Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXVII, págs. 3093-3115 (artículo completo)
Alberto Cardona, Mario Storti, Carlos Zuppa. (Eds.)
San Luis, Argentina, 10-13 Noviembre 2008

A ROBUST ALGORITHM TO DETERMINE SURFACE/SURFACE INTERSECTION IN BOTH PARAMETRIC SPACES

Fábio G. Teixeira[a] and Guillermo J. Creus[b]

[a]*Grupo Virtual Design, Pgdesign – Programa de Pós-Graduação em Design da Universidade Federal do Rio Grande do Sul, Osvaldo Aranha, 99/408, Porto Alegre, Brasil, fabiogt@ufrgs.br http://www.vid.ufrgs.br*

[b]*Grupo CEMACOM, Universidade Federal do Rio Grande do Sul, , Osvaldo Aranha, 99/408, Porto Alegre, Brasil, creus@ufrgs.br*

Keywords: Intersection, Parametric Surfaces, CAGD, Subdivision.

Abstract. An algorithm providing the intersection curves in the parametric space of both involved surfaces is presented allowing the correct union of trimmed patch surfaces to represent complex models and the generation of finite element meshes. The algorithm has four steps. On the first one, a subdivision method is used to obtain an adaptive quadtree structure of surface regions where potentially intersection curves segments can be contained. On the second one, each element of this quadtree structure is approximated by triangles; the intersection segments of triangle pairs are determined as an initial approximation of intersection curves in 3D space. On the third step, a refinement process and parametric mapping of coordinates provides the intersection points on the parametric and real spaces. In the last step, the intersection segments are reordered to obtain intersection curves in parametric form. Several examples are included to check the robustness and efficiency of the algorithm.

1 INTRODUCTION

Intersection between surfaces is a recurrent subject in computer graphics, three-dimensional geometric modeling (solids and surfaces) and computer aided design (CAD). In many applications in computer aided geometric design (CAGD) it is necessary to determine the intersection curves between two surfaces: finite element mesh generation over surfaces and solids, determination of silhouette curves of surfaces, Boolean operations, construction of blend surfaces, interference and collision detection and scientific visualization. Intersection determination has special importance in engineering, especially in automotive and aerospace industries, where computational simulation is massively used in all project stages.

This paper presents an intersection algorithm for C^1 parametric surfaces that provides the intersection curves with high precision and maps them into the parametric space of each surface. This facilitates the manipulation of trimmed surfaces using parametric sub-domains and can be used to assemble complex models and to generate finite element meshes on them.

The content of the paper is as follows. In Section 2 a critical review of previous works is given. The proposed algorithm is described in Section 3 and its four steps are detailed in Section 4 to 7. In Section 8 examples are presented to show the versatility and efficiency of the method and, in Section 9, the conclusions are given.

2 PREVIOUS WORKS

The intersection problem between surfaces is a complex subject that has been an active research field for more than three decades. Several approaches exist: some are dependent on the type of surfaces involved, while others are intended for special applications. According Hoschek and Lasser (1993), a good intersection algorithm should have the following characteristics:

- Numerical Precision – compatible with the application;
- Robustness – to determine all intersection curves, loops and singularities, independently of the surface type and position;
- Speed – compatible with the application;
- Self-control – the algorithm should not require any help from user for the correct execution.

These are conflicting characteristics. Thus, in the development of a surface intersection algorithm, an appropriate balance should be sought among these requirements in agreement with the intended application and practical usefulness. In their works, Hoschek and Lasser (1993), Krishnan *et al.* (1994) and Andrade (1998) describe five groups of algorithms types to compute intersection curves between surfaces: analytical, lattice, continuation, marching, and subdivision algorithms. Techniques that combine characteristics of different categories are designated hybrid algorithms.

Analytical methods try to solve the intersection problem through the analytical solution of the equation:

$$\|\mathbf{F} - \mathbf{G}\| = 0 \tag{01}$$

where F and G are given surface equations in vectorial form. When the surfaces are described through implicit formulations, the problem leads to a system of nonlinear equations, which can be solved by a numerical method, as Newton-Raphson, differential geometric methods (Asteasu, 1988) or algebraic techniques (Owen *et al*., 1987). In the case of parametric formulations, it is necessary to do the conversion to the implicit formulation through algebraic

transformations (Sederberg, 1987). Except for the simplest surfaces, these algorithms have high computational cost, hindering practical applications (Hoschek and Lasser, 1993).

Lattice evaluation techniques reduce the degree of complexity of the intersection problem by finding the intersections of isoparametric lines on one of the surfaces with (determined using constant values of the parametric coordinates) the other surface (Barnhill *et al.*, 1987). Points on the intersection curve are calculated by the solution of a nonlinear system of the type $F(u,v_i)=G(s,t)$, using numerical techniques. The parameter $v=v_i$ should define a sufficiently dense set of isoparametric lines on the surface $F(u,v)$ (Hoschek and Lasser, 1993). The accuracy of these methods depends on increment applied in v_i and the numerical techniques applied to solve the nonlinear systems.

Continuation methods use systems of differential equations, obtained from the parametric equations of the surfaces and their geometric characteristics; the equations are solved by numerical techniques. Performance of the method depends on the initial approximation and the complexity of the intersection curves. Problems as singularities and ramifications should be treated by specific means. Patrikalakis (1991) and Abdel-Malek and Yeh (1996) use continuation techniques in their work.

Marching methods use incremental progression along the intersection curve. These methods need starting points on the intersection curve to obtain new points evaluating the tangent direction of the curve. One of the critical stages in these methods is the search for starting points that can be found through other intersection methods, such as the subdivision method (Barnhill and Kersey, 1990; Andrade, 1998), or through specific algorithms. The marching process is another critical stage. Some works use an approximation of the tangent vector of the intersection curve to determine the marching direction. Stoyanov (1992) approximates the intersection curve locally by a parabola, which represents a Taylor expansion of the intersection curve about the current point up to the second order, generating linear equations systems. Another approach is the use of an osculating circle (Wu and Andrade,1999).

Subdivision algorithms (Lane and Riesenfeld, 1980) divide the two surfaces in parts to determine which among them intersect. The intersection points can be found by linear approximation, considering that the resultant subdivision patches are almost flat. So, the intersection problem is reduced, locally, to the case of plane/plane intersection. Proposals differ according to how subdivisions are made, and how the intersections are computed on each patch. The ending criteria for the subdivision process constitute another important aspect of the subdivision algorithms. These algorithms involve three steps: recursive subdivisions of surfaces to reach a certain level (that depends on the process), determination of the points of intersection curves, and reordering of these points to form the intersection curves. Each step has specific characteristics. Several approaches for the subdivision stage can be found in the literature. The first algorithms of this type made uniform subdivisions along all of the surfaces (Griffiths, 1975). It was expensive and difficult to use. To reduce memory and time processing, non-uniforms quadtrees are used to subdivide the surfaces in the parametric space and bounding boxes (bounding volumes) that involve each corresponding patch in 3D space. Patches whose bounding volumes intersect can contain segments of the intersection curve; other patches can be discarded, restricting the search places of the intersection curves. This is the divide-and-conquer principle. Patches where there is overlapping of bounding volumes are subdivided in the two surfaces and the process is repeated: the interference is verified and the patches without occurrences are eliminated. This process is denominated non-uniform adaptive subdivision. After several successive subdivisions, the group of remaining bounding volumes involves tightly and completely the intersection curves. Some algorithms use the size of the remaining bounding volumes as an ending criterion for the subdivision process. Other

algorithms use the local curvature of surfaces as an ending criterion (Houghton *et al.*, 1985). The subdivision process continues, for those patches where there is interference, until the ending criteria are satisfied.

The way to compute the bounding volumes varies. Gleicher and Kass (1992) use interval arithmetic (IA). Figueiredo (1996) adapts the algorithm proposed by Gleicher-Kass using affine arithmetic (AA) (Comba and Stolfi, 1993). The convergence speed is higher and the number of subdivisions is lower as compared with IA. The bounding volumes can be, basically, of two types: parallel to the global axes, min/max boxes (Houghton *et al.*, 1985), or oriented by the corresponding patch on surface (tight bounding volumes). Construction and interference computing for min/max boxes are simple. Tight bounding volumes (Houghton *et al.*, 1985; Barth and Huber, 1999; Huber, 1998) can improve the convergence speed, but have high computational cost; the work of Barth-Huber (1999) mentions a reduction of only 7% in the total processing time. The intersection curve can be obtained by a linear approximation of the patches remaining after the subdivision process. These patches are divided in triangles that are always flat.

Intersections between triangles from patches whose bounding volumes overlap determine segments that are a piecewise approximation of the real intersection curves; some additional refinement may be necessary to obtain the desired precision. It is also possible to obtain the intersection points directly on the patches, joining its central points, once the precision criteria have been satisfied. Such criterion uses the distance between the centers of closest patches of the two surfaces (Andrade, 1998).

The main advantage of the subdivision algorithms is its total independence of surface type as adaptive subdivision satisfies the local geometric properties for any surface type; moreover starting points are not needed. These reasons were decisive in the choice of this algorithm type for the first step of the proposed algorithm.

3 THE PROPOSED ALGORITHM

The intersection method adopted in this work uses adaptive subdivision controlled by the local curvature of the surfaces, which are subdivided in successive steps until no patch has flatness lower than a chosen tolerance. In this way it is possible to reduce the intersection problem, locally, to the intersection between two planes. This method is independent of type and shape of involved surfaces, and shape and complexity of the intersection curves. These factors are important to ensure the robustness and generality of the method, important characteristics in CAGD. The processing time, that is pointed as a negative factor for this type of method (Andrade, 1998), can be optimized with the use of appropriate criteria in the subdivision, in the interference computation, and in the intersection points determination, as described below.

On the other hand, the use of adaptive subdivision does not ensure an appropriate and uniform accuracy level along the intersection curves, problem that is critical when the surfaces possess variable curvature, as usual in freeform surfaces. Such limitation is solved using a refinement algorithm, to provide uniform precision along all of the intersection curves. The algorithm provides the points of the intersection curves mapped in parametric coordinates on the two involved surfaces. The precision, relative to the maximum dimension, used in the examples is 10^{-12}. The definition of the intersection curves in the parametric space turns possible the determination of sub-domains on the surfaces, making easier the process of mesh generation on trimmed surface patches.

