Three-dimensional unstructured mesh generation: Part 3. Volume meshes

Roland W. Lewis a,*, Yao Zheng a, David T. Gethin b

a Department of Civil Engineering, University of Wales Swansea, Swansea, SA2 8PP, UK
b Department of Mechanical Engineering, University of Wales Swansea, Swansea, SA2 8PP, UK

Received 9 February 1995; revised 9 June 1995

Abstract

Volume meshes are generated through three-dimensional triangulation and interior point creation based on the surface meshes. To deal with non-convex geometries, boundary surface conformity is gained via edge swapping, boundary edge and surface recovery, and the robustness of the algorithm has been discussed in terms of the accuracy of geometric judgements. The performance of the mesh generator has been investigated by means of numerical experiments, and examples have been tested to validate the mesh generator. The possibility of extension of this mesh generator to incorporate with analysis programs, in terms of adaptive analysis, has been explored. This paper is also designed to briefly address mesh quality measures, quality statistics and mesh smoothing techniques. Using mesh quality metrics, visual quality assessment has been discussed. Furthermore, several general ways of visual representation of volume meshes have been introduced.

1. Introduction

Volume meshing is the main aim of the present work. Using Delaunay triangulation and the associated point creation (i.e. Steiner triangulation), a collection of tetrahedra are to be generated, in which all the boundary points, interior points and 8 extra points of the bounding box are included. This volume meshing method is based on the triangulation, point creation and surface meshing algorithms previously mentioned in [1, 2].

If the given surface is convex, then the corresponding mesh, resulting from deleting tetrahedra connected to the points of bounding box, conforms with the original geometric boundary. Unfortunately, for general cases, the resulting mesh containing all given points, does not contain edges or faces of the boundary, which are to be presented. Therefore, the boundary integrity has to be considered carefully. Furthermore, the performance of the mesh generator becomes of an importance as the scale of physical problems to be considered increases with the improvement of numerical methods. The performance of the mesh generator is to be investigated by means of numerical experiments, and examples are to be tested to validate the mesh generator. The possibility of extension of this mesh generator to incorporate with analysis programs, in terms of adaptive analysis, is to be explored.

*Corresponding author.
One of the main problems in tetrahedral mesh generation is how to generate well-shaped tetrahedral elements, since poorly-shaped elements may cause numerical difficulties. Currently, increasing attention is being paid to the quality of the resulting elements. There are various measures to deal with the quality of elements as reported in the literature. It is intended to briefly discuss several of these measures by means of numerical experiments to determine criteria for making informed choices among these metrics.

During mesh generation, interactive procedures may be used to change control parameters, in order to improve the resulting mesh. To this end, visual presentation is a helpful approach. Mesh visualization is receiving increasing attention as its importance in validating and presenting meshes has been realized.

2. Boundary integrity

Boundary integrity is widely recognized as a problem, as real applications demand the use of a mesh generator for complex shapes. Several approaches to its solution have been proposed [3–9]. Most of the procedures involve an addition of points to ‘block’ the penetration of tetrahedra through the boundary surface. In some algorithms, the additional points were connected to their surrounding points (e.g. [7]) by Delaunay triangulation. In some cases, it was reported that this approach adds an excessive number of points, and it was difficult to find an effective strategy which demonstrates that this approach always work [9].

An alternative approach is one, in which the extra points are added and connected by means of direct construction of tetrahedra after Delaunay triangulation [4, 9]. The implementation of the algorithm adopted here is related closely that proposed in [4, 6, 9]. The approach implemented in this research is based on the fact that a finite number of direct connections can be formulated for all types of tetrahedral penetration, therefore the recovery of boundary surface can be guaranteed. However, many intersection calculations are involved in this approach, the precision and geometric judgement will affect the reconstruction of tetrahedra. Therefore, attention to geometric tolerance must be included and this will be discussed in the next section.

If boundary faces are not present in the resulting triangulation, this is due to the fact that edges and faces of the tetrahedra intersect the required faces. It is a straightforward idea to recover the missing edges first, then to deal with missing faces. It was found that, an optional edge swapping procedure applied before these treatments is helpful as it has capability to reduce significantly the computational cost involved during the edge and face recovery.

