An algorithm for intersecting and trimming parametric meshes

Luiz Cristovão G. Coelho[1] Marcelo Gattass[1] Luiz Henrique de Figueiredo[2]

[1]Department of Computer Science, PUC-Rio
Rua Marquês de São Vicente 225, 22453-900 Rio de Janeiro, RJ, Brazil
lula,gattass@tecgraf.puc-rio.br

[2]LNCC–Laboratório Nacional de Computação Científica
Avenida Getúlio Vargas 333, 25651-070 Petrópolis, RJ, Brazil
lhf@lncc.br

Abstract. We present an algorithm for intersecting finite-element meshes defined on parametric surface patches. The intersection curves are modeled precisely and both meshes are adjusted to the newly formed borders. The algorithm is part of an interactive shell modeling program, which has been used in the design of large offshore oil structures. We avoid unacceptable interaction delays by using a variant of the DCEL data structure that stores topological entities in spatial indexing trees instead of linked lists. These trees speed up the intersection computations required to determine points of the trimming curves, and also allows mesh reconstruction using only local queries.
Keywords: surface intersection; finite element generation; parametric representation; topological data structures

1 Introduction

Surface modeling and mesh generation on surfaces are important problems in computer-aided geometric design (CAGD), specially in engineering applications. One way to build complex models is to take several simple surface patches and sew them together into a single surface, trimming excess parts away. If this modeling technique is to be effective, then the problem of computing the intersection between two patches must be solved efficiently and robustly.

We consider an environment where surfaces patches are interpolated from boundary curves given in parametric form. Such surfaces are called *Coons patches* [1]. The user provides discretizations for the boundary curves, which are then used to interpolate a mesh for the surface.

We approach the surface intersection problem as a *meshing* problem, in the sense that we are given not only two parametric patches, but also meshes on them. Both the parametrizations and the meshes must be taken into account when computing the intersection. In other words, we must compute intersection curves that lie on both surfaces *and* on both meshes. Moreover, the boundary discretizations provided by the user must be respected, so that meshes on adjacent patches fit together correctly. Thus, our goal is to compute a single mesh for the union of the two patches, and this mesh must be composed largely of the original meshes, which are only modified locally, around the intersection. See Figure 1.

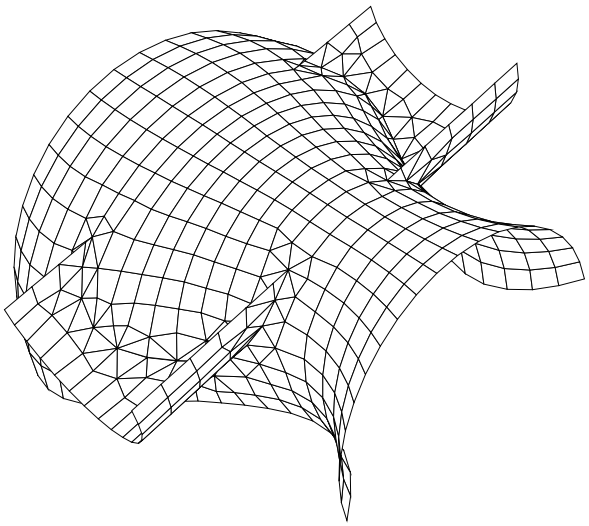


Figure 1: Composite mesh after intersection.

In this paper, we present a fully automatic algorithm that solves this mesh intersection problem. The searches required for computing the intersection curves and for re-meshing are supported by a topological data structure whose main feature is that topological entities are stored in B-trees [2] and R*-trees [3], instead of linked lists. These spatial indexing structures play a major role in the overall efficiency of the algorithm.

The algorithm has been implemented and is part of a complete shell modeling program, which has been

used in the design of large offshore oil structures. Interactive shell modeling by direct manipulation in 3D space raises several interesting user-interface issues that have been discussed elsewhere [4].

2 Previous work

Only very rarely the intersection of two parametric surfaces can be computed analytically; even when it can, it might not be practical. Hence, we must resort to numerical techniques.

The methods for solving the surface intersection problem in CAGD belong to two major classes: *Marching or continuation methods* compute the intersection curve in 3D object space by marching in the direction of its tangent vector [5–7].*Decomposition or subdivision methods* compute the trimming curves in 2D parameter space, by recursively refining the solution at each step [8].

