FAST MESH RENDERING THROUGH EFFICIENT TRIANGLE STRIP GENERATION

Murilo Vicente Gonçalves da Silva

Oliver Matias van Kaick

Hélio Pedrini

Department of Computer Science
Federal University of Paraná
81531-990 Curitiba-PR, Brazil
{murilo, oliver}@pet.inf.ufpr.br helio@inf.ufpr.br

ABSTRACT

The development of methods for storing, manipulating, and rendering large volumes of data efficiently is a crucial task in several scientific applications, such as medical image analysis, remote sensing, computer vision, and computer-aided design. Unless data reduction or compression methods are used, extremely large data sets cannot be analyzed or visualized in real time. Polygonal surfaces, typically defined by a set of triangles, are one of the most widely used representations for geometric models. This paper presents an efficient algorithm for compressing triangulated models through the construction of triangle strips. Experimental results show that these strips are significantly better than those generated by the leading triangle strip algorithms.

Keywords: triangle strips, geometry compression, rendering, mesh representation

1 INTRODUCTION

Polygonal surfaces are probably the most used representation in several scientific applications, since they are flexible and supported by the majority of modeling and rendering packages. Hardware support for polygon rendering is also becoming more available. A polygonal surface is a piecewise-linear surface defined by a set of polygons, typically a set of triangles.

Due to demand for larger and more detailed geometric datasets, a fundamental problem is to construct a compact encoding of triangular meshes in order to be able to store, transmit, and render them efficiently.

A common encoding scheme uses *triangle strips*, which exploits spatial coherence of the simplicial complex structure, enumerating 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].

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*, 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, resulting in significant saving for memory storage and transmission bandwidth.

A more general form of strips is given by *generalized 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. This can also be seen as the repetition of a vertex when two successive turns have the same orientation. Thus, the triangle sequence in Figure 1(b) can be represented as $(1, 2, 3, 4, 5, 4, 6, 7)$.

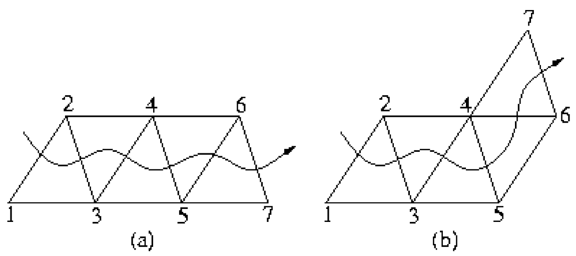


Figure 1: Triangle strips.

A crucial problem is to obtain the optimal partition of a mesh into triangle sequences $S_1, ..., S_n$, that is, a partition that minimizes *t*. Ideally, a single triangle strip covering the mesh completely should be obtained, however, this is not possible in general because the dual graph is not always Hamiltonian[1]. It has been proved that the problem of converting a given triangle mesh into the minimal set of triangle strips covering the mesh is NP-complete [Evans96a].

In this paper, we present an efficient algorithm for constructing triangle strips from triangulated models. Experimental results show that our method is significantly better than other existing approaches [Akele90, Evans96b].

In Section 2, we summarize some relevant previous work on triangle strips. Section 3 presents our method for generating triangle sequences using a local heuristic. In Section 4, the proposed method is applied to several data sets. Implementation issues and experimental results are presented and discussed. Finally, Section 5 concludes with final remarks and direction for future work.

2 RELATED WORK

Several methods for compressing triangular meshes have been proposed in literature. Such methods usually address two different tasks, the compression of numerical information associated with each vertex (such as position, elevation, texture, normal vectors) and the compression of information describing the connectivity between the surface components. These approaches are generally referred to as geometry compression and topology compression, respectively.

Geometric data associated with each vertex are usually reduced using lossy methods based on quantization, and lossless methods based on entropy encoding such as Huffman or arithmetic coding.

