Hierarchical Generalized Triangle Strips

Luiz Velho[†], Luiz Henrique de Figueiredo[‡], and Jonas Gomes[†]

[†]IMPA – Instituto de Matemática Pura e Aplicada
[‡]LNCC – Laboratório Nacional de Computação Científica

Abstract

This paper introduces a new refinement method for computing triangle sequences of a mesh. We apply the method to construct a single generalized triangle strip that completely covers a parametric or implicit surface. A remarkable feature of this application is that, our method generates the triangulation and the triangle strip simultaneously, using a mesh refinement scheme. As a consequence, we are able to produce a hierarchy of triangle strips defined at each refinement level. This data structure has many applications in geometry compression and rendering.

CR Categories and Subject Descriptors: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling; I.3.6 [Computer Graphics]: Methodology and Techniques. J.6 [Computer-Aided Engineering]: Computer-Aided Design (CAD).

Additional Keywords: geometric modeling; mesh representation; sequential triangulations; Hamiltonian paths; triangle strips; accelerated rendering; geometry compression.

1 Introduction

Triangle meshes, or *triangulations*, are one of the most widely used representations for geometric models. A triangulation is a 2-dimensional simplicial complex, a simple structure with nice combinatorial properties. Moreover, surfaces of arbitrary topology can be tessellated into a mesh of triangular patches. Also, triangles can be rendered very efficiently in both software and hardware. For this reason, they are the basic geometric primitive of the graphics pipeline.

One of the main disadvantages of triangle meshes is that, in general, they do not provide a compact surface representation, because a large number of triangles is required to faithfully describe the geometry of a complex surface.

This problem motivated the search for encoding schemes that could be used to represent triangle meshes in a more compact and efficient way. One such scheme is the *triangle strip* (and its generalizations). The mesh encoding using triangle strips exploits spatial coherence of the simplicial complex structure. It enumerates mesh elements in a sequence of adjacent triangles to avoid repeating the vertex coordinates of shared edges.

In the traditional setting, the triangle strip encoding is posed as the problem of converting a given triangle mesh into the minimal set of triangle strips covering the mesh. This problem is NP-complete [6] and, thus, existing algorithms must rely on heuristics to find sub-optimal solutions.

Triangle meshes are often used to approximate smooth surfaces, or to interpolate data points. In these cases, the mesh is generated, respectively, from a surface description (in parametric or implicit form), or from sparse samples. Based on this observation, we noticed that triangle sequences can be constructed during the process of generating the mesh. In that way, the complexity of the problem is reduced and a better solution can be found.

In this paper we describe a method for constructing a hierarchy of generalized triangle strips. The method can be integrated with the mesh creation process. In fact, with small computational effort, we are able to build triangle sequences that have many good properties.

The remainder of the paper is organized as follows: Section 2 gives definitions and some background on triangle sequences; Section 3 discusses previous work; Section 4 presents the basic method for generating hierarchical triangle sequences using refinement; Section 5 analyzes uniformly refinable sequential triangulations; Section 6 investigates the case of adaptive refinement; Section 7 shows an application of the method to obtain triangular strips that completely covers implicit and parametric surfaces; Section 8 concludes with final remarks and directions for future work.

2 Definitions and Background

In order to investigate the different ways of representing triangle meshes, we need to study their topological structure, in terms of connectivity and adjacency relations between the elements of the mesh. For that purpose, it is convenient to look at the *dual graph* of the mesh, which gives explicitly the adjacency relations between

[†] IMPA – Instituto de Matemática Pura e Aplicada,
Estrada Dona Castorina, 110, 22460-320 Rio de Janeiro, RJ, Brazil, {lvelho | jonas}@visgraf.impa.br
[†] LNCC – Laboratório Nacional de Computação Científica,
Rua Lauro Müller 455, 22290-160 Rio de Janeiro, RJ, Brazil, lhf@lncc.br

the triangles of the mesh. In the dual graph, nodes correspond to triangles, and two nodes are connected when their associated triangles have a common edge. Figure 1 shows a triangulation and its dual graph.

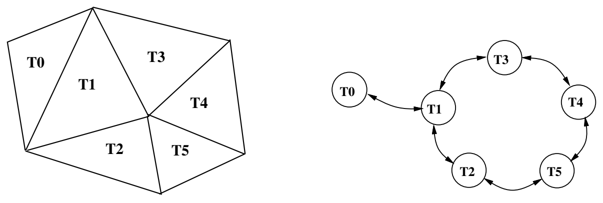
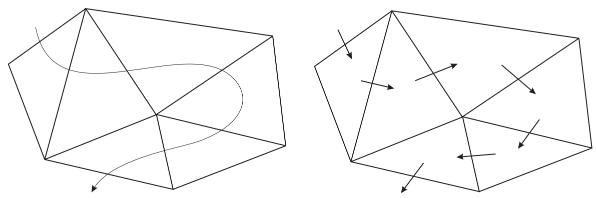


Figure 1: Triangulation and its dual graph.

A *generalized sequential triangulation*, or *Hamiltonian triangulation*, is a triangulation for which there is an ordering $T_1, \ldots, T_N$ of all its triangles such that two consecutive triangles T_j and T_{j+1} share an edge. Such an ordering exists if and only if the dual graph of the triangulation contains a Hamiltonian path. (A path in a graph is called *Hamiltonian* when it visits all nodes in the graph exactly once.)

Notice that each triangle in a Hamiltonian triangulation has an entry and an exit edge, with respect to the Hamiltonian ordering. The knowledge of these edges completely characterizes the sequence. Geometrically, a generalized triangle sequence can be represented by drawing an oriented path on the mesh domain, which visits each triangle crossing its entry and exit edges (see Figure 2(a)). This path is called a *sequential path*. Another representation consists in indicating the entry and exit edges with an arrow (see Figure 2(b)).



(a) (b)

Figure 2: Geometric representations of a triangle sequence.

In a Hamiltonian triangulation, suppose that T_{j-1} is a triangle, with a vertex v_i which is shared with the next triangle T_j (see Figure 3). Then, the triangle T_j is completely determined by specifying:

- a new vertex v_{i+1} (i.e. a vertex distinct from the vertices of triangle T_{j-1};
- the edge shared by T_{j-1} and T_j, which can be the edge to the left (counterclockwise order) or to the right (clockwise order) of the current vertex v_i.

It follows that a generalized sequential triangulation with N triangles can be encoded by a sequence of $N+2$ vertices, defining the geometric information, and a sequence of N bits, defining the connectivity information (i.e. *left* or *right* edge turn). This encoding is called a *generalized triangle strip*.

