Optimized Compression of Triangle Mesh Geometry Using Prediction Trees

Boris Kronrod Craig Gotsman
Computer Science Department
Technion - Israel Institute of Technology
Haifa 32000, Israel
kronrod@cs.techion.ac.il gotsman@cs.technion.ac.il

Abstract

Almost all triangle mesh compression algorithms to date are driven by the mesh connectivity code. The geometry code usually employs a straightforward prediction method applied to the vertex sequence as dictated by the connectivity code. This generates a suboptimal geometry code, which results in significant loss in code efficiency, since the geometry dominates the mesh information content. This paper proposes a manifold mesh code which optimizes the geometric component, at the slight expense of the connectivity code. This mesh geometry code is shown to be up to 50% more compact than the state-of-the-art geometry code of Touma and Gotsman, especially for models with non-smooth geometry, such as CAD models.

1. Introduction

A 3D triangle mesh consists of the following two components: geometry – the 3D coordinates of the mesh vertices, and connectivity - defining the edges and faces between the mesh vertices. Recently, a wealth of algorithms for the efficient coding of this type of data have been published (e.g. [1,2,4,6,7,11,13,14]). All these focus on achieving the most compact code for the connectivity data, resulting in between 1.5 and 4 bits per vertex on the average. The main disadvantage of these coding techniques is that they ignore the geometry of the model. The vertex coordinates are then coded in an order induced by the connectivity code, which is usually not optimal.

The geometric data, being floating point, is commonly quantized to a fixed number of bits per vertex before coding. Ten bits/vertex is typical, so the raw information content of a vertex is 30 bits before coding. The few published compression techniques which deal with mesh geometry (e.g. [4,8,14]) generate codes whose size is 40-50% of this. It is well known that the geometric portion of the content dominates the code, so that is the place to achieve a significant improvement in the code as a whole. However, care must still be taken to code the connectivity in a sophisticated way, as the most straightforward connectivity code may require more than 30 bits/vertex.

After having quantized the geometry, spatial geometry coding techniques then code the data in a lossless manner, meaning that this data will be recovered exactly by the decoder. In contrast, spectral techniques [8] quantize other (transform) coefficients which represent the data, so the resulting loss is less predictable. Metrics which measure the *distance* between two similar models must then be employed in order to quantify the distortion of the data.

We propose a way to significantly optimize the spatial geometry code without sacrificing too much in the connectivity code. In our approach we achieve **exactly** 4 bits/vertex for the connectivity before entropy coding, which is not as good as other published algorithms, but certainly not significantly worse.

Our approach is based on the "parallelogram" prediction method for mesh geometry, first introduced by Touma and Gotsman [14]. This method is based on the observation that two adjacent triangles in a typical mesh tend to form an almost planar shape similar to a parallelogram. Thus, if a triangle A is given, the missing vertex of the adjacent triangle B may be predicted quite reliably (see Fig. 1). Note that the prediction error is symmetric, namely that the error resulting from predicting the missing vertex of A based on B is identical to the error resulting from predicting the missing vertex of B based on A. Experimental results seem to indicate that the parallelogram rule proves itself in practice, and it has been adopted for the emerging MPEG-4 standard for mesh geometry coding [10]. Other geometric patterns will certainly be present in 3D mesh data sets, as Lee and Ko [9] have found, but the parallelogram pattern seems to be dominant, especially when the mesh data is generated by sampling free-form surfaces such as NURBS, along iso-parametric curves.