The algorithm developed involves four steps:

- Adaptive Subdivision – the surfaces are subdivided successively until all active

patches become almost flat;
- Intersection between patches – the intersection segments are calculated in the 3D space;
- Refinement of the results – the results are refined and mapped on the surfaces in the parametric space;
- Segments reordering – the consecutive segments are connected to create continuous curves of intersection.

4 ADAPTIVE SUBDIVISION

The adaptive subdivision implemented uses a non-uniform quadtrees structure to subdivide the surfaces. The goal is to determine in both surfaces the potentially places where intersection can exists. As the subdivision level enlarges, the surface elements flatness increases, allowing approximating locally the surfaces by an appropriate triangulation. Thus, the first approximation of intersection curves will be the set of intersection segments between triangle pairs.

The algorithm uses a data structure where information on the surface elements and their bounding boxes are stored. The overlapping between bounding volumes is computed at each iteration eliminating parts whose bounding volumes do not overlap. The following present the basic concepts and main steps of subdivision algorithm.

4.1 Flatness of a surface patch

The flatness of a surface element is a concept that allows evaluating the curvature degree of the element. The flatness is defined here as the smaller cosine of the angles among the normal vectors in the vertices and the center of the patch and is used to control the assembling of the bounding boxes and the subdivision processes. The flatness of an element *i* ($\hat{F}_i$) is computed by the internal product among normal vectors:

$$\hat{F}_i = Min\left(\frac{N_i}{\|N_i\|}.\frac{N_{ij}}{\|N_{ij}\|}\right)_{j=1..4} \tag{02}$$

where $\mathbf{N}_i$ is the normal vector in the center of element and $\mathbf{N}_{ij}$ is the normal vector in each vertex of the element. $\hat{F}_i$ is the smallest value of the cosines. Therefore, a flat element will have $\hat{F}_i = 1$. The largest admissible angle ($\hat{A}_{Max}$) among normal vectors corresponds to minimum admissible $\hat{F}_i$. $\hat{A}_{Max}$ is fixed angular tolerance and is expressed by

$$\hat{A}_{Max} = \cos^{-1}\hat{F}_{Min} \tag{03}$$

where $\hat{F}_{Min}$ is the minimum $\hat{F}_i$ from all elements considered.

4.2 Computing the bounding volume

Bounding volume definition is determinant for accuracy and performance of the subdivision process. The surface oriented bounding boxes result in fewer subdivisions, but time for the calculating and detecting non-intersection of surface oriented bounding boxes is far exceeds the time used by axis aligned bounding boxes (Sabharwal, 1994). Thus, this work uses axis aligned bounding box (AABB) due to its low computational cost.

Each bounding box must enclose the correspondent surface element that is assured by a local recursive subdivision method that uses the flatness of each sub-element as control parameter. When an appropriate tolerance is used, this method computes bounding box with good precision and low computational cost. To compute the bounding box of a surface

element three steps are followed:

- The surface element is subdivided while the flatness of each sub-element is greater than a tolerance;
- For each sub-element, compute the extreme 3D coordinates of its vertices;
- The final bounding box is defined by extreme 3D coordinates among all its sub-elements.

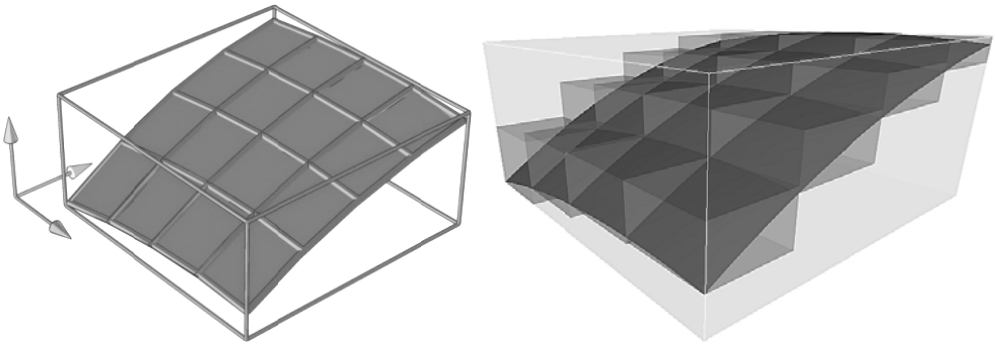


Figure 1: Scheme to build a bounding box. A surface element is subdivided in sub-elements with minimum tolerable flatness.

The accuracy is controlled by angular tolerance that defines the minimum flatness for each sub-element of a surface element. For the examples shown in this work 2.5^{o} angular tolerance is used. This is half of the angular tolerance of the subdivision process, which is five degrees and gives precision enough to compute intersections for there examples with a performance compatible with interactive applications. Figure 1 shows how the algorithm builds a bounding box for an element of a parametric surface.

4.3 Bounding box interference verification

The interference test between two axis aligned bounding boxes is a simple task. The bounding boxes are projected as rectangles (see Fig. 2) on each coordinate plane (*xy*, *yz*, *xz*). The interference on two bounding boxes occurs only when exist interference between its projections on any projection plane; in fact, only the bounding box coordinate limits need to be compared:

- $[x_1, x_2]_{BV1} \cap [x_1, x_2]_{BV2}$
- $[y_1, y_2]_{BV1} \cap [y_1, y_2]_{BV2}$
- $[z_1, z_2]_{BV1} \cap [z_1, z_2]_{BV2}$

In case of interference, the links to the corresponding surface elements are included in the quadtree data structure for each surface.

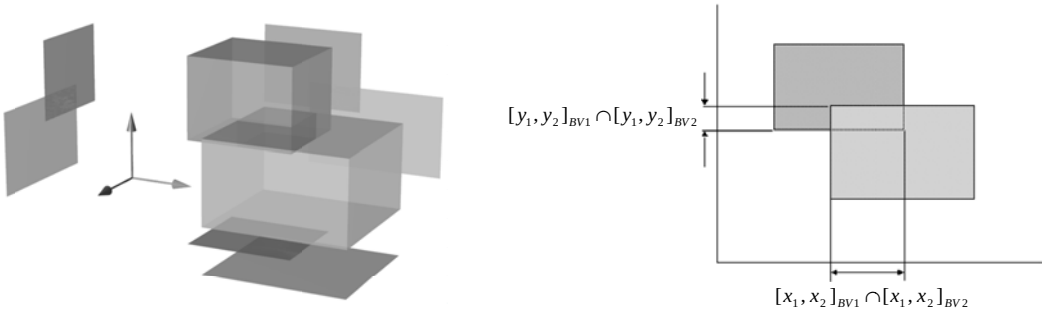
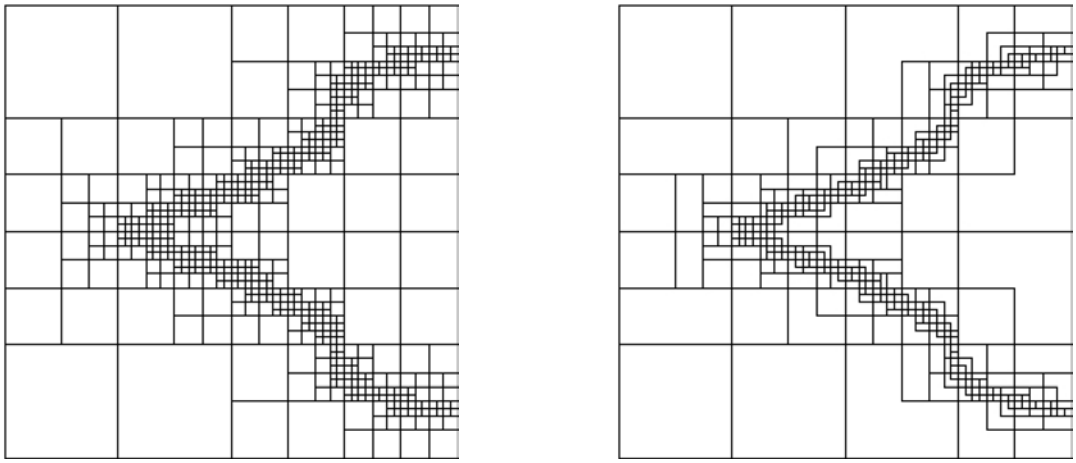


Figure 2: Bounding boxes interference computation through its projections. The intersection of interval coordinates of the two bounding boxes in the three dimensions is verified.

4.4 Subdivision of a surface element

Wherever the flatness of an active surface element is lower than the given tolerance, the element is subdivided. Each subdivided element creates four children with the same parametric dimensions that inherit the links from their mother. In parametric space the children elements have half dimensions of their mother patch. Finally, the mother element is removed from data structure and the children elements dataset is computed and added to the quadtree data structure.

A cross verification is made between the children elements and the elements linked to their mother. The child element that interferes with no element linked to its mother is eliminated avoiding unnecessary verifications between elements with false links; this additional step increases the convergence rate of the algorithm. Figure 3 shows how this procedure reduces the number of active cells in quadtree.



(a) (b)

Figure 3: Surface subdivision in parametric space made: a) without and b) with additional verification of false links.

4.5 The subdivision algorithm

The subdivision process is controlled by the surface elements flatness and their bounding box interferences determine whose elements pairs where intersection is potentially possible. Thus, in the same iteration subdivisions are made in all elements pairs whose bounding boxes overlap and whose flatness are small than specified tolerance. Then, the quadtree data structure is updated to compute the new interferences. The process is repeated until the flatness tolerance is reached for all elements. The algorithm has the following steps:

i. The two surfaces are subdivided, initially, in 2x2 meshes, resulting in four elements with the same parametric dimensions. The dataset from each element is added to database quadtree. In this initial step, all elements from one surface are linked to all elements from other surface.

ii. The overlapping bounding boxes are computed following their links and the elements whose bounding boxes are overlapping are linked, updating database. Only the data for these elements, which are potential candidates to contain segments of intersection curves, remain in the database.

iii. The flatness of all elements in the data structure is computed. The elements whose flatness is less than a given tolerance are subdivided. Their children inherit links to the elements on the other surface and substitute their mothers in the database. *No more than one level of subdivision difference is allowed between two adjacent elements.*

iv. If there is an element whose flatness is less than the tolerance, the algorithm returns to the step *ii* otherwise the algorithm stops.