Previous solutions to the surface intersection problem work well in many cases, but do not handle the mesh intersection problem as defined above. An exception is the work of Lo [9], which motivated our work. Lo [9] gave a simple algorithm for intersecting triangular meshes that automatically redefines the faces to adapt to the intersection curves. His solution does not use the continuous parametric representation of the surfaces, and thus can be used easily in most modeling systems. On the other hand, near regions of high curvature, the computed intersection points may not lie on the original surfaces, which is unacceptable for modeling.

Haber and Abel [10] pioneered the use of graphical interfaces with transfinite mappings for meshing shell structures.

3 Requirements for mesh intersection

Ideally, a good solution for the mesh intersection problem should satisfy the following requirements:

Correctness: The computed intersection curves must lie on both surfaces. If the intersection curves lie only on the faces of the meshes but not on the surfaces, then the finite-element analysis will yield false or unacceptable results. The easiest way to guarantee correctness is to compute the trimming curves in parametric space and then map them to object space.

Faithfulness: The geometry of the resulting surface must faithfully reflect the geometry of the original surfaces, because they represent the intent of the designer. In particular, new parametric patches should not be defined using the trimming curves as boundary curves, because this would yield a different geometry. See Figure 2.

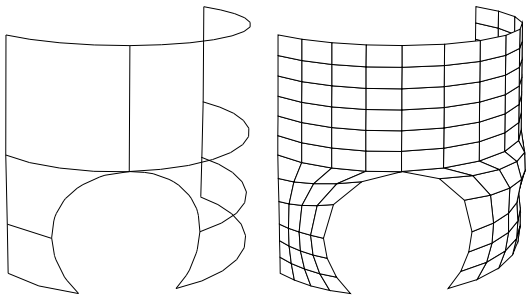


Figure 2: Bulge produced by re-parametrization.

Precision: Points in parametric space corresponding to the same intersection point must, when mapped to 3D, be at a distance smaller than a tolerance selected by the user (usually, a fraction of the size of the smallest element in the original meshes).

Automatism: The two meshes should be automatically redefined to include the intersection curves and possibly exclude trimmed regions. The user should not be required to manually edit the resulting meshes.

Quality: The elements generated during re-meshing must have a good geometric shape and must be of the same average size of the elements in the original meshes. When the original meshes have elements with very different sizes or when the element edges are close to the intersection curve, the resulting mesh may have elongated elements with sharp angular differences, if no corrections are made.

Locality: Only elements close to the intersection region should be modified during re-meshing. Elements far from the intersection region should remain unaltered. Moreover, if smoothing is used to improve mesh quality, then it too should be local.

Regions: The regions defined in parametric space by the trimming curves must be automatically identified. These regions may or may not take part in the final generated model; they can even have different attributes (loads, materials, boundary conditions).

Efficiency: The time and space required for computing the intersection should be ideally linear in the number of elements around the intersection region. Algorithms that are quadratic in the total number of elements in the meshes are too slow for large meshes and are not appropriate for interactive modeling.

Robustness: An arbitrary number of intersection curves may be generated, having different geometry, topology, and interpolation points. In particular, closed curves must be handled correctly, and must yield regions with holes when trimmed.

4 The algorithm

The algorithm we propose here for intersecting two meshed surfaces A and B has three basic steps, which aim at fulfiling the requirements listed above:

1. Determination of the intersection points:
   (a) Compute and store the intersection of edges in A against faces in B; and
   (b) Compute and store the intersection of edges in B against faces in A.
2. Determination of the trimming curves:
   (a) Link intersection points into polygonal lines representing the trimming curves;
   (b) Interpolate parametric curves passing through polygonal lines points;
   (c) Compute new points with proper spacing on these curves; and
   (d) Move these new points onto each surface.
3. Topology reconstruction:
   (a) Determine the trimming regions removing vertices and edges based on polygonal lines;
   (b) Insert new edges over the trimming curves using the new points defined in Step 2;
   (c) Triangulate the trimming regions on both surfaces; and
   (d) Smooth both meshes.