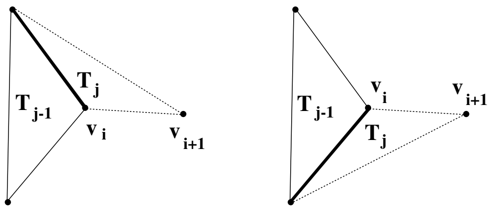
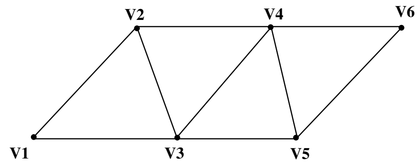


Figure 3: Two options for continuing a triangle sequence.

A *sequential triangulation* is a particular kind of generalized sequential triangulation, in which the shared edges follow an alternating left/right turn order. Because of this implied restriction, the connectivity does not need to be explicitly encoded, and the representation is given only by the sequence of $N+2$ vertex coordinates or id's. This encoding is called a *triangle strip*. Figure 4 shows an example of a sequential triangulation and its representation as a triangle strip.



$$(V_1, V_2, V_3, V_4, V_5, V_6)$$

Figure 4: A sequential triangulation and its triangle strip encoding.

The triangle strip, as well as the generalized triangle strip, are standard representations of triangle meshes and are supported by most graphics systems, including the OpenGL graphics library [3, 8].

In this paper we shall use the term *triangle sequence* to designate a sequential triangulation, as well as its generalizations.

A triangle sequence defines a total order relation on the set of triangles of a mesh. It is always possible to partition a triangle mesh $\mathcal{T}$ into a collection of sub-triangulations $\mathcal{T}_1, \ldots, \mathcal{T}_M$, such that each $\mathcal{T}_k$ is a triangle sequence. (A trivial partitioning can be obtained by using one triangle for each sequence $\mathcal{T}_k$.) The partition defines a partial order on the triangulation $\mathcal{T}$: triangles in the same sub-triangulation are related according to their order in the sequence; triangles in differents sub-triangulations are not related. Figure 5 shows a partial order on a mesh consisting of two triangle sequences.

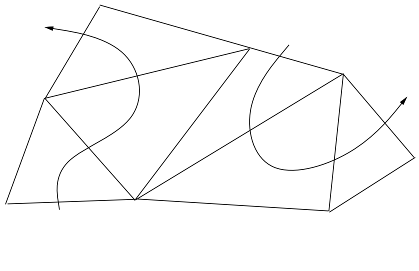


Figure 5: Partial ordering with two triangle sequences.

A natural problem is how to obtain the optimal partition of a mesh into triangle sequences $\mathcal{T}_1, \ldots, \mathcal{T}_M$, that is, a partition that minimizes M. Clearly, minimizing M is related to maximizing the length of each triangle sequence $\mathcal{T}_k$. Depending on the application, it may also be desirable to optimize the sequences of left/right edge turns within a sequence.

An optimal partition is the most compact representation of a triangle mesh among all possible sets of triangle sequence encodings. Ideally, we should have a total order among all of the triangles in a mesh, that is, a single triangle strip covering the mesh completely. However, this is not possible in general because the dual graph is not always Hamiltonian. For example, the triangle mesh of Figure 5 is not Hamiltonian.

On the other hand, the problem of finding the best collection of triangle sequences of a mesh is NP-complete [6]. Nonetheless, by adding new vertices (called Steiner points), it is always possible to refine a triangle mesh into a Hamiltonian triangulation [2]. Figure 6 shows an example of the insertion of a Steiner point to produce a Hamiltonian triangulation.

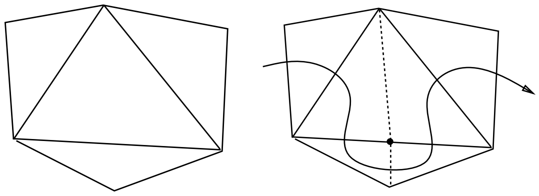


Figure 6: Hamiltonian triangulation obtained by insertion of a Steiner point.

Thus, it seems reasonable that triangulation methods based on insertion of vertices, such as mesh refinement schemes, could be used to generate Hamiltonian triangulations. This paper focus on this problem. In the next section, we review some of the strategies that have been used to compute good triangle sequences; in Section 4 we present a new method to construct Hamiltonian triangulations using refinement.

3 Previous Work and Applications

Previous work related with triangle strips fall into three major categories: algorithms to generate triangle strips for accelerated rendering; algorithms to compute sequential triangulations for geometry compression; and theoretical investigation of paths on triangle meshes.

Triangle strips are important for accelerated rendering because they can significantly increase the throughput of the visualization pipeline. First, the data rate is increased, since only $N + 2$ vertex coordinates have to be sent to the graphics engine for a sequence of N triangles. Second, viewing operations, such as matrix transforms and clipping, need to be applied only once to the elements of the data stream, further increasing the rendering performance.

Akeley, Haeberly and Burns [1] developed a program to convert triangle meshes into strips using a greedy algorithm. They build a sequence by always choosing the next triangle as the one adjacent to the least number of neighbors. Speckmann and Snoeyink [9] build triangle strips based on the dual graph of a triangulation. Their algorithm computes a minimum spanning tree of the adjacency graph and segment it using heuristics that tend to produce long strips. Evans, Skiena and Varshney [7] use a technique called patchfication, which identifies rectangular regions of a mesh consisting of quadrilaterals. Such regions can be trivially encoded as triangle strips. This last strategy is similar to the one employed in our method, in the sense that it builds the triangle strip while creating a triangulation.

The properties of sequential triangulations can be exploited in various ways for the compression of geometric models. On the one hand, the sequential structure reduces the connectivity information to one bit, or even eliminates its explicit encoding. On the other hand, the locality of reference inherent in a triangle sequence makes possible to encode geometry with fewer bits using prediction schemes.

Deering [4] introduced the concept of geometry compression based on generalized triangle sequences. His algorithm employs lossy compression for the quantization of coordinate values, as well as a vertex cache to further take advantage of spatial coherence. Taubin and Rossignac [10] construct a mesh representation using two interlocked trees: a spanning tree of triangles for connectivity; and a spanning tree of vertices. The geometry information is compressed using a variable-length lossy encoding.

The generation of paths on triangulations has been studied in computational geometry. Dillencourt [5] investigated the complexity of finding Hamiltonian paths on the edges of Delaunay triangulations. Arkin, Held, Mitchell and Skiena [2] considered several issues related to paths on the dual graph of general triangle meshes. In particular, they showed that the problem of finding optimal triangle sequences is NP-hard.

4 Triangle Sequences from Mesh Refinement

In this section we give an overview of our method to construct generalized triangle strips using mesh refinement.

A *refinable triangle sequence* is a triangle sequence whose total order can be preserved when its elements are subdivided. Note that this property depends exclusively of the subdivision scheme adopted.