2.1. Edge swapping

Edge swapping can be explained with reference to Fig. 1 (based on [9]). If in the tetrahedral construc-

![Diagram](image_url)

Fig. 1. Edge swapping in intermediate surface triangulation.
tion, faces $ACB$ and $BCD$ exist, but in the intermediate surface mesh, faces $ACD$ and $ADB$ exist, then the two can be made to agree, if edge swapping is applied to the surface mesh. However, there are some cases under which such a transformation is not allowed, these are:

(a) triangles $ACD$ and $ADB$ are not nearly coplanar;
(b) triangles $ACD$ and $ADB$ come from different surface patches.

The first is trivial to check using the normal vectors, while the second can be checked by looking up the associated surface flags.

2.2. Boundary edge recovery

The procedure to recover a resulting boundary edge involves two steps. Firstly, it is necessary to identify the faces, edges and points of tetrahedra with which the edge intersects. Then local transformations involving these tetrahedra are performed to present the edge.

With reference to Fig. 2 (based on [4]), assume that an edge joining points $A$ and $B$ is not contained in the tetrahedral construction. In this case, faces, edges or points of tetrahedra intersect with $AB$ and the edge is divided into segments by the intersecting points. All the tetrahedra intersecting with edge $AB$ form a shell associated with $AB$. All the possible cases of these intersections are depicted in Fig. 3 (based on [9]). Based on this classification of intersections, it is possible to consider all the possible intersections, then to perform direct transformations to recover the boundary edges.

The node-node case is easily dealt with when compared with the remaining five cases. For these five cases, tetrahedra splitting procedures are involved, which are illustrated in Fig. 4 (based on [9]). If the intersection involves a node-face type or face-node type, the edge $AB$ can be recovered directly by creating an edge $AB$, and converting two tetrahedra into three, provided that points $A$ and $B$ belong to two tetrahedra sharing a face (Fig. 5 based on [9]).

2.3. Boundary surface recovery

After the recovery of all boundary edges, the missing boundary faces can then be recovered. Although the edges of all boundary faces are present, it does not mean that boundary faces are present, because other tetrahedra can penetrate the interior of a face.

Consider a tetrahedron which penetrates a boundary face. There are, in total, four possibilities as shown in Fig. 6 (based on [9]). The corresponding transformation for the recovery of the missing faces are also shown in this figure, where $ABC$ is the boundary face under consideration, and points $p$, $q$, $r$ and $s$ denote the intersection points.

It should be mentioned that if one and only one edge of a tetrahedron intersects a boundary face and the edge is common to three tetrahedra, then the face can be recovered directly by deleting the edge (Fig. 7 based on [9]).

![Fig. 2. A shell consisting of the tetrahedra intersecting $AB$.](image-url)
2.4. Post treatment

From the preceding discuss it can be seen that most of the recovery transformations involve the creation of additional points, and the resulting tetrahedral constructions involve these points. However, the points formed from intersections are not necessarily on the real geometrical surface. For each of these points, the tetrahedra connected with it can be determined. After deleting these tetrahedra, an empty shell (polyhedron) will be generated. This empty shell is then triangulated directly by finding connections of points. Although this concept is straightforward it was relatively complex to implement.

Until this stage, the bounding box formed by the 8 extra points is occupied by tetrahedra completely. In order to get a valid mesh, the actual domain information needs to be considered and the extra tetrahedra are to be deleted. Then starting at one of these 8 extra points all tetrahedra can be visited through the neighbouring tree. The boundary surface in the form of an intermediate surface mesh will block some visit paths. During this search, tetrahedra are flagged indicating the domain information. To guarantee the geometric correctness of the physical domain information, it is important to get neighbouring information correct, since a part of the tetrahedra has been modified during the recovery stage.
<table>
<thead>
<tr>
<th>Transformations</th>
<th>Tetrahedra to be created</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Diagram" /></td>
<td>(1, 2, 3, p)</td>
</tr>
<tr>
<td></td>
<td>(1, 3, 4, p)</td>
</tr>
<tr>
<td></td>
<td>(1, 4, 2, p)</td>
</tr>
<tr>
<td></td>
<td>(1, 2, 3, p)</td>
</tr>
<tr>
<td></td>
<td>(1, 3, 4, p)</td>
</tr>
<tr>
<td></td>
<td>(3, 4, p, q)</td>
</tr>
<tr>
<td></td>
<td>(2, 3, p, q)</td>
</tr>
<tr>
<td></td>
<td>(1, 2, 3, p)</td>
</tr>
<tr>
<td></td>
<td>(p, q, 3, 4)</td>
</tr>
<tr>
<td></td>
<td>(q, p, 4, 2)</td>
</tr>
<tr>
<td></td>
<td>(p, q, 2, 3)</td>
</tr>
<tr>
<td></td>
<td>(1, 2, 3, p)</td>
</tr>
<tr>
<td></td>
<td>(p, q, 3, 4)</td>
</tr>
<tr>
<td></td>
<td>(q, p, 4, 1)</td>
</tr>
<tr>
<td></td>
<td>(q, q, 1, 3)</td>
</tr>
<tr>
<td></td>
<td>(1, 4, 2, q)</td>
</tr>
<tr>
<td></td>
<td>(1, 2, 3, q)</td>
</tr>
</tbody>
</table>