To avoid testing all edges against all faces in Step 1, we store topological entities in spatial indexing trees, as described in Section 4.1. Since edges and faces are curved in object space, we need a numerical procedure to determine the intersection points, which is described in Section 4.2. At the end of Step 1, edges in one mesh are paired with the faces that they intersect in the other mesh, and vice-versa. For each such pair, we also store the parametric coordinates of the intersection point.

In Step 2a, we compute the trimming curves in parametric space by linking and interpolating the intersection points computed in Step 1. In Steps 2b–d, we compute continuous representations for the trimming curves in parametric space. Equally-spaced sample points on the trimming curves are then computed and relaxed back onto the original surfaces. More details are given in Section 4.3.

In Step 3a the trimming regions are identified. These regions, actually sub-patches, are faces of the topological data structure generated by the elimination of some edges. At the end of Step 3a there are, in each surface, as many regions as the number of trimming curves. Step 3b consists of inserting the edges that represent the trimming curves, which leads to the subdivision of each trimming face in two. After that, in Step 3c, each trimming region is triangulated by inserting the edges, according to the geometric criteria described in Section 4.4.

Finally, to increase the shape quality of the faces generated in the intersection step, we use the standard Laplacian smoothing technique in parametric space. More precisely, the parametric coordinates of each vertex are changed to an average of the coordinates of its neighbors (which are easily identified using the DCEL). This averaging is repeated a number of times, until the quality of the elements reaches the desired level. Boundary and trimming vertices are never moved during smoothing.

4.1 The data structure

We store surface patches in a variant of the DCEL data structure [11], extended to handle topologies of curved elements. Moreover, topological entities (vertices, edges, and faces) are stored in trees, instead of linked lists or vectors. This allows for fast solutions for the searches necessary for the initial mesh assembly and for the mesh reconstruction.

We use B-trees [2] for storing vertices and edges, and R*-trees [3] for storing faces. The vertices are inserted into a B-tree that searches points by their parametric coordinates in lexicographical order. Each vertex gets an index such that vertices with smaller indices also have smaller parametric coordinates, in the lexicographical sense. Figure 3 shows the modified DCEL data structure used here.

The edges are considered as straight lines in the parametric space, connecting pairs of points vertices. The edges are oriented from the vertex with the smaller index to the vertex with the larger index, and are stored into a B-tree that has these indices as the search key, as in the vertex B-tree.

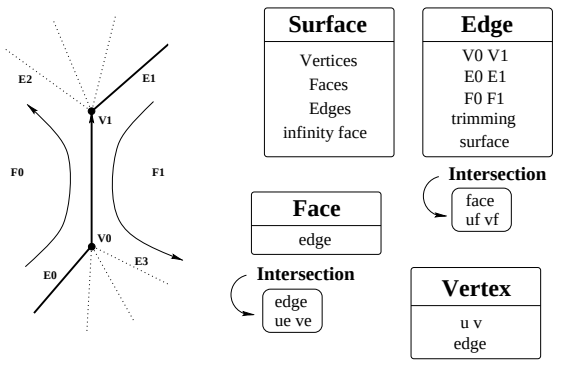


Figure 3: The modified DCEL.

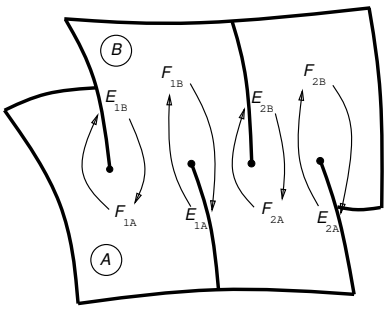


Figure 4: Topological chaining of DCELs.

The faces are inserted into a R*-tree that uses their 3D bounding boxes as the key. We use R*-trees with nodes with a minimum of four and a maximum of ten faces because present good average performance for planar subdivisions [12].

The *Intersection* field of edges and faces is used to determine the trimming curves and to reconstruct the topologies; it is also the key to the topological chaining of two DCELs. Figure 4 shows how this topological chaining between the faces and edges of two intersecting surfaces is made. The geometric information of each intersection point is stored in pairs (u_f, v_f) and (u_e, v_e) in *Intersection*.

4.2 Computation of intersection points

In Step 1 of the algorithm, the edges in one surface are tested against the faces in the other surface that can potentially intersect it. This is done by testing their bounding boxes for intersection. The face R*-tree is searched by using the edge bounding box.