A triangle subdivision scheme is called a *sequential refinement* if it is based on a subdivision template that:

- decomposes each triangle into a generalized sequential triangulation;
- the refined sub-sequence within each triangle is compatible with the ordering of the initial triangulation.

More precisely, if

$$T_1, \ldots, T_{j-1}, T_j, T_{j+1}, \ldots, T_M$$

is the initial triangle sequence, then each triangle T_j has a refinement $T_{j1}, \ldots, T_{jN_j}$, which is itself a triangle sequence, and such that the triangle mesh

$$T_1, \ldots, T_{j-1}, \underbrace{T_{j1}, \ldots, T_{jN_j}}_{T_j}, T_{j+1}, \ldots, T_M,$$

obtained by replacing triangle T_j by its refinement, is still a triangle sequence.

The sequential refinement we describe in this paper has two main parts:

1. **Initialization:** Generate a base mesh and an initial triangle sequence for it.

2. **Refinement:** Refine the base mesh, recursively propagating the triangle sequence from coarse down to finer levels.

The construction of the initial triangle sequence can (and should) take advantage of properties of the base mesh. If the mesh has a regular structure, then there is a natural order of its elements. Figure 7 shows a simple regular base mesh with a triangle sequence following a serpentine path.

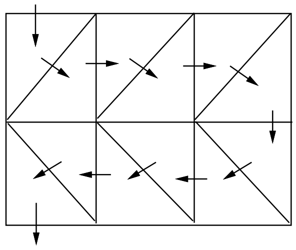


Figure 7: Base mesh and serpentine triangle strip.

Even when no structural properties of the base mesh can be exploited, just the fact that the base mesh is coarse by construction makes it feasible to compute the initial triangle sequence using conventional algorithms, such as the ones described in Section 3.

The second stage of the method takes an ordered base mesh and refines the mesh, maintaining the order as it moves from coarse to finer levels.

A description of the refinement method, along with a proof of its correctness, is done by induction. We start from an initial triangle sequence $\mathcal{T}^k$, with triangles ordered as: $T_1^k, \ldots, T_{j-1}^k, T_j^k, T_{j+1}^k, \ldots, T_{M_k}^k$. We need to subdivide each triangle T_j^k into a triangle sequence $\mathcal{T}_j^{k+1} = T_{j1}^{k+1}, \ldots, T_{jN_k}^{k+1}$, such that the refined triangle mesh

$$\mathcal{T}_1^{k+1}, \mathcal{T}_2^{k+1}, \ldots, \mathcal{T}_{M_k}^{k+1}$$

is, in this order, a triangle sequence.

Suppose, by induction, that we have accomplished the task up to the j-th triangle. That is, we have constructed the refined triangle sequence

$$\mathcal{T}_1^{k+1}, \mathcal{T}_2^{k+1}, \ldots, \mathcal{T}_j^{k+1}.$$

To complete the induction, we must devise a method to refine the triangle T_{j+1}^k into a triangle sequence $\mathcal{T}_{j+1}^{k+1}$, such that

$$\mathcal{T}_1^{k+1}, \mathcal{T}_2^{k+1}, \ldots, \mathcal{T}_j^{k+1}, \mathcal{T}_{j+1}^{k+1}$$

is also a triangle sequence.

Note that the triangle T_{j+1}^k has an entry edge and an exit edge already determined by the original triangle sequence $\mathcal{T}_k$. Also, the entry edge of T_{j+1}^k is the exit edge of T_j^k (edge CB in Figure 8). Therefore, by the induction hypothesis, this edge has been possibly subdivided in the refinement process of the triangle T_j^k, and one of the sub-edges of the refinement is the exit edge of the last triangle of the sequence $\mathcal{T}_j^{k+1}$ (sub-edge MC in Figure 8). The refinement of triangle T_{j+1}^k subdivides its exit edge, and one of the sub-edges must be chosen for the exit edge of the last triangle in the refinement sequence $\mathcal{T}_{j+1}^{k+1}$ of the triangle T_{j+1}^k (sub-edge ND in Figure 8).

The above induction step shows that the accomplishment of our goal is completely localized: we must devise a template for triangle refinement that makes the induction step possible. The template

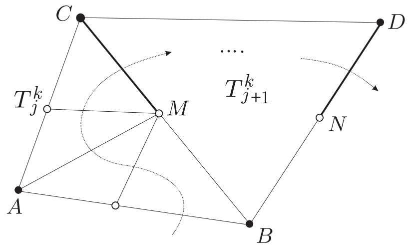


Figure 8: Propagating the triangle sequence during refinement.

must be able to provide a sequence path from the entry sub-edge, to one of the exit sub-edges. We have two strategies for the template construction:

Uniform template: We construct a single template that its used to subdivide every triangle, creating a uniform sequential refinement of the mesh.

Adaptive templates: We use different templates in order to construct an adaptive sequential refinement.

The core of the algorithm is the construction of the triangle refinement template. In the next two sections, we discuss how to determine suitable subdivision templates for both uniform and adaptive mesh refinement.

5 Uniform Sequential Refinement

Uniform mesh refinement schemes recursively subdivide all triangles of a mesh, using the same template, until some desired resolution is obtained.

In the case of triangle meshes, the most common refinement scheme is to split all edges (usually at the edge midpoint) in two parts, and apply a subdivision template to decompose each triangle into sub-triangles. In this case, there are two possibilities of subdivision templates for the uniform decomposition of a triangle. Both subdivide a triangle into four sub-triangles. These templates are shown in Figure 9.

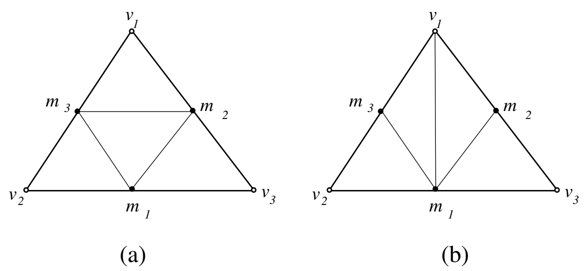


Figure 9: Uniform decomposition of a triangle: (a) isotropic template; (b) anisotropic template.

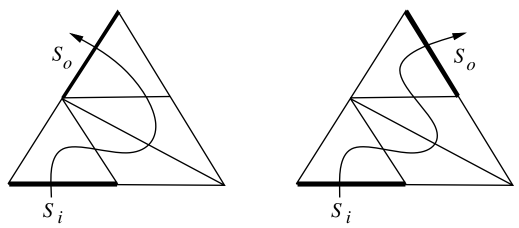
Template 9(a) induces an isotropic decomposition. It connects each new vertex m_i, at the midpoint of an edge, to vertex $m_{(i+1) \bmod 3}$, at the midpoint of the subsequent edge.