Fig. 4. Direct transformations to recover missing edges.

<table>
<thead>
<tr>
<th>Tetrahedra:</th>
<th>(1, 2, 3, A)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(1, 3, 2, B)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tetrahedra:</th>
<th>(1, 2, 8, A)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(2, 3, 8, A)</td>
</tr>
<tr>
<td></td>
<td>(3, 1, 8, A)</td>
</tr>
</tbody>
</table>

Fig. 5. Recovery of a missing edge by deleting a face.
Fig. 6. Direct transformations to recover missing faces.
3. Robustness of boundary recovery

In the boundary recovery stage, many intersection calculations have been involved. However, the precision of the calculations will affect the robustness of the boundary recovery, and that of the mesh generator finally. Therefore, more detail consideration has been taken to deal with this problem. Furthermore, a method has been adopted to deal with inconsistent geometrical judgement, which is based on epsilon geometry [10].

3.1. Exact and robust algorithms

In computational geometry, the term ‘robust’ describes algorithms whose accuracy and validity are not affected by round-off error. The development of robust geometric algorithms has been attracting increasing attention in the area of computational geometry in recent years [10, 11].

In comparing the robust and the exact algorithms for a geometry problem it is assumed that the coordinates of the input points are $B$-bit integers. The robust algorithm converts these integers into floating point numbers before performing additions and multiplications, while the exact algorithm keeps these integers unchanged. In order to avoid the round-off in all cases, the exact algorithm must use $(2B + 1)$-bit integer arithmetic, whereas the robust algorithm requires only $(B + 6)$-bit (mantissa) floating-point arithmetic to generate $B$ bits of accuracy in the output. On a typical modern computer, if the inputs are integers, then the exact algorithm can use double-precision floating-point operations and there will be little difference in cost between the robust and the exact algorithm. However, if the inputs are in double precision and double-precision accuracy is required, then the arithmetic cost of the exact algorithm may be at least ten times greater than that of the robust algorithm [11].

3.2. Framework of epsilon geometry

Epsilon geometry is based on a very general model of imprecise computations, which include floating-point and rounded-integer arithmetic. Its framework defines in a general sense the notation of an epsilon predicate as a means for expressing ‘approximate tests’. An epsilon predicate is defined as follows [10]:

Let $\mathcal{O}$ be a set of geometric objects endowed with some distance measure $\parallel \cdot \parallel$, and $P$ be a predicate defined on $\mathcal{O}$. Then, for any $X \in \mathcal{O}$ and any $\varepsilon \geq 0$, we define $\varepsilon$-$P(X)$ as shorthand for $P(X')$ is true for some $X' \in \mathcal{O}$ such that $\parallel X, X' \parallel \leq \varepsilon$. That is, $X$ is at most $\varepsilon$ away from satisfying $P(X)$. Therefore, the truth-set of $\varepsilon$-$P$ is that of $P$, ‘fattened’ by $\varepsilon$ (Fig. 8(a)). Note that $0$-$P(X)$ is the same as $P(X)$. In
the case of an n-ary predicate \( P \), we define \( \varepsilon P(X_0, \ldots, X_{n-1}) \) to mean that \( P(X'_0, \ldots, X'_{n-1}) \) is true for some \( X'_0, \ldots, X'_{n-1} \) with \( \|X_i, X'_i\| \leq \varepsilon \) for all \( i \).

