Hamiltonian Cycles in Triangle Graphs

U. Doğrusöz[1] and M. S. Krishnamoorthy[2]

[1] Tom Sawyer Software,
804 Hearst Avenue, Berkeley, CA 94710,
ugur@tomsawyer.com
[2] Department of Computer Science,
Rensselaer Polytechnic Institute, Troy, NY 12180,
moorthy@cs.rpi.edu

Abstract. We prove that the Hamiltonian cycle problem is NP-complete for a class of planar graphs named *triangle graphs* that are closely related to inner-triangulated graphs. We present a linear time heuristic algorithm that finds a solution at most one-third longer than the optimum solution and use it to obtain a fast rendering algorithm on triangular meshes in computer graphics.
Keywords: *Planar graph, inner-triangulated graph, triangle graph, NP-complete, heuristic algorithm, rendering triangular meshes.*

1 Introduction.

The Hamiltonian cycle problem (HC) is that of determining whether or not a given graph contains a cycle that passes through every vertex exactly once and has numerous applications in different areas [4, 12, 13, 14]. HC is NP-complete for various classes of graphs including cubic, 3-connected planar graphs [10] and maximal planar graphs (and therefore for inner-triangulations) [8] but a polynomial time algorithm was presented for 4-connected planar graphs by Chiba and Nishizeki [6].

The problem of finding a Hamiltonian cycle in an inner-triangulation has applications in computer vision in determining the shape of an object represented by a cluster of points [12]. Cimikowski identified a Hamiltonian subclass of inner-triangulations, namely *simply nested inner-triangulations*, and presented a linear-time algorithm for finding a Hamiltonian cycle in such graphs [9]. A three-colorable class of graphs called *triangle graphs*, closely related to inner-triangulated graphs was defined and their application to the parallel finite element solution of elliptic partial differential equations on triangulated domains was discussed in [3].

The efficiency of high-performance rendering machines (such as *OpenGL* [1] and *IGL* [5]) on triangular meshes in computer graphics is proportional to the rate at which triangulation data is sent to it. Ordering the triangles so that consecutive triangles share a face reduces the data rate by a factor of three since now, only one vertex per triangle need be specified. Such an ordering exists if and only if the dual of the given triangulation is Hamiltonian (i.e., contains a Hamiltonian path [2]). Also, note that in order to completely specify the topology of the triangulation, the insertion order of the new vertices must be specified (see Figure 1.a). *OpenGL* asks the user to issue a *swaptmesh* call whenever the insertion order deviates from alternating left-right turns. *IGL*, on the other hand, expects triangulations as "vertex-strips" and a vertex must be sent twice to get two consecutive left or right turns. Hamiltonian triangulations can be considered as triangulations satisfying a certain "Gray-code" property (consecutive triangles share a face).

In this paper, we prove that finding a Hamiltonian cycle of a triangle graph is in the class of NP-complete problems and give a heuristic algorithm to find a solution that is at most one-third longer than the optimum. This heuristic is shown to result in a fast rendering algorithm on triangular meshes in computer graphics.

2 Preliminary Definitions.

An `inner-triangulation` or an `inner-triangulated graph` is a maximal planar graph which has an embedding where each interior face or region of the graph is a triangle. We will call such an embedding a `triangular grid`.

`Merging` two vertices u and v, consists of collapsing them to form a new vertex at the same position as u which is adjacent to all vertices w that were adjacent to either u or v.

