Distance Ranked Connectivity Compression of Triangle Meshes

Patrick Marais, James Gain*
Dept of Computer Science
University of Cape Town

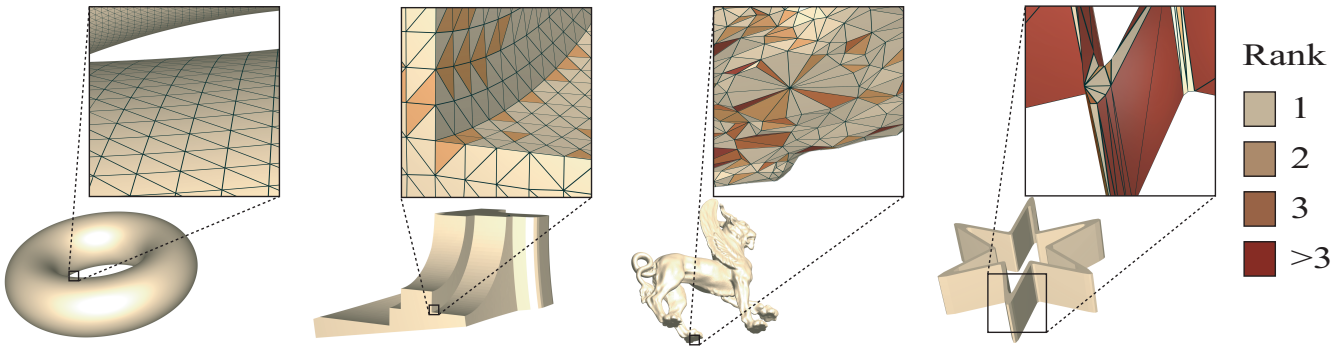

Figure 1: **Distance Ranked compression.** (a) highly regular, torus (0 bpv), (b) regular, fandisk (0.69 bpv) (c) irregular, feline (2.12 bpv)) (d) pathological, star (7.09 bpv). The colours indicate the distance rank code for each triangle.

Abstract

We present a new, single-rate method for compressing the connectivity information of a 2-manifold triangle mesh with or without boundary. Traditional compression schemes interleave geometry and connectivity coding, and are thus unable to utilise information from vertices (mesh regions) they have not yet processed. With the advent of competitive point cloud compression schemes, it has become feasible to develop separate connectivity encoding schemes which can exploit complete, global vertex position information to improve performance.

Our scheme demonstrates the utility of this separation of vertex and connectivity coding. By traversing the mesh edges in a consistent breadth-first fashion, and using global vertex information, we can predict the position of the vertex which completes the unprocessed triangle attached to a given edge. We then rank the vertices in the neighbourhood of this predicted position by their Euclidean distance. The distance rank of the correct closing vertex is stored. Typically, these rank values are small, and the sequence of rank values thus possesses low entropy and compresses very well. The paper details the algorithm as well as the predictors we have tested. Results indicate improvement on the current best valence-based schemes for many common mesh classes.

CR Categories: I.3.m [Computer Graphics]: Miscellaneous—connectivity coding

Keywords: triangle mesh, connectivity compression, geometry driven coding

*e-mail: {patrick,jgain}@cs.uct.ac.za

1 Introduction

Triangle meshes are widely used to represent 3D surface models since they are well suited to computer rendering hardware. The development of powerful consumer display cards has been accompanied by a corresponding growth in the size and complexity of 3D models. Naturally this complexity comes at the cost of greater storage requirements, and this has fuelled research into techniques that compress both the geometry (vertex positions) and the connectivity information of triangle meshes.

