Triangulating Trimmed NURBS Surfaces

Chang Shu and Pierre Boulanger

Abstract. This paper describes techniques for the piecewise linear approximation of trimmed NURBS surfaces. The problem, called surface triangulation, arises from many applications in CAD and graphics. The new method generates triangular meshes that are adaptive to the local surface curvature. We use efficient data structures for the handling of trimming curves. We also generate Delaunay triangulation on the surface to improve the quality of the meshes.

§1. Introduction

Tensor-product NURBS are widely used in today's CAD systems for describing and exchanging surface geometry. For many applications, however, piecewise linear approximations of smooth surfaces are required. Examples of these applications include finite element analysis, stereo-lithography, and visualization of geometric models. In these applications, we need to generate a triangular mesh that approximates the original surface within a given tolerance. We refer to this problem as surface triangulation, stated in the following definition.

Definition. *Given a NURBS surface $N(u,v)$, its trimming boundary, and a real number ε, the* surface triangulation problem *is to find a set of linearly parameterized triangles $\{T_i\}$ such that*

1) *Any triangle T_i satisfies* $\sup \|T_i(u,v) - N(u,v)\| < \varepsilon$.
2) *For any triangle edge not on the boundary, there is exactly one neighboring triangle sharing this edge.*

The first condition is usually called chord height tolerance, which restricts triangles to be close to the surface. The second condition requires the triangular mesh to be topologically correct.

A good surface triangulation algorithm is expected to be efficient because real world models tend to contain large numbers of surface patches. Furthermore, certain optimization factors are desirable. Two of the most important ones are triangle shape and the number of triangles in the mesh. For example, in finite element analysis, triangles with bad aspect ratio (one angle is

Curve and Surface Design: Saint-Malo 1999
Pierre-Jean Laurent, Paul Sablonnière, and Larry L. Schumaker (eds.), pp. 1–8.

ISBN 0-8265-1356-5.

significantly smaller or larger than the others) reduce the solution precision. In all applications, a mesh with a small number of triangles saves computing and transmission time as well as storage space.

Several authors [4,6,8,14] have approached the surface triangulation problem by computing a bound on the length of triangle edges in parametric space so that if all triangles have their edge lengths smaller than the bound, the resulting triangulation satisfies the chord height tolerance. Since the edge length bound applies to the whole surface, the density of the triangulation distributes uniformly across the surface and may lead to unnecessarily large mesh size. Along another line of thought, Klein and Straßer [5] considered the problem of placing points based on the surface curvature. Recently, Piegl and Tiller [11] used adaptive subdivision of the surface. Obviously, for a given chord height tolerance, adaptive algorithms generate fewer triangles than the uniform subdivision algorithms. But adaptive algorithms tend to be slower.

In this paper, we give a method that has the following features:

1) adaptive to the surface curvature,
2) efficient insertion of the trimming curves, and
3) triangle shape improvement.

Our general strategy is that we first approximate the surface with hierarchical quadrilaterals without considering the trimming curves, then we insert the trimming curves and triangulate the quadrilaterals. The result is a triangulation that satisfies the chord height tolerance. We improve the efficiency of trimming curve insertion by organizing the quadrilateral hierarchies in a quadtree structure. Also, we improve the quality of the triangles by converting the initial triangulation to a Delaunay triangulation.

§2. Curve and Surface Subdivision

We begin by discretizing the surface and its trim curves independently. We assume that the surfaces are trimmed in the parametric domain and the trimming curves are represented as NURBS with consistent orientation. Our first objective is to approximate the curves with connected line segments such that they do not deviate from the surface more than the tolerance ε. ¿From well-known results in B-Spline theory [7,10], a NURBS curve can be split into two pieces without changing its shape by inserting new knots. The consequence of this splitting is that we introduce new control points that are closer to the curve than the control points of the original curve. If we keep dividing in this way, the control polygon converges to the surface. When a sub-curve's control polygon becomes "flat" enough, we can stop the dividing process. According to the convex hull property of NURBS, the maximum distance from any point on the sub-curve to the line segment joining the two end control points is bounded by the maximum distance between control points to the segment. Therefore, we can use this bound to control the flatness of the sub-curves.

