation, however the worst case of the algorithm is not guaranteed to be faster than the traditional n^2 algorithm. The balancing algorithm of the tree is such that this

extreme case will be rare, and in practice the tree appears to run at about O(n) where n is the number of facets.

Although relatively efficient, in practice, the overhead of generating and utilizing the additional infrastructure of the R-Tree is not worthwhile for small numbers of facets. Timed tests indicated that using only the bounding box elimination method was more efficient for up to approximately 2000 facets. Where additional facets were needed to represent a surface, R-Tree was able to improve performance considerably. As a result, for the current work, R-Tree is used only when the number of facets on a surface exceeds 2000.

5.2 Closest Point on a Facet

Locating the closest point on a facet, F_i can be performed on the (hopefully) small list of facets determined from either the bounding box elimination method or R-Tree search. For G^0 surfaces, this is relatively straightforward, as illustrated by the following procedure

5.2.1 Projection to a Linear Facet

1. Determine the closest point, $\mathbf{X}_p$ to the plane defined by F_i.

$$d = \mathbf{N}_F \cdot \mathbf{X} - \mathbf{N}_F \cdot \mathbf{P}_0 \tag{54}$$

$$\mathbf{X}_p = \mathbf{X} - d\mathbf{N}_F \tag{55}$$

   where $\mathbf{N}_F$ is the facet normal and $\mathbf{P}_0$ is a vertex on F_i.

2. Determine if $\mathbf{X}_p$ is inside F_i by computing its barycentric coordinates with respect to F_i.

$$u = \frac{\left\|(\mathbf{P}_2 - \mathbf{P}_1)\times(\mathbf{X}_p - \mathbf{P}_1)\right\|}{\left\|(\mathbf{P}_1 - \mathbf{P}_0)\times(\mathbf{P}_2 - \mathbf{P}_0)\right\|} \tag{56}$$

$$v = \frac{\left\|(\mathbf{P}_0 - \mathbf{P}_2)\times(\mathbf{X}_p - \mathbf{P}_2)\right\|}{\left\|(\mathbf{P}_1 - \mathbf{P}_0)\times(\mathbf{P}_2 - \mathbf{P}_0)\right\|} \tag{57}$$

$$w = \frac{\left\|(\mathbf{P}_1 - \mathbf{P}_0)\times(\mathbf{X}_p - \mathbf{P}_0)\right\|}{\left\|(\mathbf{P}_1 - \mathbf{P}_0)\times(\mathbf{P}_2 - \mathbf{P}_0)\right\|} \tag{58}$$

3. If all parameters $u,v,w > 0$, then $\mathbf{X}_s = \mathbf{X}_p$

4. Otherwise $\mathbf{X}_p$ is outside F_i. Find the closest point to one of the facet edges or facet vertices.

   Let $\mathbf{P}_i$ $\{i=0,1,2\}$ be the vertex locations on F_i, then project $\mathbf{X}_p$ to the closest edge based on the following:

$$i = \begin{cases} u < 0, & 1 \\ v < 0, & 2 \\ w < 0, & 0 \end{cases} \tag{59}$$

$$l_i = \left\|\mathbf{P}_{i+1} - \mathbf{P}_i\right\| \tag{60}$$

$$\mathbf{E}_i = \frac{\mathbf{P}_{i+1} - \mathbf{P}_i}{l_i} \tag{61}$$

$$\mathbf{V}_i = \mathbf{X}_p - \mathbf{P}_i \tag{62}$$

$$l_e = \mathbf{V}_i \cdot \mathbf{E}_i \tag{63}$$

$$\mathbf{X}_S = \begin{cases} l_e \geq l_i, & \mathbf{P}_{i+1} \\ l_e \leq 0, & \mathbf{P}_i \\ 0 < l_e < l_i, & \mathbf{P}_i + \mathbf{E}_i l_e \end{cases} \tag{64}$$