For \( \varepsilon > 0 \), we define \( (-\varepsilon)P \) as shorthand for \( P(X') \) is true for all \( X' \in \mathcal{O} \) such that \( \|X, X'\| \leq \varepsilon \). The truth set of \( (-\varepsilon)P \) for \( \varepsilon > 0 \), therefore, is that of \( P \), 'trimmed' by \( \varepsilon \) (Fig. 8(b)). Intuitively, an object \( X \) which is \( (-\varepsilon)P \) is 'extremely \( P \)', while an \( X \) which is \( \varepsilon P \) is only 'almost \( P \')'.

The epsilon predicates defined above are exact mathematical notions which can be used in mathematical proofs. In programming, a procedure called an epsilon box can be implemented as a geometric predicate. There are three epsilon boxes which have been proposed. A T-box for a predicate \( P(X) \) is a procedure \( T.P(X) \) that computes \( P(X) \) and returns either true, false or unknown. Similarly, an E-box for \( P(X) \) is a procedure \( E.P(\varepsilon, X) \) that computes \( \varepsilon P(X) \) and also returns either true, false and unknown. That is, \( T.P(X) \) is equivalent to \( E.P(0, X) \). Finally, an I-box for predicate \( P(X) \) is a procedure \( I.P(X) \) that returns an estimate of how far \( X \) is from satisfying \( P \).

Four basic epsilon predicates are illustrated in Fig. 9. The predicate \( \varepsilon \)-Coincident\((p, q) \) is true if, and only if, points \( p \) and \( q \) lie within \( \varepsilon \) of a common point (Fig. 9(a)). And \( \varepsilon \)-Collinear\((p, q, r) \) is true if, and only if, there exists a line that passes within \( \varepsilon \) of all three points (Fig. 9(b)). Similarly, the predicate \( \varepsilon \)-Pos\((p, q, r) \) is true if, and only if, it is possible to make \( D(p, q, r) > 0 \) by displacing the three points at most by the value \( \varepsilon \) in appropriate directions, where \( D(p, q, r) \) is the determinant related to the area of triangle \( T = (p, q, r) \) (Fig. 9(c)). Finally, \( \varepsilon \)-Between\((z, pq) \) is true if, and only if, the distance from \( z \) to the segment \( pq \) is at most \( 2\varepsilon \) (Fig. 9(d)).

3.3. Correctness of boundary recovery

In the boundary recovery of 3-D mesh generation, the determination of edge-face intersection will be heavily involved (see [9]). The imprecise computation will sometimes result in failure. To this end, the implementation can be carried out following the epsilon geometry paradigm.

Moreover, a subtle error type occurs when numerical data is converted to symbolic data. Typically, such a conversion takes place when the result of a numerical computation is used in a branch procedure on zero operation. If only one such conversion from numerical to symbolic data occurs, a slight perturbation of the numerical input data should force the result of the numerical computation to agree with the conversion. Thus, no inconsistency is likely to arise from either decision. However, a program may contain a sequence of such conversions that are logically dependent. Thus, inaccuracies in the numerical computation may result in inconsistent conversion, and once symbolic data is inconsistent, some property—essential for the correctness of the algorithm—may fail. The solution to this problem is to construct the algorithm so that all conversions are logically independent [12].

Fig. 10 shows an intersection example of a tetrahedron with a boundary face \( ABC \) encountered in
the boundary recovery stage. If point 4 is very near the plane supported by points A, B and C, and the intersecting points are to be computed using the plane equation and the line equations, respectively, then an inconsistency case may occur in which the numerical results indicate that edges 14 and 24 intersect with triangle ABC while edge 34 does not intersect with triangle ABC. This inconsistency can be removed by determining whether point 4 is above the triangular plane ABC, and then finding the intersecting points as accurately as possible. This is another way to improve the correctness of the boundary recovery.
4. Performance of the mesh generation

The computational complexity (i.e. computing cost) of the Steiner triangulation (a combination of Delaunay triangulation and Steiner point creation herein), has not yet been solved theoretically and is an open problem to the computational geometry community. However, numerical experiments can be used to examine the performance in practical applications. This is a general way to investigate the computational complexity of an algorithm when the algorithm becomes sophisticated.