A simple way to represent connectivity is to use a triangle-vertex incidence table, which associates each triangle with its three bounding vertices. Since the number of triangles is approximately twice the number of vertices, the use of efficient techniques for compressing the triangle-vertex incidence table becomes an important issue. A representa-

[1] A path in a graph is called *Hamiltonian* when it visits all nodes in the graph exactly once.

tion storing each triangle as a list of 12-bit integer coordinates for each one of its three vertices would require 108 bits per triangle. Since the location of a vertex is repeated six times on average, it becomes expensive to store multiple representations of each vertex. An alternative is to store a table containing the vertex data in a sequence and a table containing three vertex references for each triangle. A vertex reference uniquely identifies the position of a vertex in the vertex data table. Since we need at most $\lceil \log_2 n \rceil$ bits per vertex reference in a triangulation with n vertices, this scheme requires a connectivity cost of $3 \log_2 n$ bits per triangle. Bar-Yehuda and Gotsman show that a buffer size of $12.72\sqrt{n}$ suffices to render any triangular mesh with n vertices, such that each vertex is transferred only once. Rossignac [Rossi99] estimates that this improvement leads to a connectivity cost of $1.25 \log_2 n + 9.75$ bits per triangle.

Progressive meshes, developed by Hoppe [Hoppe96, Hoppe98], provide a technique for transferring a mesh progressively, starting from a coarse approximation and then iteratively inserting a sequence of new vertices. A new vertex is created by expanding a vertex into an edge, which is the inverse of the edge collapse operation used in many mesh simplification techniques. Each vertex is transferred only once and 5 bits are used to identify two vertices among those adjacent, giving a total connectivity cost of approximately $(\lceil \log_2 n \rceil + 5)n$ bits.

An efficient method for compressing connectivity of 3D triangular meshes is presented by Rossignac [Rossi99]. This scheme, called *Edgebreaker*, produces results between 1.3 and 2 bits per triangle in simply connected manifold triangular meshes. It also supports meshes with holes and handles by using additional storage.

Akeley, Haeberli, and Burns [Akele90] developed a program to convert triangle meshes into strips. The sequence is constructed by selecting the next triangle as the one adjacent to the least number of neighbors. Speckmann and Snoeyink [Speck97] computes a minimum spanning tree of the adjacency graph to generate long triangle strips. The straightforward use of triangle strips does not result in high compression rates. Each vertex is encoded twice on average, and it is also difficult to obtain long strips from a generic mesh [Evans96b]. Long strips are desirable since the first two bits are the overhead for each strip. Deering [Deeri95] proposes the use of a buffer of vertices to avoid that a vertex is encoded more than once. Following this idea, Evans *et al.* [Evans96b] discuss the impact of buffer sizes on triangle strip performance, and Chow [Chow97] proposes heuristics to improve the decomposition of triangular meshes into triangle strips. Rossignac [Rossi99] suggests modifications to the idea of using a buffer of 16 positions proposed by Deering, estimating a connectivity cost of $3.75 + 0.062 \log_2 n$ bits per triangle, when a vertex is used twice on average.

3 PROPOSED METHOD

The proposed method seeks to minimize the number of vertices to be sent to the graphic pipeline. Two heuristics were considerated. One seeks to minimize the number of vertices reducing the number of strips, generating output to a hardware and a graphic library that support swap without resending a vertex. The number of necessary vertices is defined as $t+2k$, where t is the number of triangles in the mesh, and k the number of generated strips. The other heuristic minimizes the number of vertices, avoiding swap generation, producing output for a graphic library (e.g., OpenGL) that simulates swap resending a vertex.

Although our implementation is sequential, it has also investigated the generation of multiple strips simultaneously at several places of the mesh, doing the concatenation if possible.

3.1 Local Algorithm

The algorithm for choosing the next triangle to be inserted in a strip is similar to other greedy algorithms [Akele90, Evans96b]. The proposed algorithm analyzes the dual graph of the mesh taking priority for inserting triangles, which have many adjacent triangles in strips. In case of tie, our algorithm uses different look-ahead strategies, depending on the heuristic under consideration.

A description in more details of the local algorithm is now presented. Let the degree of a triangle be the number of adjacent triangles that do not belong to any strip. The selection of the next triangle is performed by using the following steps:

- if a triangle in the candidate list has degree 0, it is added immediately in order to avoid the occurrence of a singleton strip (strip containing only one triangle).
- if there is no candidate triangle with degree 0, triangles with degree 1 have now priority. In case of several candidates with this degree, a look-ahead test is performed. If the adjacent triangle has degree 1, it is inserted immediately. If all the adjacent triangles have degree 2 or 3, the algorithm seeks to insert a candidate that does not generate swap, in case of minimizing swaps. Otherwise, it is inserted the triangle which has an adjacent one with lower degree.
- if all the candidates have degree 2, the choice of the next triangle is also performed according to the heuristic under consideration. In case of strip minimization, it is inserted the triangle which has an adjacent one with lower degree. In case of swap minimization, the next triangle is one that does not generate swap.