For each selected edge-face pair, we find the point where they intersect by solving a minimization problem $F(u, v, t) = 0$, where $F(u, v, t) = S(u, v) - R(t)$. The algorithm stops when the distance from the edge point with parameter t to the face point with coordinates (u, v) is smaller than a given tolerance. This tolerance corresponds to a fraction of the smallest length of the edges involved in the intersection. Figure 5 illustrates the elements involved in the problem, where the parameter t, which defines the variation along the intersected edge, can be expressed as an interpolation given by the straight line in the parametric space (w, h) that connects V_0 and V_1.

The correction term for solving $F(u, v, t) = 0$ using the Newton-Raphson method is $\Delta p = -\mathbf{J}^{-1}(p)F(p)$, where $p = (u, v, t)$ and $\mathbf{J}$ is the Jacobian matrix of F. Sinc this Jacobian matrix is composed by the directional tangents on the patch's surface and by the tangent to the curve, which is a directional derivative on the edge's surface, one must be very careful with the length of the tangent vector, to ensure convergence. If the surfaces have different parametrizations, such as one bicubic and another bilinear, the directional derivatives must be normalized before solving $F(u, v, t) = 0$. To make the procedure robust enough to cope with the singularities that occur where the surfaces are tangent or parallel to each other, we use the modified Newton-Raphson method described by Deuflard [13].

After the convergence of the minimization algorithm, two parametric pairs: (u_f, v_f) in the face's surface, and (u_e, v_e) in the edge's surface are identified, and the relative position of the face and the intersection point must be tested inside face's parametric space. A modified *single shot* algorithm is performed to verify if the intersection point lies inside, outside, on an edge, or an a vertex of the face being tested. If the point is outside the face no updates are made in the data structure. When the point is inside the face, the edge/face pair is updated in its *Intersection* fields (see Figure 3). If the point is over an edge, besides the edge/face pair, the edge identified by the algorithm must also have it's *Intersection* field updated. When the intersection is located over a vertex of the face, all edges adjacent to this vertex must also be updated.

In all cases when the field *Intersection* has to be updated, if it is already instantiated in previous computations, no changes are necessary. As Section 4.3 will show, a single reference in each edge and face is sufficient to link intersection points, thus the first one computed does this task.

For the determination of the intersection points to be complete and for the fields necessary to the construction of the trimming curves to be filled, all procedures described in the present section must be repeated switching surfaces, that is, testing the edges of the second surface against the faces of the first one. At the end of this step, all topological and geometrical information necessary to determine the trimming curves is stored into the DCELs.

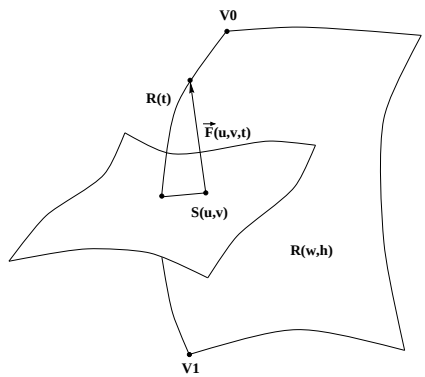
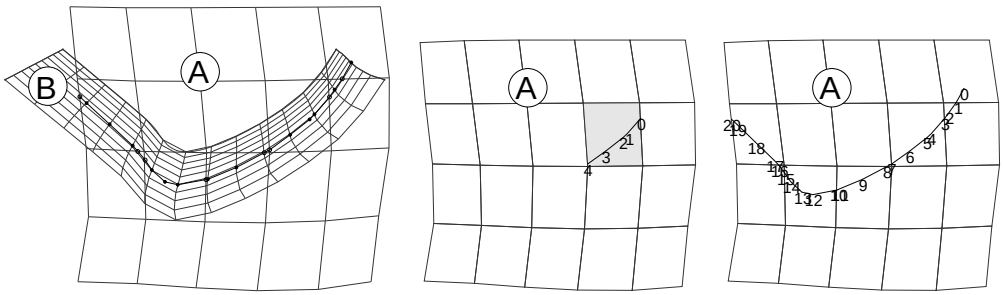


Figure 5: Notation for the intersection problem.