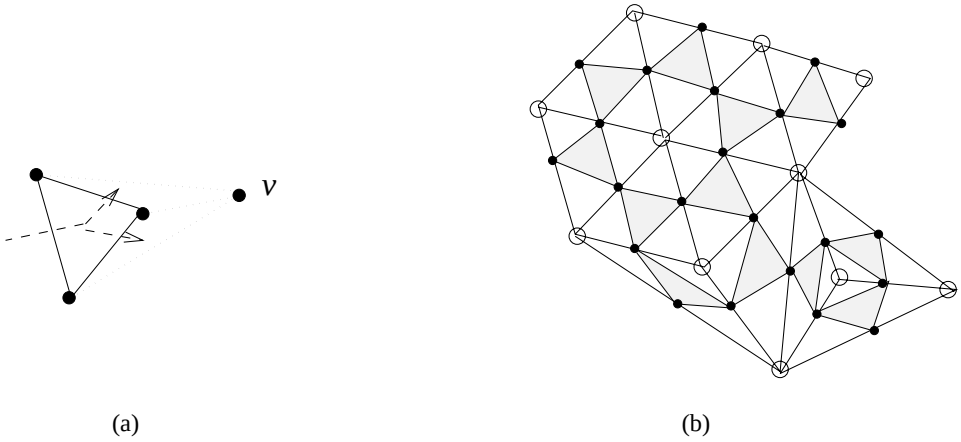


Fig. 1. **(a)** Next vertex v might specify one of the two triangles shown depending on the turn we take. **(b)** An inner-triangulation (whose vertices are represented with empty circles) and the corresponding triangle graph (whose vertices are represented with filled circles).

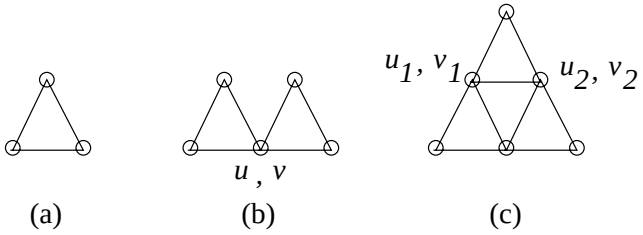


Fig. 2. An element of a triangle graph **(a)** and the construction of other triangle graphs by merging one **(b)** and two **(c)** degree-two node(s).

Definitions 1 through 4 define a class of embedded graphs named *triangle graphs*. We refer the reader to [3] for more information on triangle graphs.

Definition 1 *An* `element` *is a complete graph of three vertices with an embedding (see Figure 2.a).*

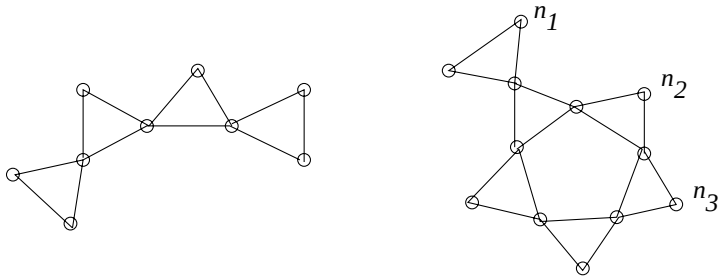
Definition 2 *A* `component` *is either a single element or a chain of elements where two and only two elements called end elements have two degree-two vertices. The remaining elements have two degree-four and one degree-two vertices (see Figure 3.a). The smallest component is a single element.*

Definition 3 *A* `triangle graph` *is an embedded planar graph G constructed according to the following rules:*

(i) A component (basic building block) is a triangle graph.

(ii) If G is a triangle graph and H is a component, and u and v are degree-two vertices of G and H, respectively, then the graph obtained by merging vertices u and v is a triangle graph, if and only if the resulting graph is planar (see Figure 2.b).

(iii) If G is a triangle graph and H is a component, and u_1 and u_2 are two neighboring (cf. Definition 4) degree-two vertices of G, and v_1 and v_2 are two neighboring degree-two vertices of H, then the graph obtained by merging vertices u_1 and v_1, and vertices u_2 and v_2 is a triangle graph if and only if the resulting graph is planar, no vertex of an interior face is of degree two, and every interior face contains at least three edges (see Figure 2.c).



(a) (b)

Fig. 3. **(a)** An example of a component of a triangle graph. **(b)** Neighboring nodes n_1 and n_2 and non-neighboring nodes n_1 and n_3 of a triangle graph.