5.1.2 Projection to a triangular Bézier Patch

To compute the closest point, $\mathbf{X}_s$ to the Bézier patch a minimization algorithm must be employed. Numerical minimization techniques are standard in the literature [27] but are not common for this specific application. For example, Ristic [28] suggests using the Downhill Simplex Method for finding the closest point to a tensor product NURBS patch with application to the NURBS registration problem. Rypl and Bittnar [9] also outline a method for projecting a point to a target triangle patch. In this case, we present a line minimization technique tuned for use with the triangle Bézier patch. This is utilized for purposes of mesh generation since in general the initial guess will be an excellent approximation to the actual point on the surface. We attempt to minimize the distance *d* to the triangle patch as:

$$d = \sqrt{\begin{aligned}&(x_X - x(u,v,w))^2 + \\ &(y_X - y(u,v,w))^2 + \\ &(z_X - z(u,v,w))^2\end{aligned}} \tag{65}$$

where $\mathbf{X}(x_X, y_X, z_X)$ is the point to be projected and the location $\mathbf{X}_S = (x(u,v,w), y(u,v,w), z(u,v,w))$ is a point on the surface which is a function of the area coordinates **A** (*u*,*v*,*w*).

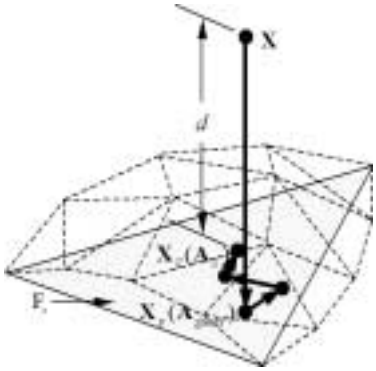


Figure 9. Projection of a point to the Bézier patch

Figure 9 illustrates the procedure used for projecting a point **X** to the triangle Bezier patch. The point is first projected to the facet F_i to determine an initial parametric guess $\mathbf{A}_{guess}$ at location $\mathbf{X}_p$ on the facet. $\mathbf{X}_p$ is defined by equation (55) and $\mathbf{A}_{guess}$ from equations (56), (57) and (58).

Given this information, the procedure progresses as follows:

1. First check if **X** is within tolerance, *tol* of one of the facet vertices. Define a convergence tolerance relative to the size of facet F_i. For example
$tol \approx \sqrt{area(F_i)} \times 10^{-3}$

   IF ($\|\mathbf{X} - \mathbf{P}_i\| < tol$) THEN $\mathbf{X}_s = \mathbf{P}_i$, RETURN.

2. Check if $\mathbf{A}_{guess}$ is outside the parametric domain of the patch and move it to the parametric boundary.

   Let $\mathbf{A} = \mathbf{A}_{guess}$

   FOR i=1,2,3; j=(i+1)%3; k=(i+2)%3

$$\text{IF } (\mathbf{A}[i] < 0) \text{THEN} \begin{cases} \mathbf{A}[i] = 0 \\ \mathbf{A}[j] = \dfrac{\mathbf{A}[j]}{\mathbf{A}[j] + \mathbf{A}[k]} \\ \mathbf{A}[k] = 1 - \mathbf{A}[j] \end{cases} \quad (66)$$

3. Let $\mathbf{X}_A = \Phi(\mathbf{A})$ be the evaluation of the Bézier patch at parametric location, **A** (see equation (5))

$$\mathbf{X}_A = \Phi(\mathbf{A}) \quad (67)$$

$$\mathbf{V}_{move} = \mathbf{X} - \mathbf{X}_A, d_{move} = \|\mathbf{V}_{move}\| \quad (68)$$

4. Keep track of the best area coordinate, ($\mathbf{A}_{best}$) the location at $\mathbf{A}_{best}$, ($\mathbf{X}_{best}$) and its distance from **X**, (d_{best}). Initialize with the values computed in (67) and (68). Also keep track of the number of iterations *niter*.

