# Efficient Generation of Triangle Strips from Triangulated Meshes

Oliver Matias van Kaick Murilo Vicente Gonçalves da Silva Hélio Pedrini
Department of Computer Science
Federal University of Paraná
81531-990 Curitiba-PR, Brazil
{oliver, murilo, helio}@inf.ufpr.br

## ABSTRACT

This paper presents a fast algorithm for generating triangle strips from triangulated meshes, providing a compact representation suitable for transmission and rendering of the models. A data structure that allows efficient triangle strip generation is also described. The method is based on simple heuristics, significantly reducing the number of vertices used to describe the triangulated models. We demonstrate the effectiveness and speed of our method comparing it against the best available program.

## Keywords

Triangle strips, mesh representation, rendering

## 1. INTRODUCTION

A crucial task in several scientific applications is the development of methods for storing, manipulating, and rendering large volumes of data efficiently. Unless compression methods or data reduction are used, massive data sets cannot be analyzed or visualized in real time.

Polygonal surfaces are probably the most widely used representations for geometric models, since they are flexible and supported by the majority of modeling and rendering packages. A polygonal surface is a piecewise-linear surface defined by a set of polygons, typically a set of triangles.

A common encoding scheme is based on *triangle strips*, which enumerates the mesh elements in a sequence of adjacent triangles to avoid repeating the vertex coordinates of shared edges. Triangle strips are supported by several graphics libraries, including IGL [Cassi91], PHIGS [ISO89], Inventor [Werne94], and OpenGL [Neide93].



***Journal of WSCG, Vol.12, No.1-3., ISSN 1213-6972***
*WSCG'2004, February 2-6, 2004, Plzen, Czech Republic.*


The set of triangles shown in Figure 1(a) can be described using the vertex sequence $(1, 2, 3, 4, 5, 6, 7)$, where the triangle $t_i$ is described by the vertices $v_i$, $v_{i+1}$, and $v_{i+2}$ in this sequence. Such triangle strip is referred to as a *sequential triangle strip* [1], in which the shared edges follow alternating left and right turns. A sequential triangle strip allows rendering of $t$ triangles with only $t + 2$ vertices instead of $3t$ vertices. This improves rendering, since its bottleneck is the vertex sending [Chow97].

A more general form of strips is given by *generalized triangle strips* [2] (for simplicity, triangle strips), where we do not have an alternating left/right turn, but each new vertex may correspond either to a left turn or to a right turn in the pattern (Figure 1(b)). To represent such triangle sequence with generalized triangle strips, the two vertices of the previous triangle can be swapped, and the sequence of vertices would be $(1, 2, 3, 4, 5, swap, 6, 7)$. This scheme is used in IGL. A swap can also be seen as the repetition of a vertex when two successive turns have the same orientation, as used in OpenGL. Thus, the triangle sequence in Figure 1(b) can be represented as $(1, 2, 3, 4, 5, 4, 6, 7)$. Note that the zero-area triangle $\Delta_{4,5,4}$ is simulating the swap.

A crucial problem is to obtain the minimal partition of a mesh into triangle strips. It is equivalent to the Hamiltonian path problem in the dual graph of the triangulation (that is NP-hard).

[1] It is called *pure sequential tristrip* in [Estko02].
[2] It is called *sequential tristrip* or *sequential tristrip with swap* in [Estko02].

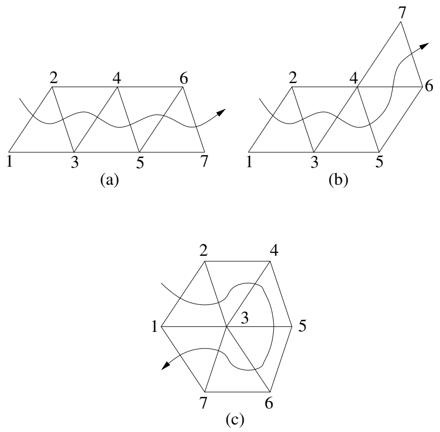



**Figure 1. Triangle strips.**

The complexity of the related problem of computing the minimal partition of the mesh into sequential triangle strips has been recently shown in [Estko02].

Nevertheless, when vertex resending is used to simulate swap, the hardness of minimizing the number of vertices is still an open problem, since neither the minimal triangle strip partition nor the minimal sequential triangle strip partition is always the best encoding.