Definition 4 *Since triangle graphs are embedded in a plane, the exterior face consists of all nodes in the exterior region. Two degree-two nodes n_1 and n_2 of a triangle graph are* `neighbors` *if and only if there is a path along the exterior face from n_1 to n_2 that does not contain any other degree-two nodes. For example, in the graph of Figure 3.b, nodes n_1 and n_2 are neighbors while n_1 and n_3 are not.*

Let $G = (V, E)$ be a planar graph. The `dual` of $G = (V, E)$ is a graph $G_d = (V_d, E_d)$ where every region in G is represented by a vertex in V_d and two vertices in G_d are adjacent if and only if the two corresponding regions share an edge in G.

The `edge adjacency graph` of a graph $G = (V, E)$ is a graph $G_2 = (V_2, E_2)$ where every edge in E is mapped onto a vertex in V_2 and two vertices in V_2 are adjacent if and only if their corresponding edges in E are adjacent.

Lemma 1 *Edge adjacency graph of a triangular grid is a triangle graph [3]. In addition, all vertices except possibly those on the outer-face are of degree four.*

Proof: We use induction on the number of triangular elements of the grid. A single triangular element maps to a single element of its triangle graph representation. Assuming edge adjacency graph of a triangular grid of $n-1$ elements corresponds to a triangle graph of $n-1$ elements, consider an arbitrary grid of n elements. Removing a boundary element, a triangular region, results in a grid of $n-1$ elements whose edge adjacency graph is a triangle graph by the induction hypothesis. Returning this element back results in one of the following two situations where:

- the triangular element adds two edges to the grid to extend the corresponding triangle graph according to rule (ii) of Definition 3, or
- the new element adds one edge to close the corresponding triangular grid with a single triangle element using rule (iii) of Definition 3.

Notice that in both cases, the vertices of the triangle graph corresponding to the edge(s) which are reused to extend or close the triangular grid will be of degree four. Only the vertices corresponding to the edges on the outer-face of the grid will be of degree two. □

Figure 1.b illustrates the relationship between triangular grids and triangle graphs with an example.

A graph is said to be `triply-connected` (or `tri-connected`) if and only if removal of no pair of vertices leaves the graph disconnected.

A graph G is called a `cubic, 3-connected planar graph` if it is planar, its every vertex is of degree three, and it is triply-connected.

Let 3PHC denote the problem of determining whether or not a cubic, 3-connected planar graph contains a Hamiltonian cycle. Also let THC represent the Hamiltonian cycle problem for triangle graphs.

3 THC is NP-complete.

Theorem 1 *THC is NP-complete.*

Proof: THC is NP-complete because,

- THC is in NP. A verification algorithm for THC in nondeterministic polynomial time is straightforward and omitted.
- THC is NP-hard. We will next show this by transforming an NP-complete problem, namely 3PHC, to THC in polynomial time.

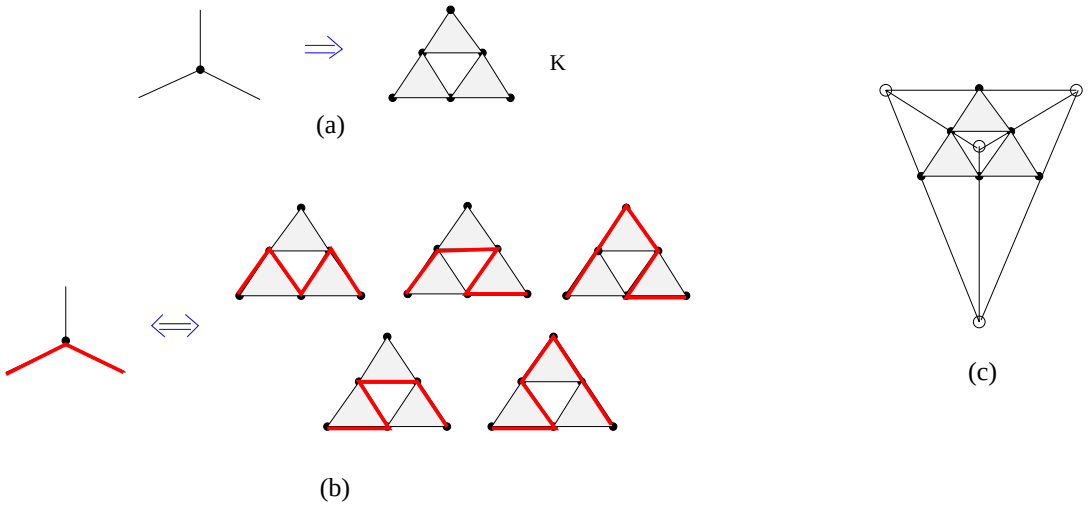


Fig. 4. **(a)** Basic building block K used for the transformation. **(b)** Possible local states of K. **(c)** Triangular grid corresponding to K.

