Engineering with Computers (2004) 20: 11–21
DOI 10.1007/s00366-004-0277-3

ORIGINAL ARTICLE

S. H. Lo · W. X. Wang

A fast robust algorithm for the intersection of triangulated surfaces

Received: 15 November 2002 / Accepted: 31 December 2003 / Published online: 4 March 2004

Abstract The use of discrete data to represent engineering structures as derivatives from intersecting components requires algorithms to perform Boolean operations between groups of triangulated surfaces. In the intersection process, an accurate and efficient method for the determination of intersection lines is a crucial step for large scale and complex surface intersections. An algorithm based on tracing the neighbours of intersecting triangles (TNOIT) is proposed to determine the intersection lines. Given the node numbers at the vertices of the triangles, the neighbour relationship is first established. A background grid is employed to limit the scope of searching for candidate triangles that may intersect. This will drastically reduce the time of geometrical checking for intersections between triangles, making the surface intersection and mesh generation a quasi-linear process with respect to the number of elements involved. In the determination of intersection between two triangles, four fundamental cases are identified and treated systematically to enhance robustness and reliability. Tracing the neighbours for the determination of intersection lines not only greatly increases the efficiency of the process, it also improves the reliability as branching and degenerated cases can all be dealt with in a consistent manner on the intersecting surfaces concerned. Five examples on a great variety of surface and mesh characteristics are given to demonstrate the effectiveness and robustness of the algorithm.

Keywords Intersection · Neighbour tracing · Triangular mesh · Surface

S. H. Lo (✉) · W. X. Wang
Department of Civil Engineering,
The University of Hong Kong,
Hong Kong, China
E-mail: hreclsh@hkucc.hku.hk

Introduction

There are, in general, two ways to represent a surface for the purpose of visualisation and computation. One is to use analytical surface patches via functions such as Coon's patches, B-splines or NURBS surfaces. Another is to use discrete data structures such as tessellate facets and triangular facets [1]. For engineering applications, surfaces defined by discrete data are widely adopted, for instance, in the presentation of complex molecules [2, 3, 4, 5], engineering design [6], visualisation [7] and computational models for analysis by the finite element method [4, 5, 6, 8], etc. There are at least three ways for the construction of complex surface models of triangulated facets. The first approach is to form triangles directly on the surface by well-known mesh generation techniques [9], such as the octree method [10], the advancing-front method [11, 12, 13, 14], the paving method [15, 16] and so on. The second is the parametric mapping approach [17, 18, 19, 20] which meshes the parametric domain followed by a mapping process of the resulting mesh onto the surface. The third approach is to construct models by using Boolean operations such as intersection [6, 8] and union between groups of surfaces composed of triangular facets. A great variety of objects can be easily created by selectively putting together various surface parts derived from surface intersections. This approach has been explored and applied in the field of finite element mesh generation [21, 22], molecules model [6], engineering design [23, 24], etc.

The intersection of triangulated surfaces consists of line segments from the intersection of triangles, which can be collected to form chains. These chains will help to define, and will be part of, the boundaries of the surface parts as a result of surface intersection. For large scale and complex surfaces, an accurate and efficient method for the determination of intersection chains is a crucial step in the intersection process. The problem we have at hand is to find out all the intersection segments when two or more surfaces of triangular facets come together. It would be very time-consuming if we try to detect intersections by examining all the triangular facets on two groups of surfaces a pair of triangles at a time. The process can be speeded up by using a bounding box technique [6, 24] to filter out all triangles of the two groups of surfaces that are too far apart and cannot possibly have intersections. Additional data structures can also be set up to further reduce searching for intersections [23, 24]. Moreover, the intersection line segments obtained, in general, do not follow any particular order. However, in actual engineering applications, these line segments have to be re-ordered to form loops or chains before they could be included as boundaries of surface parts. In this process, undetermined and unresolved cases often arise when many line segments are very close to one other. A wrong connection of line segments may result in forming incorrect chain/loop structures inconsistent with the actual intersection.

In this paper, an algorithm based on the idea of tracing the neighbours of intersecting triangles (TNOIT) is proposed in the construction of chains/loops in a progressive and consistent manner. The construction process can be initialised with any intersection segments. As the neighbouring elements of each triangular facet are known, the triangular facet to which the intersection line will extend can be easily identified. This process will be continued until a complete loop is formed or the intersection line terminates on surface boundaries to form a chain.

The algorithm

The types of surfaces under consideration are surfaces already discretised into triangular facets/elements. The intersection lines between surfaces are represented in terms of straight-line segments, which can be determined by considering a pair of triangular facets at a time. Intersection lines are represented in the form of loops and chains. This definition for a surface intersection is broad enough to cover the intersection of artificial surfaces that can be discretised into triangular facet and finite element meshes made up of surfaces of triangular elements. An intersection line will form progressively by connecting line segment end to end. Two situations can be identified: 1. the line terminates on the boundary of the intersection surfaces to form a open chain and 2. it goes back to the starting point to form a closed loop. An accurate and efficient process for the determination of intersection chains and loops is of utmost importance to the intersection of discretised surfaces. A trace by a neighbours technique is proposed, which greatly enhances reliability and efficiency as neighbours for each triangle are known and connections of segments are done by means of continuity. The following are the steps for the construction of intersection chains/loops by the neighbour tracing technique.

1. Find out and record the neighbours of each triangle on the two surfaces.
2. Reduce the population of candidate triangles by means of background grid filter.
3. Identify an intersection segment from one of the partition cells.
4. Determine the associated chain/loop by continuity tracing neighbouring elements.

The determination of neighbours

The neighbours of each triangle are a useful topological information of the triangulated surfaces. The searching time can be much reduced by making use of the continuity as provided by the neighbourhood relationship. The following is a detailed description as how neighbours of the triangles can be identified.

Some concepts and thoughts

For a given surface composed of triangular facets, generally each triangle in the surface has three adjacent triangles. If two triangles have a common edge then they are neighbours to each other. In Fig. 1, the edge *mn* is a common edge and the triangle *I* is a neighbour of triangle *J* about *mn*. In case no neighbour can be found on one of the edges (see triangle *K* in Fig. 1), the related edge is a boundary edge of the surface. For a given triangular mesh, the average number of edges connected to a particular node is quite constant and is independent of the total number of elements present in the system. Based on this idea, an efficient algorithm can be defined to determine the neighbourhood relationship by considering elements and edges connected to a node.

The data structure

A simple data structure is adopted to hold the topological information of the triangular facets, and only the three vertices of each triangle are needed and stored. As for other connectivity information, it can be derived from this basic data structure.