In Figure 1(b), the minimum number of sequential triangle strips that cover the mesh is 2, then the number of vertices required is 9. However, only 8 vertices are necessary when using one triangle strip with vertex resending. In Figure 1(c) is shown an example where the minimum number of strips is 1, in this case using 11 vertices. But if the strips $(1, 2, 3, 4, 5)$ and $(5, 6, 3, 7, 1)$ were used then 10 vertices would be sufficient to describe the mesh. Although it has not been proved here, we conjecture that the problem of minimizing vertices is also intractable.

In this paper, we present a heuristic method for constructing triangle strips from triangulated models, which is an extension of the work presented in [Silva02]. The main improvements on the new approach include an efficient data structure used to reduce the running time and a new strategy for generating the triangle strips. Instead of using simultaneous strip construction, just a single triangle strip is created at each time.

We have compared our method with `FTSG` [Xiang99], the best known stripification program, and the experimental results show that our method requires always less running time and generates better results in one of the two metrics used to compare the programs.

In Section 2, we summarize some relevant previous work on triangle strips. Section 3 presents our method for generating triangle strips, emphasizing the used data structure. In Section 4, the proposed method is applied to several data sets. Experimental results are presented and discussed. Finally, Section 5 concludes with some final remarks.

## 2. RELATED WORK

Several methods for compressing triangular meshes have been proposed in literature. Akeley *et al.* [Akele90] developed a program that creates triangle strips for a given triangulated model, trying to minimize the number of single triangle strips.

Speckmann and Snoeyink [Speck97] computed the triangle strips for triangulated irregular networks by creating a spanning tree of the dual graph, and then extracting the strips from a depth-first traversal of the tree.

Taubin *et al.* [Taubi98] also used strips to efficiently compress polygonal meshes. A method for generating and maintaining triangle strips in continuous level-of-detail is presented in [Stewa01]. Chow [Chow97] describes an efficient method for decomposing geometric models into generalized triangle meshes.

In [ElSa99], a data structure called skip strip is used to generate the triangles strips. The method also maintains a triangle-stripped progressive mesh during the refinement and coarsening process, such that strips are preserved. A method for building hierarchical generalized triangle strips is described in [Velho99].

Deering [Deeri95] introduced the concept of geometry compression based on generalized triangle meshes. The algorithm uses lossy compression for the quantization of coordinate values, and a vertex cache takes advantage of spatial coherence, decreasing vertex transfers from the CPU to the graphics pipeline.

Bar-Yehuda and Gotsman [BarY96] showed that a cache of size $O(\sqrt{n})$ is necessary to minimize vertex transmission in a mesh of size $n$. Hoppe [Hoppe99] described heuristics to construct triangle strips that are optimized for a given cache size.

Isenburg [Isenb00] describes a scheme for encoding the connectivity and the stripification of a triangle mesh, exploiting the correlation between these two information.

Evans *et al.* [Evans96] developed a program called `STRIPE`, which is based on a greedy algorithm, to generate triangle strips from polygonal models. Our method also uses a greedy heuristic, however, includes some significant differences. Whenever a new strip is created, the initial triangle is chosen as that one having fewer adjacent triangles (lower degree) in the mesh. Furthermore, when swap minimization is required, our algorithm combines a sequential triangle strip construction and the strategy of the triangle with lower degree.

A recent program, called `FTSG`, to create triangle strips based on the construction of a spanning tree in the dual graph of the triangulation is presented in [Xiang99]. We compare our method to this one, which is the best known publicly available program.

## 3. PROPOSED METHOD

The proposed method seeks to minimize the number of vertices to be sent to the graphics pipeline. Two heuristics were considerated. The first aims to minimize the number of strips, generating output to a hardware and a graphics library that support swap without resending a vertex. The second heuristic minimizes the number of vertices for models that simulate swap resending a vertex. In the first approach, less strips mean less vertices, while in the second approach there is a tradeoff between few strips and few swaps.

### 3.1. Algorithms

The algorithm for choosing the next triangle to be inserted in a strip is similar to other greedy algorithms [Akele90, Evans96]. The proposed algorithm analyzes the dual graph of the mesh, taking priority for inserting triangles which have more adjacent triangles in strips. In case of tie, our algorithm uses different look-ahead strategies, depending on the heuristic under consideration.