$$\mathbf{A}_{best} = \mathbf{A}, \mathbf{X}_{best} = \mathbf{X}_A, d_{best} = d_{move} \quad (69)$$

$$niter = 0 \quad (70)$$

5. Begin Iteration Loop. Check for convergence and ensure the maximum number of iterations, IMAX has not been exceeded.

   IF (d_{move} < *tol* OR *niter* > IMAX) THEN

   $\mathbf{X}_s = \mathbf{X}_{best}, \mathbf{A}_S = \mathbf{A}_{best}$

   RETURN

6. Parameters *u*,*v*,*w* are not linearly independent (*u*+*v*+*w*=1). Choose a system, of two variables to optimize.

   FOR i=0,1,2; j=(i+1)%3; k=(i+2)%3

   IF (**A**[i]>**A**[j] AND **A**[i]>**A**[k]) THEN

$$\eta = j, \xi = k, \psi = i \quad (71)$$

7. Define area coordinates, $\mathbf{A}_\eta$ and $\mathbf{A}_\xi$ perturbed a small distance δ from **A**. in directions η and ξ respectively.

$$\mathbf{A}_\eta[\eta] = \mathbf{A}[\eta] + \delta$$

$$\mathbf{A}_\eta[\xi] = (1 - \mathbf{A}_\eta[\eta])\left(1 - \frac{\mathbf{A}[\xi]}{\mathbf{A}[\xi] + \mathbf{A}[\psi]}\right) \quad (72)$$

$$\mathbf{A}_\eta[\psi] = 1 - \mathbf{A}_\eta[\eta] - \mathbf{A}_\eta[\xi]$$

$$\mathbf{A}_\xi[\xi] = \mathbf{A}[\xi] + \delta$$

$$\mathbf{A}_\xi[\eta] = (1 - \mathbf{A}_\xi[\xi])\left(1 - \frac{\mathbf{A}[\eta]}{\mathbf{A}[\eta] + \mathbf{A}[\psi]}\right) \quad (73)$$

$$\mathbf{A}_\xi[\psi] = 1 - \mathbf{A}_\xi[\eta] - \mathbf{A}_\xi[\xi]$$

8. Evaluate at $\mathbf{A}_\eta$ and $\mathbf{A}_\xi$ and approximate derivative vectors $\mathbf{V}_\eta$ and $\mathbf{V}_\xi$ in directions η and ξ.

$$\mathbf{V}_\eta = \frac{\Phi(\mathbf{A}_\eta) - \mathbf{X}_A}{\delta} \quad (74)$$

$$\mathbf{V}_\xi = \frac{\Phi(\mathbf{A}_\xi) - \mathbf{X}_A}{\delta} \quad (75)$$

9. Compute a search direction tangent to the Bezier patch

$$\mathbf{N}_{\eta\xi} = \mathbf{V}_\eta \times \mathbf{V}_\xi \quad (76)$$

$$\mathbf{V}_{move} = (\mathbf{N}_{\eta\xi} \times \mathbf{V}_{move}) \times \mathbf{N}_{\eta\xi} \quad (77)$$

10. Project to 2D coordinate plane based on magnitude of $\mathbf{N}_{\eta\xi}$

    FOR i=1,2,3; j=(i+1)%3; k=(i+2)%3

    IF ($|\mathbf{N}_{\eta\xi}[i]| > |\mathbf{N}_{\eta\xi}[j]|$ AND
    $|\mathbf{N}_{\eta\xi}[i]| > |\mathbf{N}_{\eta\xi}[k]|$) THEN

    I=j, J=k

Solve 2×2 linear equation in I, J system to determine the move distance d_η, d_ξ in the parameter space.

$$d_\eta = \frac{\begin{vmatrix} \mathbf{V}_{move}[\mathrm{I}] & \mathbf{V}_\xi[\mathrm{I}] \\ \mathbf{V}_{move}[\mathrm{J}] & \mathbf{V}_\xi[\mathrm{J}] \end{vmatrix}}{\begin{vmatrix} \mathbf{V}_\eta[\mathrm{I}] & \mathbf{V}_\xi[\mathrm{I}] \\ \mathbf{V}_\eta[\mathrm{J}] & \mathbf{V}_\xi[\mathrm{J}] \end{vmatrix}}, d_\xi = \frac{\begin{vmatrix} \mathbf{V}_\eta[\mathrm{J}] & \mathbf{V}_{move}[\mathrm{J}] \\ \mathbf{V}_\eta[\mathrm{I}] & \mathbf{V}_{move}[\mathrm{I}] \end{vmatrix}}{\begin{vmatrix} \mathbf{V}_\eta[\mathrm{I}] & \mathbf{V}_\xi[\mathrm{I}] \\ \mathbf{V}_\eta[\mathrm{J}] & \mathbf{V}_\xi[\mathrm{J}] \end{vmatrix}} \quad (78)$$