The surface can be approximated in the same way by quadrilaterals. At this time, we ignore the trimming boundary. Here, we insert knots in both u

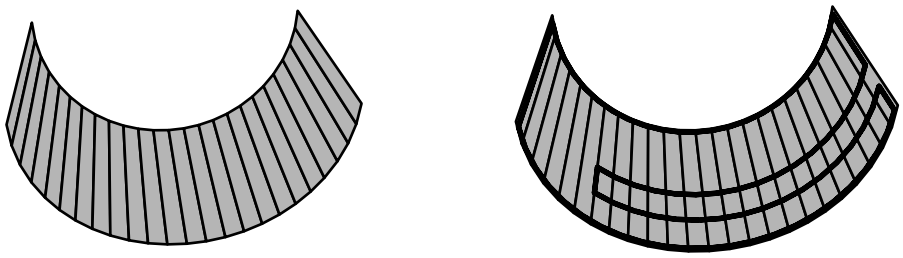

Fig. 1. Full surface subdivision.

and v directions. The flatness test is a little more complex. We examine every row and column of the control polygon and test their flatness. We also have to consider the twist factor of a surface patch. Peterson [10] gives a subdivision method, which we generally follow. Fig. 1 (left) shows a surface approximated by quadrilaterals.

We use a quadtree data structure to keep track of the surface subdivision process. A quadrilateral is divided if it does not satisfy the flatness test. Its children are subject to the same test until at a certain level they are flat enough. Therefore, more subdivisions are needed at places where surface curvature is high.

§3. Trim Curve Insertion

We assume the trimming curves are given in the parametric space of the surfaces which they delineate. They are first discretised into line segments using the same tolerance for the surface. Then the trimming curve segments are inserted into the quadtree cells by walking through the quadrilaterals using adjacency information. The right-hand figure in Fig. 1 shows an example of the insertion.

The insertion can be done completely in the parametric domain in which the quadrilaterals correspond to rectangles in two dimensions. Starting with a vertex of the trimming segments, we first find the rectangle in which this vertex is contained. This can be done efficiently by traversing down the quadtree. By following the trimming segments, we can find the segment that crosses one of the edges of the rectangle. We insert a new vertex on the intersection point and then start the insertion in the new rectangle.

For efficient insertion of the trimming segments, we make use of a data structure that can quickly find the neighboring rectangle from the edge of a rectangle. We store in each rectangle (quadrilateral) vertex the pointers of the rectangles that use the vertex. Given an edge $e = (p, q)$, we collect all the rectangles that contain both p and q, $Q_e = Q_p \cap Q_q$, where Q_p and Q_q are the sets of rectangles associated with vertices p and q respectively. This is a local operation. The number of elements in Q_e should either be 1 or 2.

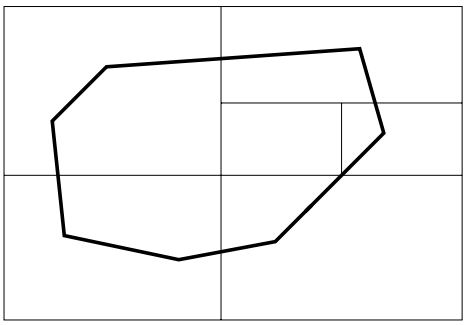

Fig. 2. Trimming curve insertion.

In the case of two elements, one of them is the neighboring rectangle we look for. When there is only one element in Q_e, our rectangle does not have a compatible neighbor. However, if we go up the quadtree, at some level there must be a rectangle that is the compatible neighbor. Then coming down the tree, we find the leaf rectangle that is partially neighboring our initial quad. This is the rectangle that the trimming segment enters.

The time complexity of the insertion process is linear in the number of quadrilaterals.