A triangle is referred to as *free* if it does not belong to any strip, and the *degree* of a triangle is the number of free triangles that are adjacent to it. It is worth mentioning that the degree of the triangle can change at each step of the algorithm.

The first heuristic is based on a greedy algorithm with one level of look-ahead in case of tie. In order to improve efficiency, the look-ahead was implemented iteratively, and triangles can be inserted immediately in some cases, without finishing the look-ahead search. These cases are: a triangle with degree 0 is found; if there are no triangles with degree 0 and a triangle with degree 1 with a free adjacent with degree 1 is found. Note that in the two cases the algorithm still behaves like an ordinary greedy algorithm, such that they were implemented only for efficiency purpose.

The second heuristic is a combination of the greedy algorithm described above and an algorithm that tries to construct sequential triangle strips. The choice for the next triangle is performed as follows: it is used the greedy algorithm and, in case of tie, the triangle that does not generate swap is chosen. Note that the two cases of immediate insertion described previously affect the result not only in terms of performance. The two cases avoid strips with one and two triangles, respectively.

The reason for choosing the next triangle not only avoiding swap is motivated by the fact that a new strip pays two vertices of penalty and a swap pays only one vertex. Of course, the strips should not have many swaps, but sometimes when reducing the number of strips, even having swaps, the number of vertices is decreased.

In both heuristics, when it is necessary to start the construction of a new strip, the triangle with the lowest degree in the mesh is chosen. This simple consideration has a great impact in the results, motivating the construction of an efficient data structure.

Before describing the data structure, it will be presented the adopted approach to constructing it from a list of vertices and triangles.

### 3.2. Dual Graph

It was used a modified *Triangle* [Shewc96] data structure to represent the dual graph of the triangulation, which is implicitly given by the adjacency list of each triangle. This list is updated by searching for adjacent triangles only if they have at least one vertex in common, which is described as follows.

First it is created a main vertex list with one entry for each vertex. This entry will be a second list that holds the information of which triangles are linked to this vertex (see Figure 2). Each entry of this second list is an item that keeps a pointer to a triangle associated with this vertex and the index to the other vertex that forms an edge in the triangle. The index in the main vertex list and the index in the second list item implicitly form the information of a triangle edge. For example, in Figure 2, the index $a$ in the main list and the index $b$ in the list item form the edge $ab$. Only three of such items are inserted, which represent the three edges of the triangle, ordered by the vertex indices.

For each triangle, it is checked if there exists a reference for any of its edges. If so, one adjacency of the triangle is updated with the pointer to the triangle of the item, and one adjacency of the pointed triangle is also updated with the current triangle. If there is no such edge, then the edge items of the current triangle are inserted in the vertex list. In this way, the adjacencies will be updated. At the end of the triangle loop, only items of the boundary edges will remain in the main vertex list, if there is any.

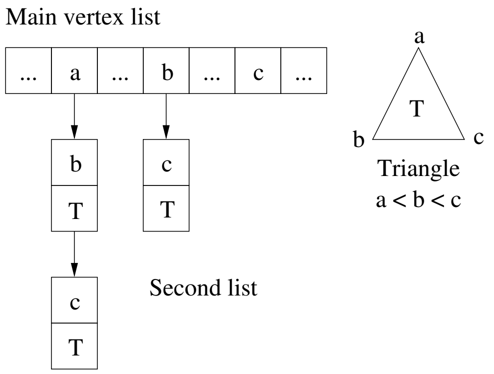



**Figure 2. Dual graph construction.**

### 3.3. Data Structure

In conjunction with our simple heuristic, another reason for the algorithm efficiency is the design of an efficient data structure (Figure 3), that allows direct triangle indexing in an *array of triangles* and allows triangle access through a list of triangle pointers sorted by degree. Furthermore, the data structure was conceived in such a way that the reordering of this list can be done in constant time.

Besides the pointers to the adjacent triangles and the triangle vertices, each triangle of the dual graph holds its degree, a reference to the strip (if there is) in that it is inserted, and a pointer to the node of the list that points to the triangle under consideration. This information is directly accessed from the array of triangles during the heuristic execution. Whenever is necessary to start the construction of a new strip, a search for the minimum degree triangle in the mesh is desirable. For this, it is used a list of indices to the triangles.