11. Compute the new trial area coordinate **A**

$$\mathbf{A}[\eta] = \mathbf{A}[\eta] + d_\eta \quad (79)$$

$$\mathbf{A}[\xi] = \mathbf{A}[\xi] + d_\xi \quad (80)$$

$$\mathbf{A}[\psi] = 1 - \mathbf{A}[\eta] - \mathbf{A}[\xi] \quad (81)$$

12. Make sure **A** remains inside the patch. Use equation (66)

13. Evaluate $\mathbf{X}_A$ at **A** and set up for next iteration. Keep track of the best evaluated location so far.

$$\mathbf{X}_{last} = \mathbf{X}_A \tag{82}$$

$$\mathbf{X}_A = \Phi(\mathbf{A}) \tag{83}$$

$$\mathbf{V}_{move} = \mathbf{X}_A - \mathbf{X}_{last}, d_{move} = \|\mathbf{V}_{move}\| \tag{84}$$

$$d_X = \|\mathbf{X} - \mathbf{X}_A\| \tag{85}$$

IF ($d_X < d_{best}$) THEN

$$d_{best} = d_X, \mathbf{A}_{best} = \mathbf{A}, \mathbf{X}_{best} = \mathbf{X}_A \tag{86}$$

14. Increment the iteration count and return to beginning of loop.

    niter = *niter*+1

    GOTO 5

5.3 Performance

In order to get a rough idea of performance for the closest_point implementation, models of varying size were tested and their results described in Table 1. Test models are shown in Figure 10 to Figure 12.

Figure 10. Cylinder Model (48 facets)

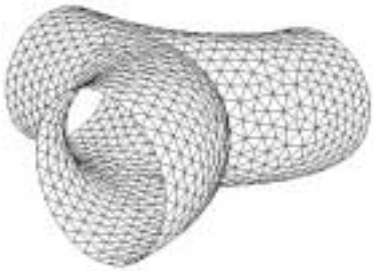


Figure 11. Diverging Corners Model (1622 facets)

Figure 12. FEA Surface Model (15,599 facets)

	number	without R-Tree Search		with R-Tree Search	
model	of facets	linear	Bezier	linear	Bezier
cylinder	48	977	792	492	443
diverging corners	1612	588	467	414	397
FEA surface	15599	88	103	357	273

Table 1. Comparison of Performance of closest point performance. (evaluations per CPU second).

For these cases, the closest_point routine was instrumented to accumulate CPU time only while in that function while meshing the surfaces shown. The total CPU time taken in the closest_point routine dvided by the number of calls to the routine is reported in Table 1. The table reports cases both with and without the use of the R-Tree spatial searching as described in the previous section. Results illustrate severe deterioration in performance for surfaces with many facets when no spatial searching is used ($O(n^2)$). With R-Tree, the performance deteriorates, but at a much more acceptable rate ($O(n\log n)$). The results also illustrate that for surfaces containing fewer facets, the overhead of using the R-Tree was more time consuming than was warranted. As a result, the R-Tree is only used where the number of facets exceeds 2000.

Also illustrated in Table 1 is the comparison of using only the linear facet representation vs. using the Bézier combined with the minimization scheme described in the previous section. Although there is some performance hit when using the Bezier representation, the overall slow-down is not significant.

In order to put these results into perspective with a standard NURBS-based geometry kernel, the ACIS geometry kernel was instrumented and a similar test performed by meshing a typical CAD model. In this case, the time reported was only that required by the ACIS third party library to return the requested closest point on the surface. The average number of evaluations per CPU second was 990. This is comparable to the linear facet representation of the cylinder model of Figure 10.

It is clear from these results that a significant amount of time is taken to locate the appropriate facet(s) to evaluate. As the number of facets increase, the performance can deteriorate substantially. This justifies the use of geometry decomposition algorithms to break the surface into smaller, logical topological surfaces. It also justified the further improvement of efficient spatial searching techniques.