The angular tolerance used in the examples is five degrees (5^o). Numerical experiments performed show that smaller angular values turn the computational cost high without significant increase in accuracy and increase the number of intersection points. Besides, it is necessary to do a refinement process to assure precision and homogeneity to the results, as is described in Section 6. This tolerance (5^o) is smaller than that used by Houghton *et al.* (1985), which was 20^o, and it is enough for a good initial precision and also to find all the intersection curve segments.

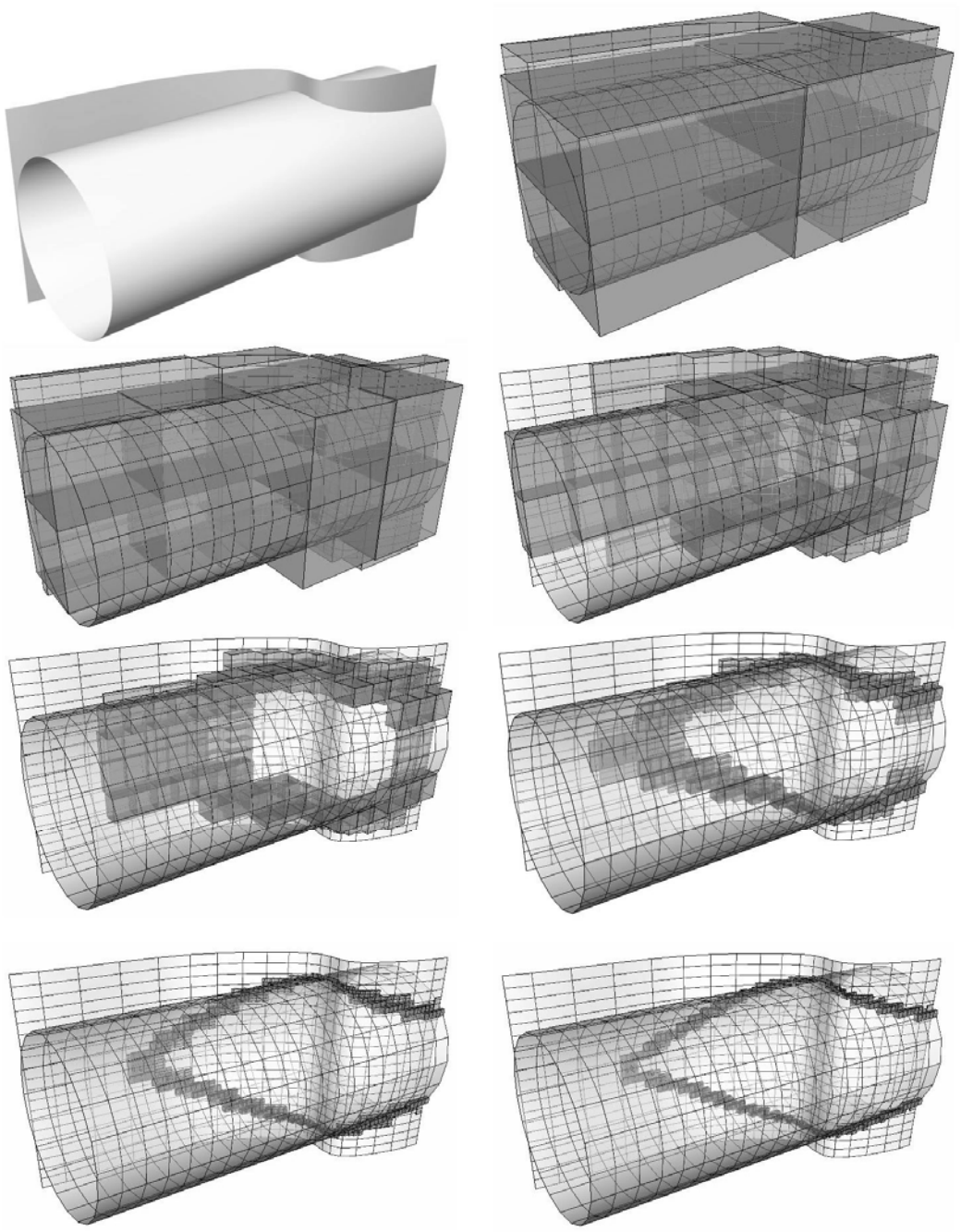


Figure 4: Example 1: Sweep surface and revolved surface. Figure shows subdivision process with bounding boxes along seven iterations in 3D space.

Figure 4 shows the example 1 where two surfaces have intersection and seven steps of the recursive subdivision process in 3D space. The bounding volumes of both surfaces and their overlaps are shown for each iteration step.

Figure 5 illustrates how the subdivision algorithm works for this example showing the quadtrees for each surface in parametric space along the same seven iterations. The patches shaded are overlapping; only these will remain in the database for the next iteration. The number of iterations depends on the surface shapes but in general, five to nine iterations are enough. At the end of this step two groups of patches linked with each other are obtained. In the next step, the intersection curve segments will be computed.

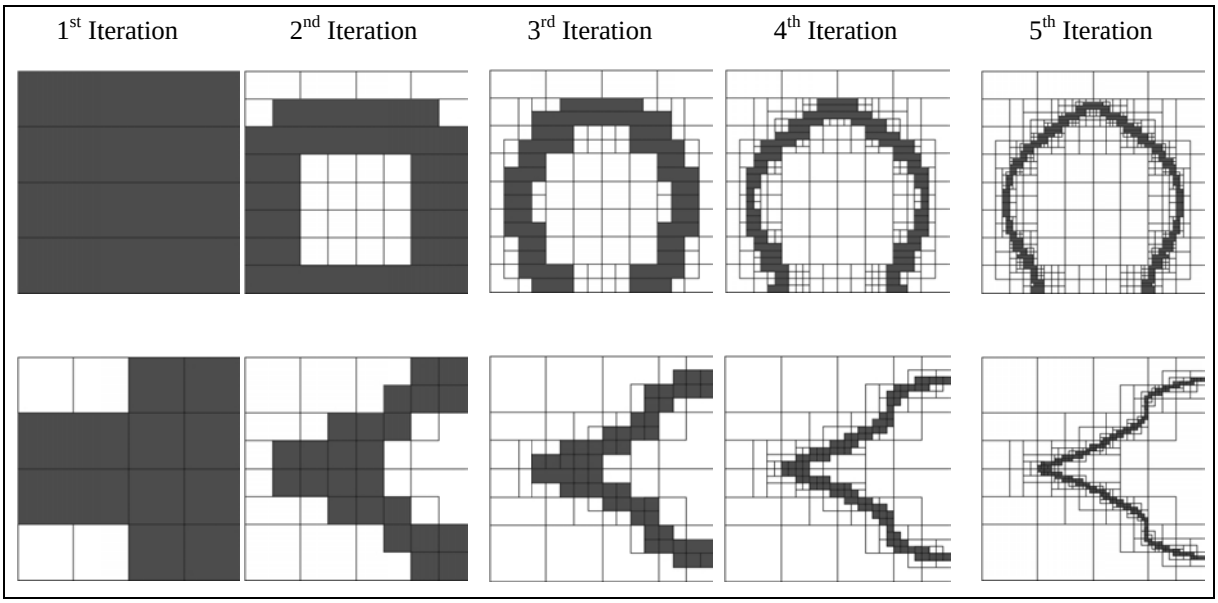


Figure 5: Illustration of the subdivision process along seven iterations in parametric spaces of the two surfaces.

5 COMPUTING INTERSECTION SEGMENTS IN 3D SPACE

In this step a first approximation of the intersection segments is computed. To this effect, the intersections in 3D space between the two linked element groups are determined. Each element is divided in triangles to guarantee the surface local flatness. The pattern of triangular division will depend on the quadtree topology of each surface that should be appropriate to assure the continuity of the intersection curves. The intersection between two elements is computed by the intersection among all triangles from one and all triangles from the other element.

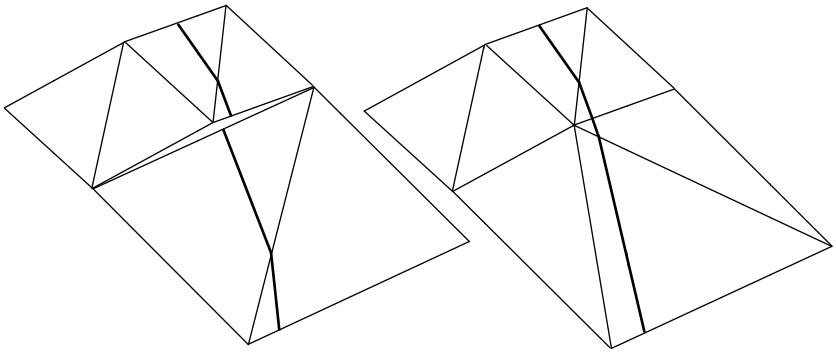


Figure 6: Linked elements are divided in triangles according quadtree topology to determine the intersection segments in 3D space.

5.1 Division in triangles

The domain subdivision obtained in the previous stage is not uniform and elements with different number of neighbors can occur. In this case, gaps could arise between the calculated segments (Fig. 6). Thus, before an element is subdivided into triangles, it is necessary to compute the number of neighbors for each edge to determine the correct topology for the triangulation. This number is restricted to one and two, because the subdivision algorithm does not allow transitions of more than one level. Five different configurations exist according to element neighborhood topology.

When the four edges of an element have one neighbor, it is subdivided in two triangles (figure 7a). When one edge of the element has two neighbors, it is divided in three triangles. In this case, a vertex is created in the middle of the edge that has two neighbors (figure 7b). If two edges have two neighbors, these can be adjacent or opposed. In both cases four triangles are created, but the topological configurations are different (figure 7c-d). Finally, if three edges have two neighbors, five triangles are created with three new intermediary vertexes (figure 7e). The situation where the four edges have two neighbors doesn't happen here because in this case the element is already subdivided in the previous step. These five configurations assure a mesh without gaps.

Each segment of the intersection curve results from the intersection between two triangles from two surface elements. A single element can have several segments depending on the number of linked elements and the number of triangles in each one (the quadtree topology). As each element can contain up to five triangles, the number of intersections among triangles can rise to twenty-five. If an element has four links, the number of intersections between triangles can increase up to one hundred.

