SURFACE MESH GENERATION ON VOXEL-BASED OBJECTS

Andrei V. Smirnov[1] Hanzhou Zhang[2]

Department of Mechanical and Aerospace Engineering
West Virginia University, Morgantown, WV, U.S.A.
[1] *asmirnov@wvu.edu* [2] *hzhang@mix.wvu.edu*

ABSTRACT

The problem of constructing surface meshes for the objects given on a set of discrete points or grid-nodes is addressed. A heuristic algorithm for generating triangulated surface mesh is proposed. The resultant meshes do not inherit the anisotropy of the underlying hexagonal grid but preserve the essential geometric features of the surface. This meshing approach is suitable for creating grid-independent surface representations of objects given by discrete data sets, such as are usually obtained from the measurements or arise in voxel-based graphical systems. Meshing strategies, such as surface curvature estimates and neighbor-detection techniques are described. Examples of applications are given.

Keywords: mesh generation, voxel, volume graphics

1. INTRODUCTION

Mesh generation is a process of dividing a physical domain into a large number of small elements with relatively simple shapes. Typically the surface of an object needs to be meshed before the application of a 3D mesh generator, which often fills in the interior of the object with tetrahedral or hexahedral elements. The most common unstructured surface meshing algorithms, which divide a surface into triangles, include Delaunay meshing methods [1], and advancing front methods [2].

While vector graphics dominate most CAD packages, 3D surfaces in many engineering and scientific applcations are usually defined by means of parametric surfaces. Thus most surface meshing algorithms are designed to deal with such. There are essentially two types of surface meshing approaches: direct and indirect. With the indirect approach, the mesh is first generated in a 2D parametric space, and then mapped back to the actual surface in 3D space [3, 4, 5, 6]. In contrast, the direct approach generates the mesh directly over the original surface in 3D space [2, 7]. With this approach surface parameteriztion is not required.

The advancement in hardware, especially cheaper and larger memories, has brought to the attention the technique of so-called voxel-based volume graphics [8]. A voxel is a volume element which is the 3D counterpart of a 2D pixel. Many geometric objects in engineering applications can be modeled with voxels. Also a great number of biomedical, geophysical and other scientific data is typically given as a collection of grid points, which can be classified as voxel-data. In all these cases no parametric surfaces are available. Although many mesh generation methods exist, few of them are designed to deal with voxel-based geometric objects [9].

In this study, a 3d surface meshing method is proposed to mesh the surface of voxel-based objects. Since no parametric surfaces exist for an voxel-based object, the algorithm proceeds directly with voxels, and in the end the surface is extracted as a triangulated mesh. Therefore, besides the purpose of generating quality mesh for numerical methods, the meshing method also serves another purpose: volume rendering, which is the objective of many surface construction algorithms, such as marching cubes algorithm [10].

2. METHOD

The main targets of the meshing method are voxel-based objects, which can be created by some simple volume graphics tools or obtained by measurements. Each voxel has a value which represents some property, such as color, texture, or material. These voxel-based objects can be rendered in different modes, such as cubes, wireframes or points.

The voxels are arranged in a regular 3D grid. For each voxel, its neighbors can be esily reached by incrementing or decrementing the respective direction indexes. Thus, the connectivity is given by one simple relation, and global neighbor connectivity lists are not required. The boundaries of an object can always be established using this neighbor information.

Each object has an id-number, which is assigned to every voxel representing that object. The meshing routine receives the id-number of the object as an input parameter and starts by scanning the voxel set for the first occurrence of this id. If such voxel was found the routine invokes the surface meshing algorithm, which creates the boundary mesh for that object. The routine continues then to search for voxels of the same type.

The *voxel grid* itself is represented by a 3D array of integer numbers, which can be 32 or 16 bit long depending on the required number of objects and memory constraints. *Surface mesh* is a linked list of triangles, where each triangle is represented by three vertexes, each of which belongs to a set of *boundary nodes*. A *boundary node* inherits the coordinates of the corresponding boundary voxel. The three vertexes of each triangle are oriented in a way such that the normal of the triangle area points outward the object volume.