Fig. 2 shows the insertion of trimming curves into the rectangles in parametric domain. Fig. 3 shows the two cases that a trimming segment enters a new rectangle: 1) entering from an edge; 2) entering from a vertex.

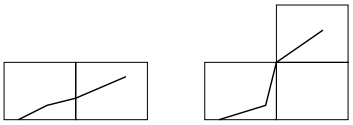

Fig. 3. Two entering cases.

§4. Initial Triangulation

After the insertion of the trimming segments, we have two kinds of rectangles: those that are cut by the trimming segments and those that do not intersect with any part of the trimming segments. For each rectangle being cut, we sort the vertices of the trimming segments inside the rectangle in counter-clockwise order to form boundaries of polygons. In general, there can be multiple polygons and each polygon can have multiple boundary loops. Those cells that lie inside the boundary are triangulated in parametric space. There is no shortage of triangulation algorithms for 2-dimensional polygonal domains. Here we adopt the algorithm from [13]. Note that most rectangles lie completely in the interior of the trimming boundary; their triangulation can simply be done by triangulating a rectangular domain [1]. If a rectangle is cut-free and lies outside the trimming boundary, we can simply ignore it. This case can be decided easily by testing if one of the vertices of the rectangle is in the interior of the polygon formed by the trimming segments. For robustness reasons, we choose the centroid of the rectangle for doing the test.

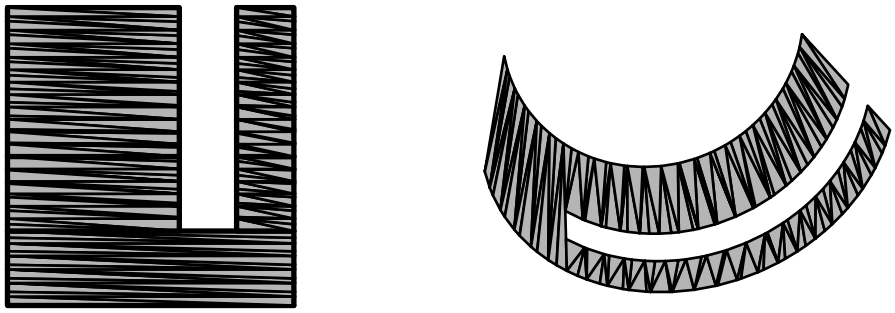

Fig. 4. Initial triangulation in parametric space and in 3-space.

An initial triangulation in 3-space is obtained by evaluating the parametric triangulation. Fig. 4 gives an example of a surface triangulated in the parametric domain (left) and the corresponding triangulation in 3-space (right).

§5. Triangle Shape Improvement

As we mentioned in Section 1, there are good reasons to make triangles that are well-shaped. In practice, it is undesirable to have triangles that are flat or pointed. These are the triangles that have one small angles or one large angles. It is well known that Delaunay triangulation for a set of points in two dimensions is optimal in the sense that it maximizes the minimum angle [2]. Delaunay triangulation that respect a set of boundary edges can be constructed. This kind of triangulation is called constrained Delaunay triangulation (CDT) [12]. Chew [3] extended the definition of CDT to the curved surfaces by replacing the empty circumcircle condition with the empty minimum circumsphere condition. Following Chew's approach, we improve the shape of the triangles by edge flipping and inserting new nodes at the circumcenters of the ill-shaped triangles.

Given a pair of triangles, if they form a convex quadrilateral, there are two choices of the diagonals, one is better than the other in terms of the shapes of the triangles. By examining each pair of adjacent triangles and flipping their diagonals if necessary, we can improve the triangulation locally (see top figures of Fig. 5). Chew [3] shows that the flipping process halts and it leads to constrained Delaunay triangulation on a surface.

A CDT is the best possible triangulation without introducing new nodes. To further improve the triangulation, we have to insert new points. Each time we insert a new point, we do edge flipping again to maintain Delaunay triangulation. New points are inserted at the circumcenters of the triangles that violate the shape criteria. The reason of this is we could improve the shape of several triangles by introducing one point. Fig. 5 illustrates the two basic operations used repeatedly for improving the shape of the triangles.