(a) Intersection points. (b) Local polyline. (c) Global polyline.

Figure 6: Local and global polylines used in Step 2 of the algorithm.

4.3 Computation of intersection curves

This step is done by visiting adjacent intersected faces, identified by the *Intersection* field shown in Figure 3, moving across intersected edges and storing references to these edges in each position. When all intersected faces have been visited, all connected components of the intersection have been found.

We use two kinds of polygonal lines: *local polylines*, which join intersection points within a fixed face; and *global polylines*, which are composed of local polylines and represent a complete trimming curve. Figure 6a shows the intersection curve and the intersection points of two surfaces A and B. Figure 6b shows the local polyline for the face drawn in gray. Points 0 and 4 are the intersections of the edges of this face with faces of B; points 1, 2, and 3 are the intersections of edges of B in the interior of the gray face in A. Figure 6c shows the global polyline obtained by propagation of local polylines through all faces of A.

To convert the global polylines into continuous intersection curves, we interpolate the intersection points using cubic splines. We use the knot sequence selection scheme defined by Foley and Nielson [14] because it gives good results for points having non-uniform spacing (it also has other interesting properties, such as invariance under affine transformations).

Once we have a continuous representation for the trimming curves, we sample them uniformly, with a spacing proportional to the size of the edges in the initial mesh. These new sample points define vertices representing the intersection curves in the combined mesh. Even though the interpolation points lie on both surfaces, the sample points may not, and we have to relax them back onto the surfaces, using a procedure similar to the one used in Step 1. As a by-product, we also obtain the parameter values for the sample points.

4.4 Determination of trimming regions

To find the trimming regions in Step 3a, we remove all edges connected to the vertex closest to each interpolation point found in Step 1. The edges belonging to a boundary or to a trimming curve determined in previous intersections cannot be removed and are split at the crossing position.

In Step 3b, the edges representing the trimming curves are inserted, dividing the faces in two: one to the left, and one to the right of each curve. Figure 7 shows the intersection of two shells and the corresponding trimming curves and regions. The edges created over the trimming curves receive the positive mark on the field *trimming* shown in Figure 3.

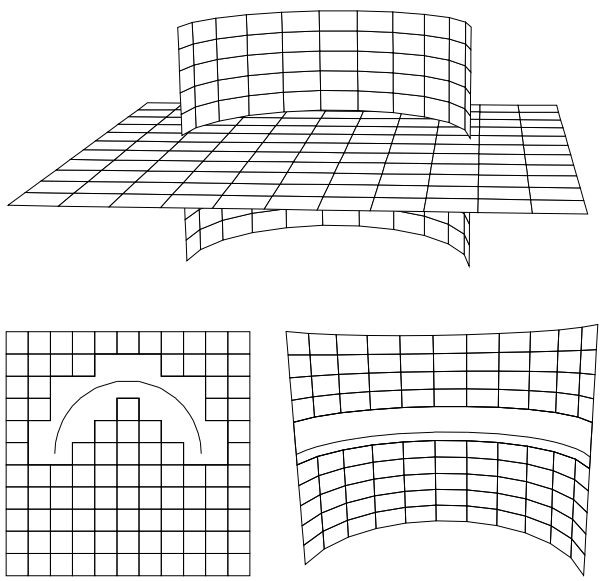


Figure 7: Two intersecting shells and the trimming regions in each suface.

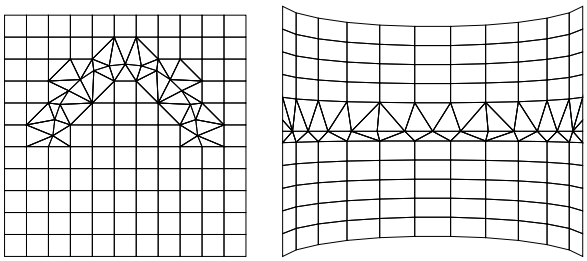


Figure 8: Triangulation of trimming regions.