Despite the dominance of parallelogram structures, there are obvious cases where it fails dismally. An easy example is a so-called *CAD model.* These models are characterized by significant flat areas, corners and folds,

and are far from being smooth. This means that adjacent triangles are either (close to) planar, or have a significant

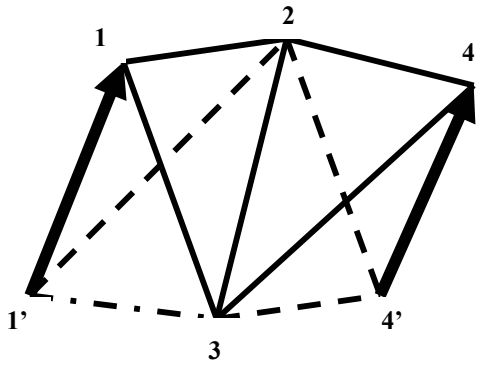


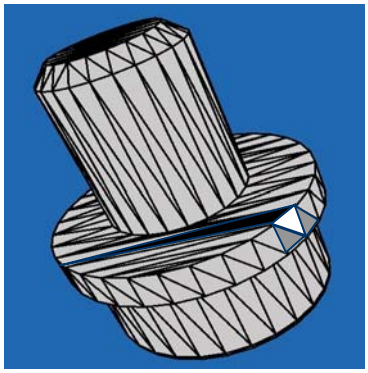
Figure 1: Two adjacent triangles in a mesh (123 and 234), the parallelogram predictions 1' and 4' of vertices 1 and 4, and the resulting (thick black) symmetric prediction error vectors.

fold between them. In the latter case, parallelogram prediction will be completely off mark, resulting in a large prediction error. See Fig. 2 for an example. This means that for these types of models, it is best to predict triangles within the almost planar regions, definitely not *between* creases. Unfortunately, a coder driven by connectivity information alone cannot tell the difference between the two cases.

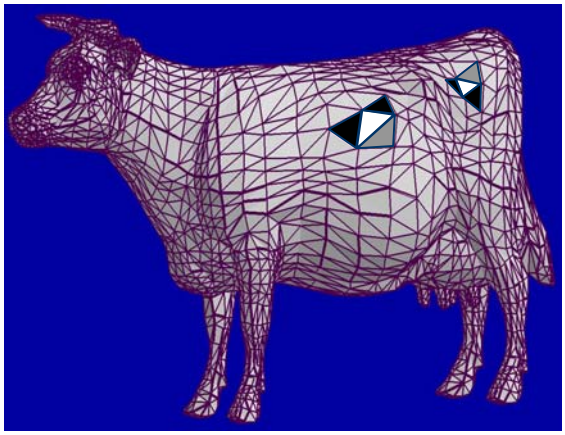
If an algorithm uses a parallelogram prediction method, then it should build a traversal structure of triangles *covering* all vertices. By "covering", we mean that *all* mesh vertices are contained in the set of vertices of the traversed triangles. The vertex coordinates are then predicted based on the previous triangle in the traversal. The cost of this code is then the entropy of the distribution of the vertex prediction errors. Since this entropy is hard to manipulate, the accepted practice is to measure the code effectiveness in the approximation sense, i.e. as the sum of the lengths of the vertex prediction error vectors. The smaller the better. This paper proposes a triangle traversal structure to which the geometric prediction is applied, which attempts to minimize this measure, which we call the *spread* of the distribution (assuming implicitly that the distribution is concentrated around the origin).

The only previous work which attempts something similar is that of Bossen [3], who tries to optimize the "Topological Surgery" coding method of Taubin and Rossignac [13]. His method, however, is applied to a tree spanning *all* the mesh triangles, which is significantly different from the tree we construct.

The rest of this paper is organized as follows: Section 2 analyzes the best we can hope for in terms of minimal prediction error. Section 3 describes our approach and experimental results are presented in Section 4. We conclude in Section 5.



(a)



(b)

Figure 2: Possible parallelogram predictions based on the white triangle: The gray is good, and the black bad. (a) Bolt. (b) Cow.

2. A lower bound

When the code is based on parallelogram predictors, a simple lower bound on the geometry prediction error spread may be established as follows. Since any mesh vertex with degree d may be predicted based on any of the d different triangles incident on the vertex, the best that can be achieved is the minimal approximation error resulting from one of these triangles. See Fig. 3.