The transformation is performed by replacing each vertex (along with its incident edges) of the graph G given for 3PHC with a graph K as shown in Figure 4.a to obtain G'. K might have five possible states in G' for each different way the corresponding vertex in G is visited (see Figure 4.b).

From the above transformation it is obvious that G' will have a Hamiltonian cycle if and only if the original graph G has a Hamiltonian-cycle. What remains to be shown is that G' is a triangle graph. Let G_d be the embedded dual of G. It is easy to see that G_d is an inner-triangulation. Furthermore, let G'_d be the embedded graph obtained when each face of G_d is partitioned into three triangle faces as shown in Figure 4.c. Notice that G'_d is also an inner-triangulation and the edge adjacency graph of G'_d corresponds to G'. Therefore, by Lemma 1, G' is a triangle graph. □

4 Approximation Algorithm.

Christofides has developed a polynomial time approximation algorithm with a worst-case bound of 3/2 for the traveling salesman problem, in which the distance between vertices satisfies the triangle inequality [7]. The Hamiltonian cycle problem is a special case of the traveling salesman problem where the distances are the shortest path lengths in a graph. Therefore, it can be used to find a spanning cycle of a planar graph that is smaller than 3/2 times the length of a shortest one (possibly a Hamiltonian cycle) in $O(n^3)$ time.

In addition, Nishizeki et al. have shown that a Hamiltonian cycle of a maximal planar graph that is at most of length $3(n-3)/2$ can be found in $O(n^2)$ time [11]. However, note that here we are interested in a Hamiltonian

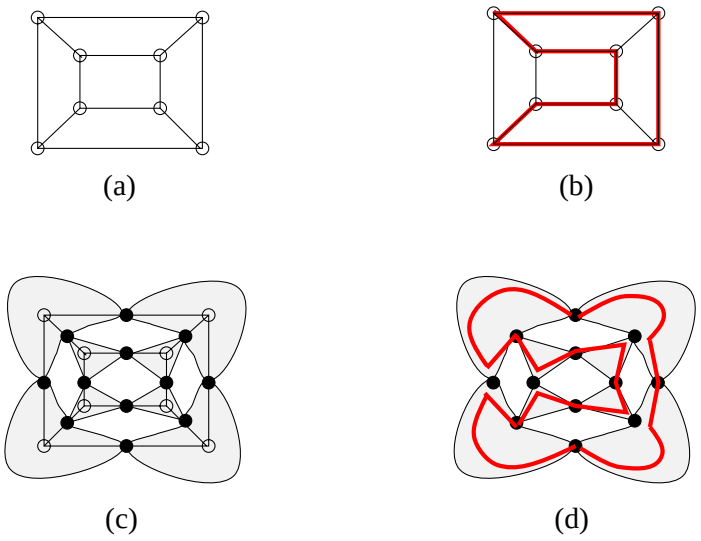


Fig. 5. An example of the transformation. Each filled region represents an instance of K. **(a)** A cubic, three-connected planar graph G. **(b)** A Hamiltonian cycle of G is shown with thick lines. **(c)** After the transformation. The outer vertices of instances of K in G' are shown with filled circles. **(d)** The Hamiltonian cycle in G' corresponding to the Hamiltonian cycle of G in (b) is shown with thick lines.

cycle of a given triangle graph or the *dual* of a given inner-triangulation as opposed to a Hamiltonian cycle of the inner-triangulation itself.

In this section, we present a linear time heuristic algorithm that finds a Hamiltonian cycle of a triangle graph whose length is at most 4/3 times that of the optimum. Then, we show how this algorithm can be used to render triangular meshes in an efficient manner.