6. EXAMPLES

To illustrate the viability of the proposed facet-based surface representation a limited number of test cases are reported here. The first example illustrates the overall process used to define the facet representation. Figure 13 shows the initial facetted representation. A feature angle of 100 degrees was used to ensure three distinct surfaces would be generated. Once the appropriate surface topology is computed, the appropriate normals at the facet vertices

are computed (Figure 13(a)). Note that multiple normals may be computed at a single vertex..

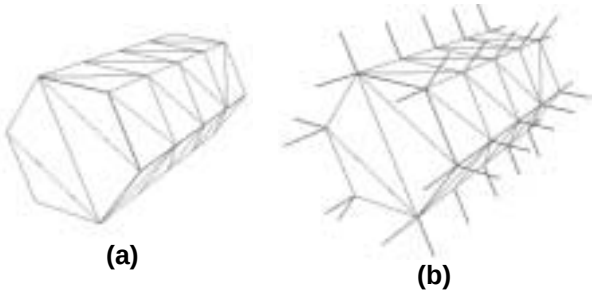


Figure 13. (a) Coarse faceting of a cylinder. (b) Computed normals at facet vertices.

Once normals are computed, the control points are computed on each FacetEdge and Facet. Figure 14 (a) shows the control point grids for each of the facets. The normal information can also be computed at this time as show in Figure 14 (b). Otherwise, the normals may be computed only as needed.

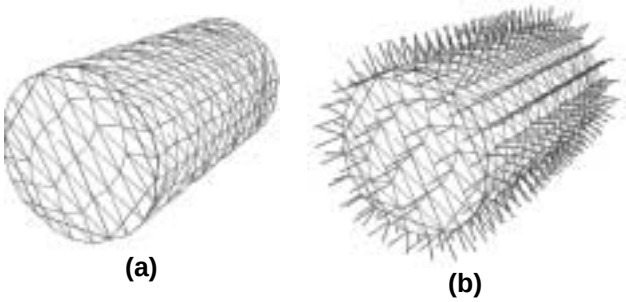


Figure 14. (a) Bezier control point grid from Figure 13. (b) Computed normals at the control points

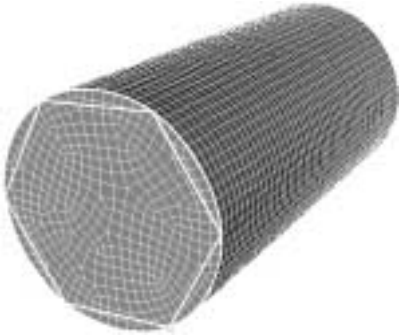


Figure 15. Facetted model meshed using paving, mapping and sweeping mesh generation algorithms

Having set up the facetted surface representation, the model is ready to be meshed. Figure 15 shows an example mesh generated on the geometry. In this case, the paving [2] algorithm is used to mesh the cylinder caps and the sub-mapping algorithm is used to mesh the sides. The interior hexahedral mesh is generated using the sweeping algorithm. Each of these mesh generation algorithms utilize the facetted surface described in this work.

It should be noted that the resulting node locations on the cylindrical surface described by the facetted surface representation are within tolerance of reproducing the shape of an exact cylinder.

The next example (Figure 16) is a contrived model intended to illustrate the use of the facet-based surface representation within an H-adaptive environment. In this case, facets representing two concentric spheres are defined. The initial mesh is defined by only quadrilateral elements, which have been split to form facets. The FEA analysis drives the adaptivity so that the quadrilateral elements are uniformly refined. In this example 3 iterations were performed resulting in 64 quadrilaterals for every initial quad on the surface. New nodes were projected to the surface defined by the Bézier facetted-surface representation. The result shown in Figure 16 (b), shows the resulting refined surface mesh. Once again, the resulting node locations of the refined model generated only from the smooth facet representation are within tolerance of an exact sphere.