This list is sorted by the degree of the triangles pointed by its nodes. Since that the possible degrees range from 0 to 3, the list has a pointer for each first node with a given degree. By doing that, the list reordering (that occurs at each triangle insertion step) is trivial. It is only necessary to remove the node, for instance with degree $k > 0$, and reinsert it in the list using the pointer to the first node with degree $k - 1$.

In the example shown in Figure 3, the nodes $T_1$ and $T_2$ of the sorted list point to triangles with degree 0. The node $T_3$ and $T_4$ point to a triangle with degree 2 and the nodes $T_5 ... T_m$ point to triangles with degree 3. Since there are no triangles with degree 1, $deg1$ points to null. If the degree of the triangle $t_{k+1}$ changes from 3 to 2, the node $T_6$ will be removed and reinserted between $T_2$ and $T_3$ and will be pointed by $deg2$. On the other hand, if the degree of the triangle $t_{k+j}$ changes from 2 to 1, $T_3$ will be removed and reinserted in the same position and will be pointed by $deg1$ while $deg2$ will point to $T_4$.

## 4. RESULTS

Our algorithm for generating triangle strips has been tested on a number of data sets in order to illustrate its performance. The experiments have been performed on a PC Pentium III 866 MHZ with 1 Gbyte RAM, running Linux operating system.

We compared our program (`Strip`) against `FTSG` [Xiang99], which is the best known publicly available program. Our algorithm generated lower number of strips in less running time, however, it generated higher number of vertices in the OpenGL model. This implies in less rendering time for swap-based models.

Table 1 shows the number of vertices and triangles for eleven data models used in our tests. It also reports the execution times required to generate the results (time for stripification, and total time for construction plus stripification).

The results of comparison between our method and `FTSG` are summarized in Table 2, which shows the total number of vertices and number of strips required to represent the models using two different heuristics, one that seeks to minimize the number of strips and other that seeks to minimize the number of vertices when vertex resending is used as swap. Figure 4 presents the results for three different data sets.

## 5. CONCLUSIONS AND FUTURE WORK

This paper presented an efficient method for generating triangle strips from triangulated models. The method is fast and significantly reduces the number of vertices used to describe a given triangulation, allowing lower memory bandwidth for real-time visualization of complex data sets.