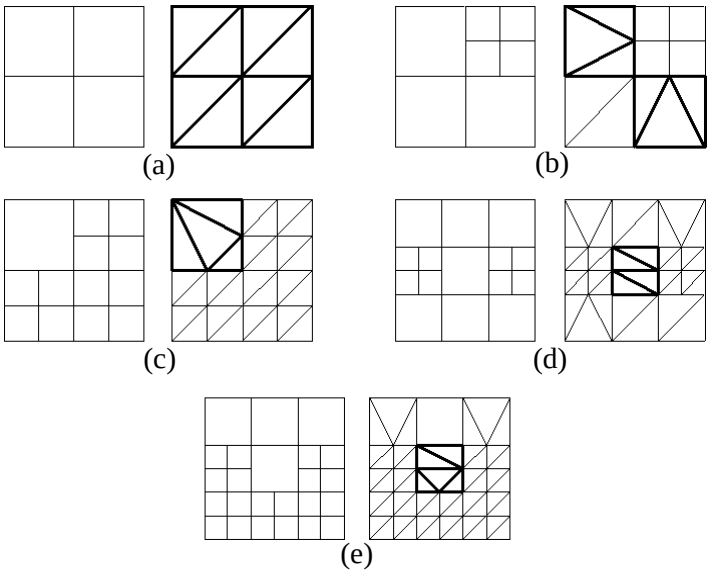


Figure 7: Topologies of triangle subdivision. a) All edges have only one neighbor. b) One edge has two neighbors. c) Two adjacent edges have two neighbors. d) Two edges opposed have two neighbors. e) Three edges have two neighbors.

5.2 Intersection of two triangles in the 3D space

The intersection between two triangles (T_1 and T_2) in 3D space is computed by two basic stages. First, the algorithm determines the intersection points of the triangle T2 with the T1 plane. The second stage uses topology criteria to compute a valid segment between the two triangles.

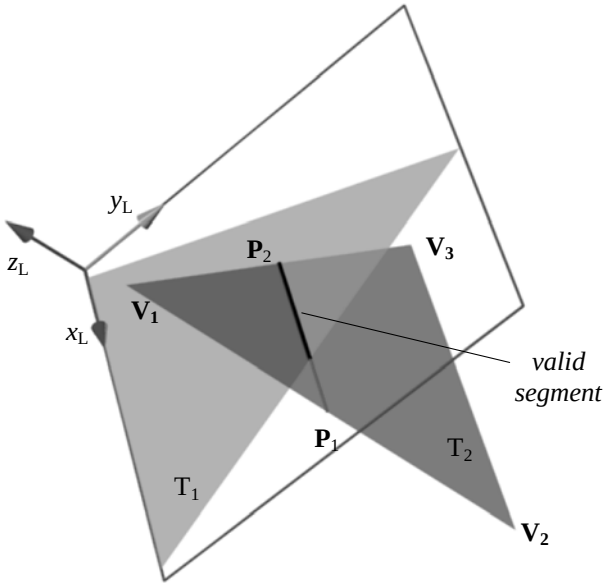


Figure 8: Intersection between two triangles (T_1 and T_2) in 3D space.

The main steps are described for triangles T_1 and T_2 (see Fig. 8):

i. A Cartesian coordinate system (x_L, y_L, z_L) is attached to T_1. The plane of the triangle is coincident with the local *xy* plane.
ii. The vertexes of T_2 are referred to this new coordinate system.
iii. If the sign of z_L coordinates of T_2 vertexes are different, then T_2 has intersection with the T_1 plane, but not necessarily with T_1.
iv. The intersection points between T_2 and the T_1 plane should have z_L coordinates equal to zero. Two points, $\mathbf{P}_1$ and $\mathbf{P}_2$, result from intersection of the two edges of T_2 with the T_1 plane (or local *xy*).
v. If any part of the segment defined by points $\mathbf{P}_1$ and $\mathbf{P}_2$ is inside T_1, then there is intersection between T_1 and T_2.
vi. A valid segment is computed.

5.3 Determination of the two points with *z* = 0 (step iii)

The parametric representations of the two edges of T_2, which cross *xy* plane, are used to compute $\mathbf{P}_1$ and $\mathbf{P}_2$:

$$\mathbf{L_1}(t) = (1-t).\mathbf{V_1} + t\mathbf{V_2} \tag{04}$$

$$\mathbf{L_2}(u) = (1-u).\mathbf{V_1} + u\mathbf{V_3} \tag{05}$$

where t where $\mathbf{P}_1$ and $\mathbf{P}_2$ are found with values of *t* and *u* that make z=0. This operation is shown below:

$$0 = (1-t)z_1 + tz_2, \cdots t = \frac{z_1}{z_1 - z_2} \tag{06}$$

$$0 = (1-u)z_1 + uz_3, \cdots u = \frac{z_1}{z_1 - z_3} \tag{07}$$

where z_1, z_2, and z_3 are the local z of T_2 vertexes. Valid values of t are in the [0,1] interval due the parameterization chosen. The x_L and y_L coordinates are computed with the t and u values in the corresponding parametric equations. Thus, $\mathbf{P}_1$ and $\mathbf{P}_2$ are obtained analytically with exact values.

5.4 Determination of a valid intersection segment (step vi)

If there is intersection between the planes, it is still necessary to verify if there is intersection between the two triangles. A valid intersection segment must belong to both triangles. Again, the parametric equations of the segments are used to solve the problem. Four segments are analyzed: the three edges of T_1 and the segment defined by $\mathbf{P}_1$ and $\mathbf{P}_2$. The intersections between the segment ($\mathbf{P}_1\mathbf{P}_2$) and the edges of T_1 are computed. The result is a parameter t of the intersection point for each segment. This allows the evaluation of the segment position in relation to T_1. $r_i(t_i)$ and $r_j(t_j)$, with j varying from 1 to 3, are respectively the parametric equations of the intersection segment and the edges of T_1. The intersection between r_i and r_j is expressed by

$$t_{ik} = Int(r_i, r_k), \text{ and } t_{il} = Int(r_i, r_l) \tag{08}$$

and the intersection between r_j and r_i is expressed by

$$t_{ki} = Int(r_k, r_i), \text{ and } t_{li} = Int(r_l, r_i) \tag{09}$$

where Int(.,.) is a function that determines the value of the parameter at the intersection point and r_k and r_l are the two edges of T_1 for $0 \leq t_{ki} \leq 1$ and $0 \leq t_{li} \leq 1$; this is the necessary but not sufficient condition to exist an intersection between the two triangles. When two values of t_{ji} are out of the interval [0,1], which is the parametric bounds, there is no intersection between T_1 and T_2 (Fig. 9). The same occurs when two values of t_{ij} are out of bounds [0,1] (Fig. 9b).

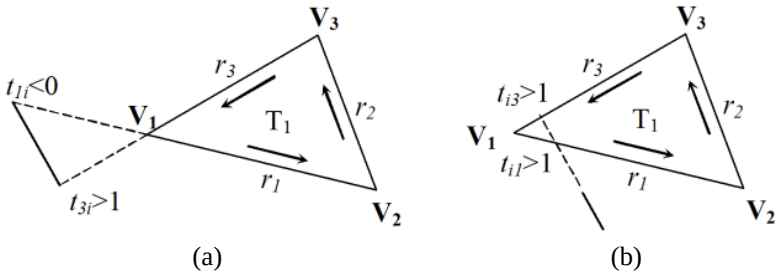
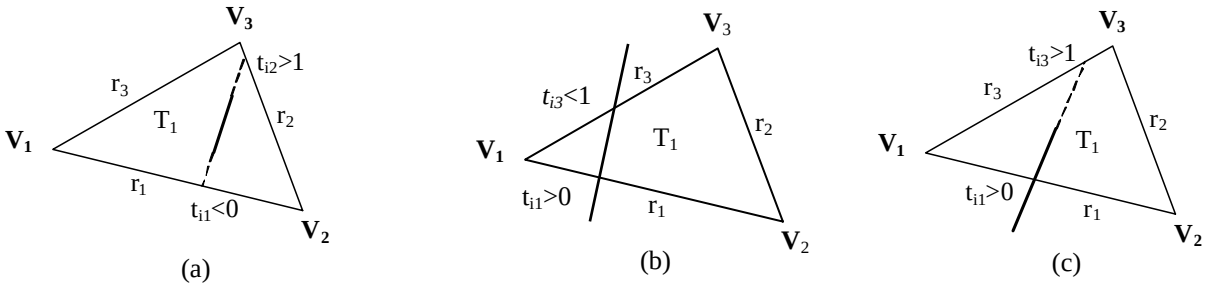


Figure 9: When the values of t_{ji} and t_{ij} are out of bounds [0,1] there is no intersection.

Still, t_{ij} and t_{ik} are analyzed to determine if the segment intersection is out of T_1, totally inside T_1 or partially inside it. If there is intersection, one of the three following situations take place:

i. $t_{il} > 0$ and $t_{ik} > 1$ – The segment is inside T_1 (Fig. 10.a);
ii. $0 < t_{ik} < 1$ and $0 < t_{il} < 1$ – Two endpoints out of T_1 (Fig. 10.b);
iii. $0 < t_{ik} < 1$ and ($t_{il} > 1$ ou $t_{il} < 0$) – One endpoint out of T_1 (Fig. 10.c).



Fiii

Figure 10: Three situations where intersection exists between T_1 and T_2: a) the segment is totally inside T_1; b) the segment is partially inside T_1, but the two endpoints are out; c) the segment is partially inside T_1 with one endpoint out.

The final intersection segment corresponds to that part that is inside T_1. Therefore, the next step is to compute the endpoint coordinates of the valid segment using the parametric coordinate t_{ij}. For situation *i* (Fig. 10a) nothing changes and $\mathbf{P}_1$ and $\mathbf{P}_2$ remain. In situation *ii* (Fig. 10b), the two points should be adjusted: $\mathbf{P}_1 = \boldsymbol{r}_i(t_{il})$ and $\mathbf{P}_2=\boldsymbol{r}_i(t_{ik})$. In situation *iii* (Fig. 10c), it is necessary to adjust one of the extremities: $\mathbf{P}_k=\boldsymbol{r}_i(t_{il})$. In this case, *k* is the endpoint (1 or 2) that is out of T_1 and t_{il} should be between 0 and 1.

Once computed all segments of intersection, the endpoints coordinates are referred to the global reference system and stored in a specific array. Segments are stored in a connectivity array that has pointers to the point array. The final result is a set of segments connected by their extremities that represent an approximation of the intersection curves between two surfaces in the 3D space (Fig. 11). These results must be refined and mapped in both parametric spaces what will be explained in the next section.