– Structure $\{T_k;//k^{th}$ triangle on the list
– $v_1{}^k$;//the first vertex of T_k
– $v_2{}^k$;//the second vertex of T_k
– $v_3{}^k$;//the third vertex of T_k
– $n_1{}^k$;//the neighbour about edge $v_2{}^k v_3{}^k$
– $n_2{}^k$;//the neighbour about edge $v_3{}^k v_1{}^k$
– $n_3{}^k$;//the neighbour about edge $v_1{}^k v_2{}^k\}$;

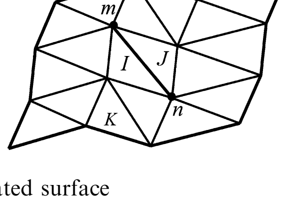


Fig. 1 A triangulated surface

The algorithm of finding out neighbours

Let N_t be the number of triangular facets on a given surface S and N_v be the number of nodal points. Let $S[i]$ be the set of triangles in S connected to node i $(1 \le i \le N_v)$.

a. Initialize $S[i]$ as null set.
b. For each triangular facet T_k, $1 \le k \le N_t$.
- Add the triangular facet T_k to the sets $S[v_1{}^k]$, $S[v_2{}^k]$ and $S[v_3{}^k]$ respectively.
- Where $v_1{}^k$, $v_2{}^k$, $v_3{}^k$ are the three node numbers of triangular facet T_k.
- End

c. For each triangular facet T_k from 1 to N_t.
- Find out and record the neighbours of triangle T_k.
- A neighbour of the triangle T_k is another triangle present in both sets $S[v_1{}^k]$ and $S[v_2{}^k]$, in which $v_1{}^k$ and $v_1{}^k$ are vertices of triangle T_k.
- End.

This is a linear process as the steps taken for the determination of neighbours for each triangle is more or less constant.

Using a background grid to filter away those triangles that do not intersect

A direct computation of intersections between two groups of surfaces by considering all the triangles on one group against all the triangles on the other group is not an economical process. As intersections only happen in a few parts of the two groups of surfaces, it is necessary to filter out triangles that do not possibly have any intersection. A background grid is introduced for this purpose, in which the search for intersection of a triangular facet can be localised to within a small volume as given by a typical cell of the background grid. The time complexity of the process is linear for triangular facets of homogeneous sizes.

Some concepts and thoughts on the background grid

A more sophisticated spatially subdivision with cells of variable sizes may be necessary for triangular facets of great differences in size [10]. Here, we only discuss the background grid with cells of equal size. Following the natural sequence of the cells in the background grid, the triangles having intersections with the cells are recorded. Within the same cell, there may be intersections if there is at least one triangular facet from each group of the surfaces. There will not be any intersections if a triangular facet from either group is missing. Hence, this condition will be used as a preliminary check for possible intersections. Figure 2 shows triangular facets cutting a typical cell, in which continuous lines represent the facets from the first group and dashed lines represent the facets from the second group.

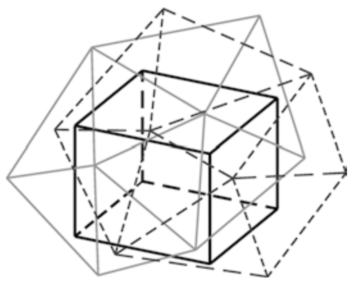


Fig. 2 Triangles intersected by a typical cell

The dimensions of a typical cell [25] are given by:

$$d_x = l_x/N_x, d_y = l_y/N_y, d_z = l_z/N_z \quad \text{(a)}$$

in which l_x, l_y and l_z srepresent the lengths on the three sides of the bounding box as shown in Fig. 3. The number of divisions $N = N_x \times N_y \times N_z$ depends on the amount of computer memory available. The size of a cell can be set equal to the average length of the edges on the surfaces. However, the dimensions of the cells can be adjusted so that the use of available computer memory can be optimised [25].

Determining the cells intersected by a triangular facet

A typical triangular facet will intersect with more than one cell in the background grid. Considering each triangular facet in turn, the cells cutting the triangular facet under consideration are recorded. At the end of the process, the triangular facets intersecting with each cell can be determined.

The cells cutting the triangular facet can be found by the following procedure.

a. Calculate the bounding box B of the triangular facet T with vertices v_1, v_2, v_3.

$$x_{\min} = \min(a_1, a_2, a_3), x_{\max} = \max(a_1, a_2, a_3) \quad \text{(b)}$$

$$y_{\min} = \min(b_1, b_2, b_3), y_{\max} = \max(b_1, b_2, b_3) \quad \text{(c)}$$

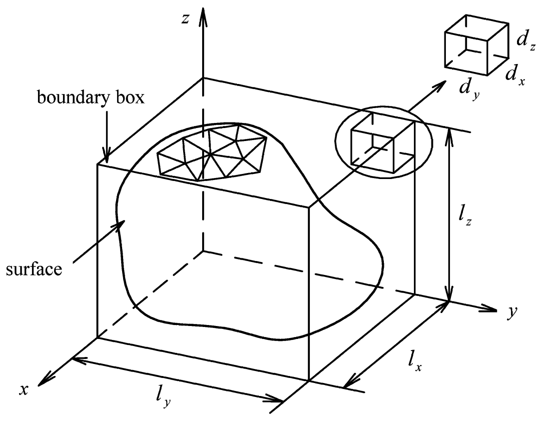


Fig. 3 The background grid for triangulated surfaces

$$z_{\min} = \min(c_1, c_2, c_3), z_{\max} = \max(c_1, c_2, c_3) \tag{d}$$

where $v_1 = (a_1, b_1, c_1)$, $v_2 = (a_2, b_2, c_2)$, $v_3 = (a_3, b_3, c_3)$.

b. Calculate the cells intersected by triangular facet T

Suppose the containing box of one group of surfaces is C:

$$[X_{\min}, X_{\max}] \times [Y_{\min}, Y_{\max}] \times [Z_{\min}, Z_{\max}] \tag{e}$$

If B is outside C, no cell in C will intersect with the triangular facet T; otherwise, find out and record the intersecting cells with address (i, j, k) using the following formulas

$$i = \left[\mathrm{Int}\left(\frac{x_{\min} - X_{\min}}{d_x}\right) + 1\right] \to \left[\mathrm{Int}\left(\frac{x_{\max} - X_{\min}}{d_x}\right) + 1\right], \tag{f}$$

$$j = \left[\mathrm{Int}\left(\frac{y_{\min} - Y_{\min}}{d_y}\right) + 1\right] \to \left[\mathrm{Int}\left(\frac{y_{\max} - Y_{\min}}{d_y}\right) + 1\right], \tag{g}$$

$$k = \left[\mathrm{Int}\left(\frac{z_{\min} - Z_{\min}}{d_z}\right) + 1\right] \to \left[\mathrm{Int}\left(\frac{z_{\max} - Z_{\min}}{d_z}\right) + 1\right] \tag{h}$$

Finding out all the candidate triangles

a. Define a background grid on the first group of surfaces S_1
- Either group can be used here, but the smaller group is usually chosen for higher computational efficiency.

b. For all the triangular facets in the first group of surfaces S_1
- Determine the cells C_i intersected by each triangular facet $T_k \in S_1$
- Record the intersecting facets T_k in cells C_i
- End

c. For all the triangular facets in the second group of surfaces S_2
- Determine the cells C_i intersected by each triangular facet $F_j \in S_2$
- Record the intersecting facets F_j in cells C_i
- End

d. For all the cells C_i from 1 to N
- Each cell C_i will be examined in turn. Cell C_i will be ignored if either T_k from S_1 or F_j from S_2 is missing. Triangles that may have a possibility to intersect are recorded as candidate elements.
- End

An example

Two spherical surfaces are shown in Fig. 4a, on each of which there are 672 triangular facets. Figure 4b shows the candidate triangles determined by using a background grid. As shown in Figs. 4c and 4d, the number of the candidate triangles for the first surface is 88 and the number for the second surface is 157. Obviously, the candidate triangles are only a small subset of the original surfaces.