Denoting this minimal approximation error for vertex v by $\mathrm{err}(v)$, the sum of $\mathrm{err}(v)$ over all vertices in a given mesh is a lower bound on the spread for that mesh. Note that this bound is quite loose, as it is probably impossible to build a coherent triangle structure which will achieve the best possible prediction advocated by the lower bound among all adjacent triangles for *all* vertices. For example,

the lower bound for the spread of the bolt model of Fig. 2a is 7.5, yet our method, as will be described in the sequel, will achieve only 136.

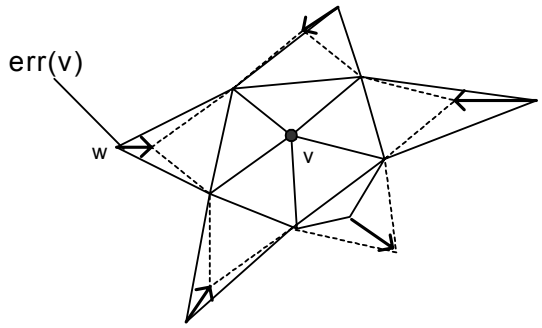


Figure 3: Possible vertex predictors and prediction errors for a vertex *v*. The best prediction for *v* in this example is from the adjacent triangle with vertex *w*. Note, however, that this does not imply that the best predictor for *w* is *v*.

3. Our approach

Any coding scheme for a class of objects may be viewed as the compact representation of members of that class. In the case of 3D triangle meshes, we need to devise representations for both the mesh connectivity and geometry. Coding the geometry will require an orderly traversal of the mesh triangles in some order which facilitates accurate causal prediction of vertex coordinates. This same (tree) structure will also represent a subset of the connectivity information, since it indicates how some of the triangles are joined together to form the mesh. Hence, we first build a "cover tree" which traverses a subset of the triangles in the mesh, such that tree edges straddle only adjacent triangles, and all mesh vertices are contained in these triangles at least once. Note that this cover tree covers only a *subset* of the mesh triangles, as opposed to the triangle cover trees used by the Topological Surgery method [13], which cover *all* the mesh triangles (vertices of the dual graph).

Unfortunately, the constraint that every vertex be covered *exactly* once is quite strong, and does not leave much freedom to optimize the tree by other measures. Hence in practice we must relax this constraint, and typically a few vertices will be covered more than once. To determine at the decoder which vertices are actually duplicates of vertices decoded before, we identify vertices with the same geometry. The cost of a repeated vertex to the geometry code is not small, but in practice it occurs very infrequently, so is quite negligible. By using a carefully designed cover tree, of which there are many, we are able to optimize it to minimize the spread of the vertex geometry prediction errors.

Since the cover tree covers only some of the mesh triangles, it represents only part of the edge information in the mesh. The remainder of the edge information is represented by coding the triangulation of the polygons whose boundaries are the cover tree boundaries. See Fig. 4. Each of these steps will be described now.

3.1 Building the cover tree

The objective is to build an optimal cover tree with minimum spread. Since each vertex's coordinates are predicted from the preceding triangle in the tree, the weight associated with a tree edge is the prediction error between the two appropriate mesh triangles. The fact that the prediction error between two adjacent triangles is symmetric implies that the total weight of the tree does not depend on which triangle is the root of the tree. Hence the total weight of a mesh cover tree *T*, which we aim to minimize, is the sum of prediction errors associated with all edges of *T*.

We cast the problem of finding an optimal cover tree as a graph-theoretic problem. Build a graph G=<V,E>, such that the node set V is the set of all mesh triangles *and* vertices, and E contains a bi-directional edge between two mesh triangles iff they are adjacent. The weight of this edge is the symmetric prediction error of the two triangles. E also contains a directed edge from a triangle to all three of its vertices. The weight of this edge is zero. G is an augmented version of the dual of the triangle mesh. The objective is to construct a tree on G which covers all nodes of G which are mesh vertices, and has minimal weight. This tree will cover all mesh vertices via a subset of the mesh triangles. See an example in Fig. 5.

