A mapping-independent primitive for the triangulation of parametric surfaces

Marco Attene, Bianca Falcidieno, Michela Spagnuolo

IMATI-GE, Consiglio nazionale delle Ricerche, Via De Marini 6, Torre Di Francia, 14149 Genova - Italy

Geoff Wyvill

Department of Computer Science, University of Otago P.O. Box 56, Dunedin, New Zealand

This paper describes a new technique for the triangulation of parametric surfaces. Most earlier methods sample the parameter domain, and the wrong choice of parameterization can spoil the triangulation or even cause the algorithm to fail. Conversely, we use a local tessellation primitive to sample and triangulate the surface. The sampling is almost uniform and the parameterization becomes irrelevant. If sampling density or triangle shape has to be adaptive, the resulting uniform mesh can be used either as an initial coarse mesh for a refinement process, or as a fine mesh to be reduced.

Key Words: Parametric Surfaces; Triangulation; Mesh Generation; Finite Element Method.

1. INTRODUCTION

Parametric surfaces are an important tool for computer-aided design (CAD) systems. Although they are considered a standard in CAD/CAM, parametric surfaces are not particularly suitable for some applications that emerged after their conquest of the CAD community. This moved a number of researchers towards the study of techniques for the discretization, or tessellation, of such surfaces for visualization or for analysis purposes. Nowadays, the most used approaches are based on advancing front [1][2] or Delaunay triangulation [3][4][5], and there are *ad hoc* methods for specific surface classes [6][7][8]. In cases such as surface reconstruction [9][10] the vertices are given, but here the *tessellator* has to sample the surface and construct a mesh connecting the samples. Most existing methods do both steps in parameter space: first, they create a triangulation of the surface's 2D domain, then the final tessellation is obtained by mapping the vertices to 3D space, without changing the connectivity. A uniform sampling of the parameter space could be quite inappropriate, so adaptive methods [11][1] are preferred where an initial coarse mesh is iteratively refined depending on some error metric. Adaptive methods, however, need a starting mesh and, if the surface is *well-behaved*, this can be achieved by a coarse uniform sampling of the parameter domain, otherwise, some problems may arise, as described in the next section.

This paper proposes a method for sampling and triangulating the surface independently of the parameterization [12], so that bad mapping properties do not spoil the final triangulation. In the rest of this paper we introduce a novel tessellation primitive, the *Normal Umbrella*, and describe how to use it to build a quasi-uniform approximation of a surface. Also, we show that this primitive can produce meshes suitable for a wide range of applications. Specifically, the quasi-uniform tessellation is by nature suitable for a Finite Element simulation, where typical numerical algorithms require the triangle shape and size to be as regular as possible. For applications requiring fast rendering, we show how to perform remeshing in order to decrease both the polygon count and the approximation error, at the price of a loss of uniformity.

2. PREVIOUS WORK

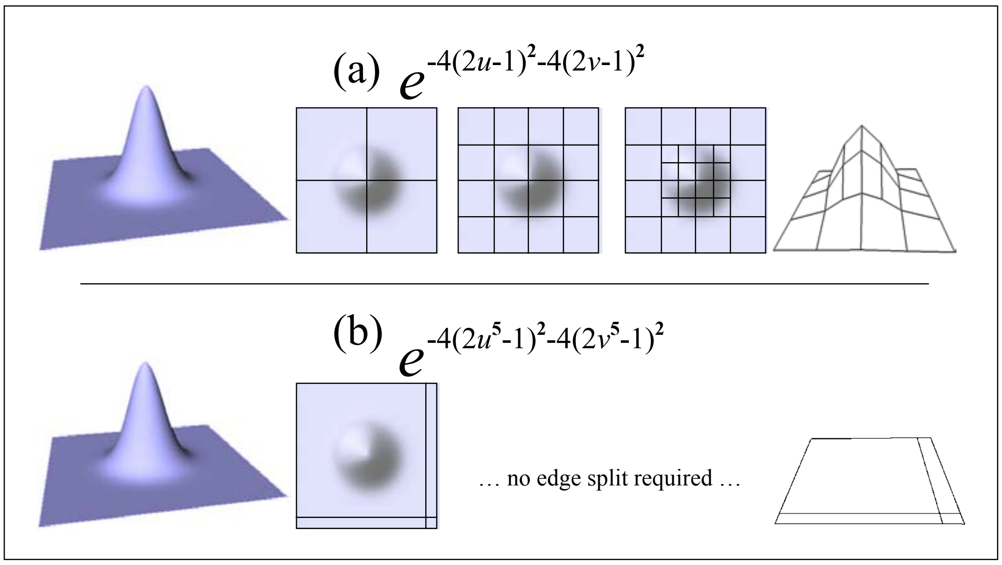
The problem of visualizing a parametric surface has been studied for over 20 years [13] and many approaches have been proposed, including a number of polygonization-based strategies. The earliest methods simply traced lines of constant u or v in the parametric space. This produces a level of detail unrelated to what is required by the display. Lane et al. [28] proposed an efficient scan-line method and by 1983 methods based on ray tracing had been reported [13][14]; here the main problem was efficiently calculating the first point of intersection of a straight half-line with the surface. A solution to this problem has been proposed in [15], where a polygonization approach is presented by which the first intersection point can be computed effectively.