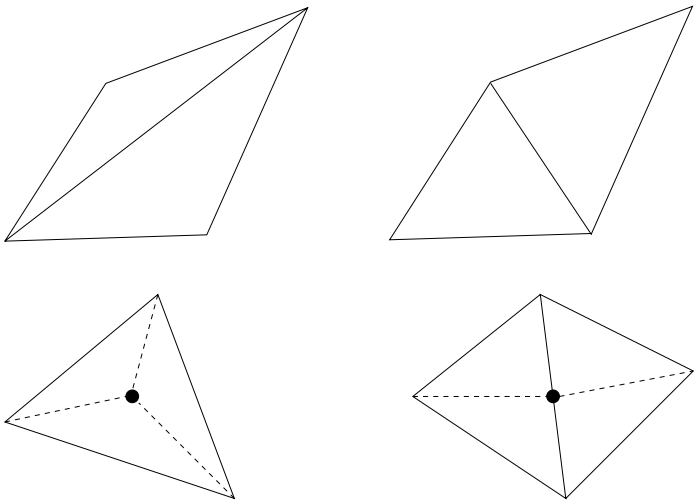

Fig. 5. Edge flipping and node insertion.

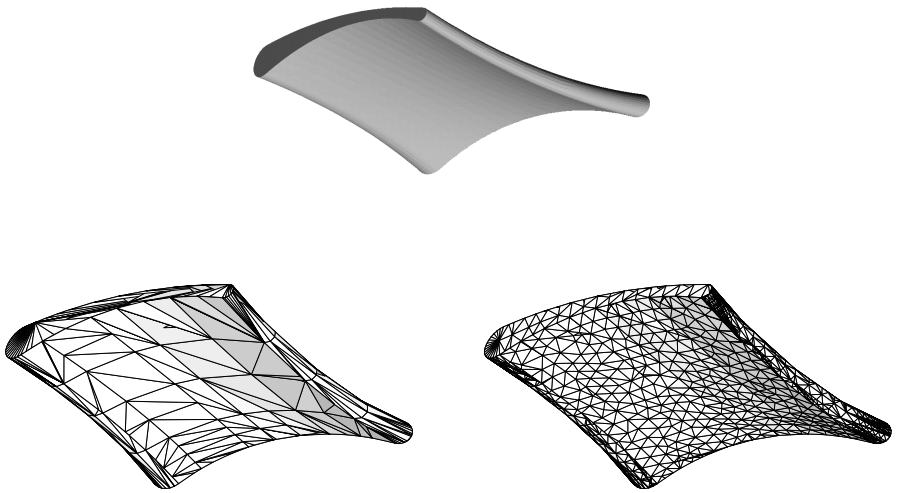

Fig. 6. Example 1.

As we flip edges, we want to preserve the error bound for the new triangulation. Given a pair of triangles that are flippable, we check the minimum distance between the current diagonal and the new diagonal. The distance should be smaller than the specified approximation tolerance ε. This does not guarantee that the resulting triangulation still satisfies the approximation tolerance. But since we are not moving any nodes on the surface and we insert additional nodes into the triangulation, there are good reasons to assume that most triangles will satisfy the tolerance. Finally, as a last step, we loop through all triangles and check their chord heights. For those few triangles that violate chord height tolerance, we subdivide them by adding points on their edges. The checking is generally expensive, but we only do this once

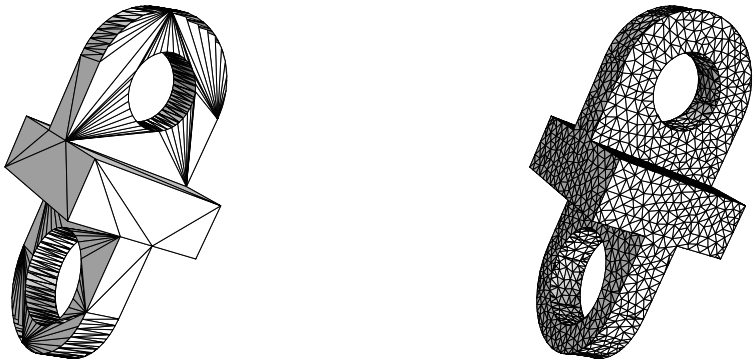

Fig. 7. Example 2.

for each triangle, and the process can be speeded up by making use of the quadtree data structure.