Unfortunately, finding this optimal cover tree is a special case of the well-known Minimal Steiner Tree problem for weighted directed graphs, which is NP-Hard [5], so the best that can be hoped for in a polynomial-time algorithm is a tree whose total weight *approximates* the optimum, in the sense that its weight is not more than a constant factor times the minimum.

We approximate the optimum with a greedy algorithm employing the simple approximation heuristic used in the undirected Minimal Steiner Tree problem: Starting from an arbitrary mesh vertex, the tree is extended by the vertex which is closest to the tree (the distance is measured as the total edge weight of a path in the graph). This approximates the optimum by a factor of two. The complexity of this (encoding) algorithm is known to be $O(n^2)$, but has been found to be significantly less in practice. The complexity of the decoding algorithm is linear.

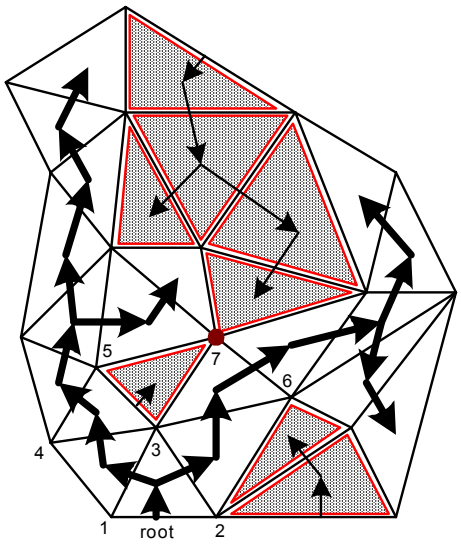


Figure 4: Coding mesh connectivity and geometry. The white triangles and all mesh vertices are covered by the (thick black) cover tree, starting from the root edge and the three initial geometries of vertices 1, 2 and 3. The geometries of vertices 4 and 6 may then be predicted, then those of vertices 5 and 7, and so on. All vertices are covered once, except for the "fat" vertex 7, which is covered twice. The gray triangles represent the remainder of the mesh connectivity, which form a set of triangulated polygons, also encoded as a set of (thin black) trees.

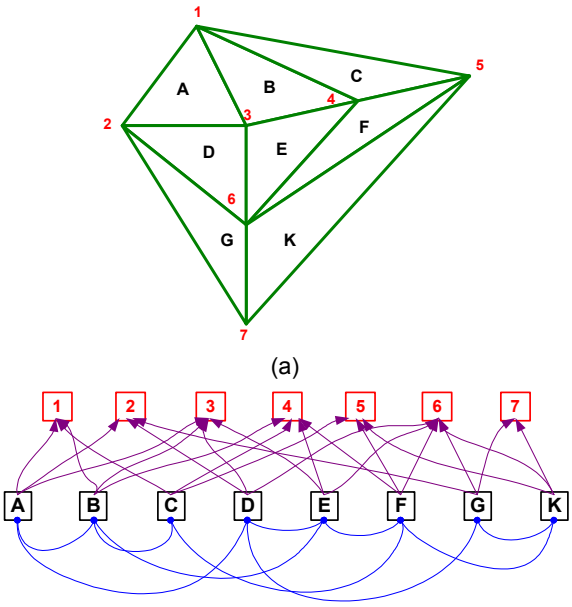


Figure 5: Building the graph from a mesh. (a) A mesh. Faces are labeled with letters and vertices with numbers. (b) The corresponding graph upon which the optimization procedure is run. Edges between mesh faces are undirected and weighted with prediction errors associated with edges incident on both faces. Directed edges between mesh faces and mesh vertices have zero weight.

3.2 Coding the remainder of the connectivity

The cover tree represents only part of the connectivity of the mesh. Thus more information is required to complete the rest of the connectivity information.