The tessellation of parametric surfaces, however, has been mainly developed for direct visualization [11] and for FEM applications [3]. As described in [16][17], the scientific literature includes approaches for generating either isotropic meshes, in which the element size and shape is roughly constant, or anisotropic ones, in which the sampling density varies depending on the surface curvature and triangles are stretched along the principal curvature directions. Typically, isotropic meshes are preferable for structural simulation and FEM applications, while anisotropic meshes are a good choice for visualization purposes. Such a distinction made most researchers to propose methods for the creation of meshes of either one or the other class, but not both. In this paper we give a more general framework, and describe a flexible method for creating both isotropic and anisotropic meshes.

Existing methods for isotropic mesh generation are generally based on iterative repositioning of an initially random vertex set [18], or on physical principles of force balancing in the mesh [3][19]. In the method described in [18], the user chooses the number of vertices n of the final mesh and a parameter q for the so-called *shape function*, which describes the local 'curvedness' of the surface. The algorithm displaces n vertices randomly on the parameter domain and iteratively adjusts their positions in order to fit the shape function. Similarly, in [3] the method takes as input the domain geometry and a node-spacing function, and then generates a mesh, or a set of connected triangles, that satisfies basic requirements such as a precise control over node spacing or triangle size. The initial nodes are placed using recursive spatial subdivision. The resulting mesh is relaxed by assuming the presence of proximity-based, repulsive/attractive internode forces and then performing dynamic simulation for a force-balancing configuration of nodes.

Unfortunately, due to their converging nature, these approaches lack a formal complexity estimate and, even more important, the number of samples must be fixed in advance, instead of depending on the surface area. An exception in this class of algorithms is the *marching method* presented in [20], which is close in spirit to the one we describe in the following sections. The marching method, that was originally defined for the polygonization of implicit surfaces, uses a local tessellation primitive consisting of a fan of triangles around a sample point. In particular, for a given surface point p, the method draws a regular hexagon with center p on the corresponding tangent plane. The boundary of the hexagon is projected on the surface using a gradient descent procedure and, for each boundary vertex, the process is repeated until the whole surface is covered. Although this approach is very effective on nearly-flat implicit surfaces, the evaluation of lengths on the tangent plane can introduce significant errors in regions of high curvature, moreover the necessity of a *numerical implicitization* for treating parametric surfaces can introduce a further error.

The creation of anisotropic meshes [21][11] is mainly based on adaptive schemes; in these cases the main drawback is the dependency on the surface definition. An example of such a problem is shown in Fig. 1; the same surface is defined in two different ways on the same 2D domain, and the adaptation criterion splits an edge in its middle 2D point if the image of this point is too far from the image of the edge in the Euclidean 3D space [11]. While in the first row (a) the initial coarse mesh has to be subdivided, it has not to be in the second case (b). On the other hand, if the initial coarse mesh were fine enough for *catching* the surface details, a huge number of redundant vertices would have to be created. In other words, the choice of the parameterization influences the resulting triangulation and, as we show in the following sections, a bad parameterization can spoil the result. From a mathematical point of view, we call "good parameterization" a function in which all the directional derivatives are close to a constant *k* throughout the whole domain.



(a) $e^{-4(2u-1)^2-4(2v-1)^2}$
(b) $e^{-4(2u^5-1)^2-4(2v^5-1)^2}$
… no edge split required …

FIG. 1. The initial coarse mesh could not detect some surface details, preventing edge splits.

In the following sections we discuss a novel approach that does not suffer from these drawbacks. The remainder of the paper is organized as follows: some remarks on definitions and notation are presented, then the *normal umbrella* and its use as a tessellation primitive is explained and, finally, we give a brief description of the necessary extensions for the creation of closed and adaptive triangulations.

3. PARAMETRIC SURFACES

The image of a position vector-valued function $f : A \subseteq \Re^2 \rightarrow \Re^3$, ***f****(u,v) = <x(u,v), y(u,v), z(u,v)>* is called a *parametric surface*. The subset of $\Re^2$ from which the two parameters *u* and *v* take their values, usually the square [0,1]×[0,1], is called the *parameter domain*. The function ***f*** is called a parameterization, or mapping, and it serves only as a representation of the surface we are interested in. The same surface can be represented by several mappings, as shown for example in Fig. 2, and the application of a given tessellator can give different results depending only on the way the surface has been parameterized.

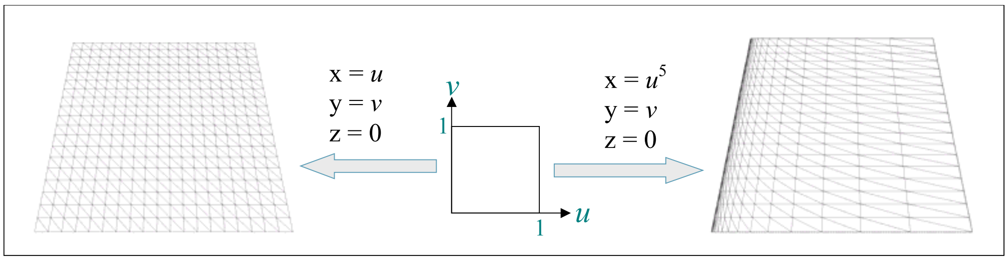


FIG. 2. The same plane can be defined in several different ways on the same domain.