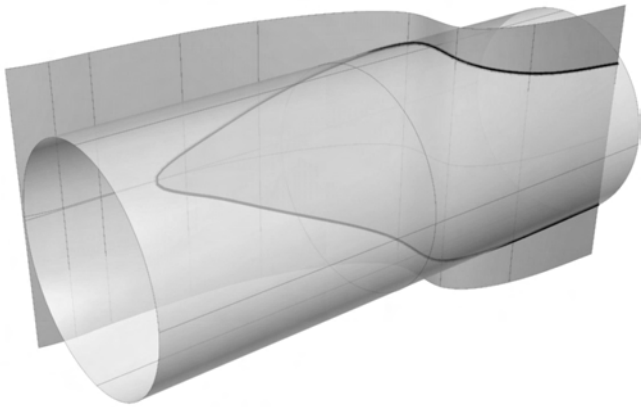


Figure 11: Intersection curve in 3D space for example 1.

6 REFINEMENT OF RESULTS AND PARAMETRIC MAPPING

The intersection points computed up to here are an approximation of intersection curves but with heterogeneous and low accuracy due to the approximation by triangulation. Thus, a refinement process was developed to increase the accuracy of results with uniformity. The algorithm developed is of Newton-Raphson type and uses the directions of the tangents on the two surfaces to improve the accuracy of the intersection points obtained in Section 5. It is based on the work of Houghton *et al.* (1985), but with important improvement because here the points are mapped in parametric coordinates in the two surfaces. The accuracy is 10^{-12} relative to maximum dimension of the involved surfaces; the points are mapped on the

surfaces with an error of 10^{-15} in parametric distance. A procedure was developed to provide this parametric mapping and assure good precision for any type of parametric surface.

6.1 Parametric mapping

The central step of the refinement process is the parametric mapping algorithm, which projects a point ($\mathbf{P}_i$) from the space onto a parametric surface. The point projection is computed in both coordinate systems: parametric and real. A generic analytical solution would depend on surface type; thus, an iterative numerical procedure that uses the tangent directions on a point of the surface is employed.

The center $\mathbf{P}_{S1}$ of the surface element is used as an initial approximation for the iterative process. The tangents on point $\mathbf{P}_{S1}$ are computed in the parametric directions (u, v) and the vector $\mathbf{V}_j$ (where $\mathbf{V}_j = \mathbf{P}_i - \mathbf{P}_{Sj}$) is projected on these tangents. These projections correct the initial approximation and the process repeats itself until the tolerance is reached. The core idea is minimize the projections length of vector $\mathbf{V}_j$ onto the tangent vectors up to the vector to coincide with the normal vector.

Considering a point $\mathbf{P}_i$ in the 3D space and a parametric surface element S_i close to $\mathbf{P}_i$, the algorithm is as follows:

i. A local conversion factor (f_{RP}) between parametric and 3D space is computed as $f_{RP} = D_P/D_R$, where D_P is parametric distance on the surface element and D_R is the equivalent 3D distance.

ii. The central point of $\mathbf{S}_i$ is computed in parametric space ($\mathbf{P}_{S1}^P$), and in 3D space ($\mathbf{P}_{S1}^P$),). This is the first approximation of $\mathbf{P}_i$ projection on to surface (j=1).

iii. The vector $\mathbf{V}_j$ is computed as $\mathbf{V}_j = \mathbf{P}_i - \mathbf{P}_{Sj}$.

iv. The tangent vectors ($\mathbf{T}_u$, $\mathbf{T}_v$) on the point $\mathbf{P}_{S1}$ of the surface are computed by finite differences with relation to u and v and normalized.

v. The vector $\mathbf{V}_j$ is projected on $\mathbf{T}_u$ and $\mathbf{T}_v$ and these projections are used to update $\mathbf{P}_{Sj}^P$:

$$\Delta_u = \mathbf{V_j}.\mathbf{T_u} \tag{10}$$

$$\Delta_v = \mathbf{V_j}.\mathbf{T_v} \tag{11}$$

$$\vec{\Delta}_p = [\Delta_u, \Delta_v] \tag{12}$$

vi. If (Δ_u and Δ_v) < Tolerance then $\mathbf{P}_S$ is $\mathbf{P}_i$ projection otherwise the new approximation is given by

$$\mathbf{P}_{Sj+1}^p = \mathbf{P}_{Sj}^P + \vec{\Delta}_p f_{RP} \tag{13}$$

$$\mathbf{P}_{Sj+1} = \mathbf{F_S}(\mathbf{P}_{Sj}^P) \tag{14}$$

vii. Return to *iii*.

At the end of the process, $\mathbf{P}_S$ is the orthogonal projection of $\mathbf{P}_i$ on the surface and $\mathbf{P}_S^p$ its parametric coordinates. The tolerance adopted in the parametric space is 10^{-15}. This high level of accuracy is needed for the convergence and precision of the refinement algorithm. Figure 12 shows how the projection is made.

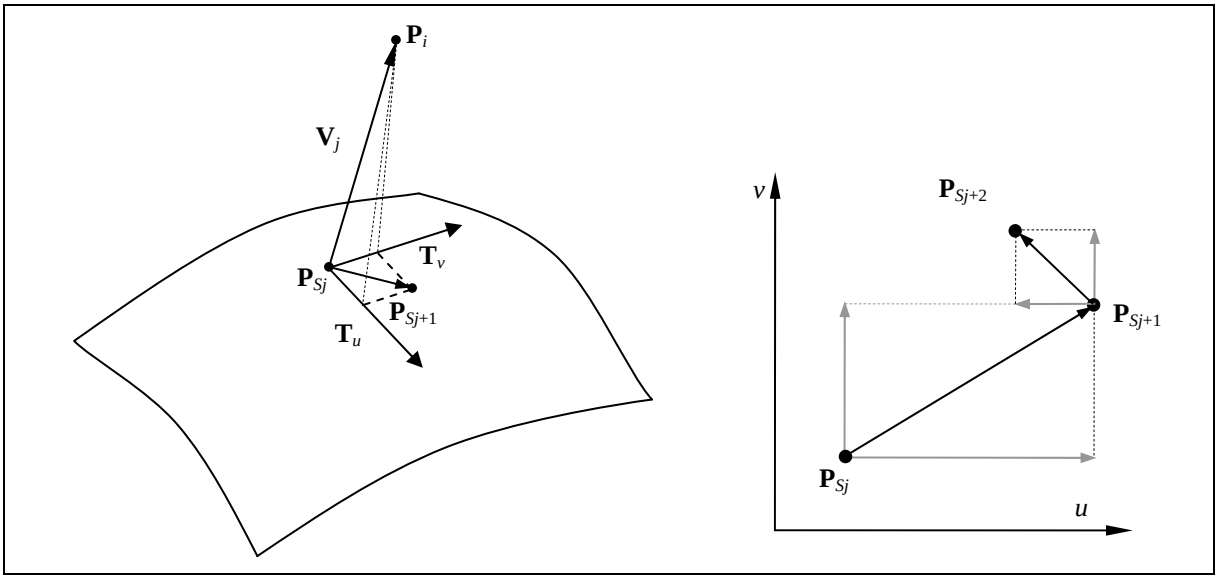


Figure 12: Projection of a point on a surface. a) Vector $\mathbf{V}_i$ being projected on tangents in $\mathbf{P}_S$. b) Correction by iteration of the coordinates of $\mathbf{P}_S$ in parametric space.

6.2 The refinement algorithm

A point in the real space projected on two surfaces, gives rise to two projections. If the point is on the intersection curve of the surfaces, the distance between these two projections vanishes. In the algorithm it is considered that a point is on the intersection if the distance between its two projections is less than a given tolerance. Whenever the distance is larger than the tolerance, an iterative process that uses the projections of the point on the two surfaces and the tangents to the surfaces in these points has to be employed.

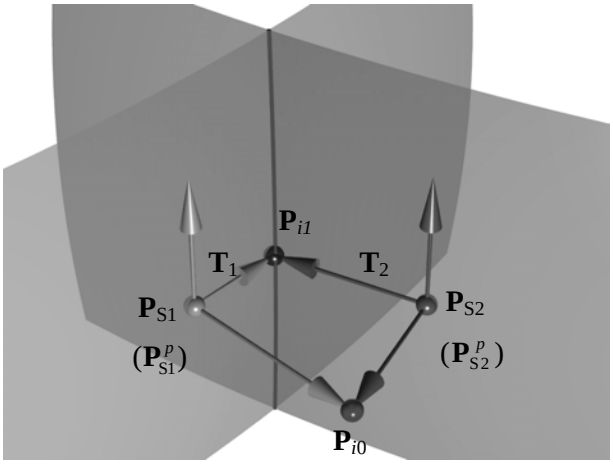


Figure 13: Refinement scheme: $\mathbf{P}_i$ is the intersection point, $\mathbf{P}_{S1}$ and $\mathbf{P}_{S2}$ are the projections on the surfaces and $\mathbf{T}_1$ and $\mathbf{T}_2$ are the tangent vectors in $\mathbf{P}_{S1}$ and $\mathbf{P}_{S2}$ in the plane defined by $\mathbf{P}_i$, $\mathbf{P}_{S1}$ and $\mathbf{P}_{S2}$.

The algorithm is described below:

i. Point $\mathbf{P}_i$ is projected on the two surfaces resulting in $\mathbf{P}_{S1}^{P}$ and $\mathbf{P}_{S2}^{P}$ in parametric coordinates and in $\mathbf{P}_{S1}$ and $\mathbf{P}_{S2}$ in real coordinates.

ii. The distance (ΔP_S) between $\mathbf{P}_{S1}$ and $\mathbf{P}_{S2}$ is computed. If ΔP_S is smaller than the tolerance, go to step *v*.

iii. The two surface tangent vectors ($\mathbf{T}_1$ at $\mathbf{P}_{S1}$ and $\mathbf{T}_2$ at $\mathbf{P}_{S2}$) are computed in such a way that they lie on the plane defined by $\mathbf{P}_i$, $\mathbf{P}_{S1}$ and $\mathbf{P}_{S2}$.

iv. The new position $\mathbf{P}_{i1}$ of point $\mathbf{P}_i$ is the intersection between lines defined by $\mathbf{T}_1$ and $\mathbf{T}_2$. Return to *i*.

v. The parametric points $\mathbf{P}_{S1}^{P}$ and $\mathbf{P}_{S2}^{P}$ are stored in arrays specific for each surface. The connectivity arrays for the two surfaces remain unchanged.