The boundaries of the cover tree form a set of triangulated *simple polygons*, since they cannot contain interior vertices. See Fig. 4. The connectivity of a triangulated simple polygon may be represented with no more than two bits per triangle, using a binary spanning tree on the triangles, as has been described before [12]. Hence 4 bits/vertex suffice to code the connectivity. This is not as good as other algorithms reported in the literature, which achieve as little as 1.5 bits/vertex, but it is not very bad either, and the sum of the connectivity and geometry code still yields a profit. Since the decoder reconstructs these simple polygons in the same order as the encoder, it is able to "zip" them into the cover tree by tracing (and identifying) the boundary edges of both in the same order. A boundary edge is an edge of the triangulated polygon which is incident on only one triangle.

This method works for a closed manifold mesh. If the mesh has boundaries ("holes"), we use the method proposed by Touma and Gotsman [14], in which the mesh is closed by adding one "dummy" vertex to each such boundary. The dummy vertex is connected to all vertices in the boundary, and assigned coordinates which are the average of the boundary vertex coordinates. The resulting closed mesh is then coded as above, and the dummy vertices marked. At the decoder, these dummy vertices are removed along with all edges incident on them. In the typical case where the mesh has but a few boundaries, the coding overhead is minimal.

4. Experimental results

We have implemented the algorithms outlined here, and compared the results with those obtained by the algorithm of Touma and Gotsman [14] (the so-called "TG" algorithm), which is widely considered to be state-of-the-art. Some of the models we experimented with are shown in Fig. 6, a mix of smooth and non-smooth meshes. For each algorithm we measured the spread of the prediction

error distribution: $\frac{1}{n}\sum_{i=1}^{n}\|err(v_i)\|$ and its entropy $-\sum_{i=1}^{m} p_i \log_2(p_i)$, where n is the number of mesh vertices, m the number of different prediction error values, and p_i the relative frequency of that value. We also compared the results of coding using the optimal cover tree generated by our algorithm with that obtained using other simple cover trees, such as that generated by Breadth-First-Search (BFS) or Depth-First-Search (DFS). Fig. 7 shows some of these trees and their performance on a sample model.

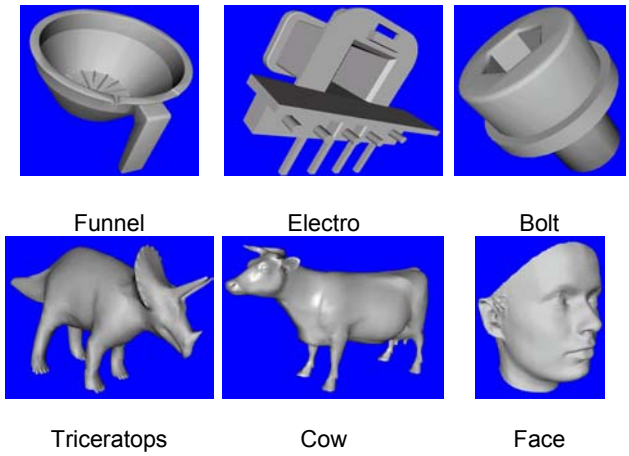


Figure 6: Sample triangle meshes

The performance of the TG algorithm on the geometric component of the mesh seems to be quite average, since our algorithm, when run using a simple BFS or DFS cover tree results in comparable, or even better, statistics for both the spread and entropy.

Our algorithm with its optimized cover tree attempts to minimize the spread of the prediction error lengths, producing a distribution of the prediction error population narrower than that produced by the TG algorithm. While a smaller spread usually results in a smaller entropy of the distribution, it is not always the case. For example, a decrease of 5% in the distribution spread will usually not result in any reduction of the entropy. In the models we experimented with (see Tables 1 and 2), applying our algorithm to smooth models resulted in an average gain in the spread of 25% relative to that generated by the TG algorithm, but only in an average gain of only 10% in entropy.