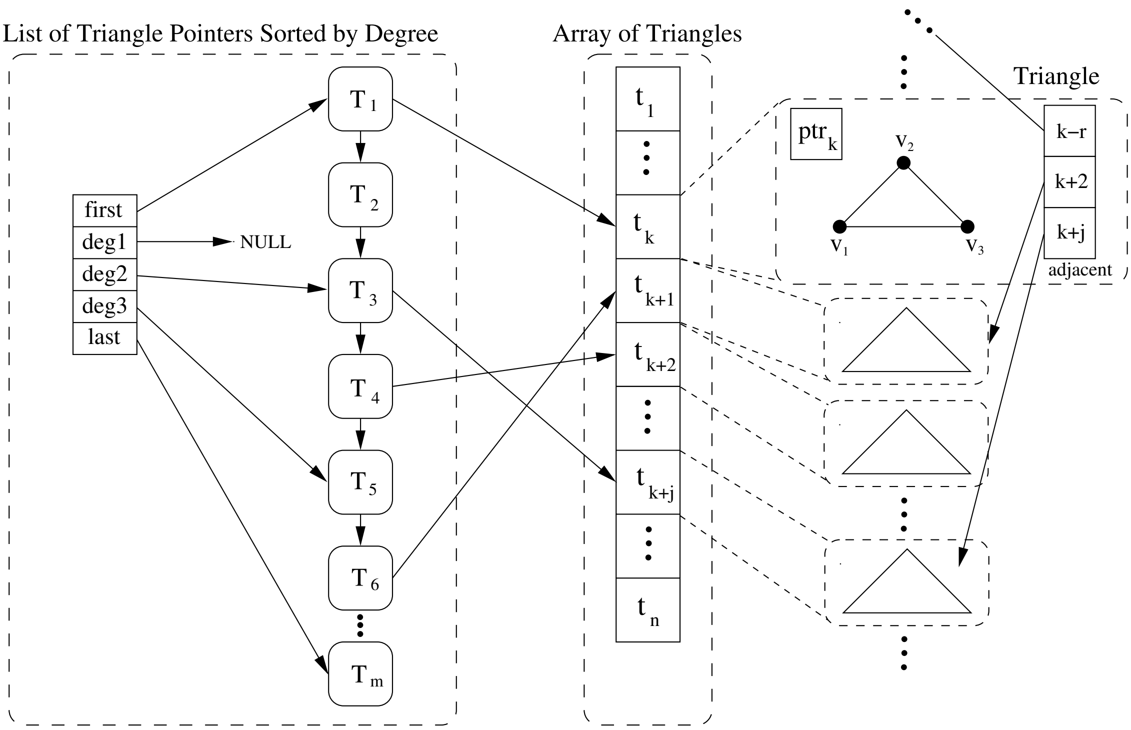



**Figure 3. Data structure: in the sorted list the pointers $deg1$, $deg2$ and $deg3$ are the pointers to the first node with degree one, two and three, respectively. The adjacent triangles to triangle $t_k$ are the triangles $t_{k-r}$, $t_{k+2}$ and $t_{k+j}$. The pointer $ptr_k$ in the triangle $t_k$ points to $T_1$. The triangle $t_k$ has degree 0, $t_{k+2}$ and $t_{k+j}$ degree 2, and $t_{k+1}$ degree 3. In this example, no triangle has degree 1. Note: only some pointers are indicated for readability purpose.**

| Model | Vertices | Triangles | Strips time | | Total time | |
|---|---|---|---|---|---|---|
| | | | FTSG | Strip | FTSG | Strip |
| buddha | 543652 | 1087716 | 3.63 | 1.73 | 6.78 | 5.06 |
| bunny | 35947 | 69451 | 0.26 | 0.11 | 0.49 | 0.29 |
| canyon | 47088 | 93980 | 0.35 | 0.16 | 0.64 | 0.44 |
| champlain | 100000 | 198996 | 0.74 | 0.33 | 1.36 | 0.98 |
| crater | 107903 | 214808 | 0.77 | 0.36 | 1.44 | 1.05 |
| dragon | 437645 | 871414 | 2.92 | 1.39 | 5.47 | 4.14 |
| emory | 36500 | 72712 | 0.27 | 0.12 | 0.50 | 0.34 |
| hand | 327323 | 654666 | 2.10 | 0.98 | 3.85 | 2.77 |
| mars | 8971 | 17820 | 0.06 | 0.03 | 0.12 | 0.07 |
| rice lake | 200000 | 399166 | 1.47 | 0.69 | 2.74 | 2.06 |
| roseburg | 40343 | 80423 | 0.28 | 0.14 | 0.53 | 0.37 |

**Table 1. Model characteristics and execution times in seconds.**

| Model | Strips | | Vertices | |
|---|---|---|---|---|
| | FTSG | Strip | FTSG | Strip |
| buddha | 25576 | 19640 | 1398464 | 1421420 |
| bunny | 618 | 563 | 81412 | 81908 |
| canyon | 2297 | 1738 | 120884 | 123152 |
| champlain | 4357 | 3339 | 255236 | 260369 |
| crater | 4563 | 3468 | 278565 | 283208 |
| dragon | 20571 | 15943 | 1121151 | 1140173 |
| emory peak | 1744 | 1325 | 93403 | 95060 |
| hand | 10394 | 8493 | 806855 | 816202 |
| mars | 462 | 369 | 23010 | 23383 |
| rice lake | 9668 | 7322 | 514734 | 523862 |
| roseburg | 1802 | 1400 | 102920 | 105108 |

**Table 2. Comparison of triangle strip algorithms.**

Future work includes an investigation of the impact of the buffer size on transmission cost, when hardware has additional buffer space, beyond the usual storage for two vertices.

## 6. ACKNOWLEDGEMENTS

The authors would like to thank the Stanford 3D Scanning Repository, United States Geological Survey, Georgia Institute of Technology, and Nasa's Planetary Data System for the models.

## 7. REFERENCES

[Akele90] K. Akeley, P. Haeberli, and D. Burns. `tomesh.c`: Program on SGI Developer's Toolbox CD, 1990.

[BarY96] R. BarYehuda and C. Gotsman. Time/space trade-offs for polygon mesh rendering. *ACM Transactions on Graphics*, 15(2):141–152, April 1996.