Figure 13 shows how the refinement algorithm works. The convergence is very fast when the surfaces have a uniform curvature. When the surfaces have variable curvature, the iterations increase, but generally it is less than four. A tolerance of 10^{-12} relative to the maximum dimension of the involved surfaces was adopted in this work, providing the precision needed in CAGD. The processing time in this step is dominant among the four that compose the intersection algorithm because of the great number of floating-point operations.

Following the application of the refinement algorithm, only the parametric coordinates of the intersection points on each surface are retained. The segments that are determined by these points constitute the solution of the intersection problem. Figure 14 represents, in both parametric spaces, the intersection segments obtained after the refinement with parametric mapping for Example 1 (see figures 4, 5 and 11).

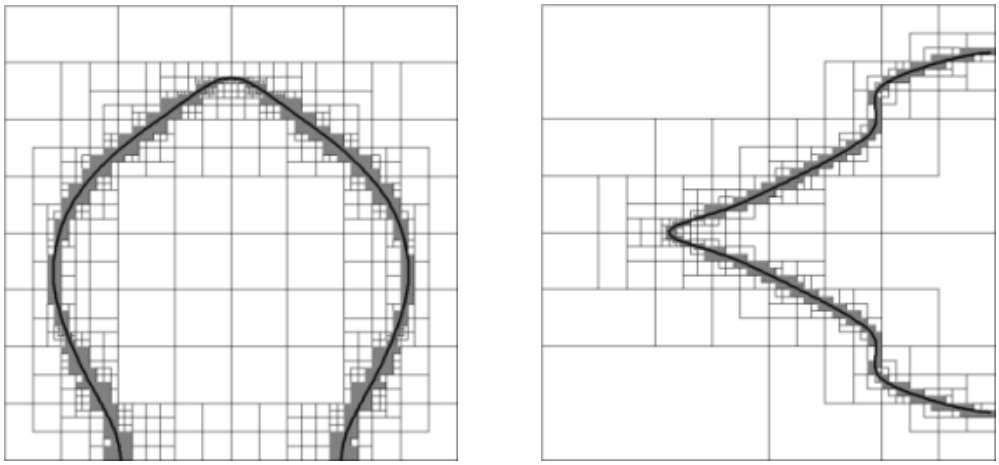


Figure 14: Intersection segments in the parametric space of two surfaces. The patches shaded are those that correspond to overlapping bounding volumes. The intersection curve is represented in the parametric spaces of both surfaces.

7 SEGMENTS REORDERING AND PARAMETRIC REPRESENTATION

The objective of this stage is to transform the set of segments into a set of successive points that defines a continuous intersection curve. The algorithm determines topology of the intersection curve based on segments topology. Thus connectivity of segments are analyzed to find the intersection curves endpoints and to identify adjacent segments. Open curves have two endpoints and closed curves have none. A point that belongs to a single segment is an endpoint. Points that belong to more than two segments are also endpoints. These endpoints are used as starting points in the assemblage of a curve. In closed curves the starting point is chosen arbitrarily.

Each point added to a curve is called pivot. The first pivot is an endpoint. The algorithm finds the point connected by a segment to this pivot. Such point is the new pivot. The process repeats until the other endpoint.

After reordering, the intersection curves are given by an ordered set of points that define straight segments in the parametric space. However, this is not the best representation for an efficient manipulation of the intersection curves, a parametric representation $\mathbf{f}(t)$ being more useful. Such representation is obtained using the real lengths of the curve in 3D space. As the

curve **f**(*t*) is unique in the 3D space, it is possible to use the same parametric representation for the curve in both surfaces. The parametric representation is determined in the following way:

i. The length of the curve (*LT*) is calculated by adding the distances between consecutive points in the 3D space.

ii. The accumulated length (LA_i) at a given point is calculated by adding the distances between all of the previous points.

iii. Parametric coordinate of a point is calculated by $t_i = \frac{LA_i}{LT}$

iv. The parametric function uses a linear interpolation in the parametric space to calculate the points on the intersection curve:

$$\mathbf{f}_C(t) = \mathbf{P}_j^P(1 - t_m) + \mathbf{P}_{j+1}^P t_m \tag{15}$$

$$t_m = \frac{t - t_j}{t_{j+1} - t_j} \tag{16}$$

Eq. 15 provides the parametric coordinates (*u*, *v*) for the curve on the corresponding surface. All the intersection curves are represented in parametric form on the both surfaces. This allows the independent manipulation of the two surfaces.

8 EXAMPLES

In this section, examples involving several types of surface are studied to test the efficiency and versatility of the proposed algorithm. Table 1 shows a summary of results for examples 1 to 15.

Example 1 (Fig. 4) was used in the description of the method.

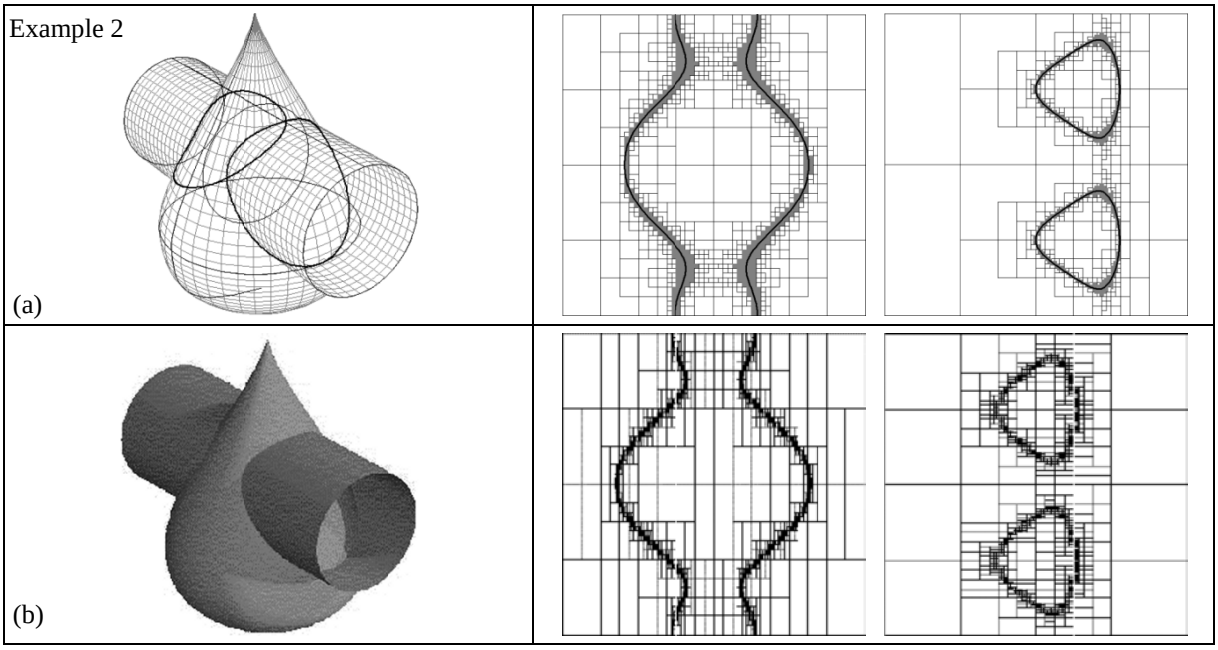


Figure 15: Example 2: two revolved surfaces. a) Proposed algorithm: S1: 568 patches and S2: 352 patches. b) Barth and Huber work: S1: 3264 patches and S2: 2947 patches.

Example 2 (Fig. 15a) was taken from Barth and Huber (1998) (Fig.15.b). The subdivision obtained is comparable, graphically, to that obtained by Barth-Huber. Their work does not use angular tolerances, but the condition: $A_{Di} = A_{D0}/10000$, where A_{Di} is the area of element i in parametric space and A_{D0} is the surface area in parametric space that leads to a great subdivision density (2947 elements). The algorithm proposed in this paper solves this intersection problem with 568 surface elements working with axis aligned bounding boxes (AABB).

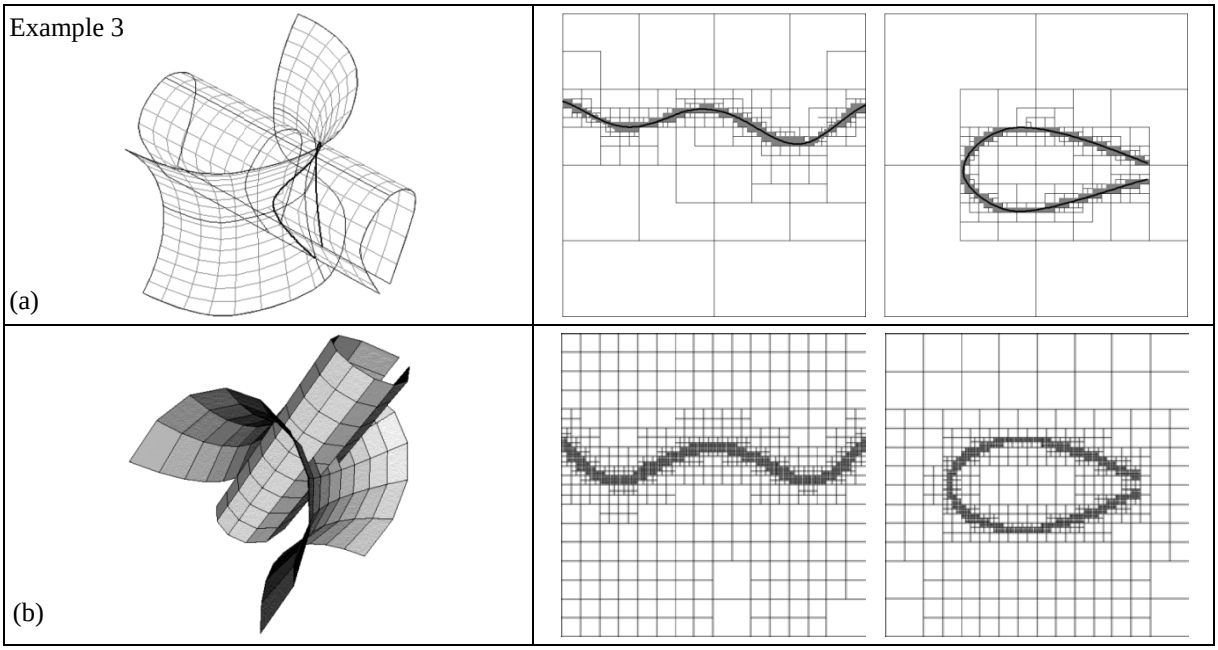


Figure 16: Example 3: two bi-cubic patches. a) Proposed algorithm: S1: 411 elements (173 remaining elements), and S2: 325 elements (133 remaining elements). b) Figueiredo algorithm: total: 3066 and remaining elements: 1280.