Figs.6 and 7 give two results of the algorithm before and after shape improvements.

§6. Concluding Remarks

The main results of this work are:

1) a surface triangulation algorithm that guarantees correct mesh topology,
2) an efficient trimming curve insertion scheme, and
3) triangle shape improvement by Delaunay triangulation.

We have only discussed the problem of triangulating a single surface. However, in real world problems, a model usually consists of many NURBS surface patches. The triangulation between two neighboring surfaces have to be compatible. There should be a post-processing step that stitches the triangulation of different surfaces. This can be done with a kd-tree data structure, which facilities locating nearest nodes in 3-space quickly. Therefore, we can propagate a node on the boundary of one surface to the boundary of its neighboring surface.

References

1. Baehmann, P. L., S. L. Wittchen, M. S. Shephard, K. R. Grice and M. A. Yerry, Robust, geometrically based, automatic two-dimensional mesh generation, International Journal for Numerical Methods in Engineering **24** (1987), 1043–1078.
2. Bern, M. and D. Eppstein, Mesh generation and optimal triangulation, in *Computing in Euclidean Geometry, 2nd ed.*, D.-Z. Du and F. K. Hwang, (eds.), World Scientific, 1995, 47–123.
3. Chew, P., Guaranteed-quality mesh generation for curved surfaces, Proc. of the 9th Annual Symposium on Computational Geometry, 1993, 115–127.

4. Filip, D. J., R. Magedson and R. Makot, Surface algorithms using bounds on derivatives, Comput. Aided Geom. Design **3** (1986), 295–311.
5. Klein, K. and W. Straßer, Large mesh generation from boundary models with parametric face representation, Proc. of the 3rd ACM Symposium on Solid Modeling and Applications, 1995, 431–440.
6. Kumar, S. and D. Manocha, Efficient rendering of trimmed NURBS surfaces, Computer-Aided Design **27** (1995), 509–521.
7. Lane, J. M. and R. F. Riesenfeld, A theoretical development for the computer generation and display of piecewise polynomial surfaces, IEEE Trans. Pattern Analysis and Machine Intelligence **2** (1980), 35–46.
8. Lane, J. M. and R. F. Riesenfeld, Bounds on a polynomial, BIT **21** (1981), 112-117.
9. Peterson, J. W., Tessellation of NURBS surfaces, in *Graphics Gems IV*, P. S. Heckbert (ed.), Academic Press, New York, 1994, 286–320.
10. Piegl, L. and W. Tiller, *The NURBS Book*, Springer-Verlag, 1997.
11. Piegl, L. and W.Tiller, Geometry-based triangulation of trimmed NURBS surfaces, Computer-Aided Design **30** (1998), 11–18.
12. Ruppert, J., A Delaunay refinement algorithm for quality 2-dimensional mesh generation, Journal of Algorithms **18** (1995), 548–585.
13. Seidel, R., A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons, Computational Geometry: Theory and Applications **1** (1991), 51–64.
14. Sheng, X. and B. E. Hirsch, Triangulation of trimmed surfaces in parametric space, Computer-Aided Design **24** (1992), 437–444.

Chang Shu and Pierre Boulanger
Institute for Information Technology
National Research Council of Canada
Montreal Road, Ottawa, Ontario
Canada K1A 0R6
chang.shu@iit.nrc.ca, pierre.boulanger@iit.nrc.ca