[Cassi91] R. Cassidy, E. Gregg, R. Reeves, and J. Turmelle. *IGL: The Graphics Library for the i860*, 1991.

[Chow97] M.M. Chow. Optimized geometry compression for real-time rendering. In *Proceedings of IEEE Visualization'97*, pages 347–354, 1997.

[Deeri95] M. Deering. Geometry compression. In *SIGGRAPH'95 Conference Proceedings*, Annual Conference Series, pages 13–20, Los Angeles, California, USA, 1995.

[ElSa99] J. ElSana, E. Azanli, and A. Varshney. Skip strips: Maintaining triangle strips for view-dependent rendering. In *Proceedings of IEEE Visualization*, pages 131–138, San Francisco, California, United States, 1999.

[Estko02] R. Estkowski, J.S.B. Mitchell, and X. Xiang. Optimal decomposition of polygonal models into triangle strips. In *Proceedings of the 18th annual symposium on Computational geometry*, pages 254–263, Barcelona, Spain, 2002. ACM Press.

[Evans96] F. Evans, S. Skiena, and A. Varshney. Optimizing triangle strips for fast rendering. In *Proceedings of IEEE Visualization'96*, pages 319–326, 1996.

[Hoppe99] H. Hoppe. Optimization of mesh locality for transparent vertex caching. In *Proceedings of the 26th annual conference on Computer Graphics and Interactive Techniques*, pages 269–276. ACM Press/Addison-Wesley Publishing Company, 1999.

[Isenb00] Martin Isenburg. Triangle strip compression. In *Graphics Interface*, pages 197–204, May 2000.

[ISO89] ISO. Information Processing Systems - Computer Graphics - Programmer's Hierarchical Interactive Graphics System (PHIGS). Technical Report ISO/IEC 9592, International Organization of Standardization, 1989.

[Neide93] J. Neider, T. Davis, and M. Woo. *OpenGL Programming Guide: The Official Guide to Learning OpenGL*. Addison-Wesley, New Jersey, 1993.

[Shewc96] J.R. Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Lecture Notes in Computer Science*, volume 1148, pages 203–222. Springer-Verlag, May 1996.

[Silva02] M.V.G. da Silva, O.M. van Kaick, and H. Pedrini. Fast mesh rendering through efficient triangle strip generation. *Journal of WSCG*, 10(1):127–134, February 2002.

[Speck97] B. Speckmann and J. Snoeyink. Easy triangle for TIN terrain models. In *Canadian Conference on Computational Geometry*, pages 239–244, 1997.

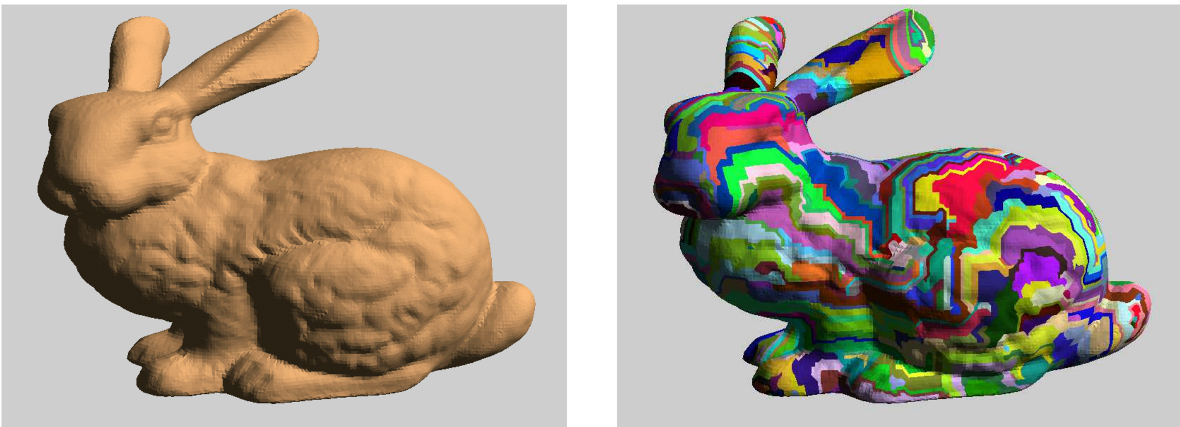
[Stewa01] A. J. Stewart. Tunneling for triangle strips in continuous level-of-detail meshes. In *Graphics Interface*, pages 91–100, June 2001.