The earliest approaches [Deering 1995] exploited the adjacency structure of a triangle mesh to avoid storing unnecessary connectivity information and applied simple quantisation and delta encoding to represent vertex positions. In general, the issue of connectivity compression is deemed more pressing, since, while it is possible to quantise vertex positions quite coarsely and still maintain a good surface approximation, connectivity information must be represented exactly which usually requires many bits per triangle. Subsequent techniques [Taubin and Rossignac 1998; Rossignac 1999] thus focused on ways to reduce the cost of encoding the mesh topology. Great strides have been made in this area. Edgebreaker [Rossignac 1999] is a "face-based" scheme, which reduces the cost of encoding connectivity to at most 2 bits per triangle, or equivalently, 4 bits per vertex (bpv). Subsequent improvements [Rossignac 2001] have refined this bound. The next great breakthrough arrived with "valence-based" schemes [Touma and Gotsman 1998], which use the number of edges attached to a vertex to derive a compact coding algorithm which, in most cases, provides far better results than face-based techniques. In these schemes, however, one cannot derive a general bound for the cost of connectivity encoding without some simplifying assumptions [Khodakovsky et al. 2001]. Valence-based schemes generates roughly *half* the number of codes compared to face-based schemes (per vertex, rather than per triangle). Valence-based connectivity encoding works well because the entropy of the valence codes typically tracks the mesh *valence entropy*, which is often very low, particularly for highly regular meshes, as demonstrated by the excellent results obtained [Touma and Gotsman 1998; Khodakovsky et al. 2001; Lee et al. 2002]. However, there are many meshes which are irregular,

and for which the valence entropy is commensurately higher.

To improve on these results we need to look for additional sources of *prior* information to reduce the code size. The encoding of geometry and connectivity are usually interleaved: the mesh is rebuilt step by step, with new vertices continually added according to the decoded connectivity information. Consequently, a significant source of prior geometric information, the set of mesh vertices, is not available in these methods. We propose the use of *global vertex information* to produce our compact coding. The recent development of algorithms to compress vertex information separately [Lee and Ko 2000; Devillers and Gandoin 2000], at rates competitive with interleaved encoding, makes this approach feasible.

For a given edge of a processed mesh triangle, we need to find the vertex which completes the attached triangle. If the underlying triangulation is highly regular, we can simply reflect the third vertex of the current triangle through this edge and assume the closet neighbouring vertex is the one we seek. If this is not the case, then the second closest vertex is probably the one we need. We can continue in this fashion, checking nearest neighbours until a match is found. Thus we are reduced to predicting a point from the information we have, and storing a "distance rank". In the ideal case, such as a smooth surface composed of regular triangles, all these ranking codes will be one (the 1st closest point will close each triangle) and the entropy of the sequence will approach *zero*. When we do not have such a regular mesh, we need to use the vertices we have not yet processed to further constrain our prediction.

The remainder of the paper is structured as follows: Section 2 discusses related work. In Section 3, we present the compression algorithm, along with a motivation for the predictors we employ and a brief discussion of issues pertaining to entropy coding. This is followed by an analysis of the results in Section 4. Finally, we present our conclusions and suggest areas for future work in Section 5.

2 Related Work

There is a large and growing literature on triangle mesh compression, and the interested reader is referred to [Alliez and Gotsman 2003] for a summary. We confine our discussion to single-rate compression connectivity schemes, since our compression technique falls into this category. Such schemes generally come in two flavours: face-based and valence-based. Face schemes are usually derivatives of *Edgebreaker* [Rossignac 1999; Attene et al. 2003], while valence schemes are modifications or extensions of Touma and Gotsman's valence-based encoder [Touma and Gotsman 1998; Khodakovsky et al. 2001]. Both these approaches interleave the encoding of geometry and connectivity.

Our approach assumes *separate* encoding for vertex and connectivity information, and that the entire quantised vertex set is available for both the encoding and decoding steps. Although there has been little work in this area, the notion of *geometry-driven* connectivity encoding has seen some support. The work of Coors and Rossignac [Coors and Rossignac 2004] provides such a scheme in which *Edgebreaker* is modified to predict the next symbol based on the geometry and connectivity of the processed mesh. However, an incorrect guess incurs a substantial bit code penalty, with the result that meshes which do not conform to the prediction scheme yield comparatively poor results[1].

Another approach which uses geometry to drive the connectivity encoding is *Angle Analyzer* [Lee et al. 2002]. This technique adopts an Edgebreaker-like traversal strategy and exploits the intrinsic properties of quad and triangle meshes to reduce the number of codes required for such meshes. Their technique improves on the best valence scheme in many instances, but for very regular meshes other valence-based techniques perform better.