Since the tessellator should approximate the surface, and not its representation, such behavior is not a good characteristic. Worse than that, the *wrong* choice of parameterization for a surface could lead to quite bad results, as shown in Fig. 3.

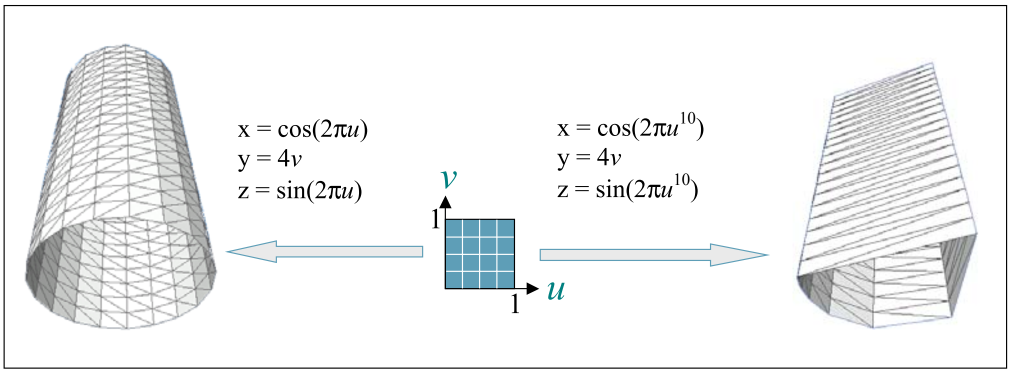


FIG. 3. The wrong choice of the parameterization can spoil the tessellation of the cylinder.

4. UNIFORMLY SAMPLING A SURFACE

In this section we analyze the problem of sampling a parametric surface independently of its definition. We discuss how to tackle the problem for 2D parametric curves, and show that there is an intrinsic hitch for generic 3D surfaces.

The problem of generating an isotropic mesh independently of the parameterization can be approached from different points of view and, among the others, we tried to answer the following questions:

1. How to find a sampling that is as uniform as possible on the surface rather than on its domain ?
2. Is it possible to re-parameterize the surface so that a uniform grid in parameter space maps uniformly on the surface ?

Let us consider the second question in the case of plane curves. For a regularly parameterized curve ***c****(t)* = *<x(t), y(t)>*, the arc length between two points can be computed using the integration of the curve tangent vector. Starting from this, it is not difficult to derive a parameterization of the same curve based on the *curvilinear abscissa*. The curve is the same, but for the new mapping a uniform sampling of the parameter space maps uniformly on the curve. In [22] the problem has been approached in order to find a piecewise linear approximation of a plane curve in which each segment forms the same angle with the next one. In the same paper they attempted to define a reasonable extension for surfaces, but the author himself could not prove whether a solution exists or not. In the case we want to create an isotropic tessellation, it is not difficult to find a counterexample, as shown in Fig. 4.

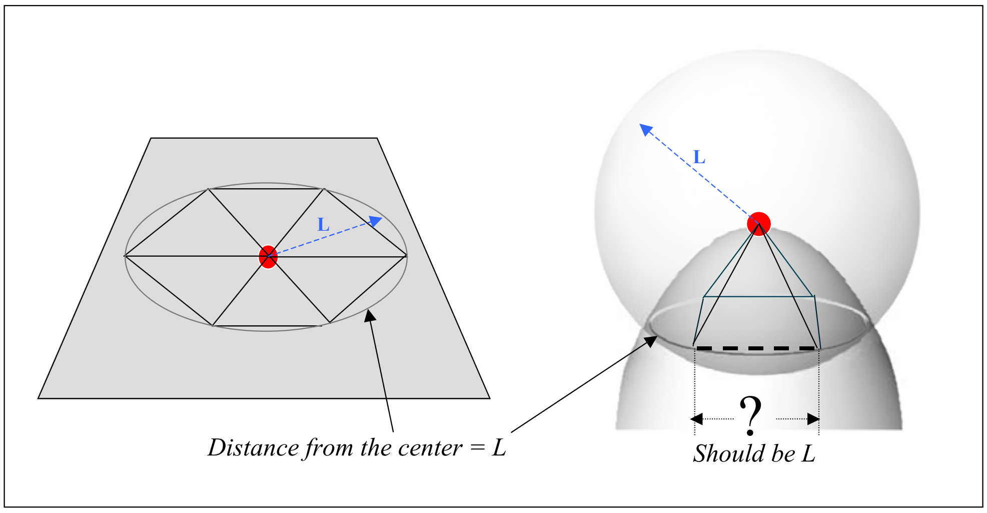


FIG. 4. An exactly uniform mesh does not always exist on curved surfaces.

While in the plane it is always possible to create a closed fan of equilateral triangles around each point, that is, a triangulation in which all the edges have the same length, this cannot always be achieved for generic 2-manifolds.

5. THE NORMAL UMBRELLA

In the previous section it has been shown that, for a generic surface, an approximating mesh that is perfectly uniform (i.e. all the edges have exactly the same length) may not exist. For most applications a good approximation is enough, so we define a local quasi-isotropic approximation called the *Normal Umbrella, or NU.*

Definition: 6th degree normal umbrella

Given a point p_p in the parameter domain whose image in the surface is p, let $H = \{p_1, p_2, .., p_6\}$ be a regular hexagon centered on p and lying on the tangent plane at p, Tp(p). For each p_i, draw a curve on the surface from p to a point q_i so that its projection on Tp(p) is a straight line parallel to $\boldsymbol{v}_i = p_i - p$, and $\|q_i\text{-}p\|=r$. Connect each end-point, q_i, with its neighbor, q_{i+1}. This process defines a 6th degree normal umbrella of radius r and center p which will be triangulated by the star $\{(p,q_1,q_2), (p,q_2,q_3), ..,(p,q_6,q_1)\}$. The result of this process is shown in object and parameter space in Fig. 5.