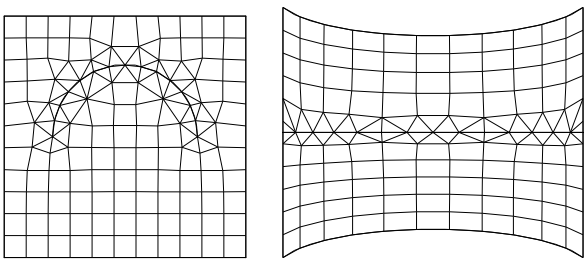


Figure 9: Smoothed meshes.

In Step 3c, the trimming regions are triangulated by selecting edges according to the following geometric criteria:

Consistency: Edges should be completely contained in the parametric region limited by the edges, without intersecting other edges.

Proximity: Edges should connect two vertices that are near each other.

Curvature: The area of the sector formed by the chord and by the curve which follows the surface is the smallest one.

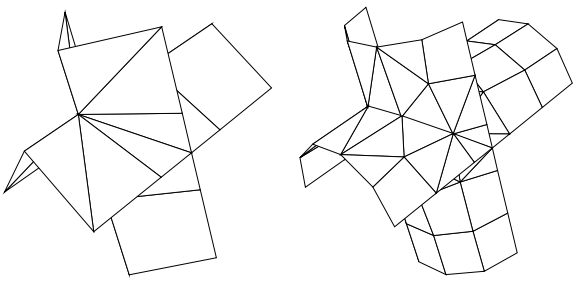
Delaunay: Edges should obey the standard Delaunay criterion projected on a plane whose normal vector is the average of the normal vectors of adjacent faces.

Figure 8 shows the regions of Figure 7 triangulated by the algorithm using these criteria. The edges created to subdivide the trimming regions are drawn with thicker lines. Figure 9 shows the resulting meshes after smoothing.

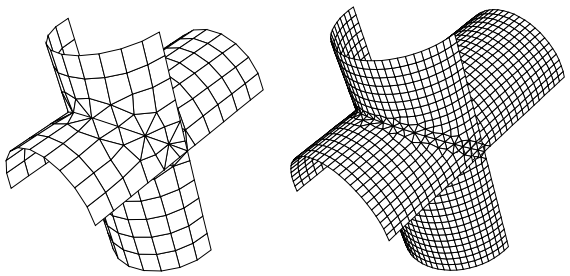
5 Examples

We now show some examples of our algorithm in action. The first example has already appeared in Figure 1: it is the union of a Coons patch with a cylinder, whose bottom part has been trimmed away. Note that the two intersections curves are symmetrical, due to symmetry in the original surfaces.

The next example is the union of two cylindrical shells, modeled as bilinear Coons patches. Figure 10 shows the results of the algorithm for meshes of increasing complexity. In Figures 10a and 10b, the intersection region extend up to the boundaries, so that the smoothing repositions a few interior vertices. Figures 10c and 10d show that, in higher resolution meshes, only the few elements close to the intersection region are affected, and the original meshes are largely preserved. This can also be seen in Figure 1.



(a) 2×3. (b) 5×5.



(c) 10×11. (d) 30×30.

Figure 10: Intersections between cylindrical shells meshed with increasing resolution.

Figure 11 shows an example of an intersection that has two connected components. The patches are a cylindrical shell and a toroidal shell. Figure 12 shows a detail. In this example, three distinct regions can be identified on each patch. Figures 13a and 13b show the elements in parametric space, with distinct regions drawn with different line widths. Note that the elements do not have good shapes in parametric space but they do in object space. Figure 14 shows the mesh resulting from removing the central region of the cylindrical shell.

The example in Figure 15 shows that the algorithm correctly handles closed intersection curves.

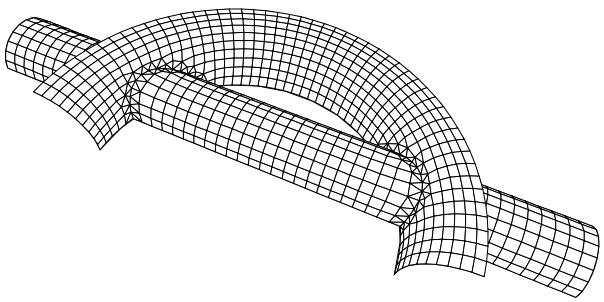


Figure 11: Intersection with two components.