Inspired by their earlier success with vertex encoding [Devillers and Gandoin 2000], Gandoin and Devillers [2002] introduce a progressive encoding scheme for geometry and connectivity encoding. The progressivity arises from the space subdivision scheme they use to encode their vertex data — this structure is augmented with additional topological codes to progressively recover connectivity as new vertices are extracted. While the results they achieve are good in relation to other progressive schemes, they lag behind single-rate compression schemes.

It should be noted that all these approaches interleave geometry and connectivity encoding, and cannot therefore use global mesh information.

3 The Algorithm

The basic algorithm involves only one fundamental operation: estimating the point that completes the triangle on the current edge. We choose an initial triangle on the surface and then process all the edges of the triangulation using a breadth-first traversal based on these 3 starting edges. For each edge we wish to identify the vertex (from the set of all vertices) which completes the attached triangle. This is accomplished by using a prediction scheme based on the parts of mesh already visited *and* (potentially) the complete set of vertices. Given a predicted estimate for the closing vertex position, we rank the neighbouring vertices based on their Euclidean distance from the predicted point and record the rank which corresponds to the correct closing vertex. The two new closing edges are then placed on the BFS queue and the process continues.

```
Build kd Tree //using quantised vertices
Read in start Triangle (V0, V1, V2)
Q.enqueue: Edge(V0, V1), Edge(V1, V2), Edge(V2, V0)
Initialise edge counts
while Q ≠ empty
    E ← Q.dequeue
    (Vi, Vj) ← OrientedEdgeVertices(E)
    if E is boundary
        ENCODE_SYMBOL 0
    else if E is OPEN // triangle required
        P ← PredictPoint(V, ProcessedMesh)
        I ← ClosestPointRank(P, PTarget, V)
        ENCODE_SYMBOL I
        Update edge counts // used for vertex culling
        Q.enqueue: Edge(Vi, PTarget)
        Q.enqueue: Edge(PTarget, Vj)
```

Table 1: The Encoding Algorithm

If the prediction is accurate the (first) closest vertex will be correct, otherwise we will need to examine vertices with successively larger ranking numbers. In any event, a single integer value will be generated for each such edge, and for the most part these values will be quite small. If the edge happens to be a boundary edge, the escape code 0 is generated. Any positive value is assumed to correspond to a distance rank value. By keeping track of the number of triangles attached to each edge, we can appropriately prune the BFS traversal

[1] Worst case error bounded above by 3 bpv

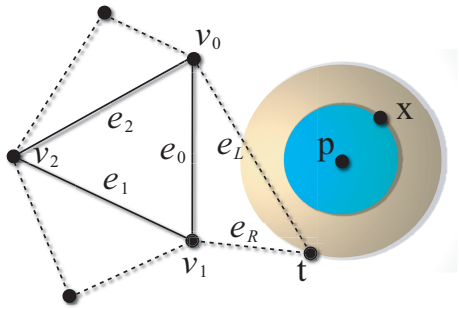

Figure 2: **Distance ranked prediction.** Give the current processed triangle, (v_0, v_1, v_2), we estimate the closing vertex, **p**. Vertex **t** is the point we are trying to find — this is the *second* closest vertex to **p**, the closest being **x**. We thus generate the code **2**, and enqueue the new edges e_L and e_R to allow further traversal of the mesh surface.

and ensure that each triangle is processed only once. The scheme thus generates $T-1$ integer values for a closed surface, where T is the number of triangles. If the surface has B boundary edges, then each boundary edge will generate an additional 0 value, and we will need to encode $T+B-1$ integers. Figure 2 shows the steps involved in encoding an edge from a given base triangle and Table 1 presents the encoding algorithm.

To recover the connectivity information, we simply reverse the process: we start off with the same initial triangle (which is specified) and then build the BFS queue in the same way. For each edge, we compute the predicted point and search for the Kth nearest neighbour (K is the value of the ranking code) to this predicted point in the complete vertex set. We then add the new edges (if any) and continue processing the BFS queue until it is empty.