In general, we define an nth degree normal umbrella for every $n \geq 3$. Thus, drawing an equilateral triangle on the tangent plane leads to a 3rd degree NU, a square leads to a 4th degree NU, and so on.

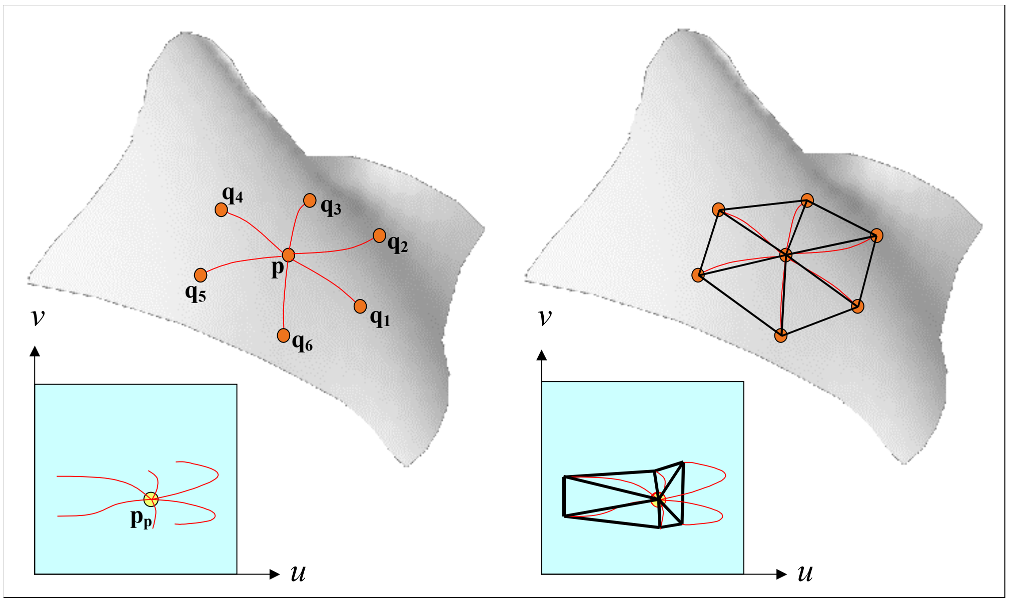


FIG. 5. A 6th degree normal umbrella and its pre-image in parameter space.

Although the Normal Umbrella is based on a regular hexagon on the plane, as the surface curvature increases the edges connecting the end points q_i - q_{i+1} become shorter. Since we want the edges to have about the same length, this behavior suggests that the number of paths to be tracked, n, should depend on the local curvature, specifically, high positive Gaussian curvature implies $n < 6$, while high negative curvature implies $n > 6$. Actually the curvature can change along the path, so we need a more accurate evaluation: consider all the paths emanating from the point p whose projection on the tangent plane is a straight line and whose end-point distance from p is r. Except for special cases, the end-points of these paths constitute a closed surface line that we call a *Normal Umbrella- Front (NU-front)*. If the length of the NU-front is L, then the optimal number of triangles n around the point p is the closest integer to L/r. In this context, "optimal" means that the piecewise linear approximation of the *NU-front* is composed of segments whose length is as close as possible to r. In the case of a plane, for example, the NU-front is the circle of radius r, so its length L is $2\pi r$ and $n = \text{round}(2\pi) = 6$. The length of the NU-front of a paraboloid's tip is $\leq 2\pi r$, so $n \leq 6$ while in the case of a saddle point $L \geq 2\pi r$, so $n \geq 6$. Special cases arise when an excessive radius of the NU for the given surface point is required; for example, the NU-front on the unit sphere when r=2 collapses on a single point (the antipodean of the NU center), while the NU-front on a radius 1 cylinder becomes multiply connected if r>2. If, at some point of the polygonization process, the NU-front is not a single surface line, we declare that the chosen sampling step is too big for the given surface.

5.1. Computing the Normal Umbrella

The main difficulty in computing the normal umbrella comes from the fact that the definition is based on tracking surface paths and, since we can only play with parameters, tracking such paths is not a trivial operation. Fortunately, differential geometry [23] provides all the tools we need.

5.2. Normal Sections

The paths to be tracked for building the normal umbrella are parts of *normal sections*, which are defined for each point as the intersection of the surface and a normal plane. Although a normal section can be made of several connected components, we want to preserve the topology [24], so we are only interested in the component containing the generating point. By means of differential calculus, it is not difficult to prove that the following system of differential equations represents the component we are interested in:

$$\begin{cases} u' = (n \times d) \cdot f_v(u,v) \\ v' = (d \times n) \cdot f_u(u,v) \\ u(0) = u_0 \\ v(0) = v_0 \end{cases}$$

where n is the normal of the generating point $f(u_0, v_0)$, d is a tangent unit vector belonging to the intersecting normal plane, and f_u and f_v are the two derivatives of the mapping function.