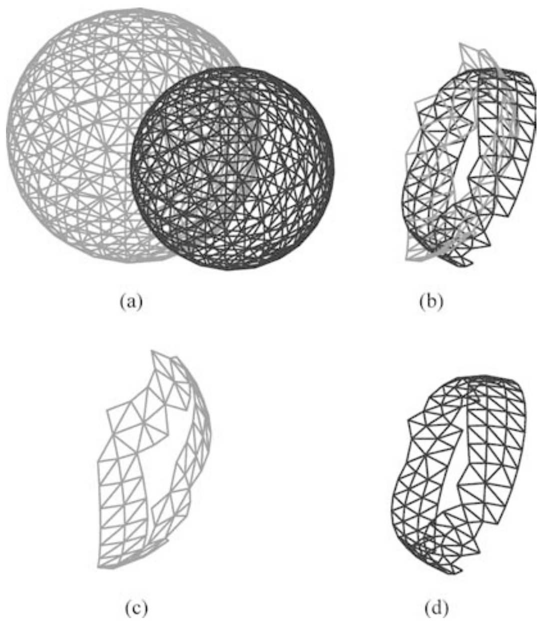


Fig. 4 Candidate triangles on two intersecting surfaces

Calculating the intersecting straight-line segment between a pair of triangles

The two given surfaces $T = \{A_i\ B_i\ C_i,\ i = 1,\ N_T\}$ and $F = \{D_j\ E_j\ G_j, j = 1,\ N_F\}$ are in fact two sets of triangular facets. The intersection between surfaces T and F can be determined by considering in turn the intersection of a pair of triangles, each one taken from each of the two surfaces. The intersection between a typical triangular facet ABC from surface T and a triangular facet DEG from surface F can be determined as follows [8].

In order to find out where triangle DEG cuts triangle ABC, we have to consider the intersection between triangle ABC with edges DE, EG and GD of triangle DEG as shown in Fig. 5a. Line segment DE will have intersection with triangle ABC if:

1. points D and E are on the opposite sides of the plane containing triangle ABC, and
2. intersection point P is inside triangle ABC.

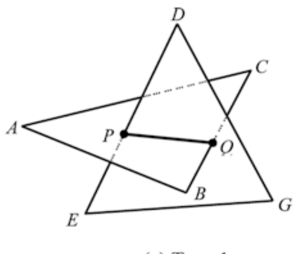
Condition 1 can be verified by checking if

$$\left(AD \cdot \overrightarrow{N}\right)\left(AE \cdot \overrightarrow{N}\right) \leqslant 0 \tag{1}$$

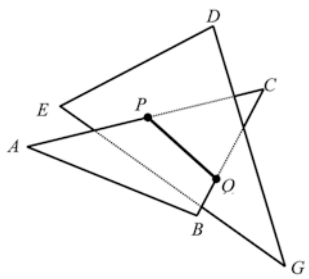
where normal vector $\overrightarrow{N}$ of triangle ABC is given by

$$\overrightarrow{N} = AB \times AC \tag{i}$$

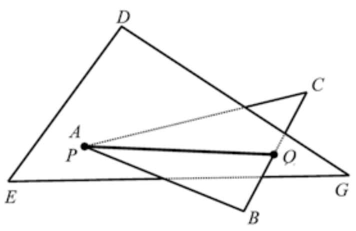
If Eq. 1 is satisfied, compute the point of intersection P.



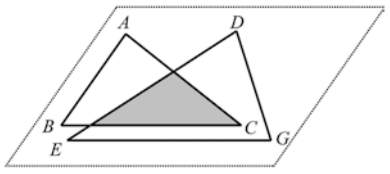
(a) Type 1



(b) Type 2



(c) Type 3



(d) Type 4

Fig. 5 Intersection between two triangular facets; **a** Type 1; **b** Type 2; **c** Type 3; **d** Type 4

$$P = tE + (1 - t)D \tag{j}$$

where $t = d/(d-e)$, with $d = AD \cdot \overrightarrow{N}$ and $e = AE \cdot \overrightarrow{N}$

Finally, P is inside triangle ABC if

$$(AB \times AP) \cdot \overrightarrow{N} \geqslant 0, (BC \times BP) \cdot \overrightarrow{N} \geqslant 0, (CA \times CP) \cdot \overrightarrow{N} \geqslant 0 \tag{k}$$

Similarly, the second point where triangle ABC cuts triangle DEG can be determined by considering the intersection between triangle DEG with the edges AB, BC and CA of triangle ABC. Let Q be the second point where triangle ABC cuts triangle DEG, then the line segment PQ will be the intersection between the two triangles (Fig. 5). An alternative method to calculate the intersection points and line segment is to use signed volumes [6, 24].

For the intersection of a pair of triangular facet, four general cases can be identified as follows:

1. The intersection point P is inside triangle ABC as shown in Fig. 5a.
2. The intersection point P is on an edge of triangle ABC as shown in Fig. 5b
3. The intersection point P is on an vertex of triangle ABC as shown in Fig. 5c
4. The intersection is more than one point and is a common planar domain as shown in Fig. 5d.

Intersections are first classified into one of these four types, and will be treated accordingly. This classification enables a consistent treatment for the same intersection on the two surfaces.

Forming intersecting chains/loops by TNOIT

As explained before, the intersection between surfaces can be represented by chains or loops of line segments. Along each intersection line segment the intersecting triangles on each surface are neighbours to one another. Making use of this neighbouring relationship, an intersection line can be constructed by tracing neighbouring triangles one after the other.

In the neighbour tracing process, the type of intersection will indicate to which neighbour the intersection line will extend. The following gives the details how neighbouring triangles are traced when the intersection point is 1. inside the triangle, 2. on an edge, 3. at a vertex and 4. at a planer zone of intersection.