Whenever a new strip is created, a low-degree triangle is chosen as the starting one.

3.2 Multiple Strip Construction

The method uses a strategy based on a simultaneous construction of strips. The algorithm maintains s strips being built and at each step adds an adjacent triangle to one of the strips.

Figure 2 exemplifies a case of four strips being created at same time. In this example, the next triangle to be added is chosen among the list of candidates $T_1, T_2, \ldots, T_9$. The candidate is inserted in a strip according to Section 3.1. If the triangle chosen is T_2, strips 1 and 4 can be concatenated together. Case this concatenation occurs, the strip 1 encompasses the strip 4, and another strip is created in order to maintain the number of strips in construction. The location of the new strip is chosen based on a restriction that the start triangle of the new strip is not adjacent to the extremities of an existing strip, avoiding the immediate concatenation of the new strip. If there is no more candidate triangle for insertion to any strip extremity, then s new strips are created for construction.

If $s > 1$ and a triangle having either degree 0 or 1 is inserted, two strips may be concatenated together. The possible cases to be considered are:

- insertion of triangle with degree 0 (T_1 in strip 1 shown in Figure 3, cases (a), (b), and (c)).

  Case 1: the other two triangles adjacent to T_1, besides strip 1, are strip extremities. Strip 1 is concatenated to one that contains fewer triangles.

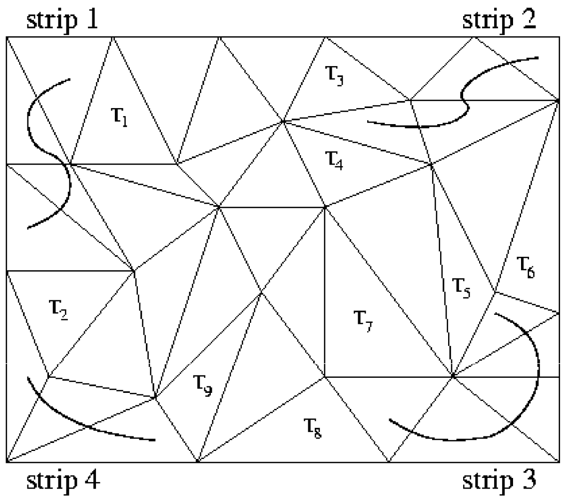


Figure 2: Simultaneous strip construction.

Case 2: only one triangle adjacent to T_1 is extremity of other strip The concatenation is straightforward.

Case 3: there are no extremities of other strips adjacent to T_1. No concatenation is performed.

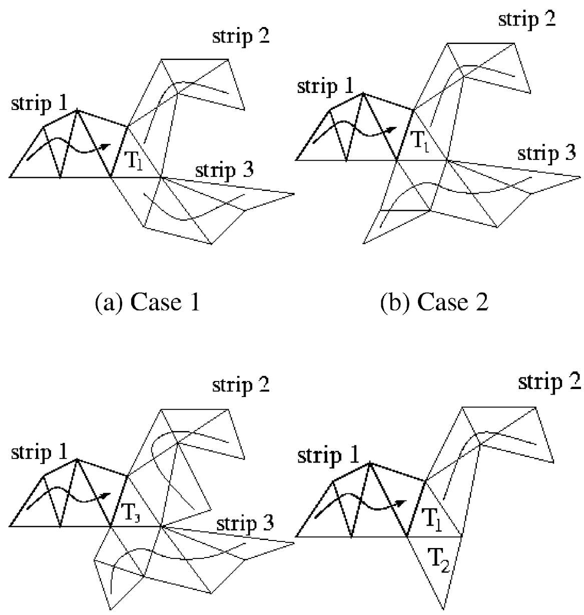
- insertion of triangle with degree 1 (T_1 in strip 1 shown in Figure 3, case (d))

Case 4: If T_1 has degree 1, then there is a triangle T_2 that does not belong to any strip. The concatenation is performed in case of T_2 having degree greater than 1. Otherwise, this union will generate a singleton strip, therefore the strips will not be joined.

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 450 MHZ with 128 Mbytes RAM, running LINUX operating system. The source code is available upon request to the authors.

We compared our algorithm against `STRIPE 2.0` [Evans96b], which is the best known publicly available program. In all tests, our algorithm generated better results for all parameters under consideration, producing lower number of vertices, lower number of strips, less memory usage, and less CPU time. It is worth mentioning that I/O operations have been excluded from timing in order for `STRIPE` and our algorithm to report the same type of statistics data.