When building a normal umbrella of a given radius at a point, a direction d for each path must be chosen. It is important to consider that the normal umbrella is not unique, in fact, there exist infinitely many regular hexagons[1] on the tangent plane. And since we have no particular preference, the direction of the first path is arbitrary, while the others have to be computed according to this first one. Once a path has been tracked, its projection on the tangent plane is a straight segment, and the projection of each path must produce an angle α of exactly 60 degrees with the previous one. In general, when building an nth degree normal umbrella, the angle α between two consecutive projections must be $2\pi/n$.

5.3. The NU as a tessellation primitive

The normal umbrella makes it possible to create a simple triangulation around a sample point, and such a triangulation is independent of the surface definition. To extend this characteristic to triangulations of complete surfaces, we can use the NU as a tessellation primitive. Given a parametric surface and a desired edge length, let us choose some point of the parameter domain and build an appropriate normal umbrella around its image. Now pick a vertex, *v*, of the boundary of the NU and complete its fan, treating the triangles meeting at *v* as if they were already part of a normal umbrella around *v*. Repeating the latter step until all the parameter domain is spanned and avoiding overlaps, gives a rough solution of the problem. Now let us see how to perform these steps in detail.

5.4. Completion of a partial approximate NU

Except for the starting NU, each step must complete the fan of triangles around a vertex of the current boundary in the following way (with reference to Fig. 6.):

1. Compute the part of NU-front between the two boundary edges.
2. Divide it in equal parts so that the distance between the end-points of each part is as close as possible to the desired edge length.
3. Consider the end-points of the paths separating two adjacent parts.
4. Connect these end-points by edges and triangulate with the NU center.

[1] For simplicity we restrict the explanation to the 6th degree NU, but it holds for every degree ≥ 3.

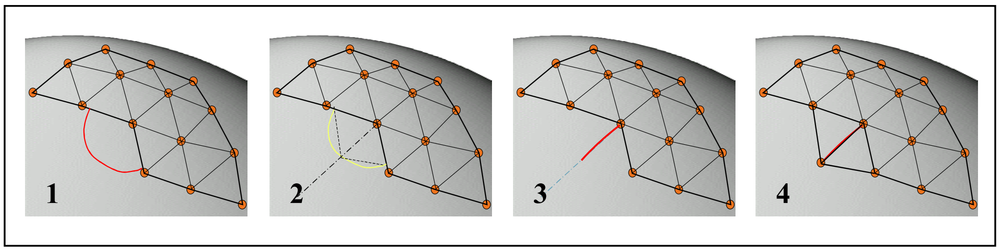


FIG. 6. The four steps for completing a partial approximate NU around a boundary vertex.

The procedure described above surrounds each vertex with a variable number of triangles. When the desired edge length is small enough, we have experienced that this number (i.e. the *valence* of the vertex) is exactly 6 on planar regions, less than 6 on sufficiently convex/concave regions, and more than 6 on saddles.

6. THE TILING ALGORITHM

A few more details must be considered in order to design a working algorithm. Firstly, at each step a vertex must be processed and it must be chosen from those of the current boundary. It is not difficult to see that the order of the processing influences the final result, in particular, the insertion of some *short* edges (that is, thin triangles) may be required in some cases, since nothing of what we said till now prevents the boundary forming acute angles at any step. By experiment we have found that choosing the vertex with the smallest angle to be triangulated drastically reduces such cases. So we sort the boundary vertices in a queue and, at each step, we pop the first vertex from it. A similar method has been used by Hartmann in [20], where the expansion on the implicit surface proceeds using the same sorting technique for choosing the boundary vertex to be triangulated. Summarizing, the following is a sketch of the algorithm:

1. INPUT: Surface definition (mapping function), parameter domain (bounds or trimming curves) and desired edge length (sampling step);
2. Choose a random point of the parameter domain and build the starting NU;
3. Build the queue containing the boundary vertices whose pre-image is not on the boundary of the parameter domain;
4. Pop the first vertex from the queue, complete its approximate NU and update the queue;
5. While the queue is not empty go to 4.

As explained in section 5, the number of paths to be tracked around the first vertex (that is, the degree of the NU) depends on the length of the NU-front and the radius. In our implementation we compute this length by linearly interpolating the end-points of a fixed number of normal sections around the vertex. Of course, the higher this number, the more precise is the result. The adaptive step-size Runge-Kutta method [25] is used to track the normal sections. When tracking a path, we prevent crossings of the domain boundary by breaking the process on the first intersection point. This makes the final triangulation less uniform near the boundary.

6.1. Intersection checks

In cases where the curvature is particularly high it may happen that the iterative expansion of the mesh makes the boundary intersect itself, so, before inserting a new edge, we perform an intersection test between the edge and the other parts of the current boundary.

7. IMPLEMENTATION AND COMPLEXITY

We have implemented our method, including the extensions outlined in the following sections, using standard C++ under Linux. The prototype tessellates surfaces defined over a rectangular domain whose bounds must be specified by the user. The mapping function must be defined using the standard C syntax and it must be coupled with its partial derivatives; in the current version, derivatives must be provided by the user, however they could be automatically computed using a symbolic calculus library.