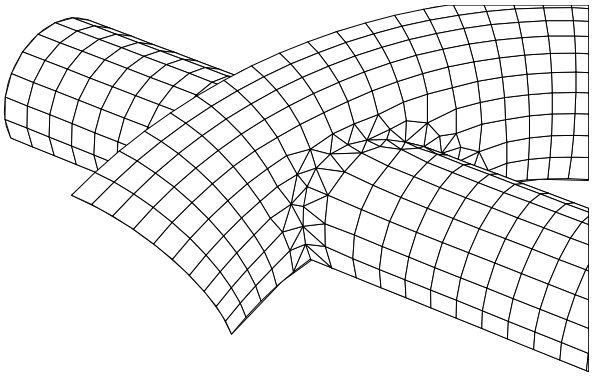
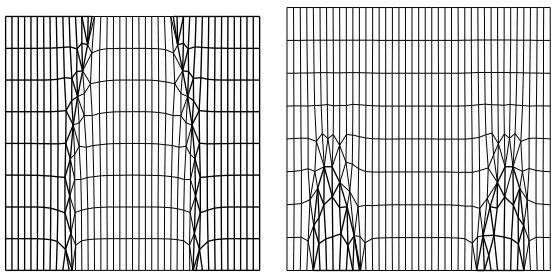


Figure 12: Detail of the intersection in Figure 11.



(a) Cylindrical shell. (b) Toroidal shell.

Figure 13: Distinct regions in parametric space.

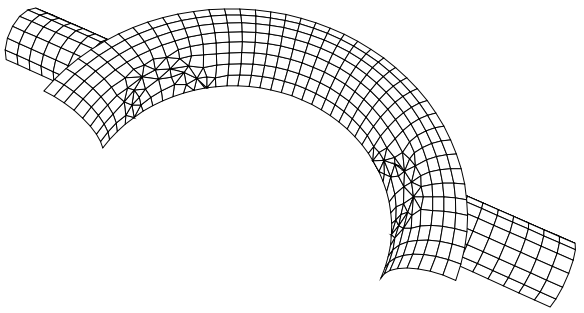


Figure 14: The distinct regions resulting by eliminating the central part of the cylindrical shell.

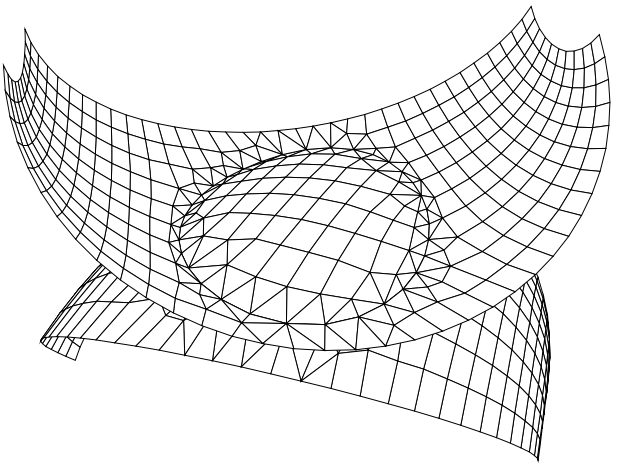


Figure 15: Closed intersection curve.

6 Conclusion

An algorithm that determines the intersection curves and the resulting meshes of two parametric patches was developed and implemented with the DCEL topological data structure, using trees for storing entities. These spatial structures greatly reduce the complexity of intersection computations by quickly discarding large sections in one mesh that cannot intersect the other mesh. The topological data structure used to represent the mesh of each surface patch allows the construction of the trimming curves without global searches. The topological information is also used in the re-meshing algorithm to obtain elements with improved quality, suitable for finite-element analyses.

The results show that the algorithm is very fast if compared with the construction of the individual patches, and can be used in interactive shell modeling with composite patches. Experiments with meshes having up to one million faces suggest that the time required by the algorithm grows linearly with the number of faces. This can be explained by the small number of faces intersections actually processed due to the efficient searches provided by the spatial trees.

Step 3 of the algorithm is the most time consuming of all three steps (an average of 60% of the total time), and its current implementation uses the speed-up described by O'Rourke [15], which still is $O(n^2)$, where n is the total number of edges in the intersection region. In practice, n is much smaller than the total number N of edges in both meshes. Typically, $n = O(\sqrt{N})$, which is consistent with the experimental results mentioned above.