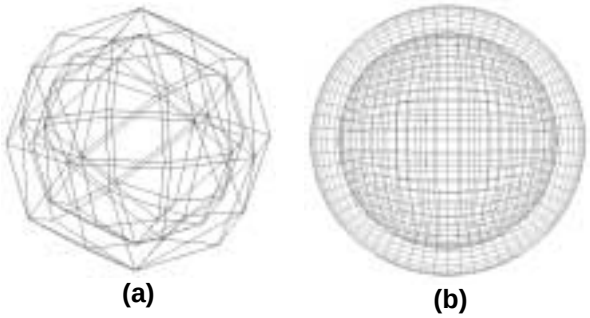
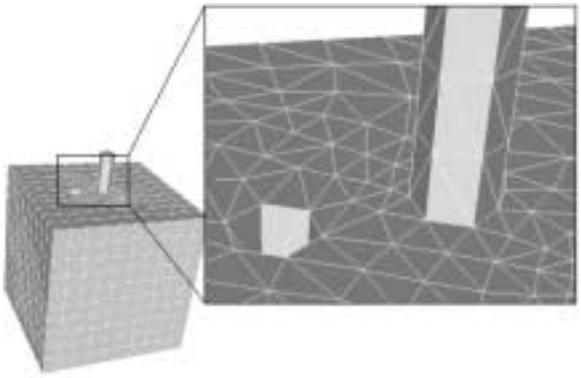


Figure 16. (a) Coarse faceting of concentric spheres (b) FEA mesh after 3 iterations of H-Adaptivity

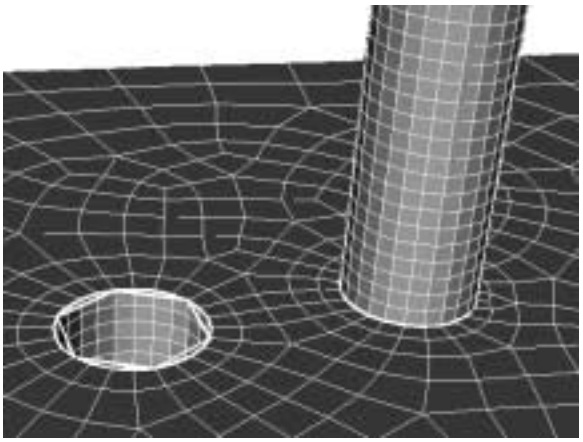
Figure 17. shows an example of the use of the proposed facet-based surface representation to improve the resolution of small features within an existing FEA model. Figure 17 (a) shows the initial facetted representation of the model. In this case, the model has small features that are not sufficiently resolved for purposes of the required analysis. After setting appropriate mesh sizing constraints on the model, the surfaces are remeshed using the paving algorithm with results shown in Figure 17 (b). Note the improved resolution of the interface curves between adjoining surfaces. The bold lines illustrate the difference between the initial linear representation and the final Bézier representation of the curves.

One final example (Figure 18) illustrates the use of the facet-based surface representation to convert a model meshed with triangles into a different element type or to modify its resolution by remeshing. In some cases, when only the FEA mesh is available, to perform design studies, it may be required to change the element type or mesh resolution. This can be troublesome when there is no underlying geometry representation. In this example, the triangle surface mesh Figure 18 (a) is used as input to the

facet-based surface representation and Bézier patches are generated. The paving algorithm is once again used to mesh over the original triangle surface mesh. Figure 18 (b) and Figure 18 (c) show the triangle surface mesh remeshed with different resolutions of a quadrilateral mesh. In (b) the mesh is coarser than the original mesh and (c), the mesh is finer.



(a)

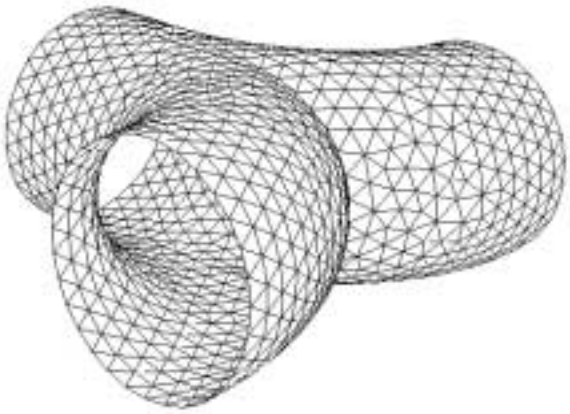


(b)