We give the complexity as a function of the area of the surface to be triangulated. Let A be this area; the number of elements of the boundary is $O(A^{1/2})$. The number of triangles is linearly proportional to A, and the computation of each one of them requires $O(A^{1/2})$ intersection checks. The priority queue contains only the boundary vertices, so its size is $O(A^{1/2})$, and the insertion of each new vertex requires an update of the queue, that is, $O(\log(A^{1/2}))$ operations. Summarizing, the algorithm performs $O(A)$ steps, and the dominating operation of each step takes $O(A^{1/2})$ time, thus the global complexity is $O(A^{1.5})$.

8. EXTENSION TO CLOSED SURFACES

The basic algorithm, as it has been described, can only manage open surfaces with the same topology as their parameter domains. In order to build the tessellation of surfaces with different topology, we look for paths of (u,v) points that map on a unique 3D image (cutting line). For example, in the typical definition of the sphere by parallels and meridians, the two lines (0,0)-(0,1) and (1.0)-(1,1) of the parameter domain map on the same meridian (Fig. 7.).

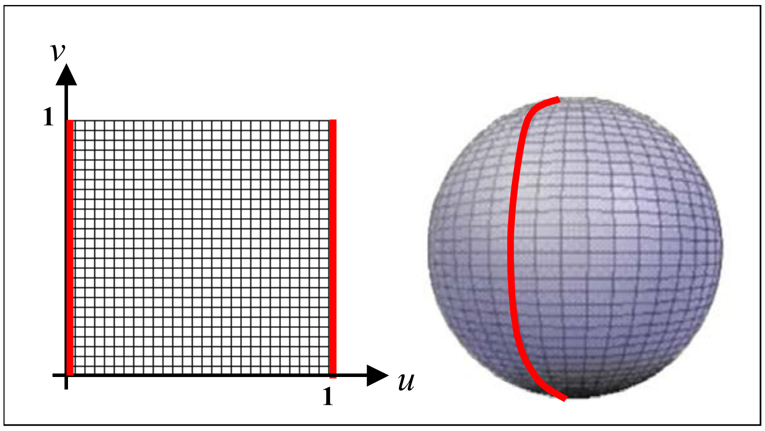


FIG. 7. Two edges of the parameter square map on the same line on the sphere.

We have implemented a sewing procedure based on a split-and-merge of pairs of boundary elements that *touch* each other, without being properly connected. Specifically, once the whole parameter domain has been covered, we pick a boundary vertex v and check if it is too close (i.e. the distance is less than half the desired edge length) to a boundary edge e whose vertices are both different from v. If such an edge exists, we split it by v, and exploit adjacencies to zip the boundary performing edge splits where needed, as shown in Fig. 8, until the vertex-edge distance exceeds half the threshold distance. If e does not exist for the chosen vertex, we perform the test on another one, until all the boundary vertices have been checked.

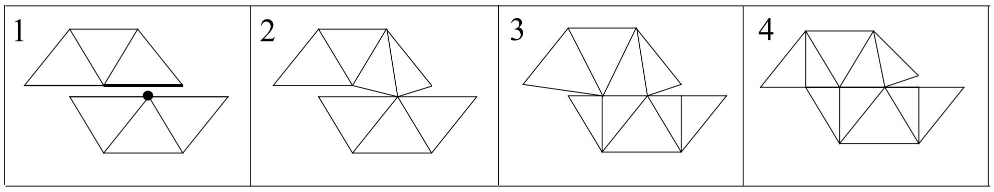


FIG. 8. Joining two parts of the boundary. The gap has been enhanced to show the lacking topological connection.

Our algorithm can tessellate every regularly parameterized C^1 2-manifold and, by this simple extension, the resulting mesh also reflects possible closures of the input surface through a coherent connectivity graph. Moreover, since we compute distances in the Euclidean 3D space, the same procedure can be used to join several parametric patches that were triangulated using the same sampling step. In Fig. 9 the triangulation of a torus is shown as it has been computed by our implementation of the algorithm.

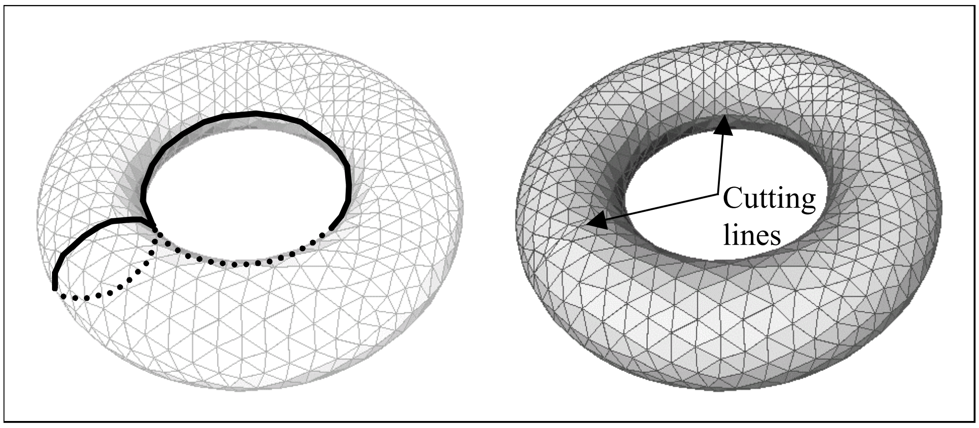


FIG. 9. Triangulation of a torus with corrected connectivity.