1. If the intersection point is inside a triangle then there is no need to trace its neighbour. As shown in Fig. 6a

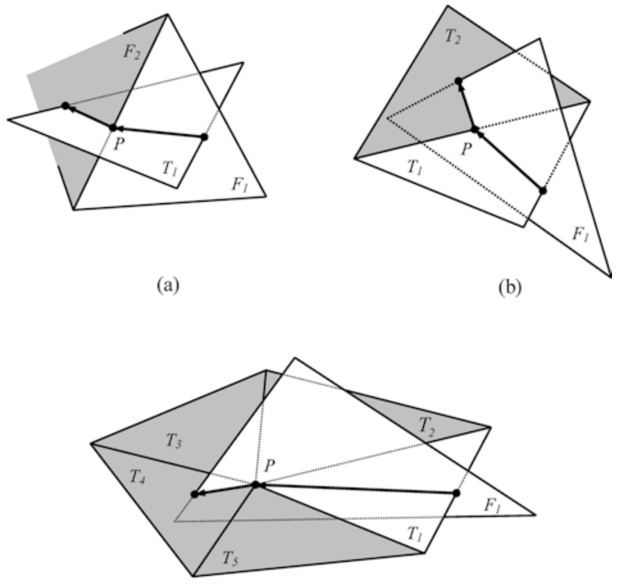


Fig. 6 The construction of the intersection line by the tracing of neighbours

for intersection type 1, intersection point P is inside triangle T_1. In this case, there is no need to trace for neighbour of T_1, Instead, P is on the common edge of F_1 and F_2, hence by neighbour tracing the intersection line continues into F_2.

2. As shown in Fig. 6b, the intersection point P is on an edge of T_1. The triangle having this common edge with T_1 is T_2. Hence, the intersection line grows from T_1 to neighbouring triangle T_2.
3. If the intersection point is on a vertex of triangle T_1 then all the triangles connected to this node have to be considered. Consider the intersection between triangles F_1 and T_1 as shown in Fig. 6c. The intersection point P is on a vertex of triangle T_1 then triangles T_2, T_3, T_4, T_5 having P as common vertex will be examined for intersection with triangle F_1.
4. If the two intersecting triangles are on a common plane, there is no need to compute the intersection. In other words, intersection will only be calculated if the two triangles concerned are not on a common plane, in which case they can be again classified into 1, 2 or 3 as described above. However, if the boundary edge is involved, the intersection has to be calculated along the boundary part as shown in Fig. 7. Two triangles are on a common plane, MN is a boundary edge of one surface and PQ will be the intersection line segment between T_1 and F_1. Figure 8 shows the intersection between a planar surface with an internal opening (Fig. 8a) and a cylindrical surface with a closed bottom (Fig. 8b). Two intersection loops are shown in Fig. 8d. The outer loop is the result of the intersection between the vertical cylindrical surface and the flat surface. The inner loop is the intersection between the inner boundary of the flat surface and the bottom plate of the cylinder (Fig. 8c).

Intersection lines can be constructed by linking up individual line segments to form open chains or closed loops. In the tracing process, if the intersection line goes back to the starting point, a closed loop will be formed; otherwise, a chain can be defined with the end points on surface boundary. Figure 9 shows the loop of intersection between two spheres shown in Fig. 4a. There are 114 line segments in the loop as shown in Fig. 9b. Figure 9c and 9d show the intersection loop and neighbouring triangles on the two surfaces. Figure 10 shows the intersection of two open surfaces. The intersection line in the form of an open chain terminates on the boundary of the hemisphere. An intersection chain can be defined in the neighbour tracing process when no neighbour can be found on a boundary edge.

When two groups of surfaces intersect, the intersection may contain more than one loops/chains as shown in Fig. 11. In this case, the intersection loop/chain can be traced out one by one. To improve efficiency, pairs of candidate triangles already considered for intersections are flagged. In the construction of a new chain/loop, what we need is the first intersection segment, and the

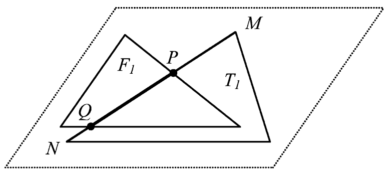


Fig. 7 An intersection line segment with the boundary edge involved

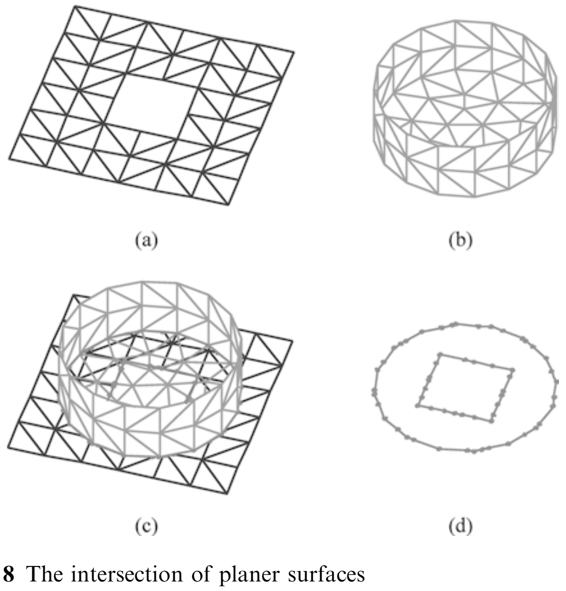


Fig. 8 The intersection of planer surfaces

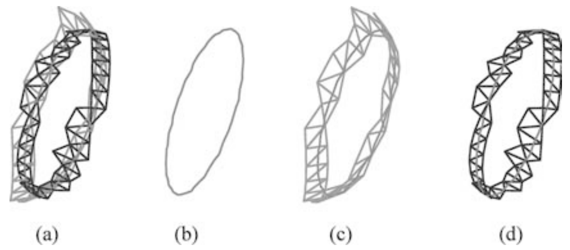


Fig. 9 The intersection loop and neighbouring triangle relationship

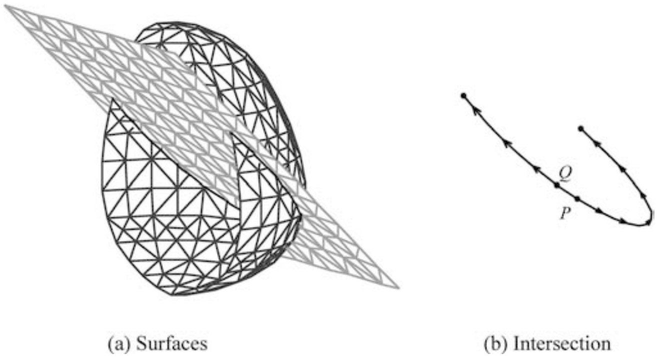


Fig. 10 The intersection chain from the intersection of open surfaces; **a** Surfaces; **b** Intersection

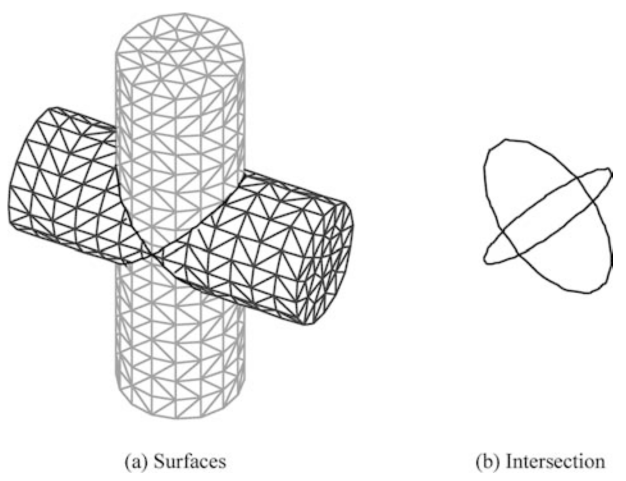


Fig. 11 The intersection of cylindrical surfaces; **a** Surfaces; **b** Intersection

line of intersection can be easily determined by neighbour tracing. When all pairs of candidate triangles are flagged indicating there are no more intersections, the entire process terminates.