[Taubi98] G. Taubin and J. Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17(2):84–115, 1998.

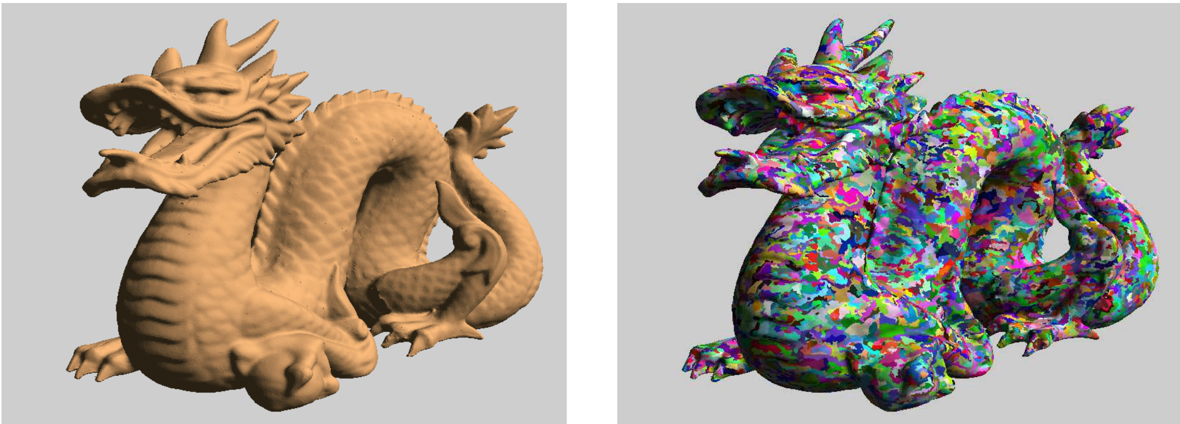
[Velho99] L. Velho, L. H. de Figueiredo, and J. Gomes. Hierarchical generalized triangle strips. *The Visual Computer*, 15(1):21–35, 1999.

[Werne94] J. Wernecke. *The Inventor Mentor*. Addison-Wesley, 1994.

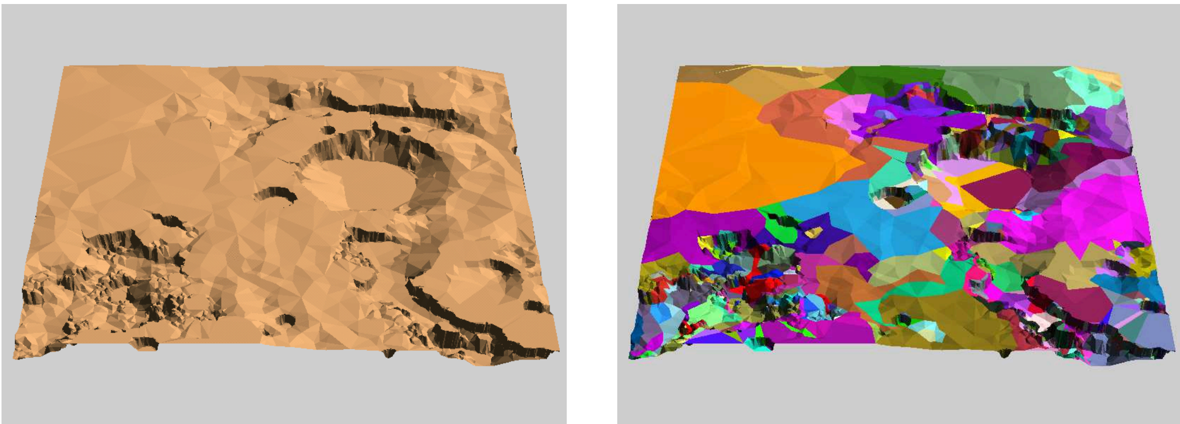
[Xiang99] X. Xiang, M. Held, and J.S.B. Mitchell. Fast and effective stripification of polygonal surface models. In *Proceedings of ACM Symposium on Interactive 3D Graphics*, 1999. http://www.ams.sunysb.edu/∼xxiang/strip.html.



(a) Bunny



(b) Dragon



(c) Mars terrain

**Figure 4. Results for three data sets. Each colored area is covered by one strip.**