9. ADAPTIVITY

The method described so far can tessellate a surface in an almost uniform manner and whatever its parameterization. If a more precise mesh is necessary, we have developed a scheme to refine the initial coarse triangulation iteratively. The user chooses the expected edge-length, representing the dimension of the smallest feature that must be detected, then he sets the maximum acceptable distance, ε, of an edge from the surface. Once the initial mesh has been constructed, each edge is split at its farthest point from the surface if this point is farther than ε. In Fig. 10 the triangulation of a volcano shape is shown [26]; the initial mesh is shown on the left. No adaptation has been used and the number of triangles, nt, is 354. The other three are adaptive meshes obtained by refining the first one with different tolerances. Notice how the density and shape of the triangles follow the curvature and its principal directions. This *stretching* can be seen even better in Fig. 11; here the triangles are elongated in the direction of the cylinder axis. All the depicted triangulations were computed through our prototype.

Let us suppose that the user requires a short sampling step because, for example, some small features must be captured. In this case, if the surface has large and flat regions, a significant number of redundant triangles is used to tessellate such regions. This may happen even if the adaptive scheme described above is used. Thus, we further extend our method by the means of a mesh simplification procedure driven by the underlying parametric surface. In particular, we iteratively perform edge-contractions as described in [27] and stop when any further contraction would produce a distance edge-surface bigger than ε. The application of the refinement followed by the simplification makes our method able to produce anisotropic meshes that are particularly suitable for fast rendering (Fig. 11c).

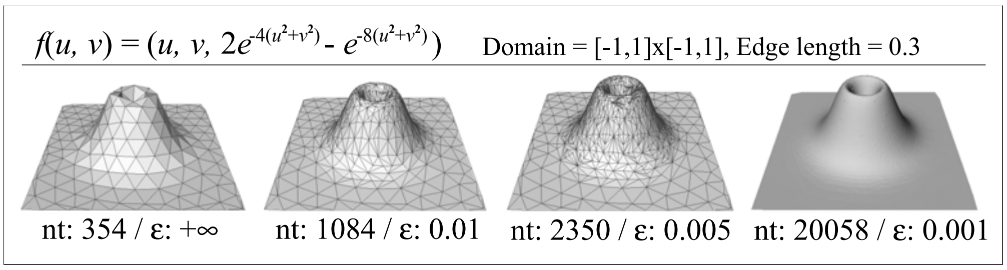

$f(u, v) = (u, v, 2e^{-4(u^2+v^2)} - e^{-8(u^2+v^2)})$
Domain = [-1,1]x[-1,1], Edge length = 0.3
nt: 354 / ε: +∞
nt: 1084 / ε: 0.01
nt: 2350 / ε: 0.005
nt: 20058 / ε: 0.001

FIG. 10. Adaptive triangulation of a volcano with different tolerances.

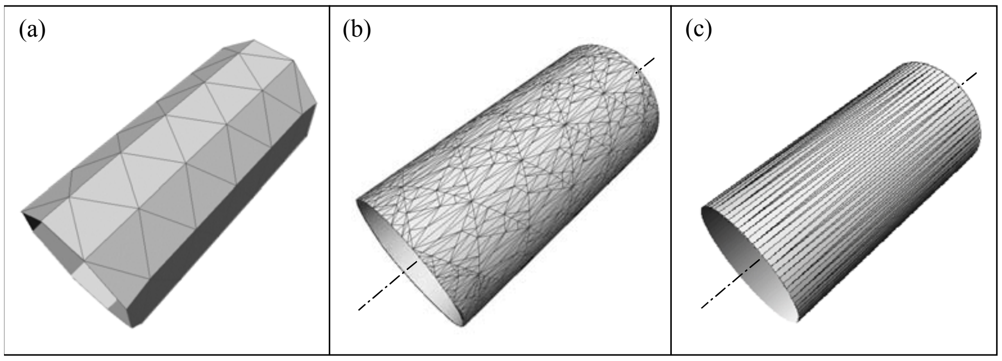


FIG. 11. Uniform triangulation of a cylinder (a), after the refinement (b) and after the simplification (c). Triangles are stretched along the direction of the axis.

10. CONCLUSION

In this paper we presented a method for overcoming the limits of existing algorithms for the polygonization of parametric surfaces. The normal umbrella allows us to build a triangulation depending only on the surface's intrinsic characteristics, avoiding the problems of *bad* parameterization. The algorithm works on all regularly parameterized surfaces of class C^1, provided that the surface is a 2-manifold. The method for *sewing-up* the boundary of closed surfaces can be used to join several parametric patches, as usual in CAD applications. The output meshes are rather uniform, so that they are suitable for applications of the FEM or other kinds of numerical simulation. If the initial triangulation is fine enough, the adaptation criterion used allows the construction of bounded error meshes. Moreover, the simplification described makes us able to create anisotropic triangulations for applications requiring fast rendering.

ACKNOWLEDGEMENTS

This work is part of the bilateral research agreement "Shape-based meshing techniques" – University of Otago and IMATI-GE/CNR. Marco Attene's work on this project was partially supported by the EC project ARION (EC project #IST-2000-25289) and the Marsden Fund in New Zealand. The authors thank all the members of the Computer Graphics Group of the IMATI-GE/CNR and the ones of the Computer Graphics and Vision Lab of the University of Otago, for their encouragement in writing the paper and their helpful advice.