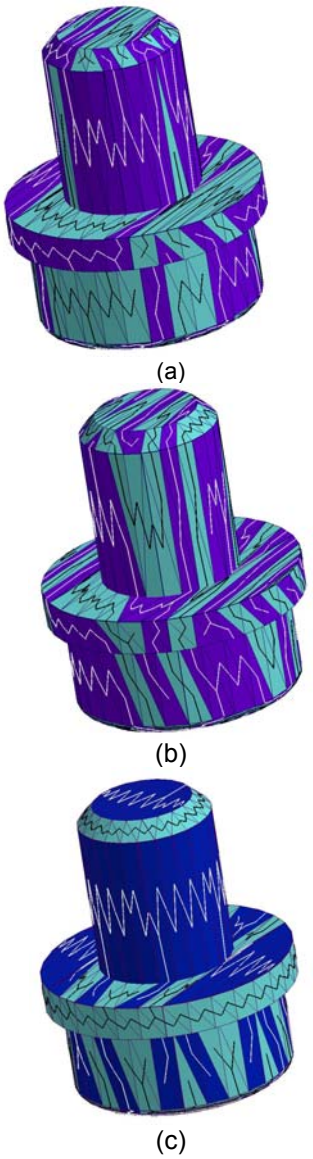


Figure 7: Comparison between different trees used to code the geometry of the bolt model (quantized to 8 bits/coordinate), containing 328 vertices. Cover tree is drawn in black, and faces that it covers colored light (cyan). Other polygons to complete the connectivity code are colored dark (blue). (a) BFS cover tree. No vertices are covered more than once and only one extra polygon is needed to complete the connectivity. Geometric prediction error entropy = 13.3 bpv. (b) DFS cover tree. No vertices are covered more than once and only one extra polygon is needed to complete the connectivity. Geometric prediction error entropy = 13.0 bpv. (c) Optimal cover tree. Note how it does not cut thru creases unless absolutely necessary. Only one vertex is covered twice and two extra polygons are needed to complete the connectivity. Geometric prediction error entropy = 9.9 bpv.

A parallelogram predictor will not predict well if the two adjacent triangles are not parallel, or if there is a significant crease between them, since the parallelogram rule assumes planarity. The latter is not a serious problem in smooth models, but is for models containing sharp corners and creases, such as CAD models. It is on this class of models that we expected our algorithm to perform the best, and, indeed, for these we obtained an average gain of 75% in the spread relative to that generated by the TG algorithm, and an average gain of 45% in the entropy.

Increasing the number of quantization bits per coordinate from 10 to 12 (adding 6 bits to the raw vertex data) seems to add almost 6 bits per vertex to the resulting entropies of all the codes, and a factor of almost 4 to the spreads, indicating that these extra bits are probably just noise, hence uncompressible, and redundant in a sense.

The Bolt model is small (328 vertices) compared to the others, so even quantization of 8 bits per coordinate is sufficient to preserve its fidelity. It this case the gain in the spread was 54%, corresponding to 30% in the entropy.

Model		Vertices	Spreads				Entropy (bits/vertex)			
			TG	KG (optimal)	KG (BFS)	Gain %	TG	KG (optimal)	KG (BFS)	Gain %
Non smooth	**Funnel**	4,193	37.6	8.8	38.7	**77**	16.9	9.5	14.5	**41**
	Electro	6,556	48.0	14.2	49.8	**70**	12.5	6.3	10.2	**50**
	Bolt	328	100.0	43.0	106.0	**57**	16.0	13.6	18.1	**15**
Smooth	**Triceratops**	2,832	8.9	6.2	10.3	**30**	13.6	12.0	14.1	**12**
	Cow	2,604	9.4	7.4	11.6	**21**	13.9	12.9	14.6	**7**
	Face	12,530	5.5	4.6	6.8	**16**	11.6	10.7	12.4	**8**

Table 1. Experimental results. TG – Touma and Gotsman algorithm. KG (optimal) – our algorithm. KG (BFS) is our method using a simple BFS cover tree. Gain is the relative savings when using between KG (optimal) vs. TG. All model geometries were quantized to 10 bits/coordinate before coding.