Time complexity and memory management

Let's examine the time complexity for all the processes involved in the surface intersection algorithm.

1. *Establishing the neighbouring relationship of the elements*. The neighbouring relationship can be established by scanning through all the elements once, hence this is a process of linear time complexity.
2. *Define the background grid and determine the elements in a bin*. This can be achieved by determining the bins intersected by each element. As the bins are contiguous and sequential, this is also a process of linear time complexity.
3. *Intersection between elements*. This is a more a complicated process depending on the element distribution and the amount and the characteristics of intersection. Only a rough estimate of time complexity will be given.

Let M_1 and M_2 be, respectively, the number of elements in group 1 surfaces and group 2 surfaces. For even element distribution in a grid of N bins, average number of elements in a bin is given by $n_1 = \frac{M_1}{N}$ for group 1 surfaces, and $n_2 = \frac{M_2}{N}$ for group 2 surfaces.

The number of combinations for intersection between elements within a bin:

$$n = n_1 n_2 = \frac{M_1 M_2}{N^2} \tag{l}$$

If we have to look into all N bins for intersections, then the total number of operations is

$$n_t = \frac{M_1 M_2}{N^2} N = \frac{M_1 M_2}{N} \tag{m}$$

Assume $M_1 = M_2 = M$, and suppose N is a fraction of M say 10%, then

$$n_t = \frac{M^2}{0.1M} = 10M \tag{n}$$

That is, if elements are evenly distributed, or made into such a case by advanced spatial subdivision techniques, then the search for intersection is of linear time complexity. However, other measures could and have been taken to speed up the element intersection process in general and in the worst situation.

a. The min/max comparison will be applied between a group of elements and two elements before rigorous intersection tests are performed.
b. Intersection chain/loop is recovered by neighbour tracing and all intersected elements are flagged and deleted from the list of candidate elements. In fact, all intersections can be recovered by locating only one intersection point for each intersection line.

For the examples of different characteristics, the time taken for searching for neighbours, the construction of background grid and surface intersection are more or less the same, indicating the linear time complexity of all these process. Moreover, the number of surface components in a surface group will have little influence on the computational cost, as this information is never used and individual surface components need not be extracted or identified in the entire intersection process.

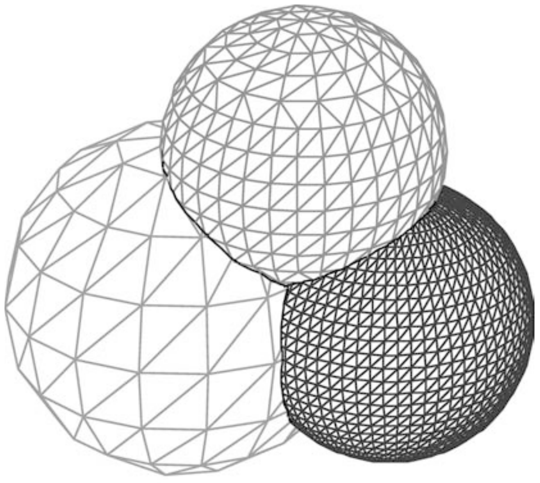
The main additional memory requirement is for the construction of the background grid. An efficient method for recording the list of elements cutting each bin is to make use of a pointer. Suppose the background grid consists of N bins; then, a single vector $\mathbf{L}$ is needed to store the information of elements in each bin.

For i = 1,N, elements in bin i = $\{\mathbf{L}(j), j = P_i + 1, P_{i+1}\}$

Auxiliary pointer vector P is of size $N+1$, and the size of $\mathbf{L}$ equal the total number of intersections between elements and bins. For instance, if each element intersects with three bins on average, then size of $\mathbf{L}$ equals three times of the number of elements.

Mesh generation along the intersection line

Based on [8], by keeping the features of the intersection line, the nodes on the intersection lines are reposition according to local element size. Elements cut by the intersection line are removed, and the void as a result of element removal is meshed by the proper connection of nodes between the intersection line and the surface boundary. While this method is simple, additional care is needed to maintain the original geometry of the surface during mesh generation. In this paper, a more robust method is proposed ensuring that elements generated are always on the triangulated surface. The steps are summarised as follows.



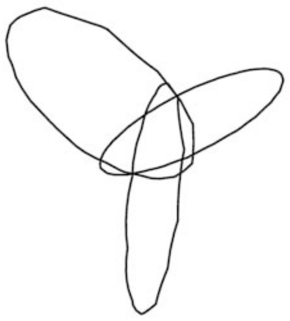
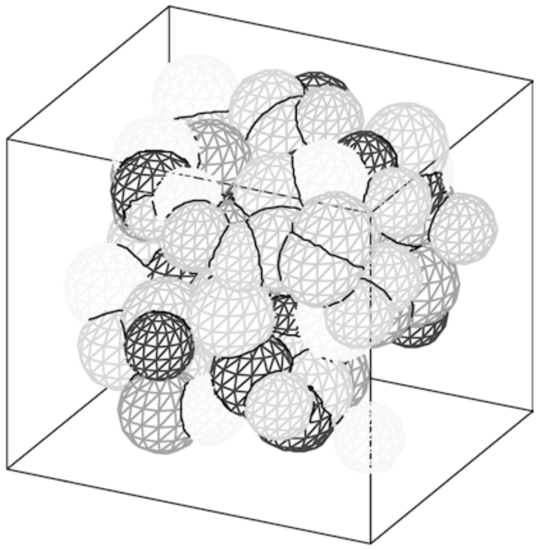


Fig. 12 The intersection of surfaces discretised into elements of different sizes; **a** Three surfaces of 4448 triangles; **b** Three intersection loops of 422 segments