Our heuristic algorithm is simple and based on *"peeling off"* the given triangle graph by removing the elements that share at least one common edge with the exterior region in this embedding, layer by layer. At each step, we combine the optimal solutions that we obtain from each layer to eventually form a long cycle that visits all the vertices as we will explain in a moment.

Let us first illustrate our idea with an example before we sketch a formal algorithm. Suppose we want to find a cycle of the triangle graph G (shown in Figure 6.a) that visits all the vertices of G once, or possibly twice. The elements of G sharing common edges with the exterior region are filled in to be easily distinguished. Let G_1 represent the subgraph that contains all such triangle elements of the first layer. The cycle marked with thick lines in Figure 6.a shows a cycle that visits all the vertices of G_1. Also, let C_1 be the cycle obtained for G_1.

Then, we remove all these exterior elements and find the biconnected components of the remaining triangle graph $G - G_1$ (see Figure 6.b). Suppose there is k of them and they are denoted by $G_{1,1}$ through $G_{1,k}$. The neighboring biconnected components are shown with blank and filled regions, respectively. The vertices that are marked with filled circles are the ones that connect these components to the previous layer.

After that, for each biconnected component $G_{1,i}$, $i = 1, ..., k$ independently, we recursively find a cycle $C_{1,i}$ that visits each vertex of $G_{1,i}$ once or twice. While doing so, we make sure that each such cycle starts from a randomly chosen marked vertex and comes back to it, without going through any of the other marked vertices in that biconnected component as shown in Figure 6.c.

The next step is to combine these cycles obtained for each biconnected component with the cycle obtained for the first layer as in Figure 6.d. During this process, we eliminate duplicate traversal of each vertex v shared by biconnected components $G_{1,i}$ and $G_{1,j}$ by letting vertex v get visited by only one of $G_{1,i}$ and $G_{1,j}$. Such vertices are shown with small circles near them in Figure 6.d. The final form of the cycle obtained is shown in Figure 6.e. The vertices visited twice are marked with small crosses near them. A pseudo-code for this algorithm follows:

algorithm *THC_Approx(G)*
// G is a triangle graph (an embedded graph as defined earlier).

if G is not biconnected **then**
for each biconnected component $G_{1,i}$ of G **do**
$C_{1,i} \leftarrow$ *THC_Approx*$(G_{1,i})$
return $(C_{1,1} \cup C_{1,2} ... \cup C_{1,k})$
else
$G_1 \leftarrow$ outer triangle elements of G
$C_1 \leftarrow$ cycle obtained from G_1 // Randomly choose one of the marked (in the previous
// step) vertices as the start and the end vertex for this cycle and do not visit
// other marked vertices.
if $(G - G_1)$ is empty **then**
return (C_1)
else
return $(C_1 \cup$ *THC_Approx* $(G - G_1))$

It is relatively straightforward to write a linear time implementation of our algorithm. Assume the elements or the vertices on the outer face of the embedding of the triangle graph G are stored in a list (this can be done by keeping track of such elements or vertices during the construction of the triangle graph or its corresponding triangular grid). Then finding C_1 as well as finding biconnected components of graph G can be performed by visiting each triangle element or vertex a constant number of times. Notice that an outer vertex is a cut-vertex of this graph if and only if it is repeated on the outer face. Furthermore, the elements or vertices encountered in between repetitions of cut-vertices belong to the same biconnected component. The algorithm needs space linear on the size of the graph as well.