We use edge information to decide whether a vertex returned from a closest point query is admissible. We cull a vertex if all the edges attached to that vertex are closed (have triangles on either side) or if the only open edges are boundary edges (which are explicitly coded). By eliminating these points we reduce the maximum size of the ranking codes, and thus reduce the overall entropy of the code sequence. The scheme automatically deals with arbitrary topologies and boundaries, but cannot deal with non-manifold triangulations.

A possible complication arises from neighbouring points which share the same Euclidean distance from the predicted point. To deal with this problem, we use the vertex *indices*: if several points have the same Euclidean distance, we rank them according to their vertex index. To avoid issues with floating point precision we perform all calculations using doubles, and then cast the results to floats before making comparisons.

We assume that the vertex compression scheme will preserve the point ordering, so that both the encoding and decoding steps will be consistent. If this is not the case for the scheme used, one can re-index the input mesh as follows prior to encoding:

1. Construct a bounding box around the point data,
2. Sub-divide the bounding box into cells of fixed size, with the choice of the cell size being such that each cell contains at most one vertex,
3. Re-index the mesh based on the order in which populated cells are encountered as you sequentially scan through all the cells.

If one stores the cell size, the decoder can quickly perform this re-indexing prior to initialising the decompression scheme and a consistent ordering of vertices is guaranteed.

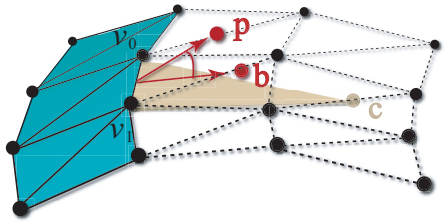

Figure 3: **Correcting for surface curvature.** Given the prediction edge (v_0, v_1), we predict the closing vertex as **p**. We search in a neighbourhood about **p** to find a small number of mesh vertices, and compute the centroid, **c**, of this set. In many cases the centroid will lie close to the underlying surface of the mesh. Finally, we rotate **p** about (v_0, v_1) onto the plane spanned by (v_0, v_1, c) to yield our final prediction, **b**.

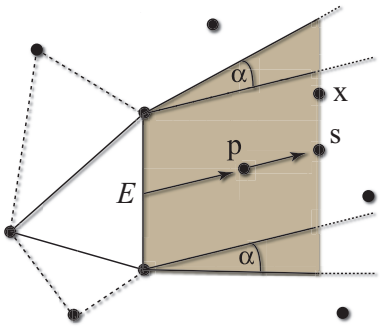

Figure 4: **Triangle size changes.** We predict our closing vertex **p**. Then, we search for the *first* vertex which lies in the region ahead of E, bounded by 2 planes parallel to the prediction direction. In this case, we find the vertex **x**. We then scale the prediction in this direction to form the final prediction, **s**. α can be used to widen the search region, and may be determined empirically across a wide range of models.

3.1 Distance rank codes

The success of the closest-point coding scheme depends on our ability to construct good predictions for a range of different mesh classes. We can identify *three* core attributes of triangle mesh geometry that affect our ability to make accurate predictions: *surface curvature*, *triangle regularity* and *triangle size*.

If the surface curves unexpectedly, a prediction based on the recovered mesh will perform poorly. Fortunately, in our scheme one can use the distribution of the points ahead of the prediction front to constrain the prediction. By using the centroid of the points clustered ahead of the prediction edge to define a plane which approximates the curvature, and then bending the prediction onto this plane, we can largely overcome this problem — see figure 3. We also implemented higher order surface fitting, but the additional gains were small and did not justify the computational expense.

If the triangulation is very regular (triangles are similar), then a simple parallelogram rule can be used to predict the closing vertex, using the 3 vertices of the base triangle. If the mesh consists of many different triangle shapes, a parallelogram rule will generally yield a

poor prediction. If the mesh has little or no regularity, then all one can reasonably say is that the prediction is *most likely* to lie some distance in front of the prediction edge. We have developed what we call the *midpoint predictor*, which predicts the closing vertex as being somewhere along a ray perpendicular to the midpoint of the base edge. For a highly regular mesh, a prediction based on on a parallelogram suffices, otherwise we use the midpoint predictor.