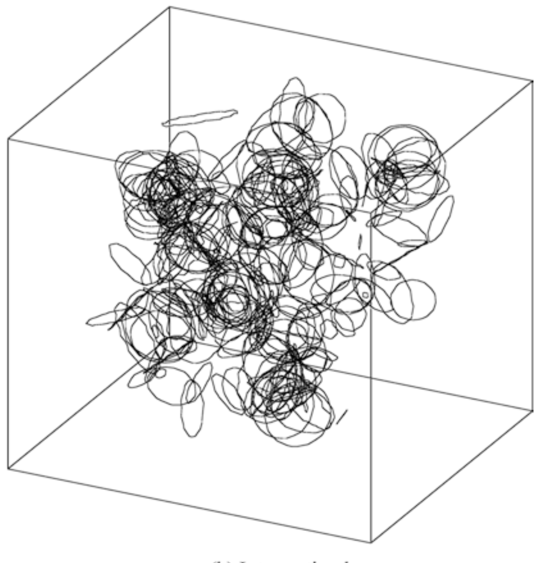


Fig. 13 100 randomly placed spheres and their intersection lines; **a** 100 spherical surfaces **b** Intersection loops

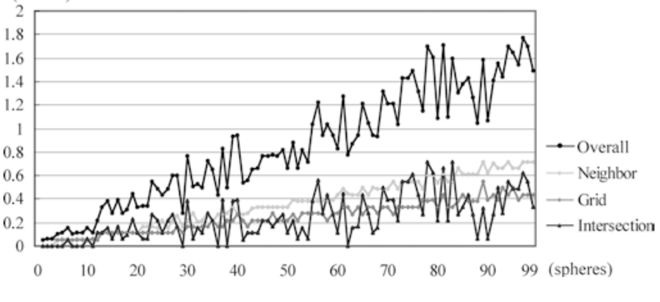


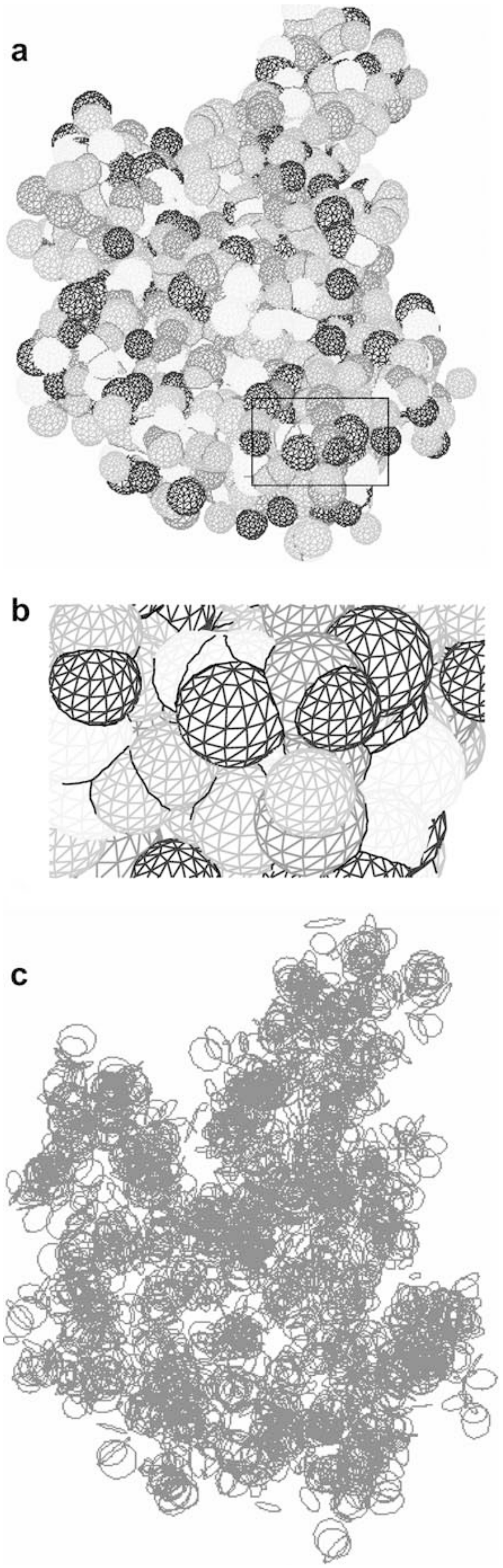
Fig. 14 CPU time plots for the intersection of 100 spheres

1. *Insert nodes on the intersection line to each surface separately.*

This process is easy and fast as elements associated with each intersection node are recorded in the neighbour tracing process. The intersection node can be classified into three cases according to its position relative to the element. 1. The node is at the interior of the element—the element is divided into three triangles. 2. The node is on an edge of the element—the edge will be divided and two new elements are generated. 3. The node is close to a vertex of the element—the vertex is shifted to the node and no element is generated.

2. *Local mesh modification.*

Excessive nodes are removed or repositioned along the chain. Elements near the intersection chain are improved by node repositioning and a swap of diagonals. In fact, all the techniques employed to improve a planar triangular mesh can be used here. However, for all the optimisation procedures, due cares have to be exercised to maintain surface features and surface curvatures, and this can be achieved simply by verifying the geometry of the patch of elements involved in the process.



◀

Fig. 15 **a** 1000 randomly placed spherical surfaces; **b** A magnified view of Fig. 15a; **c** Intersection loops for the intersection of 1000 spheres

Examples

Five examples of various surface characteristics are given in this section to demonstrate how intersections are determined by using the neighbour tracing technique on