(a) Case 1 (b) Case 2

(c) Case 3 (d) Case 4

Figure 3: Strip concatenation.

The experiments used data sets available from Stanford Graphics Lab, United States Geological Survey, Georgia Institute of Technology, and Viewpoint DataLabs. Table 1 shows the number of vertices and triangles for ten data models used in our tests.

The results of comparison between our method and `STRIPE` are summarized in Table 2 and Table 3, which show 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 vertices while reducing the number of strips (default mode) and other that seeks to minimize the number of swaps, re-

Model	Vertices	Triangles
bunny	35947	69451
cow	2904	5804
crater	107903	214808
dragon	437645	871414
cannyon	20000	39885
buddha	543652	1087716
horse	48485	96966
champlain	100000	198996
foot	2154	4204
hand	327323	654666

Table 1: Sample of models.

Model	Stripe		Ours	
	Strips	Vertices	Strips	Vertices
bunny	918	91705	599	85831
cow	102	7646	80	7607
crater	4194	310800	3561	297879
dragon	-	-	16222	1216698
cannyon	891	58293	798	55743
buddha	-	-	20071	1520115
horse	1630	124403	842	122324
champlain	3946	289593	3372	275452
foot	110	6048	82	5668
hand	-	-	8440	866729

Table 2: Comparison of triangle strip algorithms minimizing strips.

spectively. It is worth observing that the number of vertices shown in Table 2 corresponds to the OpenGL cost model. In case of models having built-in swap, the actual number of vertices can be trivially calculated by $t + 2k$.

Table 4 reports the execution times required to construct the representations. Our algorithm behaves linearly with respect to the input size. Table 5 shows no significant change in the number of strips as multiple strips are constructed simultaneously. The values shown in Tables 2, 3, and 4 were obtained by using $s = 1$. Figure 4 presents the results for four different data sets.

Model	Stripe		Ours	
	Vertices	Strips	Vertices	Strips
bunny	82128	1230	81856	1147
cow	7123	137	7092	137
crater	283804	5860	283047	5320
dragon	-	-	1139294	23427
cannyon	52258	1188	52110	1089
buddha	-	-	1421383	29128
horse	117621	1918	117506	1867
champlain	260712	5505	260169	5010
foot	5417	121	5363	114
hand	-	-	816267	14662

Table 3: Comparison of triangle strip algorithms minimizing vertices.

Model	Stripe	Ours
bunny	2.01670	0.21096
cow	0.14076	0.01599
crater	4.72040	0.66261
dragon	-	2.53687
cannyon	0.83277	0.12145
buddha	-	3.15003
horse	2.47020	0.29593
champlain	4.42200	0.61523
foot	0.06577	0.01189
hand	-	1.79598

Table 4: Execution times in seconds.

Model	1	2	4	8	16
bunny	599	575	601	591	601
cow	80	78	73	83	90
crater	3561	3428	3461	3519	3449
dragon	16222	16402	16304	16281	16313
cannyon	798	789	772	772	794
buddha	20071	19878	19993	19891	19921
horse	842	806	811	884	868
champlain	3372	3374	3357	3426	3390
foot	82	88	101	119	131
hand	8440	8318	8227	8156	7980

Table 5: Results for different values of s.

5 CONCLUSION AND FUTURE WORK

We have presented an efficient method for constructing 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.

Future work includes the investigation of new local heuristics, a more detailed study of simultaneous generation of a variable number of strips.

6 ACKNOWLEDGEMENTS

This work was partially supported by a grant from Conselho Nacional de Desenvolvimento Científico e Tecnológico, CNPq, Brazil. The authors would also like to thank the Programa Especial de Treinamento (PET) (Special Training Program) of the Computer Science Department.

REFERENCES

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

[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.

[Evans96a] F. Evans, S. Skiena, and A. Varshney. Completing sequential triangulations is hard. Technical report, Department of Computer Science, State University of New York at Stony Brook, USA, 1996.

[Evans96b] F. Evans, S. Skiena, and A. Varshney. Optimizing triangle strips for fast rendering. In *Proceedings of IEEE Visualization'96*, pages 319–326, 1996.

[Hoppe96] H. Hoppe. Progressive meshes. In *Computer Graphics, SIGGRAPH'96 Proceedings*, pages 99–108, New Orleans, Louisiana, USA, 1996.