Model		Vertices	Spreads				Entropy (bits/vertex)			
			TG	KG (optimal)	KG (BFS)	Gain %	TG	KG (optimal)	KG (BFS)	Gain %
Non smooth	**Funnel**	4,193	150	36.5	154	**76**	21.3	14.2	19.3	**34**
	Electro	6,556	191	54.3	198	**72**	15.7	7.7	12.5	**50**
	Bolt	328	387	158.0	402	**65**	21.6	18.9	23.4	**12**
Smooth	**Triceratops**	2,832	35.2	23.9	40.7	**32**	19.4	17.3	19.5	**11**
	Cow	2,604	37.4	29.6	44.8	**21**	19.8	18.8	19.8	**5**
	Face	12,530	21.5	17.7	26.9	**18**	17.4	16.1	18.1	**7**

Table 2: Experimental results. TG – Touma and Gotsman algorithm. KG – Our algorithm. All model geometries were quantized to 12 bits/coordinate before coding

5. Discussion and future work

This paper has shown how to optimize triangle mesh codes by exploiting a major source of savings, largely ignored in previous works – the mesh geometry. Rather than the connectivity code drive the geometry code, here we do the opposite. Optimizing the geometry code has led to significant savings in the overall code size. Casting the problem in an optimization setting will allow similar results to be obtained for more elaborate prediction schemes.

The main drawback of our method is the complexity of the encoder. Due to the need to run an optimization procedure at the encoder, it is up to one order of magnitude slower than, e.g. the TG encoder. Our decoder, however, is very fast, so for the many applications where the encoding is done offline, the encoder speed is not an impediment.

Future work will address the question of how to directly optimize the prediction error entropy, as opposed to its spread, leading to a more compact code. We will also deal with non-manifolds and non-triangular meshes. The most interesting questions, in our opinion, relate to lower bounds on the number of bits needed to code a typical triangle mesh. This implies, of course, the need to impose a (subjective) probability distribution on the class of triangle meshes, in order to compute the entropy.

Acknowledgements

Thanks to Roni Raab for implementing the algorithm described in this paper and performing the experiments.

This work was supported by German-Israel Fund (GIF) grant no. I-627-45.6/1999.

References

[1] P. Alliez and M. Desbrun. Valence-driven connectivity encoding for 3D meshes. Proceedings of Eurographics, pp. 480-489, 2001.

[2] V. Bajaj, V. Pascucci and G. Zhuang. Single resolution compression of arbitrary triangular meshes with properties. Computational Geometry: Theory and Applications 14, pp. 167-186, 1999.

[3] F. J. Bossen. On the art of compressing three-dimensional polygonal meshes and their associated properties. Ph.D. Thesis. Ecole Polytechnique Federale de Lausanne, 1999.

[4] M. Deering, Geometry compression, Computer Graphics, Proceedings of SIGGRAPH‘95, pp. 13-20, ACM, 1995.

[5] M. Garey and D. Johnson, Computers and Intractability: A guide to the theory of NP-completeness, pp. 208-209, 1978.

[6] S. Gumhold and W. Strasser. Real time compression of triangle mesh connectivity. Proceedings of SIGGRAPH '98, pp. 133-140, ACM, 1998.

[7] M. Isenberg and J. Snoeyink. Face Fixer: Compressing polygon meshes with properties. Proceedings of SIGGRAPH’00, pp. 263-270, ACM, 2000.

[8] Z. Karni and C. Gotsman. Spectral compression of mesh geometry. Proceedings of SIGGRAPH’00, pp. 279-286, ACM, 2000.

[9] E. Lee and H. Ko, Vertex data compression for triangular meshes, Proceedings of Pacific Graphics’00, pp. 225-234, ACM, 2000.

[10] MPEG-4 Standard. http://www.mpeg-4.com

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

[12] G. Taubin, A. Gueziec, W.Horn and F. Lazarus. Progressive forest split compression. Proceedings of SIGGRAPH’98, pp. 123-132, ACM, 1998.

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

[14] C. Touma and C. Gotsman, Triangle mesh compression, Proceedings of Graphics Interface ‘98, pp. 26-34, 1998.