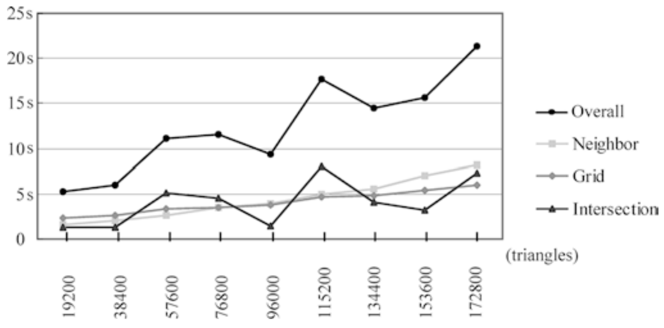


Fig. 16 CPU time for the intersection of 1000 spheres

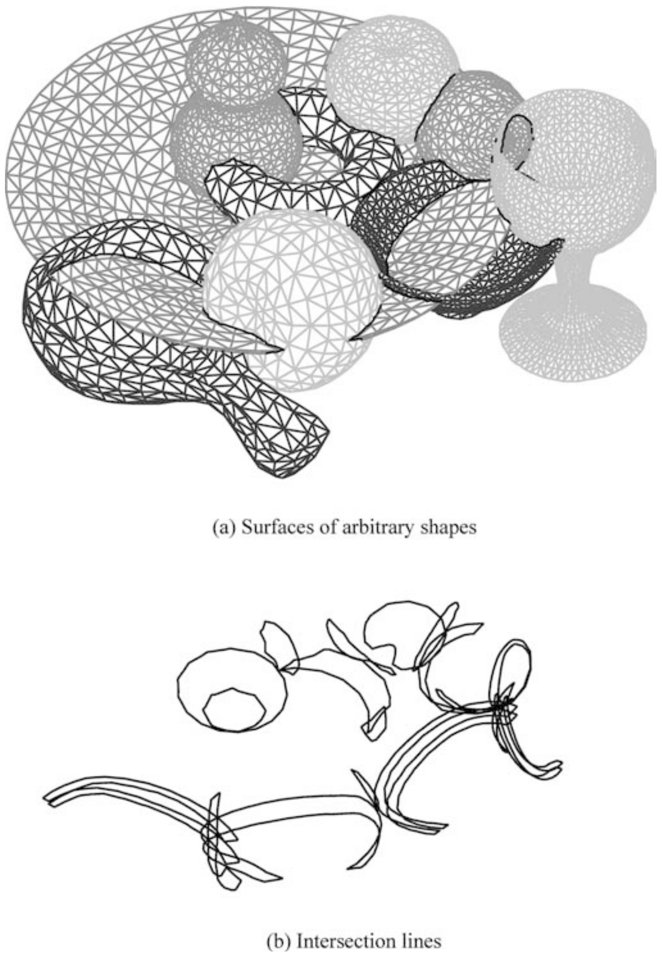


Fig. 17 The intersection of surfaces of arbitrary shapes; **a** Surfaces of arbitrary shapes; **b** Intersection lines

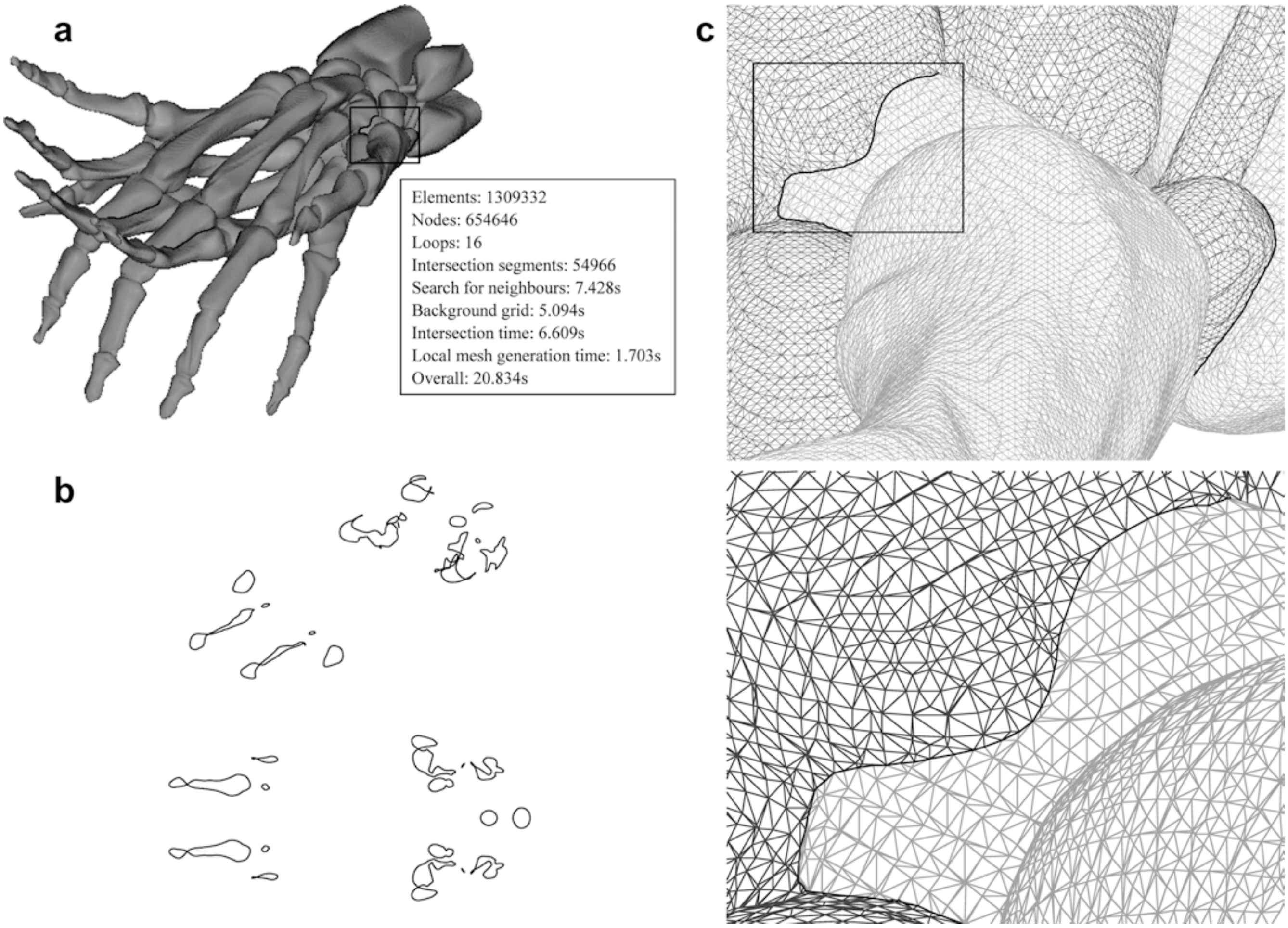

a
Elements: 1309332
Nodes: 654646
Loops: 16
Intersection segments: 54966
Search for neighbours: 7.428s
Background grid: 5.094s
Intersection time: 6.609s
Local mesh generation time: 1.703s
Overall: 20.834s
b
c

Fig. 18 **a** Hands, courtesy of the Stereo Lithography Archive at Clemson University. Elements: 1309332, nodes: 654646, loops: 16, intersection segments: 54966, search for neighbours: 7.428s, background grid: 5.094s, intersection time: 6.609s, local mesh generation time: 1.703s, overall: 20.834s; **b** The intersection loops; **c** Local views

a relatively slow PC with a CPU speed of 150 MHz and 128 RAM.

The first example consists of three spheres having 192, 672 and 3584 triangular facets, respectively, as shown in Fig. 12a. The sizes of the triangles on the sphere are made different to test the proposed algorithm. The intersection consists of three intersection loops in 422 line segments as shown in Fig. 12b.

The second example consists of 100 spheres randomly placed in a box one by one as shown in Fig. 13. Each sphere consists of 192 triangular facets. The intersection of two spheres is first considered, and the resulting surface consists of 384 triangles. Then the third sphere is added, and this process is repeated until the last sphere is placed in the box. At the final stage, the updated surface of 19008 triangles intersects with the last sphere and the resulting surface contains 19200 triangles. For the intersection of these 100 spheres, there are 13539 intersection line segments from which 277 loops can be traced out. Figure 14 shows the statistics of the CPU times for the major processes in the determination of intersections. In Fig. 14, the CPU times for calculating neighbours, for the determination of candidate triangle by background grid, for calculating intersection loops by neighbour tracing and the overall CPU time are shown by four separate graphs. From the graphs, the CPU time increases when the number of sphere increases, and a quasi-linear relationship can be observed. The local fluctuation is probably due to the random nature of the intersecting spheres.