Theorem 2 *The solution found by the heuristic algorithm above for any given triangle graph G is at most one-third longer than the exact solution (which is possibly a Hamiltonian cycle).*

Proof: Consider any outer triangle element T of G that contains only degree four vertices (please see Figure 7.a; notice that if there is no such triangle element, G must consist of a simple cycle of triangle elements, which is a trivial case). T could be connected to the rest of the triangle graph in two ways. In the first case, T is attached to a biconnected component of a single triangle element (as in Figure 7.b). The second case as shown in Figure 7.c is when T is connected to a biconnected component of three or more triangle elements. In either case, at least four vertices (shown marked with empty circles in Figure 7) are visited once and exactly one vertex (vertex v marked with a filled circle) twice during this step of our heuristic. However, since the vertices visited once are shared between neighboring outer triangle elements and might be shared with other biconnected components of the rest of the graph, there is at most one vertex visited twice for every *three* vertices $(4/2 + 1 = 3)$ for each step of our algorithm. This applies to the successive steps of our algorithm since the same process is repeated in a recursive fashion. Therefore, overall there is at most one vertex visited twice for every three vertices of G. That's why the cycle obtained is at most one-third longer than the optimum solution. □

There are different ways this algorithm could be modified to result in a possibly shorter cycle with an order of constant time slow-down. One example is to change the algorithm to identify biconnected components which can be connected to the previous layer using two neighboring vertices that belong to the neighboring triangle elements of the previous level (see Figure 8 for an example).

This heuristic yields a fast rendering algorithm if we simply feed the triangles of the triangular mesh to be rendered into the machine in the order the triangle elements of the given triangulation are visited by the heuristic.

Lemma 2 *The heuristic can be used to find an encoding sequence for any triangulation whose length is at most twice*[3] *that of optimal in $O(n)$ steps.*

[3] Note that being able to find a Hamiltonian cycle/path that visits all triangles would help us find a sequence that is at most one and a half times that of optimal.