Fig. 11. Boundary representation and a mesh of a cube with a hole.

Fig. 12. CPU time vs number of points in the final meshes.
For the purpose of investigating the performance of the mesh generator, a cube with a hole as shown in Fig. 11 has been meshed using different element sizes. In this example, some of the points are generated temporarily and are located within the hole. These points have been removed during the boundary recovery and the post treatment. Fig. 12 illustrates temporal breakdown in terms of curves of CPU time vs. number of points in the final meshes. The numerical experiments have been conducted on a Silicon Graphics workstation (IRIS Indigo Elan, CPU R3000). It can be seen that the time for volume meshing is much greater than the time for boundary recovery. It should be mentioned that the swap space in the computer affects the CPU time statistics as the problem scale increases further.

5. Examples of 3D meshes

5.1. Benchmark tests

On completion of the development of the 3D meshing algorithm, it has been used to demonstrate its robustness and accuracy, and a set of examples has been considered to test all aspects. Fig. 13 illustrates a simple shape, of which the boundary information is given by using T3 type patches, where 192 patches are involved. This figure gives its surface representation, intermediate surface mesh and volume mesh presented with a half cut.

Fig. 14 depicts the surface description and the corresponding volume mesh of a second example, of which the shape is defined by 400 Q4 type patches. A third example is to consider a casting component, which is described by a collection of 36 T3, Q4 and Q8 surface patches. Fig. 15 illustrates the corresponding volume mesh shown as a whole volume and with a half cut.

The NURBS patch is a form of surface description, which has been included in this mesh generator. After a surface mesh of a closed object has been obtained, the remaining procedures in the mesh generation are the same as for objects given by any basic patches. A torus is shown in Fig. 16, which is

---

Fig. 13. Surface description, intermediate surface mesh and volume mesh (with a half cut) of a mechanical component.
described by one single NURBS patch. The edges of the patch are curves as shown in Fig. 16 and the edges merge to form the torus. The corresponding volume mesh is shown in Fig. 17.

Example five (Fig. 18) is more complicated. It involves three mechanical components: a crankshaft, a connecting rod and a piston. They are defined initially by T3 patches, and an unified element size is required for the three components. The details of the resulting meshes are given in Table 1.

All these examples above show valid triangulation and boundary conformity to the given collections of patches. These examples have been used as benchmark problems in the development of the mesh generator.

5.2. Towards adaptivity

If a mesh generator is able to produce meshes according to the point spacing required by the analysis program, then this mesh generator can be incorporated into the analysis program, and automatic adaptive analyses may be performed. In order to examine the current mesh generator in terms of this capability, an example is designed, which is related to a cube \([-1, 1] \times [-1, 1] \times [-1, 1]\). An artificial point spacing

<table>
<thead>
<tr>
<th>Mesh Type</th>
<th>Surface Points</th>
<th>Surface Elements</th>
<th>Volume Points</th>
<th>Volume Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crankshaft</td>
<td>647</td>
<td>4077</td>
<td>8150</td>
<td>8365</td>
</tr>
<tr>
<td>Pin-rod</td>
<td>390</td>
<td>1529</td>
<td>3458</td>
<td>1675</td>
</tr>
<tr>
<td>Piston</td>
<td>292</td>
<td>2013</td>
<td>4018</td>
<td>2461</td>
</tr>
</tbody>
</table>
function \( d \) is given as follows (refer to Fig. 19):

\[
d = \begin{cases} 
0.024, & \text{if } z > 0 \text{ and } 0.5z^2 < 1.5x^2 + y^2 < 0.7z^2; \\
0.12, & \text{otherwise.}
\end{cases}
\]  
(1)

The resulting mesh produces reasonable agreement with the point spacing requirement, as shown in Fig.
Fig. 17. Volume mesh presented as a whole and the other with a half cut.

Fig. 18. Surface descriptions, surface meshes and volume meshes (with quarters cut) of three mechanical components.