Example 3 (Fig. 16.a) was taken from Figueiredo (1996), Fig. 16.b, which uses affine arithmetic to compute the bounding volumes in the subdivision process. Figueiredo algorithm uses a total of 1786 elements; of those, 728 remaining elements are used to compute the intersection curve. It is mentioned that the original algorithm proposed by Gleicher-Kass (1992) needed 8038 total elements and 5508 remaining elements for same example. The algorithm described in the present paper uses 736 total elements and 306 remaining elements to compute the intersection curve for example 3.

The next examples, 4 to 15 (Fig. 17, 18), show the versatility of the proposed algorithm. Examples 4 to 7 show intersections between two cylinders. Examples 3 to 6 are typical junctions of industrial pipes. Example 7 checks the algorithm robustness solving a surface tangent problem. Examples 8 to 10 show the intersection between cylinders and other three geometric surface types: cone, torus, and sphere. Examples 12 to 15 (see Figs. 18 – 19) show intersections between surfaces with complex geometry.

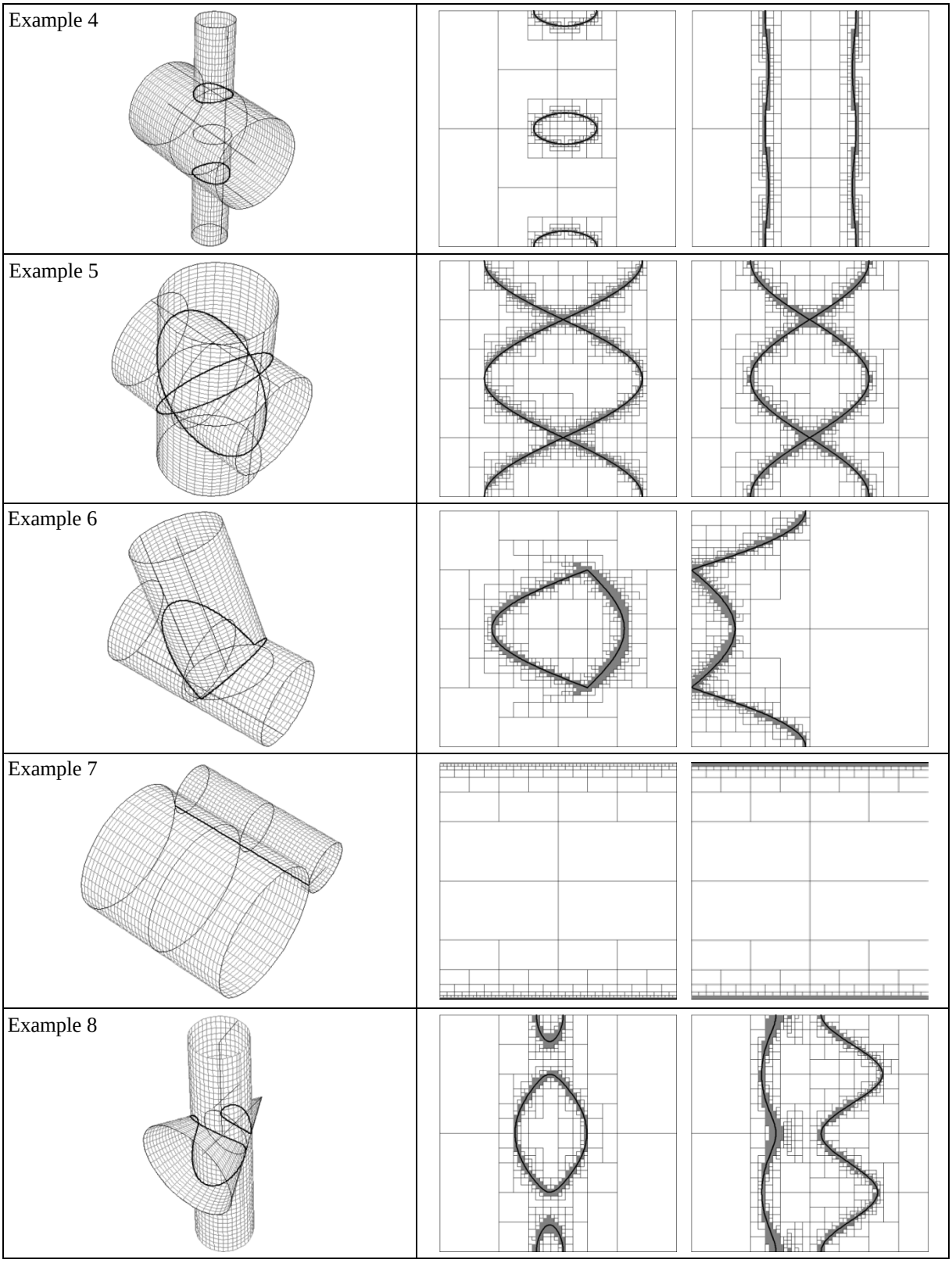


Figure 17: Example 4: two perpendicular cylinders with different diameters. Example 5: two perpendicular cylinders with same diameter. Example 6: two oblique cylinders with same diameter. Example 7: two parallel cylinders. Example 8: cone and cylinder with perpendicular axes.

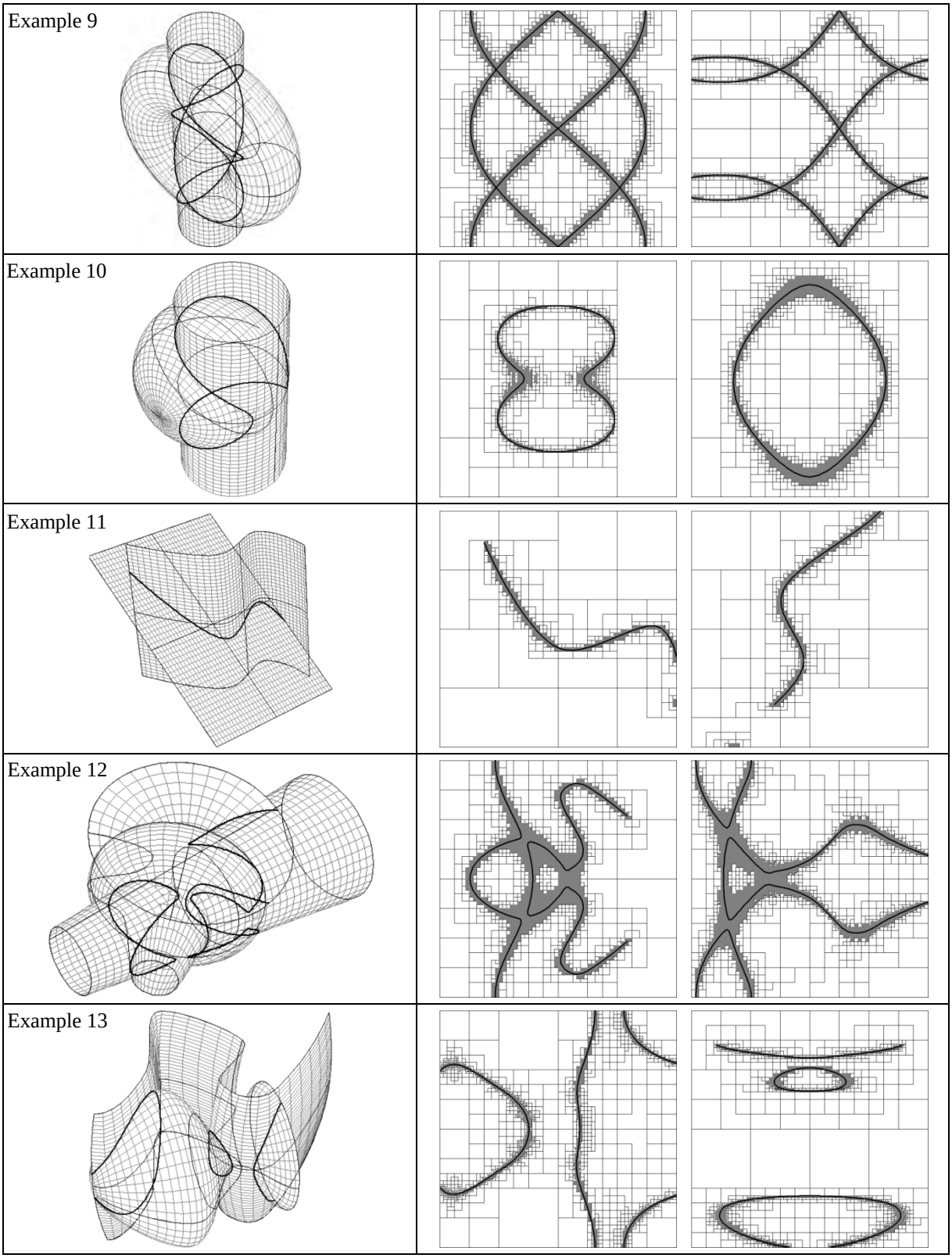


Figure 18: Example 9: cylinder and torus with same diameter. Example 10: cylinder and sphere. Example 11: plane and sweep surface. Example 12: two revolved surfaces, one is a truncated cone and the other is a spline revolved. Example 13: revolved and sweep surface. Example 14: two revolved surfaces, one is a cylinder and the other is a spline revolved. Example 15: Coons and ruled surface.

Table 1 shows the mean and maximum errors in the computed intersection points (error is defined as the relative distance between the points on the two surfaces that correspond to a single intersection point). In all examples the maximum error was less than 10^{-12} and the mean error less than 10^{-13} (except in example 3 where the mean error was less than 10^{-11}). As the usual accuracy in CAD applications is 10^{-6} to 10^{-8} the precision of the proposed algorithm is, in general, greater. The CPU time given corresponds to a Pentium IV 2.4 GHz with 512MB. Refinement is the most time-consuming step. Processing speed increases when tolerance is reduced. Example 13 needs 36 sec with tolerance 10^{-12} and 25 sec with tolerance 10^{-8}. Tolerance can be customized on implementation for the required applications, according to the needed accuracy.