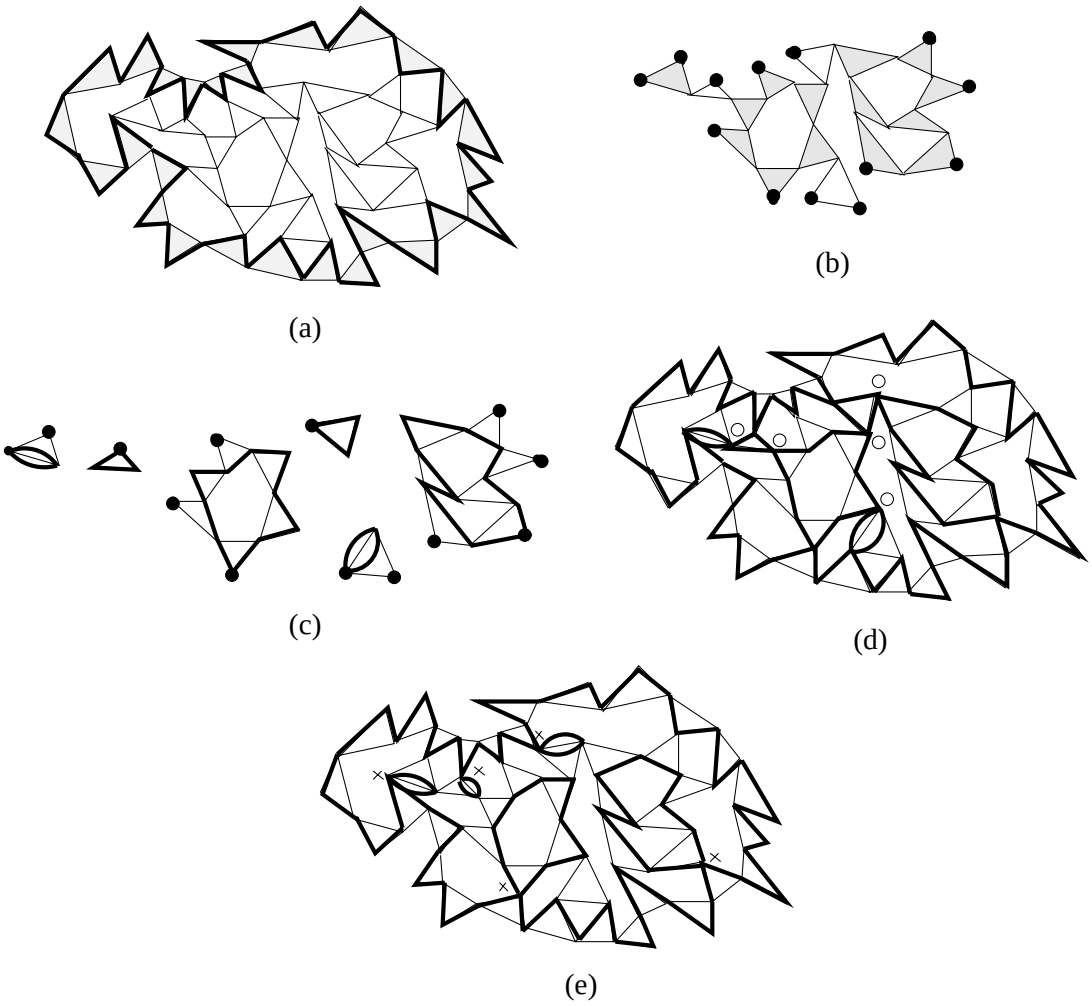


Fig. 6. (a) An embedding of a triangle graph G. Exterior triangle elements (G_1) are filled in and the cycle that visits all the vertices of these elements (first layer) is marked with thick lines. **(b)** $G - G_1$. The neighboring biconnected components are filled in differently and the vertices that connect them to the previous layer are marked. **(c)** We find separate cycles that visit all the vertices of these biconnected components recursively. These cycles (marked with thick lines) start and end at any one of the vertices marked in (b) but do not go through any of the remaining marked vertices. **(d)** The cycles obtained from the biconnected components of $G - G_1$ are combined with the cycle obtained from G_1. The vertices common to neighboring biconnected components (marked with circles near them) are visited twice. **(e)** The final cycle is acquired after the duplicate traversals are eliminated (for the vertices marked with circles in (d)). The vertices marked with small crosses near them are the only ones that are visited twice in the final cycle.

Proof: The heuristic finds a solution that is at most one-third longer than that of optimal and in the worst case, the turns are either all right or all left turns and half of these triangles in the sequence need be adjusted so that a left-right alternating sequence will be obtained. Therefore,

$$(\frac{4}{3} \cdot S_{opt}) \cdot \frac{3}{2} = 2 \cdot S_{opt}$$

where S_{opt} is an encoding sequence of minimum length for a given triangulation. □

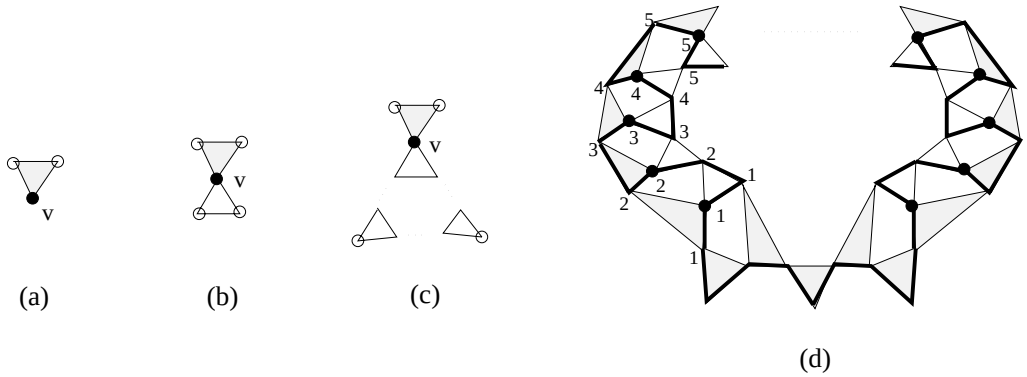


Fig. 7. **(a)** An outer triangle element, T. The vertex marked with a filled circle, v, is visited twice since it connects T to the next layer. The vertices marked with empty circles (the ones that are shared between two neighboring outer triangle elements), on the other hand, are visited only once by the heuristic algorithm. **(b)** First of the two possibilities for the biconnected component that is attached to vertex v shown in (a). The biconnected component consists of only one triangle element in this case. **(c)** The second of the two cases in which v connects the outer layer to the next one. The biconnected component consists of at least three triangle elements. **(d)** An example of a triangle graph for which our heuristic yields a solution that is about one-third longer than the optimum solution (shown with thick lines). Marked vertices are the ones our algorithm ends up in visiting twice. Notice why the number of these vertices is about one-third of the total number of vertices in the entire graph.