When the mesh is being constructed the algorithm operates on nodes and edges, which can be of 2 types: *open* or *closed*. An open edge is an edge that belongs to only one surface triangle. An *open node* is a node, which is not completely surrounded by triangles. *Open edges* and *open nodes* are simply edges or nodes which are at the front of the newly constructed surface mesh, that is, the part which is currently under construction (Fig.1). We call the front-edges *open* because each of them belongs to only one triangle, whereas every edge inside the mesh belongs to two triangles. Likewise, every node at the front (*open node*) belongs to at least two open edges, whereas each node inside the mesh doesn't belong to any open edges. When the mesh is completely done all the nodes and edges should become closed and the front reduces to a zero-set. The mesh front is essentially a one-dimensional segmented line. Each open node has two pointers pointing to 2 neighboring open nodes, designated as *prev* and *next*, The direction from the *prev* node to the *next* node

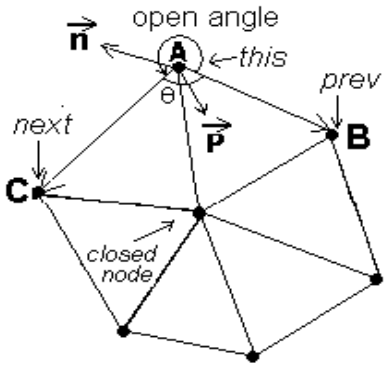


Figure 1: Node *this*, its *prev*, *next* nodes and open angle.

following the unmeshed area is in counter-clock wise direction with respect to the surface outward normal vector. The angle between node's two open edges on the unmeshed area side is called the *open angle*. Figure 1 shows an active node A, its *prev* node B and *next* node C, and its open angle. A closed node is also shown in this figure.

One step of the algorithm consists of closing a single node. For this purpose the algorithm selects a current node from the set of open nodes, which is called *this* node, and performs the *closure operation*. The selection of active nodes can be done in a preferential manner so as to provide for a more uniform angle distribution inside the surface triangles.

The very first node of the mesh selected on the boundary has its *prev* and *next* pointers point to itself, and its open angle is set to 2π[1]. When a node is closed, its open angle is 0. The set of open nodes is stored as a linked list. Each closure operation of the node may lead to the appearance of other open nodes, which are newly introduced nodes of the mesh. The meshing algorithm repeats the closure operation on all open nodes until the list becomes empty.

The meshing algorithm can be summarized as the following:

1. Construct the first triangle, and add its three nodes to the linked list;
2. Select the node from the list with the smallest open angle. Call it *this* node;
3. Recursively create surrounding triangles until *this* node is closed; In this procedure use either

[1] Strictly speaking, it can be less than 2π depending on the curvature of the surface at that point.

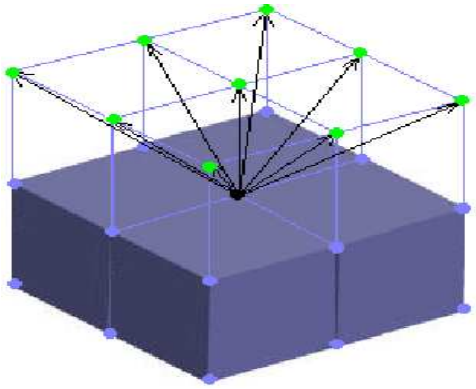


Figure 2: Estimating surface normal of a boundary point

existing open nodes for new triangle vertexes or introduce new open nodes;

4. When *this* node is closed, remove it from the list, and consider if its neighbor nodes should be removed as well.

5. If there are open nodes in the list repeat from step 2.

The essential procedures performed by the algorithm are the following:

Construction of a surface normal. Surface normal vector is needed to determine the node-closure direction. Given a rigid character of the surface in a voxel-based representation when looked at in the vicinity of only few grid-nodes, one needs to perform some averaging to get a fair estimate of the surface normal direction at each voxel. A quick way to do this estimate is to account of all the neighboring voxels, which don't belong to the current object (outside voxels). For each such voxel the distance vector, $\vec{P}_i$, between *this* node and the neighbor node, i, is calculated. The sum of these vectors will approximately point in the surface normal direction. Thus, the normal vector $\vec{n}$ is estimated using the following equation (see also Figure 2):

$$\vec{n} = \frac{\sum_{i=1}^{k} \vec{P}_i}{|\sum_{i=1}^{k} \vec{P}_i|} \tag{1}$$