[Hoppe98] H. Hoppe. Efficient implementation of progressive meshes. *Computer & Graphics*, 22(1):27–36, 1998.

[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] Jackie Neider, Tom Davis, and Mason Woo. *OpenGL Programming Guide: The Official Guide to Learning OpenGL*. Addison-Wesley, New Jersey, 1993.

[Rossi99] J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61, 1999.

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

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

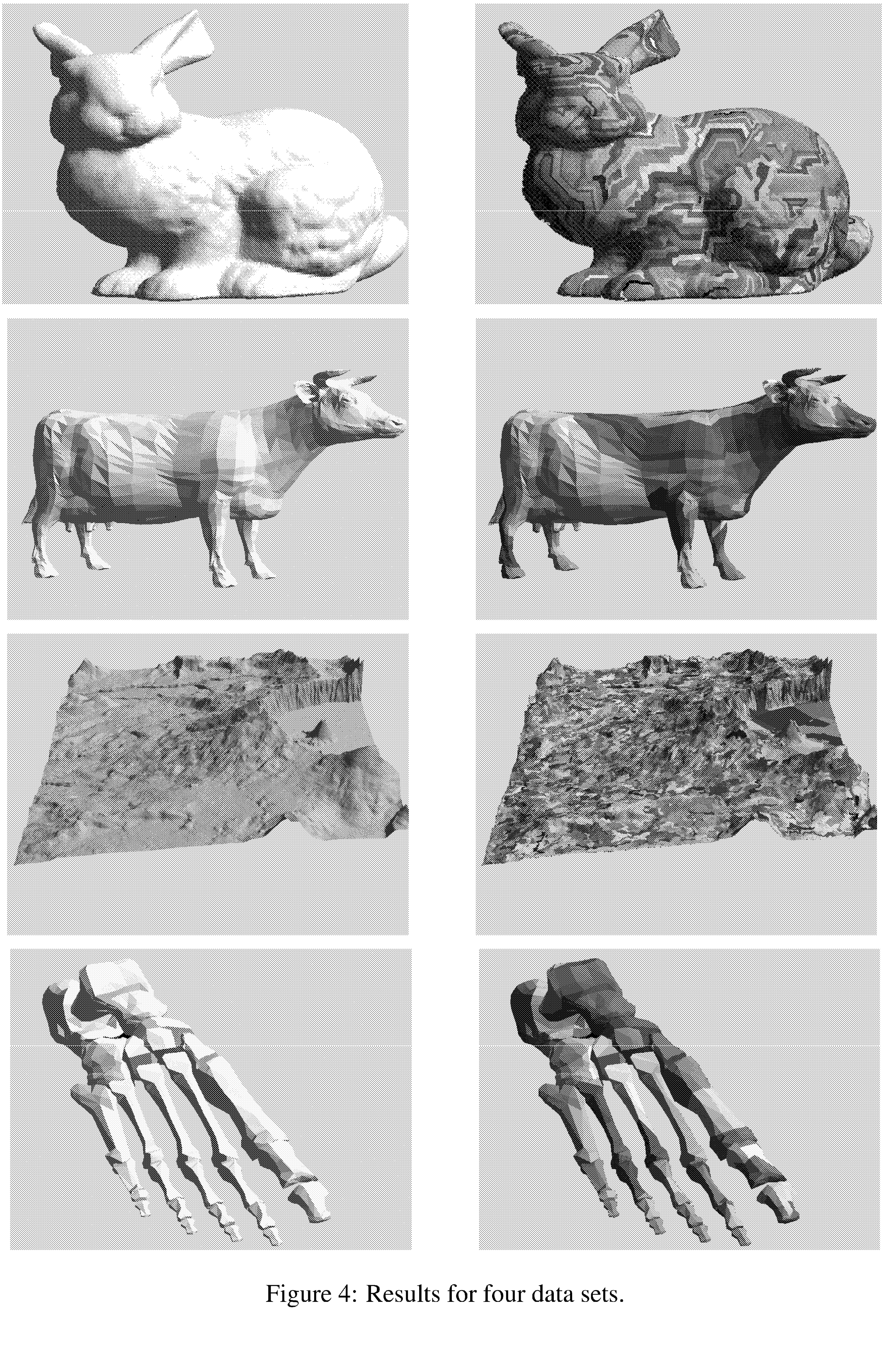


Figure 4: Results for four data sets.