20, where three configurations have been cut by different boxes in order to display the interior parts of the mesh. In this example, the mesh concentration is quite high, as can be found in some real adaptive analyses. It can be concluded that the current mesh generator is suitable to incorporate into analysis programs to perform adaptive simulation.
6. Mesh quality

6.1. Quality measures

Fig. 21 shows some common configurations of poorly-shaped tetrahedral elements. Various measures have been proposed in the past to characterize the tetrahedral shapes. From [9, 13, 14], 10 formulations have been collected as listed in Table 2. The optimal values are the corresponding values of these measures for an equilateral element. Using these optimal values, the measures can be normalized for easy comparisons. The global quality can be defined as an average value of the values for all elements in the domain concerned. Among these formulations, the definitions of mean ratio $\eta$ and minimum solid angle are given as follows [14].

Let $T(t_0, t_1, t_2, t_3)$ denote the tetrahedron of interest, and $R(r_0, r_1, r_2, r_3)$ be an optimal tetrahedron having same volume as $T$. An affine transformation from $R$ to $T$ can be written as $t_i = M r_{p(i)} + b$ ($0 \leq t \leq 3$), where $(p(0), p(1), p(2), p(3))$ is a permutation of $(0, 1, 2, 3)$, and $b$ is a translation vector. Then define $\eta = 3\sqrt{\lambda_1 \lambda_2 \lambda_3}/(\lambda_1 + \lambda_2 + \lambda_3)$ where $\lambda_1, \lambda_2$ and $\lambda_3$ are the eigenvalues of matrix $M^TM$. A geometric interpretation is that the $\lambda_i$ are proportional to the squares of the half-lengths of the three principal axes of the inscribed ellipsoid of $T$, formed from applying the affine transformation $Mx + b$ to the inscribed sphere of $R$. As mentioned in [14], the expression of $\eta$ with respect to $V$ and $S_i$ ($i = 0, \ldots, 5$) in Table 2 has been proved.

The solid angle $\theta_i$ at vertex $t_i$ of tetrahedron $T(t_0, t_1, t_2, t_3)$ is defined to be the surface area formed by projecting each point on the face not containing $t_i$ to the unit sphere centered at $t_i$ (Refer to Fig. 22). The minimum solid angle $\theta$ is defined to be the minimum of $\theta_i$ ($i = 0, 1, 2, 3$). It is shown that $0 \leq \sum_{i=0}^{3} \theta_i \leq 2\pi$. Therefore, a very large solid angle (near $2\pi$) implies that $T$ has small solid angles. In 2D triangular meshes, the minimum interior angle of a triangle is a commonly used quality measure, and Delaunay triangulation satisfies a max–min angle criterion. The minimum solid angle is a natural extension of the minimum interior angle to three dimensions. The max–min criterion has been used to improve the quality of 3D Delaunay triangulation by local transformation [15].

In [14], it has been proved that, for any pair formed from $1/\beta$, $\eta$ and $\theta$, there exist positive constants
Table 2
Quality measures proposed in the literature

<table>
<thead>
<tr>
<th>Quality measures</th>
<th>Notations</th>
<th>Optimal values</th>
<th>Ranges</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circumradius</td>
<td>$\beta = \frac{R}{R_0}$</td>
<td>3.0</td>
<td>$[1, \infty)$</td>
<td>[13, 14, 20]</td>
</tr>
<tr>
<td>Inradius</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum edge length</td>
<td>$\sigma = \frac{S_{\text{max}}}{R_0}$</td>
<td>4.898979</td>
<td>$[1, \infty)$</td>
<td>[13, 21]</td>
</tr>
<tr>
<td>Minimum edge length</td>
<td>$\omega = \frac{S_{\text{min}}}{R_0}$</td>
<td>0.6123725</td>
<td>$(0, \infty)$</td>
<td>[13, 21]</td>
</tr>
<tr>
<td>Circumradius</td>
<td>$\tau = \frac{R_{\text{max}}}{R_{\text{min}}}$</td>
<td>1.0</td>
<td>$[1, \infty)$</td>
<td>[9, 13, 21]</td>
</tr>
<tr>
<td>4th power of sum of square surface areas</td>
<td>$k = \frac{1}{3} \sum_{i=1}^{3} r_i^4$</td>
<td>4.572474e-4</td>
<td>$(0, 1)$</td>
<td>[13, 22]</td>
</tr>
<tr>
<td>3rd power of average edge length</td>
<td>$\alpha = \frac{S_{\text{avg}}}{V}$</td>
<td>8.4852816</td>
<td>$[1, \infty)$</td>
<td>[13, 23]</td>
</tr>
<tr>
<td>3rd power of root mean square of edge lengths</td>
<td>$\gamma = \frac{S_{\text{rms}}}{V}$</td>
<td>8.4852816</td>
<td>$[1, \infty)$</td>
<td>[13, 24]</td>
</tr>
<tr>
<td>Maximum dihedral angle</td>
<td>$\delta = \frac{\sum_{i=0}^{3} S_i^2}{V^3}$</td>
<td>1.2309594</td>
<td>$[1, 2.5522]$</td>
<td>[9]</td>
</tr>
<tr>
<td>Mean ratio</td>
<td>$\eta = \frac{1/2 \sum_{i=1}^{3} S_i^2}{V^3}$</td>
<td>1.0</td>
<td>$(0, 1)$</td>
<td>[14]</td>
</tr>
<tr>
<td>Minimum solid angle</td>
<td>$\vartheta = \frac{1}{2 \sum_{i=1}^{3} S_i^2}$</td>
<td>0.5512856</td>
<td>$(0, 1)$</td>
<td>[14]</td>
</tr>
</tbody>
</table>