Open angle estimates. The open angle estimation procedure is illustrated in Figure 1. First the dot product of the two vectors $\vec{AB}$ and $\vec{AC}$ is computed to find the angle θ between them. Then the actual open angle could be this angle or its complimentary of 2π. To decide on this we need extra information, which comes from the normal vector $\vec{n}$ and the cross product $\vec{P}$ of vectors $\vec{AB}$ and $\vec{AC}$. If $\vec{n}$ and $\vec{P}$ are on the same side of the surface, i.e. their scalar product is positive, then the open angle is θ, otherwise the open angle is $2\pi - \theta$. Thus, we have:

$$\theta = \arccos\left(\frac{\vec{AB} \cdot \vec{AC}}{|\vec{AB}||\vec{AC}|}\right) \tag{2}$$

$$\vec{P} = \vec{AB} \times \vec{AC} \tag{3}$$

$$if(\vec{n} \cdot \vec{P} < 0) \Rightarrow \theta = 2\pi - \theta \tag{4}$$

Quality of a triangle. The quality of triangles is important for the accuracy of numerical methods. Generally speaking, having small angles is bad. One measurement for the quality of a triangle is called aspect ratio, which is defined as the value of the longest edge divided by the smallest height. We know the smaller the aspect ratio, the better the quality. And the best triangle is a equilateral triangle.

Construction of a triangle. To construct a new triangle for an open node, the following procedure is used (Fig. 3). We consider that at this point an open node was selected as *this* node:

1. Check nearby open nodes to *this* node, and see if there is one suitable for constructing a triangle, consisting of that node, *this* node and *prev* node, or consisting that node, *next* node and *this* node. A suitable node is a node which will lead to a triangle with good quality, e.g, from the respective of aspect ratio. If such node is found:

   - Then construct such triangle and call the newly found node the *old* node. Check if *this*, *old* node, and *prev* (or *next*) nodes are now closed. Remove any closed node from the open node list.

2. Otherwise:

   - create a *new* open node by searching an appropriate boundary voxel, such that the distance to that voxel is consistent with the current mesh length-scale, and the angle between *this* $\rightarrow$ *prev* and *this* $\rightarrow$ *new* directions is around 60^o (see below).

3. Change *prev* and/or *next* pointers of any involved node.

As can be seen this algorithm takes care of the *meeting front* problem, when the open nodes from different parts of the front begin to converge on each other.

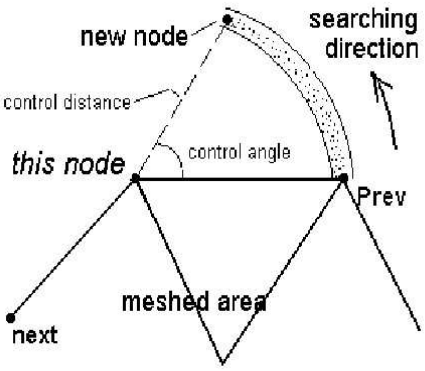


Figure 3: Constructing a new triangle

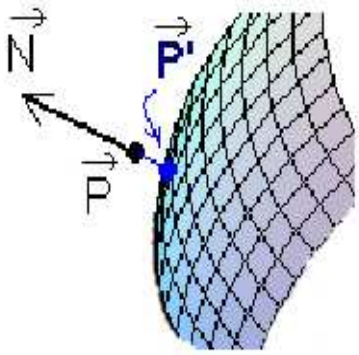


Figure 4: Locating the closest point.

That's why the existing nodes are tried first before creating *new* nodes in step 1 instead of generating new nodes to form a triangle.

Generating a new node. A new node is to be generated from a boundary voxel, inheriting its coordinates. Two parameters are used in searching for a suitable boundary voxel. One is a control angle. We try to make it close to 60 degrees by dividing the open angle by an integer:

$$control\_angle = \frac{open\_angle}{round(open\_angle/60.0)} \quad (5)$$

Another parameter is a control distance, which is related to local mesh length-scale (see below).