Template 9(b) induces an anisotropic decomposition. It selects the midpoint m_i of one edge, and connects it to the opposite vertex of the triangle, v_i, as well as to the midpoints of the two adjacent edges, $m_{|(i-1) \bmod 3|}$ and $m_{|(i+1) \bmod 3|}$. This decomposition scheme can have three different realizations, each corresponding to the selection of one of three edge midpoints. Note that the use of one of these particular configurations implies choosing a preferred direction to split the triangle. We have exploited this extra degree of freedom to conform the mesh geometry to a surface (see Section 7).

Simple inspection reveals that template 9(a) does not produce a generalized sequential triangulation, whereas template 9(b) does. Therefore, only template 9(b) is suitable to define a sequential refinement scheme. This result is summarized in the following

Theorem 1 *A triangle sequence is uniformly refinable if all its elements are subdivided using template 9(b) in a orientation compatible with the sequence order.*

Proof. From the discussions in the previous section, we need to determine an exit sub-edge and construct a sequence path from the entry sub-edge to the exit sub-edge. There are two possible choices for the exit sub-edge corresponding to the bottom-left and the bottom-right. Figure 10 shows a template subdivision and the sequence path for each case. □



left exit right exit

Figure 10: Realization of template 9(b) compatible with Bottom-Left and Bottom-Right order.

We remark that the solution shown above is not unique. In fact, template 9(b) gives two possible orientations for the case where the exit edge is not adjacent to the entry sub-edge. One alternative is shown in the right side of Figure 10; the other alternative would be a configuration connecting the midpoint of the entry edge to the opposite vertex.

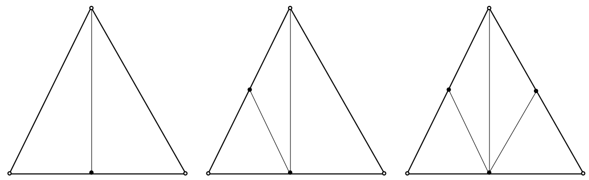
In the next section, we construct the templates for adaptive sequential refinement.

6 Adaptive Sequential Refinement

Adaptive refinement schemes decompose the mesh in a non-uniform manner.

In non-uniform subdivision, triangle edges are also split in two parts (usually at the midpoint), but unlike uniform schemes, not all three edges of a triangle are subdivided. Mesh elements are subdivided up to different levels based on some local adaptation criteria. The recursive subdivision of a triangle stops when all three edges satisfy the adaptation criteria.

A consequence of the adaptive nature of non-uniform refinement is that the subdivision templates are more complex. They must take into account the cases of one, two and three edges splitting. Therefore, the templates consist of all simplicial decompositions generated by internal edges connecting edge midpoints and/or triangle vertices. Figure 12 shows the subdivision templates corresponding respectively to: (i) one edge split; (ii) two edge split; and (iii) three edge split. Note that template (iii) is the same one used for uniform subdivision. Obviously, the realization of these templates includes all permutations of different edges splitting, as well as all configurations for connecting them.



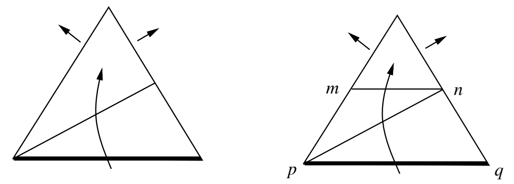
(i) (ii) (iii)

Figure 12: Templates for non-uniform subdivision.

As for uniform refinement, in order to prove that we can construct a refinable triangle sequence, it is sufficient to show that templates (i), (ii) and (iii) produce triangle sequences that are compatible with the parent triangle sequence order.

Since the cases involving template (iii) have already been analyzed in Section 5, there are three cases remaining to be considered, depending on the types of entry and exit edges:

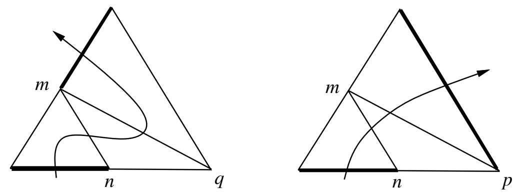
1. *Entry edge does not split.* This is the simplest case. The triangle is subdivided by a main internal edge, transversal to the sequence path. The exit edge could be either to the left or to the right. This configuration is illustrated in Figure 13. Note that there are two possible realizations for the configuration shown in Figure 13(b), i.e. a subdivision of the quadrilateral $pmnq$ into the triangles pmn and pnq, or pmq and mnq.



(a) (b)

Figure 13: Subdivision when entry edge does not split.

2. *Entry edge splits, and the adjacent edge to the entry sub-edge also splits.* As shown in Figure 14, in this case, the internal edges mn and mp produce a refinable triangle sequence that begins at the entry sub-edge and can exit either at one of the two other edges.

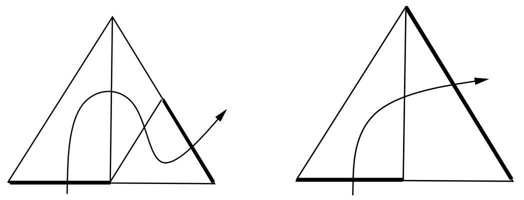


(a) (b)

Figure 14: Subdivision when adjacent edge splits.

3. *Entry edge splits, and the adjacent edge to the entry sub-edge does not split.* This is the most difficult case, and we break it down in two sub-cases depending on the exit edge:

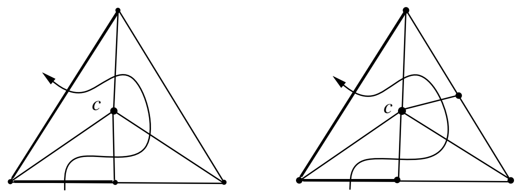
3-a. When the exit edge is not adjacent to the entry sub-edge, the refined triangle sequence can be constructed easily, as shown in Figure 15.



(a) (b)

Figure 15: Subdivision when adjacent edge does not split, and exit edge is not the adjacent edge.

3-b. When the exit edge is adjacent to the entry sub-edge, it is not possible to produce a triangle sequence with templates (ii) and (iii). To overcome this situation, we create two new templates, by inserting an extra vertex, c, in the interior of the triangle, and connecting it to the other vertices. These new templates, (iv) and (v), are illustrated in Figure 16.



(iv) (v)

Figure 16: New templates for subdivision when adjacent edge does not split, and exit edge is the adjacent edge.

In summary, the above case analysis demonstrated that templates (i)–(v) are sufficient to produce refinable non-uniform triangle sequences. They can be used with any adaptive mesh refinement method that is based on edge subdivision.

In the next section, we apply the adaptive sequential refinement to compute triangle strips for implicit and parametric surfaces.