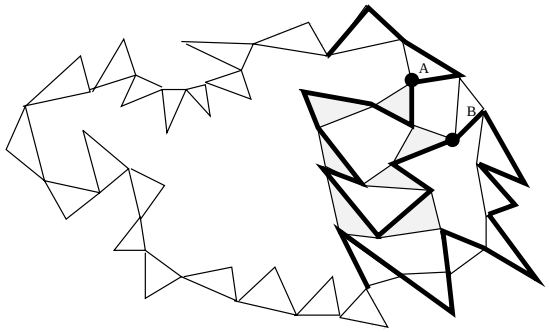


Fig. 8. One possible modification to improve our heuristic. The two marked vertices belong to two neighboring triangle elements A and B of the previous layer. Notice how combining this biconnected component to the previous layer in this fashion eliminates the duplicate traversal of one vertex.

5 Concluding Remarks.

We have proven that Hamiltonian cycle problem for a class of graphs closely related to inner-triangulations, namely triangle graphs, is NP-complete. Then, we presented a simple and efficient heuristic algorithm that produces a solution that is at most one-third longer than the optimum and can be improved at the cost of a slower performance. This heuristic can be directly applied to fast rendering on triangular meshes in computer graphics.

We also hope that this work will also bring insight to the problem of finding a larger (than simply nested inner-triangulations) Hamiltonian subclass of inner-triangulations, which has applications in some areas of computer vision.

References

1. *Silicon Graphics, Inc. Graphics Library Programming Guide.* 1991.
2. E. M. Arkin, M. Held, J. S. B. Mitchell, and S. S. Skiena. Hamiltonian triangulations for fast rendering. In J. van Leeuwen, editor, *Algorithms – ESA'94*, LNCS 855, pages 36–47, Utrecht, NL, September 1994.
3. M. Benantar, U. Dogrusoz, J. Flaherty, and M. Krishnamoorthy. Triangle graphs. *Applied Numerical Mathematics*, 17:85–96, 1995.
4. J. C. Bermond. *Hamiltonian graphs, in Selected Topics in Graph Theory ed. by L. W. Beinke and R. J. Wilson.* Academic Press, 1978.
5. R. Cassidy, E. Gregg, R. Reeves, and J. Turmelle. *IGL the Graphics Library for the i860.* 1991.
6. N. Chiba and T. Nishizeki. The Hamiltonian cycle problem is linear-time solvable for 4-connected planar graphs. *J. Algorithms*, 10:187–211, 1989.
7. N. Christofides. Worst case analysis of a new heuristic for the traveling salesman problem. Technical Report 388, Carnegie-Mellon University, Management Sciences Research Report, 1976.
8. V. Chvatal. Hamiltonian cycles. Technical report, School of Computer Science, McGill University, Montreal, Canada, 1981.
9. R. J. Cimikowski. Finding Hamiltonian cycles in certain planar graphs. *Information Processing Letters*, 35:249–254, 1990.
10. M. R. Garey, D. S. Johnson, and R. E. Tarjan. The planar Hamiltonian cycle problem is NP-complete. *SIAM J. Computing*, 5(4):704–714, 1976.
11. T. Nishizeki, T. Asano, and T. Watanabe. An approximation algorithm for the Hamiltonian walk problem on maximal planar graphs. *Discrete Applied Mathematics*, 5:211–222, 1983.
12. J. O'Callaghan. Computing the perceptual boundaries of dot patterns. *Comput. Graphics Image Process.*, 3:141–162, 1974.
13. F. Preperata and M. Shamos. *Computational Geometry: An Introduction.* Springer, New York, 1985.
14. G. Toussaint. Pattern recognition and geometrical complexity. *Proc. Fifth International Conference on Pattern Recognition*, pages 1324–1347, 1980, Miami Beach.

This article was processed using the LaTeX macro package with LLNCS style