Figure 3 illustrates the searching process. Starting from the *prev* node, the searching process walks through the boundary voxels from neighbor to neighbor in counter clock wise direction keeping the distance from *this* node in a narrow strip, until a point with the right angle is found.

Mesh length-scale control The local mesh length-scale is selected on the basis of two control sizes: one is a global size, set by the user and serving as an upper bound of mesh-size; and another is local size, related to the local surface curvature. Highly curved surfaces should use triangles of smaller sizes. Therefore, to achieve the goal of local size control, surface curvature evaluation is necessary.

To find the local curvature one can first fit a surface patch with a set of local points around the point of interest, and then estimate curvature on the fitted surface. Many different surfaces can be used for this purpose. In the current implementation a quadric surface patch fit with least square method is used (Appendix A).

After the quadratic surface fit was found the closest point on that surface patch from the point of interest is identified (since the fitted patch does not have to pass though that point) and the curvature at that point is used to approximate the curvature at the point of interest. The closest point is assumed to lie on the intersection between the boundary normal at the point of interest computed earlier and the fitted surface (Fig.4).

If $\vec{P}$ is the coordinate vector to the point of interest and $\vec{N}$ is the boundary normal vector at that point then the intersection point $\vec{P'}$ can be represented as: $\vec{P'} = \vec{P} + k\vec{N}$, where k can be obtained by substituting the coordinates of $\vec{P'}$ into the fitted surface equation. This will give a quadratic equation, which is easy to solve.

Curvature evaluation. The curvature evaluation method used for 3D implicit quadric surface is shown in Appendix B.

3. DISCUSSION

The surface meshing method described here uses an advancing front technique, which is similar to traditional advancing front methods [2]. However in our method the front does not start from the boundary of a surface. Instead it can start from a random point. Mesh quality might be improved if the front starts from corners and edges of the object volume, but in that case corner detection and edge detection need to be addressed before the meshing process. Another difference in the proposed method is that it relies on the node closure routine introduced in the previous section. This routine constitutes a single step of the al-

gorithm, which is repeated until no open nodes are left in the mesh. A conventional approach to front propagating uses *edge closure* rather than node closure, in which for each open edge a new mesh triangle is constructed with this edge making one side of the triangle (Fig.5).

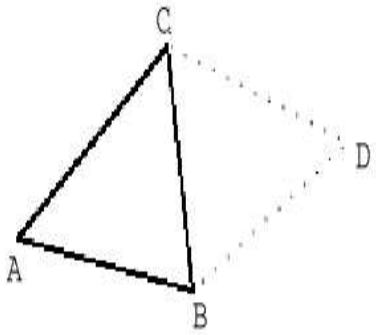
Comparing the two approaches one may note that the node closure algorithm is more complex in implementation than the edge closure one. However, the former has a potential for a better control of the angles, which may lead to a higher uniformity of the mesh. This is because the open angles for each node at the front are constantly computed and sub-divided into the appropriate number of sub-angles. This means that the node-closure algorithm will require less post-processing steps of mesh quality improvements.

The node-closure algorithm has also a wider range of *visibility* when it looks out for other front nodes as potential candidates to be joined with a current node. This is because the lookout radius for each node is about the size of the local edge-length of the mesh, whereas the lookout radius for the edge in the edge-closure case is usually smaller (Fig.5). It is still possible to employ the same strategy in the edge-closure case of using a larger lookout radius for both nodes of a given edge. However, this will lead to a considerable increase of the processing time, since each node will be processed as many times as there are edges connected to it, which may be on the order of 4 to 6 times more for surface meshes.