7 Examples

We have implemented an algorithm that employs the sequential mesh refinement method described in this paper to compute simultaneously, both a hierarchical triangulation and a hierarchy of triangle sequences. The algorithm produces superior results compared with methods where the triangle strip computation is done separately starting from a pre-computed triangulation.

The first stage of the algorithm, as outlined in Section 4, creates a base mesh that will be used in the refinement stage. The second stage of the the algorithm, recursively refine the base mesh, using the either the uniform or the non-uniform subdivision templates that we introduced, respectively, in Sections 5 and 6.

We now present some examples of triangulations and triangle sequences computed using the algorithm.

Unit Square Figure 17 shows the first three levels of a hierarchy of triangle sequences generated using uniform mesh refinement of the square $[0, 1] \times [0, 1]$ of the plane.

The left part of the figure displays the triangle meshes and the right part displays the corresponding triangle sequences, shown as a polygonal path connecting the midpoints of entry and exit edges.

The base mesh was a decomposition of the square, into two triangles, along one diagonal. Note that the refinement process produces a triangle sequence that follows the path of a space-filling curve (a Sierpinski-like curve).

Height field. Figure 18 shows a height field surface given by the regular sampling of a terrain elevation model. The base mesh was a 2×2 rectangular grid, with the initial triangle sequence following a serpentine path. The mesh was refined down to 4 levels using uniform subdivision.

Sphere. Figure 19 shows the adaptive tessellation of a sphere without the polar caps, described parametrically by:

$$x = \cos u \cos v, \quad y = \sin u \cos v, \quad z = \sin v;$$
$$u \in [0, 2\pi], \quad v \in [\pi/2 + 0.2, 3\pi/2 - 0.2]$$

Figures 19(a) and (b) display respectively the decomposition of the parametric domain and the triangulation of the surface in 3-space. Figures 19(c) and (d) display the paths corresponding to the triangle sequence respectively in parameter space and on the surface. Notice that the triangulation forms a restricted structure, while the triangle sequence follows the path of a space-filling curve that is adapted to the surface.

Parametric torus. Figure 20 shows the adaptive tessellation of a torus, described parametrically by:

$$x = \cos u(r + \cos v), \quad y = \sin u(r + \cos v), \quad z = \sin v;$$
$$u \in [0, 2\pi], \quad v \in [0, 2\pi], \quad r = 1.6.$$

As in the previous example, the triangle sequence also follows a space-filling path.

Trimmed Bézier patch. Figure 21 shows the polygonization of a trimmed Bézier surface patch with control points

$$\begin{bmatrix} (-1,0,0) & (0,3,0) & (3,3,0) & (4,0,0) \\ (1,0,1) & (1,-3,1) & (3,3,1) & (2,0,1) \\ (0,0,2) & (0,3,2) & (3,3,2) & (3,0,2) \\ (-1,0,4) & (0,3,3) & (3,-3,3) & (4,2,3) \end{bmatrix}.$$

Despite of the complex topology of the parametric domain, we were able to generate a well adapted mesh, as well as a triangle sequence that coves the entire patch.

8 Conclusions and Future Work

We have presented a new method for constructing generalized triangle strips from an initial, coarse sequence. The method uses both uniform and adaptive refinement schemes. We have also applied the method to obtain refinable generalized triangle sequences that approximate parametric or implicit surfaces.

The recursive nature of the process actually produces a hierarchy of triangle sequences, one for each level of refinement. This structure can be employed with advantages in the compression of multiresolution models, and in accelerated progressive rendering.

As future work, we plan to investigate several issues, including: the properties of space filling curves on manifolds; the potential of locality of reference to increase geometric compression; and applications of triangle sequences to artistic rendering of surfaces.

REFERENCES

[1] K. Akeley, P. Haeberly, and D. Burns. tomesh.c: C program on SGI developer's toolbox CD, 1990.

[2] E. Arkin, M.Held, J. Mitchel, and S. Skiena. Hamiltonian triangulations for fast rendering. In *Second Annual European Symposium on Algorithms*, pages 36–47. Springer Verlag, 1994. Lecture Notes in Computer Science 855.

[3] Open GL Architecture Review Board. *OpenGL Reference Manual*. Addison Wesley, 1993.

[4] Michael F. Deering. Geometry compression. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conferenece Series, pages 13–20. ACM SIGGRAPH, Addison Wesley, August 1995. Held in Los Angeles, California, 06-11 August 1995.

[5] M. Dillencourt. Finding Hamiltonian cycles in Delaunay triangulations is NP-complete. In *Proc. 4th Canad. Conf. Comput. Geom.*, pages 223–228, 1992.

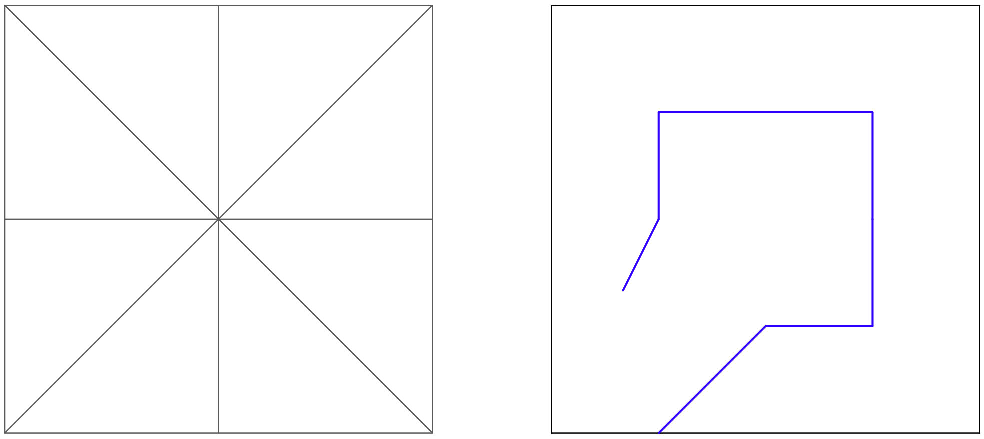
[6] F. Evans, S. Skiena, and A. Varshney. Completing sequential triangulations is hard. Technical Report Department of Computer Science, State University of New York, Stony Brook, March 1996.

[7] Francine Evans, Steven S. Skiena, and Amitabh Varshney. Optimizing triangle strips for fast rendering. In *IEEE Visualization '96*. IEEE, October 1996. ISBN 0-89791-864-9.