Fig. 20. A mesh of a cube with artificial element size requirement.
Fig. 21. Some poorly-shaped tetrahedral elements.

Fig. 22. The definition of solid angle.

c_0, c_1, e_0, and e_1 such that c_0 \mu^{e_0} \leq \nu \leq c_1 \mu^{e_1}, where \mu or \nu denote one of 1/\beta, \eta and \theta. This means that these three measures are equivalent.

6.2. Statistics analysis

All these measures have been implemented in the present work. Based on an example given in Fig. 23, the behaviour and performance of these measures will be discussed briefly. In this example, 52 T3 patches are given to describe the geometry boundary. The volumetric mesh was created at an early stage of the development of the mesh generator, and 8718 points and 46476 elements have been involved. Table 3 lists the details of these measures applied to the example, where the relative CPU times are also given. It should be mentioned that CPU times depend upon the implementation. Therefore, it cannot be used as an independent criterion to choose between these formulations. In Table 3, the minimum, maximum and average values of the measures for all elements are also given. Figs. 24–33 illustrate the distribution of all these measures.

A comparison of the first seven formulations has been given in [13], where four sensitivity tests have been carried out. Through these tests, the observation was made that: \alpha, \beta, \sigma, \gamma and \kappa are able to characterize distortion of all the four tests, while \omega and \tau have their limitations. Considering the computational cost it was recommended to use \gamma.
Although the CPU time statistics depend on the implementation of the measures, the result listed in Table 3 roughly agree with the observations made in [13]. Different measures have their own distribution characteristics for a particular mesh (Figs. 24–33). Furthermore, if one considers a family of meshes, it is hard to choose between these formulations. However, the sensitivity analysis of some artificial cases is able to provide ideas to discard a particular measure. Also, the computational performance is a factor which can be used in the evaluation of these formulations.

6.3. Quality improvement

Theoretically, the mesh smoothing technique allows the relaxation of interior points of a volume mesh, and results in a mesh of higher quality. The smoothing formulation is analogous to that of surface meshes. With the experience of the present work, the relaxation coefficient \( \varphi \) needs to be set far less than 1.0, otherwise, invalid meshes with elements of negative volumes are more likely to result.

7. Visual representation

It is much more difficult to visualize volume meshes than to do surface meshes, as the interiors of the volume meshes are difficult to see by common techniques. In order to visually examine the meshes and to represent the details of their interiors, several ways have been studied [9, 16, 17] such as domain splitting,
Fig. 24. Statistics of mesh quality: (1) $\beta$ values.

Fig. 25. Statistics of mesh quality: (2) $\sigma$ values.

Fig. 26. Statistics of mesh quality: (3) $\omega$ values.
Fig. 27. Statistics of mesh quality: (4) \( \tau \) values.

Fig. 28. Statistics of mesh quality: (5) \( \kappa \) values.

Fig. 29. Statistics of mesh quality: (6) \( \alpha \) values.
Fig. 30. Statistics of mesh quality: (7) $\gamma$ values.

Fig. 31. Statistics of mesh quality: (8) $\delta$ values.