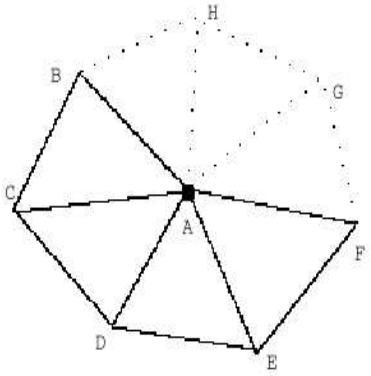
Even though each step of the node-closure algorithm takes more operations to accomplish, the overall efficiency may be comparable to the edge closure procedure, since several new mesh triangles can be generated during one step of the former, while the latter only leads to the creation of one new triangle at a time. This is a consequence of the fact that there are usually several times more edges than nodes in a typical mesh.

The advantage of using node-cloure operation as compared to an edge-closure operation used in a conventional advancing front method is that on the average it spans a larger area while examining a neighborhood of the node as compared to the edge. Indeed, to close an edge one needs to create a single node that will form a triangle with other two nodes on the edge. To close a node one needs to create more than one new nodes, therefore there is a wider range of possibilities and in fact an optimization can be used to select the new nodes. There is also a larger area examined in the process of selection of several nodes compared to just one. This all amounts to an improved distribution of nodes with respect to the selected mesh optimality criteria (aspect ratios, angles, etc.)

Speaking about the accuracy of surface representa-



(a) Edge closure: new triangle CBD is to be constructed to close edge CB



(b) Node closure: new triangles AFG, AGH, and AHB are to be constructed to close node A

Figure 5: Edge vs node closure approaches to front propagation.

Exp.	*Grid*	N	α_{max}	α_{min}	r_{max}
1	$70 \times 70 \times 70$	1158	114^o	24^o	3.35
2	$100 \times 100 \times 60$	1342	112^o	25^o	3.21
3	$40 \times 60 \times 100$	1880	113^o	20^o	3.08

Table 1: Characteristics of example meshes.

tions, the mesh generated in the present approach provides the resolution of surface details down to 4 to 6 voxels in size. This is because the node closure operation requires a certain minimum separation of the open nodes to provide a more even angle distribution and to avoid inheriting the hexagonal unisotropy of the voxel grid. Although this property sets a limit on the resolution of surface details, such as sharp angles, it provides for the grid independence of the mesh.

4. RESULT

Figures 6, 7 and 8 show surface meshing examples of spheres, joint spheres and other composite objects. The voxel grid size, the number of tiangles, the maximum/minimum angle, and the maximum aspect ratio r are shown in Table 1. The global mesh length-scale was chosen to be 6.

The quality of triangles are affected by the resolution of the object and the selected global length scale. Since all coordinates of voxels are integer values, during the procedure of searching a new node, we cannot make the angle exactly equal to the control angle. The smaller the control distance (or the global length scale), the larger the error from the control angle. Therefore if we have high resolution voxel grid for the object, we can then choose larger length scale, and thus make the mesh quality better.

5. CONCLUSIONS AND FUTHER WORK

The proposed mesh generation scheme belongs to a family of propagating front methods, but exploits the idea of using nodes and their triangle closure as basic steps in building the surface mesh. The extension of this idea to 3D meshing is relatively straightforward. In this case the node has to be closed by the envelop of tetrahedral elements filling in the spherical region around the node. The construction of these elements can be done using a 2D algorithm described here for meshing the surface of the sphere, with subsequent connections of each newly generated node to the sphere-central node. Implementation of this idea will be the logical continuation of this work.

The method suggested here works best on smooth surfaces of objects, since it has a minimum resolution