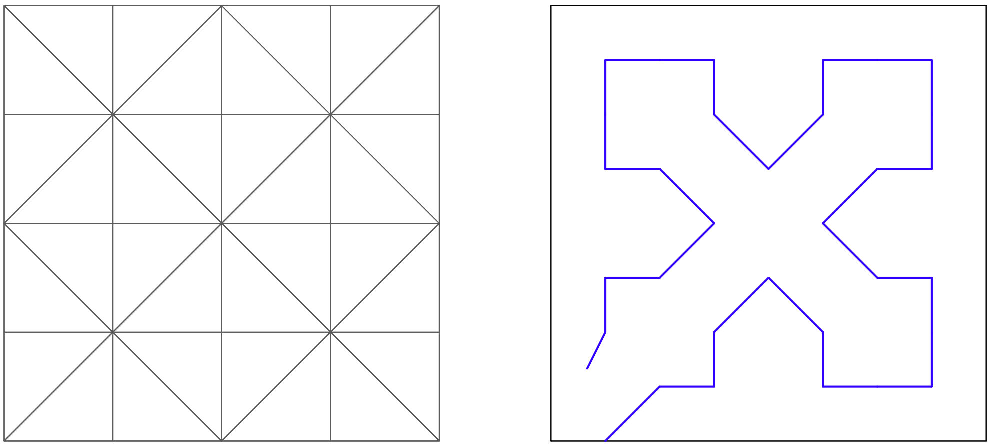
[8] Mark J. Kilgard. Realizing OpenGL: Two implementations of one architecture. In Steven Molnar and Bengt-Olaf Schneider, editors, *1997 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 45–56, New York City, NY, August 1997. ACM SIGGRAPH / Eurographics, ACM Press. ISBN 0-89791-961-0.

[9] B. Speckmann and J. Snoeyink. Easy triangle strips for TIN terrain models. In *Canadian Conference on Computational Geometry*, pages 239–244, August 1997.

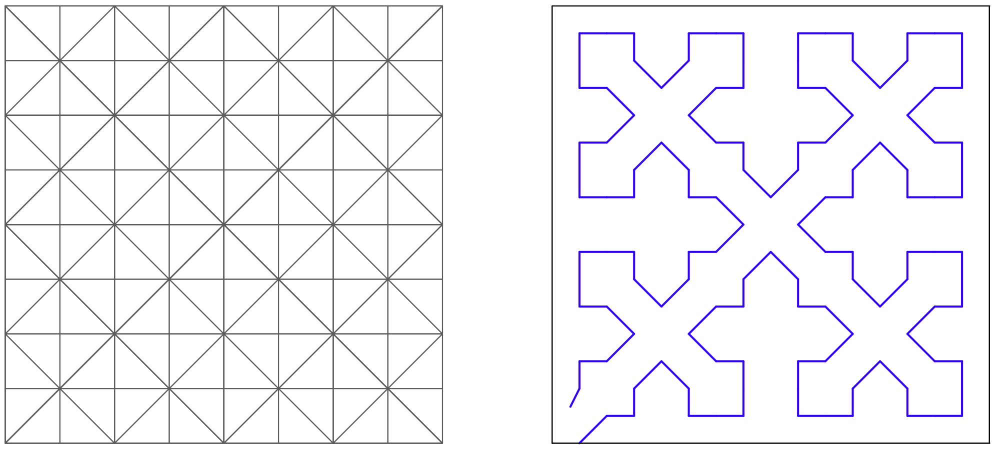
[10] Gabriel Taubin and Jarek Rossignac. Geometric compression through topological surgery. Technical Report RC 20340, IBM T.J. Watson Research Center, 1996.