1000 spheres are studied in the third example as shown in Fig. 15a. These 1000 spheres were divided into 10 groups of 100 spheres each. A group of 100 spheres was put in a box at a time until all 10 groups of spheres were processed. A magnified view for the intersection between the spheres is shown in Fig. 15b. The intersection of these spheres consists of 138807 intersection line segments which form 1965 loops as shown in Fig. 15c. The CPU times for the various stages of the intersection process are shown in Fig. 16.

Example 4 deals with the intersection of surfaces of various geometrical shapes as shown in Fig. 17a. These surfaces were divided into nine groups having 416, 576, 900, 1088, 1102, 1776, 2260, 2688 and 4864 triangles, respectively. The intersection consists of 22 intersection loops in 2608 line segments as shown in Fig. 17b.

Example 5 is the most complicated. Two triangulated surfaces constructed from a surface scan were downloaded from the website http://lodbook.com/models/ as shown in Fig. 18a. There are 1309332 elements in the two triangulated surfaces, in which there are many pointed triangles connecting points from scanned images to model the finger bone geometry. As shown in Figure 18b, 54966 intersection segments in 16 loops are recovered, and the total CPU time required is 20.83 s. Figure 18c shows the magnified view of the hands. It can be seen that intersection lines have already been incorporated into the surfaces by a local mesh modification procedure. After mesh optimisation, elements of size and shape compatible with those on the surfaces are generated along lines of intersection.

Discussions and conclusions

An algorithm based on tracing by neighbours TNOIT for the determination of intersection lines has been proposed in this paper. The reliability of the process is greatly enhanced as intersections are classified into four general types, based on which a clear direction as how the intersection line should progress following a neighbouring relationship can be determined. While TNOIT ensures a consistent treatment of intersection on the surfaces concerned, the determination of intersections is also tremendously speeded up, as the search for intersection is kept to a minimum using the neighbourhood information.

A simple procedure has also been included in identifying neighbours. This is a linear process, and the working arrays used in finding neighbours can also be reused later in the construction of background grid to save computer memory.

In order to speed up the process of detecting intersections, the use of a background grid is proposed. In this way, the search for intersections is localised to within cells, each of which contains, in general, only a few triangulate facets from each group of surfaces. The determination of triangular facets cutting the background cells is also a linear process, hence little overhead is added in the intersection procedure.

Based on the tracing by neighbour technique, the intersection line segments will form loops/chains following a natural order. This can save a lot of searching work, but more importantly, it enhances the overall reliability of the process as undetermined degenerated and branching cases can all be dealt with in a consistent manner.

In the largest example, the intersection of 654666 triangles with 654666 triangles took about 20.83 seconds as shown in Fig. 18. The graphs of CPU time plot show that the method is very efficient and a quasi-linear relationship is expected for large problems involving many surfaces and triangles.

Acknowledgements Financial support from the Committee of Research and Conference Grants for the project "Automatic mesh generation of 3D hexahedral elements" is highly appreciated.

References

1. Löhner R (1996) Regridding surface triangulations. J Comp Phys 126(1):1–10
2. Akkiraju N, Edelsbrunner H (1996) Triangulating the surface of a molecule. Disc Appl Math 71(1–3):5–22
3. Heiden W, Schlenkrich M, Brickmann J (1990) Triangulation algorithms for the representation of molecular surface properties. J Comp-Aid Mol Des 4(3):255–269
4. Laug P, Borouchaki H (2000) Automatic generation of finite element meshes for molecular surfaces. In: Proceedings of the European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS 2000), Barcelona, Spain, September 2000
5. Laug P, Borouchaki H (2001) Molecular surface modeling and meshing. In: Proceedings of the 10th International Meshing Roundtable, Newport Beach, CA, 7–10 October 2001
6. Shostko AA, Löhner R, Sandberg WC (1999) Surface triangulation over intersecting geometries. Int J Num Meth Eng 44:1359–1376
7. Bruyns CD, Senger S (2001) Interactive cutting of 3D surface meshes. Comp Graph 25(4):635–642
8. Lo SH (1995) Automatic mesh generation over intersecting surfaces. Int J Num Meth Eng 38:943–954
9. Owen SJ (1998) A survey of unstructured mesh generation technology. In: Proceedings of the 7th International Meshing Roundtable, Dearborn, MI, 26–28 October 1998
10. Shephard MS, Georges MK (1991) Automatic 3-dimensional mesh generation by the finite octree technique. Int J Num Meth Eng 32(4):709–749
11. Lo SH (1988) Finite element mesh generation over curved surfaces. Comp Struct 29:731–742
12. Lau TS, Lo SH (1996) Finite element mesh generation over analytical surfaces. Comp Struct 59(2):301–309
13. Anastasiou K, Chan CT (1996) Automatic triangular mesh generation scheme for curved surfaces. Comm Numer Meth Eng 12:197–208
14. Canann SA, Liu Y-C, Mobley AV (1997) Automatic 3D surface meshing to address today's industrial needs. Fin Elem Anal Des 25:185–198
15. Cass RJ, Benzley SE, Meyers RJ, Blacker TD (1996) Generalized 3-D paving: an automated quadrilateral surface mesh generation algorithm. Int J Num Meth Eng 39(9):1475–1489
16. Hartmann E (1998) A marching method for the triangulation of surfaces. Vis Comp 14:95–108
17. Borouchaki H, Laug P, George P-L (2000) Parametric surface meshing using a combined advancing-front generalized Delaunay approach. Int J Num Meth Eng 49:233–259
18. Cuillière JC (1998) An adaptive method for the automatic triangulation of 3D parametric surfaces. Comp-Aid Des 30(2):95–161
19. Lee CK, Hobbs RE (1998) Automatic adaptive finite element mesh generation over rational B-spline surfaces. Comp Struct 69(5):577–608
20. Lee CK (1999) Automatic metric advancing front triangulation over curved surfaces. Eng Comp 16:230–263
21. Cebral JR, Löhner R, Choyke, Yim PJ (2001) Merging of intersecting triangulations for finite element modeling. J biomech 34(6):815–819
22. Cristovão L, Coelho G, Gattass M, de Figueiredo LH (2000) Intersecting and trimming parametric meshes on finite element shells. Int J Num Meth Eng 47:777–800
23. Bonet J, Peraire J (1991) An alternating digital tree (ADT) algorithm for geometric searching and intersection problems. Int J Num Meth Eng 31:1–17
24. Aftosmis MJ, Berger MJ, Melton JE (1998) Robust and efficient Cartesian mesh generation for component-based geometry. AIAA J 36(6):952–960
25. Lo SH (1998) Analysis and verification of the boundary surfaces of solid objects. Eng Comp 14:36–47