REFERENCES

1. J.C. Cuillière, An adaptive method for the automatic triangulation of 3D parametric surfaces, in *Computer Aided Design*, Vol. 30, No. 2, 1998, pp.139-149.
2. J.R. Tristano, S.J. Owen and S.A. Canann, Advancing Front Surface Mesh Generation in Parametric Space Using a Riemannian Surface Definition, in *7th Intl. Meshing Roundtable Proceedings*, 1998.
3. K. Shimada and D.C. Gossard, Automatic triangular mesh generation of trimmed parametric surfaces for finite element analysis, in *Computer Aided Geometric Design*, Vol 15, 1998, pp.199-222.
4. X. Sheng and B. Hirsch, Triangulation of trimmed surfaces in parametric space, in *Computer Aided Design*, Vol. 24, No. 8, 1992, pp.437-444.
5. H. Chen and J. Bishop, Delaunay Triangulation for Curved Surfaces, in *6th Intl. Meshing Roundtable Proceedings*, 1997, pp. 115-127.
6. C. L. Bajaj and A. Royappa, Triangulation and Display of Rational Parametric Surfaces, in *IEEE Visualization'94 Proceedings*, 1994, pp.69-76.
7. S. Kumar, Interactive Rendering of Parametric Spline Surfaces, *PhD thesis, Department of Computer Science, University of N. Carolina at Chapel Hill*, 1996.
8. L.A. Piegl and A.M. Richard, Tessellating trimmed NURBS surfaces, in *Computer Aided Design*, Vol. 30, No. 1, 1998, pp.11-18.
9. M. Attene and M. Spagnuolo, Automatic Surface Reconstruction from Point Sets in Space, in *Computer Graphics Forum*, *EUROGRAPHICS 2000 Proceedings*, Vol. 19, No. 3, 2000, pp.457-465.
10. N. Amenta, S. Choi and R. Kolluri, The power crust, in *Sixth ACM Symposium on Solid Modeling and Applications*, 2001, pp. 249-260.
11. L. Velho and L. H. De Figueiredo, Optimal adaptive polygonal approximation of parametric surfaces, in *SIBGRAPI'96 Proceedings*, 1996., pp.127-133
12. M. Attene and G. Wyvill, Mapping independent triangulation of parametric surfaces, Technical Sketch in *SIGGRAPH 2001 Conference Abstracts and Applications*, 2001, pp. 224.
13. J.T. Kajiya, Ray tracing parametric surfaces, in *Computer Graphics, SIGGRAPH'82 Proceedings*, Vol. 16, No. 3, 1982, pp. 245-254.
14. K.I. Joy and M.N. Bhetanabholta, Ray tracing parametric surfaces patches utilizing numerical techniques and ray coherence, in *Computer Graphics, SIGGRAPH'86 Proceedings*, Vol. 20, No. 4, 1986, pp. 279-285.
15. V. Vlassopoulos, Adaptive polygonization of parametric surfaces, in *The Visual Computer*, Vol. 6, 1990, pp.291-298.
16. M. Vigo, N. Pla and P. Brunet, Curvature Adaptive Triangulations of Surfaces, *Technical Report LSI-00-16-R, Universitat Politècnica de Catalunya*, 2000.
17. M. Bern and D. Eppstein, Mesh Generation and Optimal Triangulation, in *Computing in Euclidean Geometry*, 2nd Edition, 1995, pp.47-123.
18. S.Z. Li, Adaptive sampling and mesh generation, in *Computer Aided Design*, Vol. 27, No. 3, 1995, pp.235-240.
19. F.J. Bossen and P.S. Heckbert, A Pliant Method for Anisotropic Mesh Generation, in *5th Intl. Meshing Roundtable Proceedings*, 1996, pp.63-74.
20. E. Hartmann, A marching method for the triangulation of surfaces, in *The Visual Computer*, Vol. 14, No. 3, 1998, pp.95-108.
21. K. Shimada, Anisotropic Triangular Meshing of Parametric Surfaces via Close Packing of Ellipsoidal Bubbles, in *6th Intl. Meshing Roundtable Proceedings*, 1997, pp. 375-390.
22. M. Kosters, Curvature-dependent parameterization of curves and surfaces, in *Computer Aided Design*, Vol. 23, No. 8, 1991, pp.569-578.
23. M.M. Lipschutz, Shaum's Outline of Differential Geometry, *Ph.D., Hahnemann Medical College*, McGraw-Hill, 1969.
24. M. Attene, S. Biasotti and M. Spagnuolo, Re-Meshing Techniques for topological analysis, in *Shape Modeling and Applications Proceedings*, 2001, pp.142-151.
25. W.H. Press, Numerical Recipes in C: The art of scientific computing. *Cambridge ; New York : Cambridge University Press*, 1992.
26. W. Seibold and G. Wyvill, Near-Optimal Adaptive Polygonization, in *CGI'99 Proceedings*, IEEE Computer Society, 1999, pp. 206-213.
27. M. Garland and P.S. Heckbert, Surface Simplification Using Quadric Error Metrics, in *Computer Graphics, SIGGRAPH'97 Proceedings*, 1997, pp. 209-218.
28. J.M. Lane, L.C. Carpenter, T. Whitted and J.F. Blinn, Scan Line Methods for Displaying Parametrically Defined Surfaces, in *ACM Communications*, Vol. 23, No. 1, 1980, pp. 29-34.