level 1

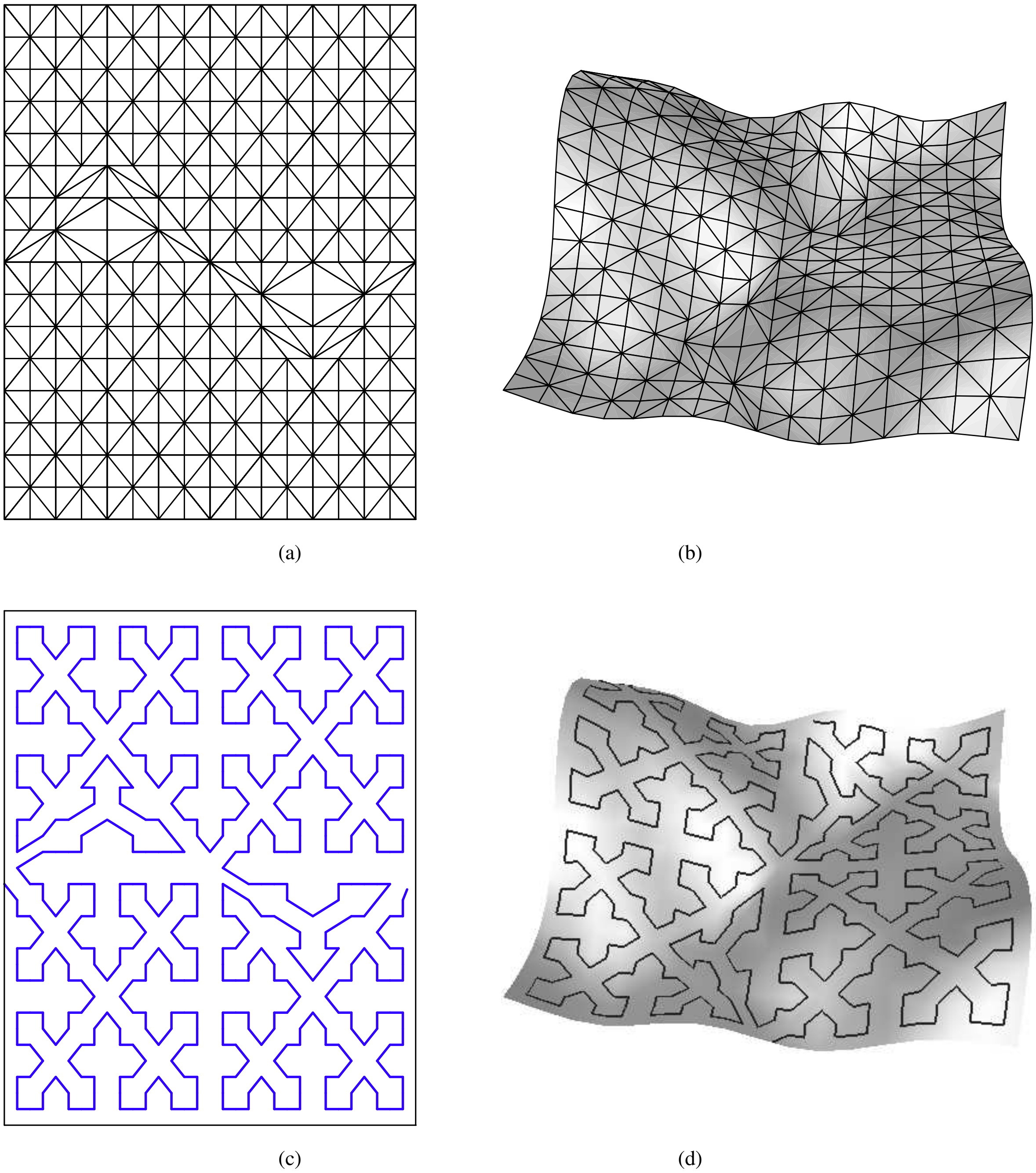


level 2



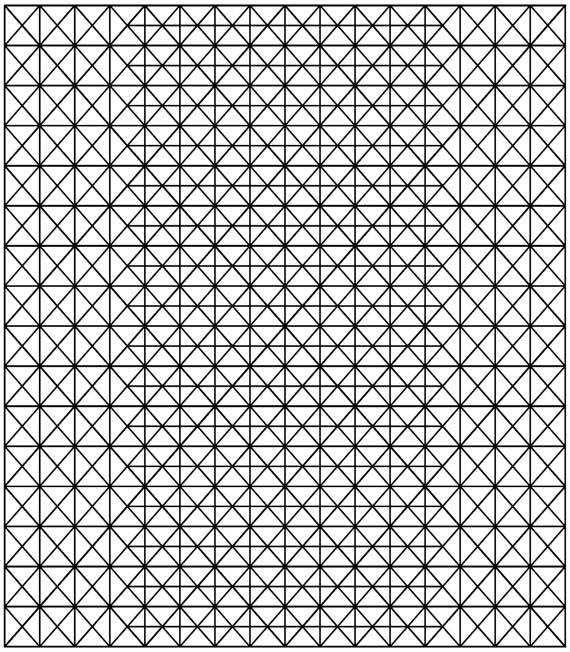
level 3

Figure 17: Hierachical mesh and triangle sequences.

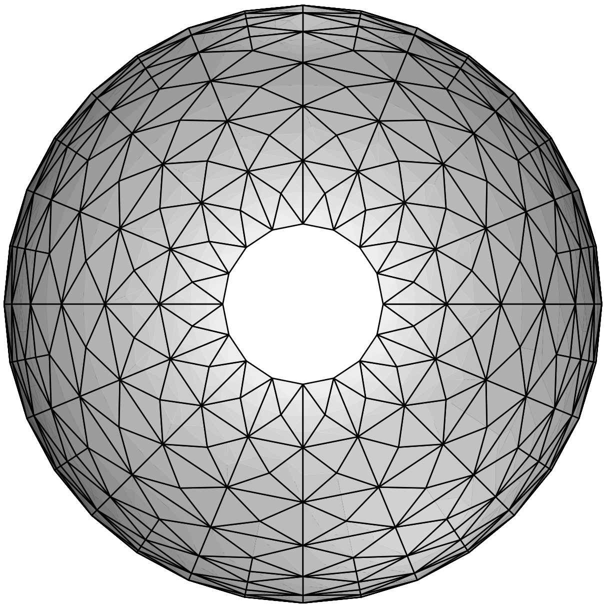


(a) (b) (c) (d)

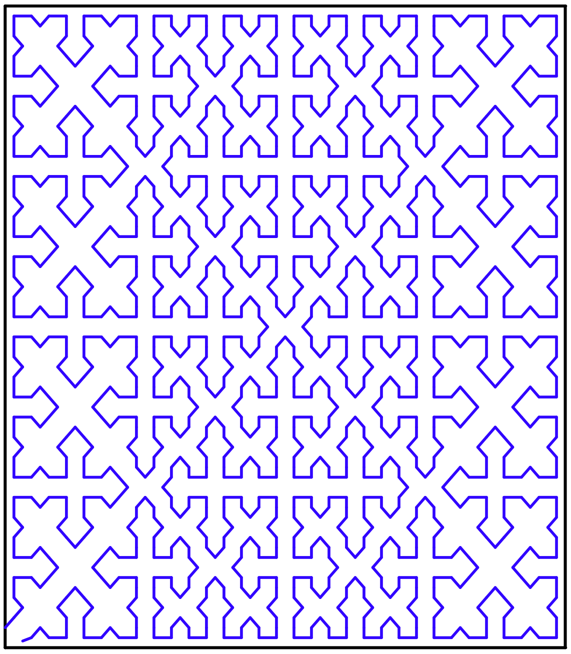
Figure 18: Uniform Tesselation of Terrain Model



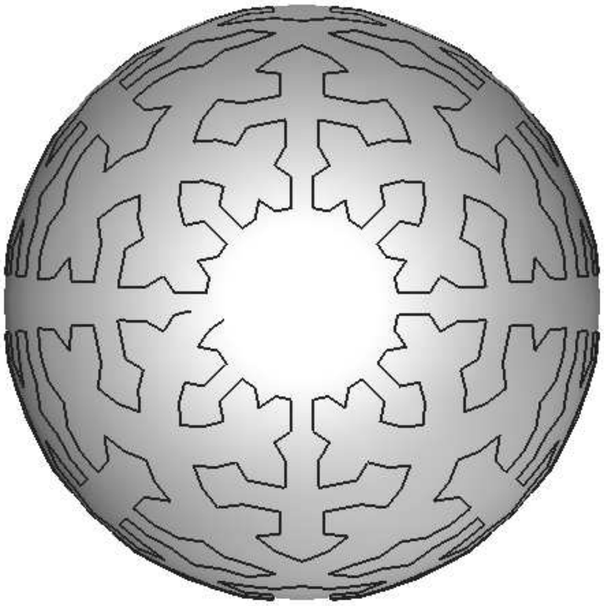
(a)



(b)



(c)



(d)