The DCEL data structure is simple and compact, and is a good choice for the shell modeling program.

For applications that must handle all topological degeneracies that appear in more general intersections (faces with holes, self-edges or edges inside faces), the half-edge data structure [16] would be a better choice. However, the half-edge data structure is not compact and its impact on memory use would have to be further evaluated, specially for large meshes.

The algorithm does not need to order the intersection points in a face using geometric tests as done in [17]. The algorithm also does not require the global search for chaining the individual segments, needed in Lo's algorithm [9], because it performs this task as the global polylines are computed with topological tests on the extremities of the trimming curve. It also handles trimming curves with sharp curvatures and several connected components.

The geometric quality of some trimming curves generated by the algorithm were tested computing the distance from points evaluated over them with the analytical solution, and the results show that they are very close. Thus, the interpolation method using Foley's parametric variation [14] presented good results in our tests.

The regions created by the inserted trimming edges are implicitly represented in the data structure and a simple topological algorithm can quickly identify them.

We are currently working on mesh uniformization, converting triangular elements into quadrilaterals, using the method proposed by Potyondy et al. [18].

Acknowledgements. The shell modeling program has been developed for PETROBRAS in TeCGraf, the Computer Graphics Technology Group of PUC-Rio. Maurício Riguette Mediano implemented the R*-tree library and gave many useful suggestions. The authors are partially supported by research grants from the Brazilian Council for Scientific and Technological Development (CNPq). This research is part of the first author's Ph.D. thesis.

References

[1] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, 1990.

[2] D. Comer. The ubiquitous B-tree. *ACM Computing Surveys*, 11:121–131, 1979.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and retangles. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 322–332, May 1990.

[4] L. C. G. Coelho and C. S. de Souza. Comunicação de problemas e soluções geométricas em uma interface 3D. In *Anais do VIII SIBGRAPI*, pages 233–240, 1995.

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

[6] R. Barnhill and S. Kersey. A marching method for parametric surfaces/surface intersection. *Computer Aided Geometric Design*, 7:257–280, 1990.

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

[8] E. Houghton, E. Emnett, R. Factor, and L. Sabharwal. Implementation of a divide-and-conquer-method for the intersection of parametric surfaces. *Computer Aided Geometric Design*, 2:173–184, 1985.

[9] S. H. Lo. Automatic mesh generation over intersecting surfaces. *International Journal for Numerical Methods in Engineering*, 38:943–954, 1995.

[10] R. Haber and J. F. Abel. Discrete transfinite mappings for the description and meshing of three-dimensional surfaces using interactive computer graphics. *International Journal for Numerical Methods in Engineering*, 18:41–66, 1982.

[11] F. P. Preparata and M. I. Shamos. *Computational Geometry – An Introduction*. Springer Verlag, New York, 1990.

[12] M. R. Mediano, M. Gattass, and M. A. Casanova. HPS-tree: Um método de acesso para armazenar mapas longos com multi-resolução geométrica e topológica. In *Anais do IX SIBGRAPI*, pages 219–226, 1996. `http://www.visgraf.impa.br/sibgrapi96/trabs/abst/a32.html`.

[13] P. Deuflard. A modified Newton method for the solution of ill-conditioned systems of non-linear equations with applications to multiple shooting. *Numer. Math.*, 22:289–315, 1974.

[14] T. A. Foley and G. M. Nielson. Knot selection for parametric spline interpolation. In L. Schumaker, editor, *Mathematical Methods in CAGD*, pages 445–467. Academic Press, 1989.

[15] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, New York, 1993.

[16] M. Mäntylä. *An Introduction to Solid Modeling*. Computer Science Press, Rockville, Maryland, 1988.

[17] P. R. Cavalcanti, P. C. P. Carvalho, and L. F. Martha. Nonmanifold modeling: an approach based on spatial subdivision. *Computer-Aided Design*, 29:209–220, 1997.

[18] D. O. Potyondy, P. A. Wawrzynek, and A. R. Ingraffea. An algorithm to generate quadrilateral or triangular element surface meshes in arbitrary domains with applications to crack propagation. *International Journal for Numerical Methods in Engineering*, 38:2677–2701, 1995.