Fig. 32. Statistics of mesh quality: (9) $\eta$ values.
element shrinking, displaying point clouds, using wireframe representation, and elements intersected by a user defined plane.

Fig. 33 shows a domain split mesh representation. This can provide information about the sizes of interior elements. Unfortunately, the angles involved in elements are not easy to see. Fig. 35 provides an illustration of elements shrunk, which is able to give a limited idea of the shapes of elements through gaps between these shrunk elements. Although point clouds are able to show point spacing for 2D meshes clearly, this is difficult to be imagined in three dimensions as in Fig. 36. Interactive representation helps a great deal via manipulations such as rotation, translation and zooming. However, there is no connectivity information which can be provided, and the use is limited to certain scale of problems. For a very large

Fig. 34. Domain splitting.
Fig. 35. Element shrinking.

Fig. 36. Point cloud representation.
scale problem, the point clouds only show a screen of dots of a part of the domain. Fig. 37 shows the wireframe representation of the mesh. The fifth method is to display edges of elements which intersect with a given plane. Normally, the wireframe, or edges of elements intersecting with a given plane is a screen of ‘spaghetti’, or for fine cases a white display. Even for a coarse mesh, it is hard to see the trees (elements), and only to get the forest [16].

However, the above approaches give the user some ideas rather than nothing as there are very limited number of effective techniques capable of representing the interior of a volume mesh for the time being.

If there are some scalar values associated with the mesh, then element clouds can be presented to show the parts (or elements) of interest. It is called weathermap representation in [17]. Similarly, if the values of mesh quality are assigned to individual elements, then this method can be used to visually assess the quality of the mesh.

8. Visual quality assessment

Histogramming is a traditional method to address mesh quality assessment. It is capable of giving the user a statistical senses of the mesh quality throughout the entire domain [16]. With reference to Figs. 24–33, it is observed that these histograms show different shapes of the quality distributions. All the histograms can be used as a basis of visual assessment, provided that the corresponding quality measure is sensitive enough to any distortion of an element.

Under an interactive environment, meshes can be displayed according to quality threshold setting. With a given setting, the corresponding elements will be drawn on the screen. In this way, the user is able to inspect elements, and further to improve the mesh in a particular region by changing control parameter settings at the mesh generation stage.

Fig. 38 illustrates the element representations when the quality ranges are set to be (0, 0.1), (0, 0.05) and (0, 0.02), respectively. The quality measures used herein is minimum solid angle $\theta$. For this mesh, the minimum, maximum and average values of $\theta$ for all the elements are 3.13e-3, 0.9412 and 0.42106, respectively.
9. Conclusions

Volume meshes can be generated through three-dimensional triangulation and interior point creation based on the surface meshes. To deal with non-convex geometries, boundary surface conformity is gained via edge swapping, boundary edge and surface recovery, and the robustness of the algorithm has been discussed with regard to the accuracy of geometric judgements. In order to gain a robust algorithm for the boundary recovery, special attention must be paid to the geometric tolerance requirements.

The performance of the mesh generator has been investigated by means of numerical experiments. Several examples have been tested to validate the mesh generator, these were employed as benchmark problems during the development of the mesh generator. By generating test meshes with different point spacing requirements, it is observed that this mesh generator is able to deal with adaptive meshing.

Ten quality measures have been addressed in terms of definitions. A practical mesh has been examined using these measures with quality statistics and the corresponding histograms. It was experienced that good care should be taken during the mesh smoothing stage, as it is likely to lead to a mesh with negative volume elements for a large relaxation coefficient.

During the mesh generation, interactive procedures may be used to change control parameters, in order to improve the resulting mesh. Visual representation of volume meshes is a rather difficult due to the nature of the problem, several techniques have been discussed in this paper. Using mesh quality metrics, visual quality assessment has been introduced to display elements within a range of quality values. This is of a particular benefit in the mesh generation stage towards a good quality mesh.

Acknowledgements

This work has been carried out under financial support by the Engineering and Physical Science Research Council of the United Kingdom (Grant Number GR/G 35107), which is gratefully acknowledged. The geometry visualization programs used in this work are Geomview [18] and FEView [17]. FEView.
making use of Forms GUI library [19], works with Geomview for finite element applications. We would like to thank the creators of Geomview for permission to use their software, and Prof. Mark Overmars for the use of the Forms Library.

References