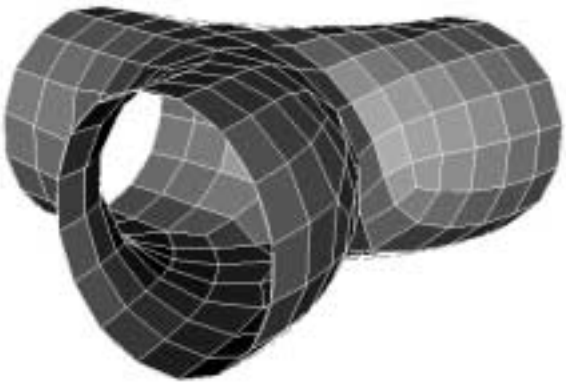
Figure 17. (a) Facet-based model. Enlarged section shows small features insufficiently represented by the facets. (b) Paving algorithm applied to the facet-based surface improves resolution.

7. CONCLUSION

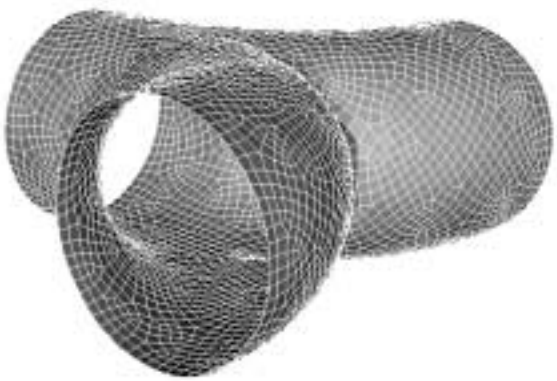
Facet-based surface representations have become an important tool for simulation. Existing mesh generation tools can utilize facet-based surfaces in a similar manner to NURBS-based surfaces through a common API. We have proposed a variety of technologies necessary to successfully implement curve and surface representations specifically for use within an existing mesh generation toolkit.



(a)



(b)



(c)

Figure 18. (a) Initial triangle mesh. (b) remeshed using Paving. (c) at finer resolution

An object oriented facet representation has been proposed that permits flexibility and reuse within different applications. Both G^0 and G^1 facet representations have been presented, allowing the user to trade off between accuracy and efficiency of the resulting mesh. The Walton patch was proposed as the basis for the G^1 facet

representation, and its implementation within the context of a geometry kernel for 3D mesh generation was presented.

Additionally, techniques for evaluation of the facet representation during the mesh process were described. It was found that in cases with small numbers of facets, that the NURBS-based and facet-based representations were equivalent. However, for significant numbers of facets, the performance of the facet-based representation was less efficient. A spatial searching technique, R-Tree, was used to improve performance for significant numbers of facets.

The proposed research has been implemented within the context of the Cubit mesh generation software toolkit [1] and is currently utilized by a wide variety of users at the U. S. National Laboratories.

REFERENCES

[1] CUBIT Mesh Generation Tool Suite: Automatic Unstructured Hex, Tet Quad and Tri Meshing and Solid Model Geometry Preparation. Web Site: http://endo.sandia.gov/cubit (2002)

[2] Blacker, Ted D., "Paving: A New Approach To Automated Quadrilateral Mesh Generation", *International Journal For Numerical Methods in Engineering*, No. 32, pp.811-847, (1991)

[3] Roger J Cass, Steven E. Benzley, Ray J. Meyers and Ted D. Blacker. "Generalized 3-D Paving: An Automated Quadrilateral Surface Mesh Generation Algorithm", *International Journal for Numerical Methods in Engineering*, No. 39, 1475-1489 (1996)

[4] Knupp, Patrick M., "Next-Generation Sweep Tool: A Method For Generating All-Hex Meshes On Two-And-One-Half Dimensional Geometries", *Proceedings, 7th International Meshing Roundtable*, pp.505-513, October 1998

[5] White, David R. and Timothy J. Tautges, "Automatic scheme selection for toolkit hex meshing", *International Journal for Numerical Methods in Engineering*, Vol 49, No. 1, pp.127-144, (2000)

[6] Borden, Michael, Steven Benzley, Scott A. Mitchell, David R. White and Ray Meyers, "The Cleave and Fill Tool: An All-Hexahedral Refinement Algorithm for Swept Meshes", *Proceedings, 9th International Meshing Roundtable*, pp.69-76, October 2000