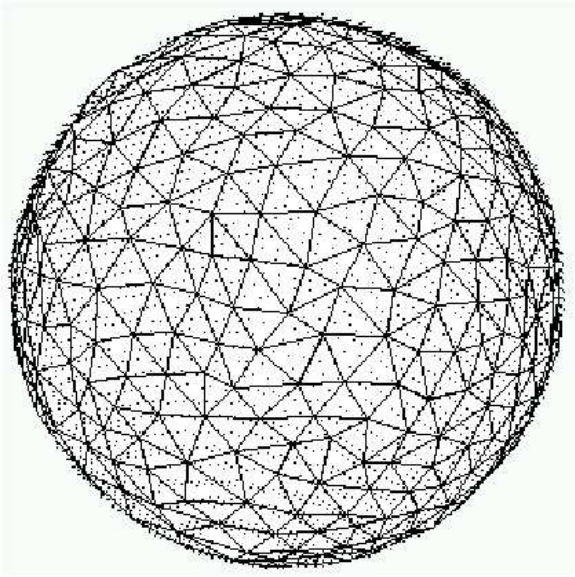


Figure 6: Example 1: Surface mesh on a sphere

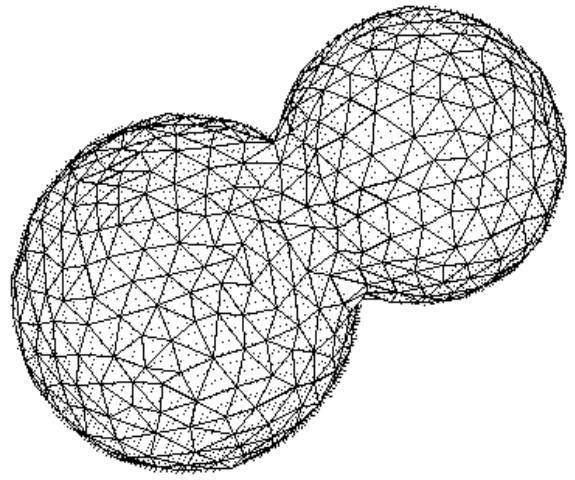


Figure 7: Example 2: Surface mesh on two overlaying spheres

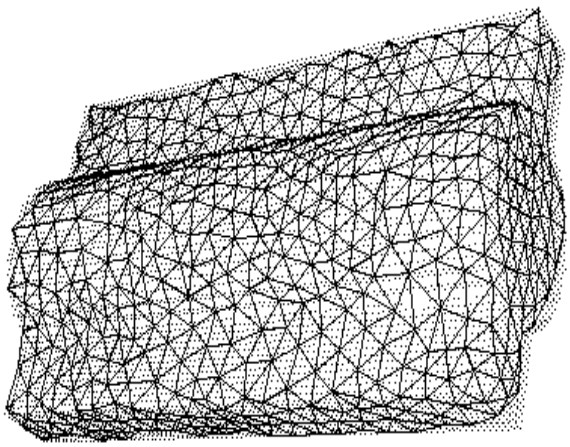


Figure 8: Example 3: Surface mesh on overlaying cylinder and cube

limit of several voxel sizes. Thus the method may not preserve the sharp corners and edges. To correct this situation some improvements in curvature evaluation algorithms may be needed, as well as edge detection methods.

The method can be used to create grid-independent surface representations of data given on discrete set of points, and in voxel-based sculpting systems.

A. LEAST SQUARE QUADRIC SURFACE FITTING

One approach to fit a set of 3d points is least-squares surface fitting with a quadric function:

$$f(x,y,z) = CV^T \tag{6}$$

where C is a vector of coefficients:

$$C = (c0, c1, c2, c3, c4, c5, c6, c7, c8, c9) \tag{7}$$

and

$$V = (x^2, y^2, z^2, xy, yz, xz, x, y, z, 1) \tag{8}$$

We need at least 10 points to find the ten unknown coefficients. Suppose the number of points N is greater than 10 points, generally they cannot be on the same quadric surface. To fit the best surface, we can attempt to choose vector C to minimize the error function:

$$E(C) = CMC^T \tag{9}$$

$$M = D^T D \tag{10}$$

where D is a Nx10 matrix whose ith row is:

$$(x_i^2, y_i^2, z_i^2, x_iy_i, y_iz_i, x_iz_i, x_i, y_i, z_i, 1), i = 1, 2...N \tag{11}$$

When all coefficients are 0 the error function has the minimum value zero. Therefore there is a degree of freedom in this function. To get rid of it, we can choose C to be a unit-length vector as a constraint. Now this problem is an eigenvalue problem. The minimum error will be the smallest eigenvalue of M. And the eigenvector corresponding to the smallest eigenvalue is the coefficient vector C.

In the above construction, none of the sample points is guaranteed to be on the fitted surface.