Figure 19: Sphere

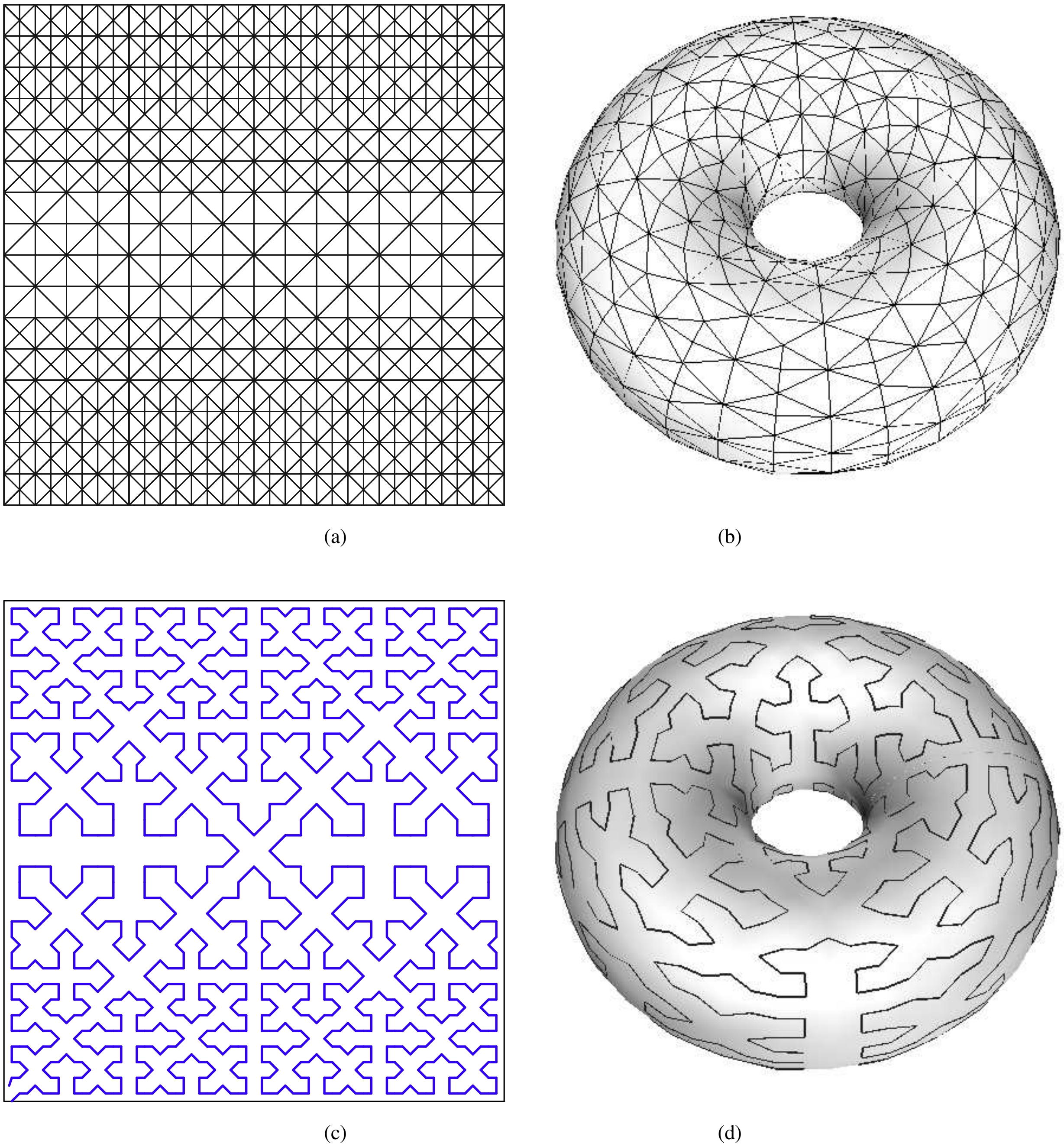


Figure 20: Torus

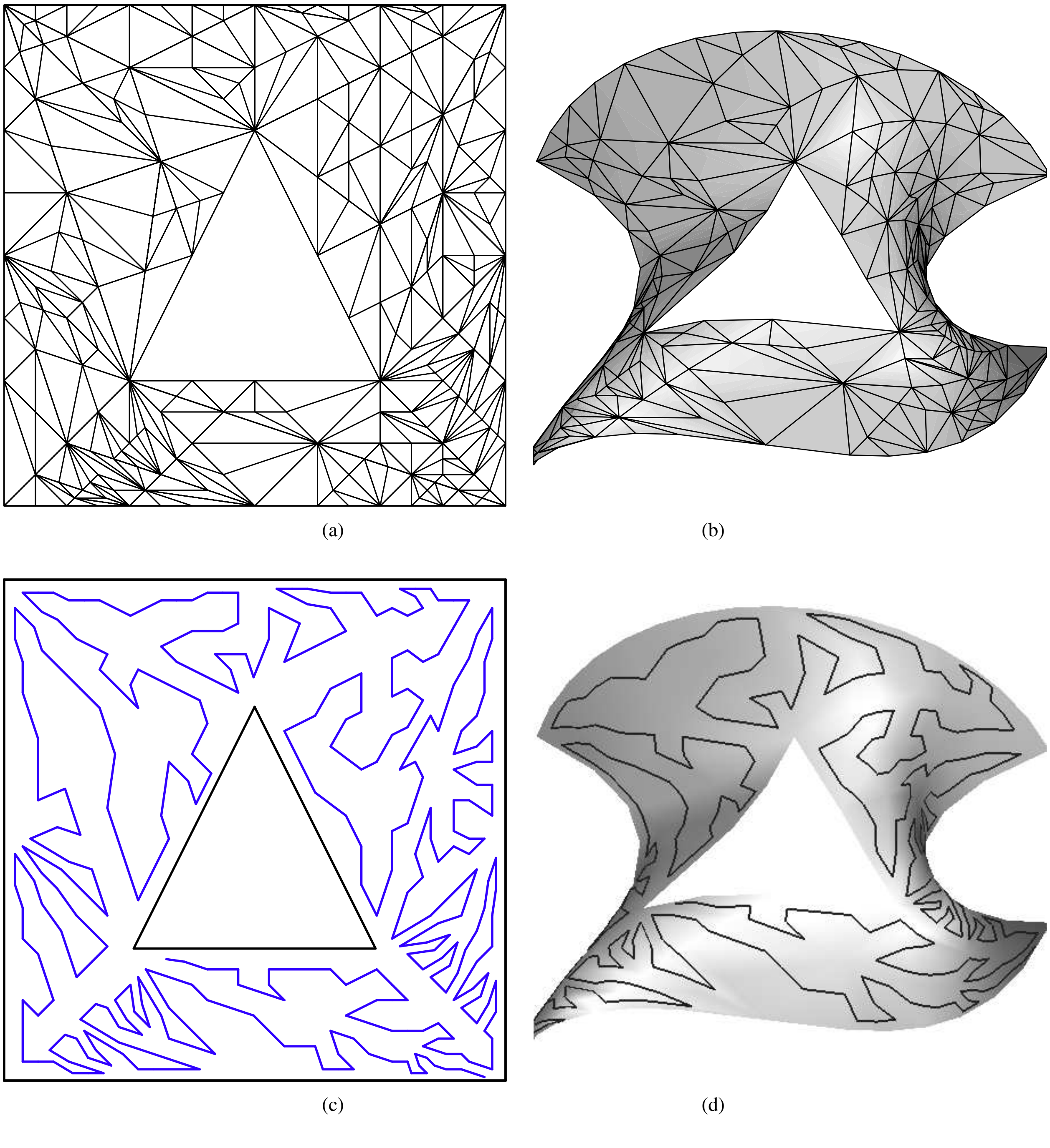


Figure 21: Bezier Surface