	S1 patches		S2 patches		Inters.	Curves		Relative Error		CPU
Ex.	Total	Rem.	Total	Rem.	points	S1	S2	Max	Med.	Time (sec)
1	986	348	516	222	608	1	1	2.157009e-11	2.088392e-12	7
2	1116	568	1178	352	832	2	2	3.443295e-11	2.759666e-12	5
3	411	173	325	133	372	1	1	1.288117e-10	4.007308e-11	13
4	536	260	468	144	616	3	2	1.788262e-11	3.192538e-13	2.05
5	2471	1339	951	334	1242	6	6	2.154993e-11	2.166294e-12	4.57
6	658	235	1031	523	541	1	1	2.027047e-11	2.766512e-12	2.21
7	508	256	252	128	129	1	1	4.898425e-16	4.898425e-16	0.60
8	552	209	1004	442	679	3	2	2.426449e-11	1.156698e-12	3.13
9	1789	690	1772	922	1488	12	14	2.509866e-11	2.162789e-12	8.39
10	1195	416	1604	872	890	1	1	1.738579e-11	2.159538e-12	4.93
11	346	134	434	165	329	1	1	2.047152e-11	1.067454e-12	3.82
12	1468	690	2634	1412	1225	2	3	4.471619e-11	2.518051e-12	11
13	1721	622	2017	936	1472	4	3	7.022341e-10	2.883347e-11	36
14	1250	538	2856	1576	1216	2	3	1.918307e-05	7.844887e-08	11
15	1036	566	1004	544	510	1	1	4.239043e-12	7.908833e-13	5

Table 1: Results obtained with proposed algorithm applied on examples 1 to 15.

9 CONCLUSIONS

An algorithm that provides the intersection curves of two surfaces in parametric space has been described. The algorithm uses a subdivision process together with a refinement procedure. The adaptive subdivision process uses a sample points technique to compute the bounding volumes and it is controlled by the flatness tolerance; this assures that the number of intersection points will be adequate. The subdivisions converge very fast to the neighborhood of the intersection curves, as shown by the examples. The initial approximation in 3D space is refined with an algorithm that increases accuracy and projects the curves onto the two surfaces in their parametric spaces; the results are parametric curves in the parametric spaces of the two involved surfaces. The curves in the parametric space can be used to manipulate the surface geometry through sub domains. In this way, it is possible to build complex models with great number of trimmed patches to use in mesh generation (see Fig. 19), numerical analysis, CAGD, and general computer graphics applications.

This work introduces some important contributions as the practical use of the subdivision algorithm to solve completely the surface intersection problem for quadric parametric surfaces. These algorithms are common used only to compute start points to other techniques. Therefore it is possible to take the main advantage of this method type that is the total independence of the shape of the intersection curves and its immunity to singularities points. Other original contribution is the method to compute the bounding volumes, which uses a local recursive subdivision. Others two important contributions are the method to project a

point onto surface and the method to reordering the intersection points. These methods provide the same parametric representation for an intersection curve in the parametric spaces of the two involved surfaces, what is important to geometric manipulation of these curves.

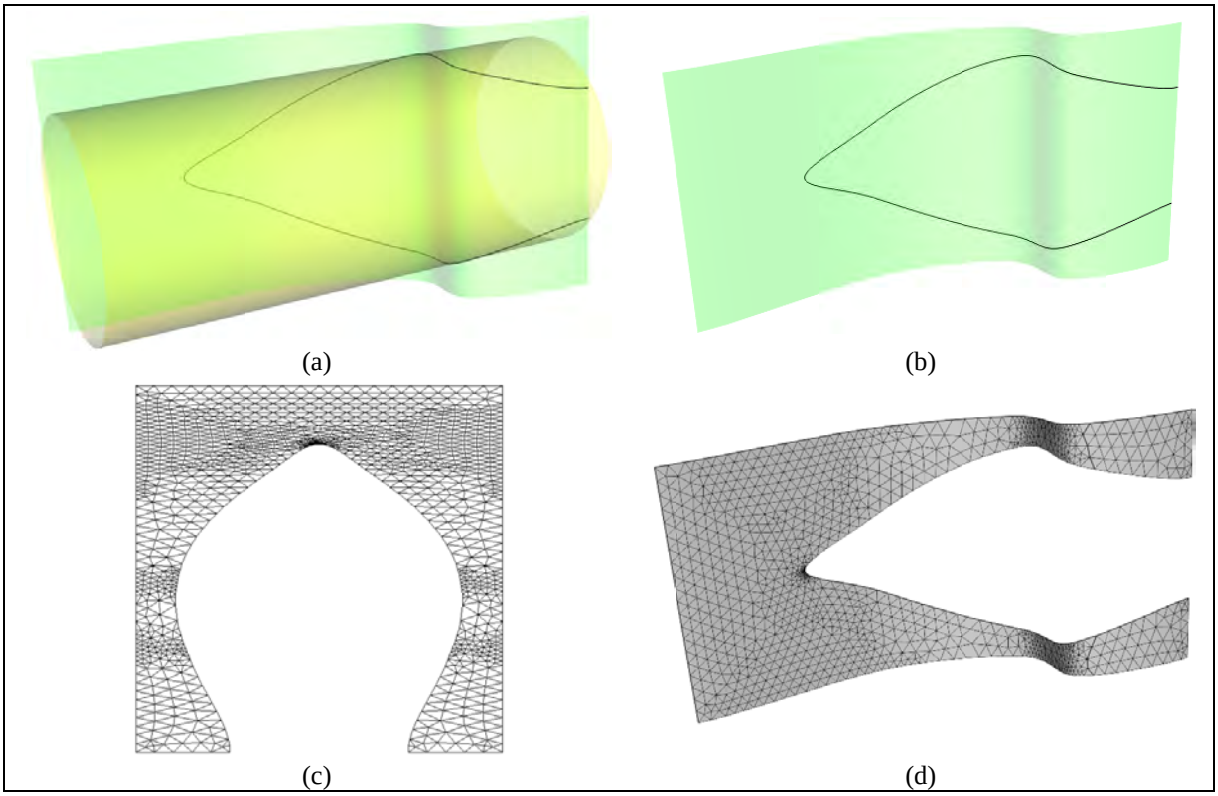


Figure 19: Meshing a sub domain of one surface: (a) and (b) show example 1 after determining the intersection curve with proposed algorithm; (c) and (d) show the mesh in parametric space and 3D space.

10 REFERENCES

Abdel-Malek, K., Yeh, H.J., Determining intersection curves between surfaces of two solids. *Computer-Aided Design* 28:6/7, 539 – 549, 1996.

Andrade, L.N., *Traço de interseção de superfícies com passos circulares*. Phd thesis, FEEC, Unicamp, Campinas, Brazil, 1998.

Asteasu, C., Intersection of arbitrary surfaces. *Computer-Aided Design* 20:9 (1988), 533–538.

Barnhill R., Farin G., Jordan M., Piper B.: Surface/surface intersection. *Computer Aided Geometric Design* 4, 3 – 16, 1987.

Barnhill, R., Kersey, S., A marching method for parametric surface/surface intersection. *Computer-Aided Design* 7, 257 – 280, 1990.

Barth, W., Huber, E., Computations with tight bounding volumes for general parametric surfaces. In *Proceedings of the 15th European Workshop on Computational Geometry - CG'99*, Antibes, France, 123 – 126, 1999.

Comba, J.L.D., Stolfi J., Affine Arithmetic and its Applications to Computer Graphics. In *Proceedings of SIBGRAPI '93 – VI Brazilian Symposium of Computer Graphics and Imaging Processing*, Recife, Brazil, 9 – 18, 1993.

Figueiredo, L.H., Surface intersection using affine arithmetic. In *Proceedings of Graphics Interface '96*, 161 – 170, 1996.

Gleicher, M., Kass M., An interval refinement technique for surface intersection, in: *Proceedings of Graphics Interface '92*, 242 – 249, 1992.

Griffiths, J.G., A data structure for elimination of hidden-surface algorithms. *Computer-Aided*

Design 11, 71 – 78, 1975.

Lane, J., Riesenfeld, R., A Theoretical Development for the Computer Generation and Display of Piecewise Polynomial Surfaces. *IEEE Transaction on pattern analysis and machine intelligence* vol. PAMI-2, No. 1, 1980.

Hoschek, J., Lasser, D., *Fundamentals of Computer Aided Geometric Design*. A K Peters, Ldt, Wellesley, Massachusetts, 1993.

Houghton, E.G., Emnett, R.F., Factor, J.D., Sabharwal, C.L., Implementation of a divide-and-conquer method for intersection of parametric surfaces. *Computer Aided Geometric Design* 2, 173 – 183, 1985.

Huber, E., Intersecting general parametric surfaces using bounding volumes. In *proceedings of the 10th Canadian Conference on Computational Geometry – CCCG'98*, Montreal, Canada, 1998.

Krishnan, S., Manocha, D., *An efficient surface intersection algorithm based on lower dimensional formulation*. Technical Report, Department of Computer Science, University of North Carolina, 1994.

Owen, J.C., Rockwood, A.P., Intersection of general implicit surfaces. In *Geometric Modeling: Algorithms and New Trends* (1987), G. Farin (editor), SIAM, 335 – 346.

Patrikalakis, N.M., Maekawa, T., *Intersection Problems*. Technical report, MIT Sea Grant College Program, MIT, 1991.

Sabharwal, C. L., A fast implementation of surface/surface intersection algorithm. In *Proceedings of ACM symposium on Applied computing*, Phoenix, Arizona, United States, pp. 333 – 337, 1994.

Sederberg, T.W., Algebraic geometry for surface and solid modeling. In *Geometric Modeling: Algorithms and Trends*, G. Farin (Editor), SIAM, 29 – 42, 1987.

Stoyanov, Tz.E., Marching along surface/surface intersection curves with an adaptive step length. *Computer Aided Geometric Design* 9, 485 – 489, 1992.

Wu, S.T., Andrade, L.N., Marching along a regular surface/surface intersection with circular steps. *Computer Aided Geometric Design* 16:4, 249 – 268, 1999.