B. CURVATURE EVALUATION FOR AN IMPLICIT QUADRIC SURFACE

Let $F(x,y,z) = ax^2 + by^2 + cz^2 + exy + fyz + gxz + lx + my + nz + d = 0$ represent an implicit quadric surface in R3. F_x, F_y and F_z are the first order partial derivatives of F. F_{xx}, F_{xy}, ..., F_{zz} are the second order partial derivatives of F. The following equations are used to calculate the curvature of surface F(x, y, z) [11]:

$$|\Delta F| = \sqrt{F_x^2 + F_y^2 + F_z^2} \tag{12}$$

$$L = \frac{1}{F_z^2|\Delta F|} \begin{vmatrix} F_{xx} & F_{xz} & F_x \\ F_{zx} & F_{zz} & F_z \\ F_x & F_z & 0 \end{vmatrix} \tag{13}$$

$$M = \frac{1}{F_z^2|\Delta F|} \begin{vmatrix} F_{xy} & F_{yz} & F_y \\ F_{zx} & F_{zz} & F_z \\ F_x & F_z & 0 \end{vmatrix} \tag{14}$$

$$N = \frac{1}{F_z^2|\Delta F|} \begin{vmatrix} F_{yy} & F_{yz} & F_y \\ F_{zy} & F_{zz} & F_z \\ F_y & F_z & 0 \end{vmatrix} \tag{15}$$

$$E = 1 + \frac{F_x^2}{F_z^2} \tag{16}$$

$$H = \frac{F_x F_y}{F_z^2} \tag{17}$$

$$G = 1 + \frac{F_y^2}{F_z^2} \tag{18}$$

$$A = \begin{bmatrix} L & M \\ M & N \end{bmatrix} \tag{19}$$

$$B = \begin{bmatrix} E & H \\ H & G \end{bmatrix} \tag{20}$$

The engenvalues of $B^{-1}A$ are the principle curvatures $k1$ and $k2$. The values of the Gaussian curvature K and the mean curvature H are then given by:

$$K = k1 \cdot k2 \tag{21}$$

$$H = \frac{1}{2}(k1 + k2) \tag{22}$$

References

[1] Baker T.J. "Automatic Mesh Generation for Complex Three-Dimensional Regions Using a Constrained Delaunay Triangulation." *Engineering with Computers*, vol. 5, 161–175, 1989

[2] Lau T., Lo S. "Finite Element Mesh Generation Over Analytical Surfaces." *Computers and Structures*, vol. 59, no. 2, 301–309, 1996

[3] Chen H., Bishop J. "Delaunay Triangulation for Curved Surfaces." *6th Interlational Meshing Roundtable*, pp. 115–127. 1997

[4] Cuilliere J.C. "An adaptive method for the automatic triangulation of 3D parametric surfaces." *Computer-Aided Design*, vol. 30, no. 2, 139–149, 1998

[5] Tristano J.R., Owen S.J., Canann S.A. "Advancing Front Surface Mesh Generation in Parametric Space Using a Riemannian Surface Definition." *7th Interlational Meshing Roundtable*. 1998

[6] Borouchaki H., Laug P., George P.L. "Parametric surface meshing using a combined advancing-front generalized Delaunay approach." *International Journal for Numerical Methods in Engineering*, vol. 49, 233–259, 2000

[7] E. H. "A marching method for the triangulation of surfaces." *The Visual Computer*, vol. 14, 95–108, 1998

[8] Kaufman A., Cohen D., Yagel R. "Volume Graphics." *Computer*, vol. 26, no. 7, 51–64, 1993

[9] Frey P., Sarter B., Gautherie M. "Fully Automatic Mesh Generation for 3-D Domains Based Upon Voxel Sets." *International Journal for Numerical Methods in Engineering*, vol. 37, 2735–2753, 1994

[10] Lorensen W., Cline H. ""Marching Cubes: a high resolution 3D surface reconstruction algorithm." *Computer Graphics (Proceedings of SIGGRAPH '87)*, vol. 21, pp. 163–169. 1987

[11] E. H. "Blending an Implicit with a Parametric Surface." *Computer Aided Geometric Design*, vol. 12, no. 9, 825–835, 1995