[7] "The Common Geometry Module (CGM): A Generic, Extensible Geometry Interface", *Proceedings, 9th International Meshing Roundtable*, Sandia National Laboratories, pp.337-348, October 2000

[8] 3D ACIS Modeler, Spatial Technologies, Web Site: http://www.spatial.com/products/3D/modeling/ACIS.html, (2002)

[9] Daniel Rypl and Zden ěk Bittnar, "Discretizatoion of 3D Surfaces Reconstructed by Interpolation Subdivision," *7th Conference on Numerical Grid Generation in Computational Field Simulations*, pp. 679-688 (2000)

[10] Pascal J. Frey and Paul-Louis George, "Mesh Generation: Application to Finite Elements," Hermes Science Publishing, Chapter 13.4, pp. 445-448 (2000)

[11] Pascal J. Frey, "About Surface Remeshing", *Proceedings, 9th International Meshing Roundtable*, pp. 123-126 (2000)

[12] P. Laug and H. Borouchaki, "Adaptive Parametric Surface Meshing Based on Discrete Derivatives," *7th Conference on Numerical Grid Generation in Computational Field Simulations*, pp. 719-728 (2000)

[13] Steven J. Owen and David R. White, "Mesh-based geometry: A systematic approach to constructing geometry from a finite element mesh", *Proceedings, 10th International Meshing Roundtable*, pp. 83-96 (2001)

[14] Tristano, Joseph R., Steven J. Owen, and Scott A. Canann**,** "Advancing Front Surface Mesh Generation in Parametric Space Using a Riemannian Surface Definition", *7th International Meshing Roundtable*, pp.429-445, October 1998

[15] George P. L., and H. Borouchaki, *Delaunay Triangulation and Meshing Application to Finite Elements*, Editions HERMES, Paris, 1998

[16] Cuilière, J. C., An adaptive method for the automatic triangulation of 3D parametric surfaces, *Computer Aided Design*, Vol 30(2), pp.139-149, 1998

[17] Sheffer, Alla and E. de Struler, "Surface Parameterization For Meshing by Triangulation Flattening", *Proceedings, 9th International Meshing Roundtable*, pp.161-172, October 2000

[18] Michael S. Floater, "Parametrization and Smooth Approximation of Surface Triangulations," *Computer Aided Geometric Design*, Vol 14, pp.231-250, 1997

[19] Lau, T.S. and S.H. Lo, "Finite Element Mesh Generation Over Analytical Surfaces", *Computers and Structures*, Elsevier Science Ltd, Vol 59, No. 2, pp.301-309, 1996

[20] Gerald Farin, "Curves and Surfaces for CAGD", 5th Ed. Academic Press (2001)

[21] Praveen Kashyap, "Geomeric interpretation of continuity over triangular domains," *Computer Aided Geometric Design* No. 15 pp. 773-786 (1998)

[22] B. Piper, "Visually Smooth Interpolation with Triangular Bezier Patches", in Farin, G. (Ed.) Geometric Modeling: Algorithms and New Trends SIAM Philadelphia (1987) pp 221-233

[23] D. J. Walton and D.S. Meek, "A triangular G^1 patch from boundary curves", *Computer-Aided Design*, Vol 28, No. 2, pp. 113-123 (1996)

[24] J. L. Bentley and J. H. Friedman, “Data structures for Range Searching,” *Computing Surveys* No. 11, 4 pp. 397-409 (1979)

[25] Antonin Guttman, “R-Trees, A Dynamic Index structure for spatial Searching,” SIGMOD Conference 1984. pp. 47-57

[26] J. L. Bentley, “Multidimensional Binary Search Trees used for Associative Searching,” *Communications of the ACM* 18, 9 pp. 509-517 (1975)

[27] Willian H. Press et. al., “Numerical Recipes in C: The Art of Scientific Computing,” Cambridge University Press, Chapter 10. pp. 394-455 (1995)

[28] Mihailo Ristic and Djordje Brujic, “Efficient registration of NURBS geometry,” Image and Vision Computing, No 15 pp. 925-935 (1997)