The final ingredient for a good prediction scheme is the ability to deal with rapidly changing triangles sizes. It is self evident that a parallelogram prediction based on a small triangle will provide a very poor estimate if the closing triangle is very large. We overcome this problem by scaling along the prediction direction by an amount that places the predicted point close to the first vertex it encounters in a "frustum" centred on the base prediction edge. See figure 4. While this is not always optimal, it tends to deal well with abrupt transitions in triangle size.

We implemented the parallelogram and midpoint predictors. Both use the mesh vertex set to deal with curvature and triangle scaling. The midpoint predictor generally places the prediction point along the ray perpendicular to the midpoint of the prediction edge. However, for meshes with highly symmetric vertex sets, such as triangulated quad meshes, the midpoint placement leads to a "tie" for closest point, which inflates the distance codes. To deal with this issue, the midpoint position is biased slightly in a direction that follows the triangle skew of the adjacent processed triangles. This generally ensures that if the closest point is not the target vertex, then the second closest point will be.

In order to choose the appropriate predictor, we apply the same triangle traversal to the mesh and *approximate* the predictor solutions, P_i, using only the attached triangle. This local information is enough to estimate the scaling and curvature behaviour referred to above, without any expensive closest point queries. We maintain a counter, C_i, for each predictor. We increment C_i if P_i is closer to the target vertex, T, than any vertices attached to T. If none of the P_i satisfy this condition (note that *both* can), we increment the counter for the predictor *closest* to T. The predictor with the highest counter is chosen to encode the surface.

The ranking values we generate typically span the first few positive integers, with a probability distribution heavily skewed towards small values. We use a standard *adaptive arithmetic coder* [Moffat et al. 1998] to compress the connectivity sequence. In general, the probability of the symbols 1 or 2 occurring is far higher than any other symbol, and this is quickly picked up by the adaptive arithmetic coder. However, the conditional probability of a 1 following a 2 (and the 3 other permutations of 1 and 2) is also generally useful and not simply random. We have found that by storing the conditional probabilities $P(1|2)$, $P(2|1)$, $P(2|2)$ and $P(1|1)$ and using these to modify the probability estimates we can improve compression performance by 1%-5%.

4 Results and Discussion

Table 2 presents our results for connectivity encoding. We have endeavoured to compare these accurately with other schemes, but in many cases the lack of freely available code implementations and inconsistent model sizes hampers comparison. Tests were run on a P4 3GHz machine with 512MB of memory. All the meshes are processed in memory. A kd-tree [Mount and Arya 1997] is used to accelerate closest points queries. This kd-tree implementation does not support incremental queries and the run times are thus somewhat higher than a properly optimised implementation would allow. As expected, the compression time scales linearly with the number of triangles. The header information required by the algorithm (the four prior probabilities, starting vertices for the initial triangle and a few flags) typically amounts to about 30 bytes for a single component mesh, and is not included in the compression cost. For every separate triangulated component, another 3 integers are required to specify the starting triangle.

Models	$\|V\|$	**Nopt VDr**	**TG**	**EB**	**AA**	**DR**	**Gain %**	T s
Regular								
armadillo	172974	1.65	1.83			**0.86**	48	18.9
bunny	34834	1.07	1.29			**0.61**	43	3.4
fandisk	6475	0.93	1.08			**0.69**	26	0.5
horse	48485	1.33	1.51	1.76		**0.66**	50	5.2
horse-lres	19851	**2.25**	2.34		1.35	**0.47**	65	1.8
rabbit	67039	1.47	1.66			**0.81**	45	6.3
mannequin	11704	**0.37**	0.46	1.2		0.38	-2	1.0
Irregular								
dinosaur	14070	2.25	2.39		1.69	**1.10**	35	1.4
feline	49864	2.20	2.38		1.50	2.12	-40	5.1
venus	50002	2.05	2.20			**1.98**	3	5.2
venus-lres	8268	**2.71**	2.82		1.95	2.52	-29	0.8
molecule	10028	1.80				**1.51**	16	1.0
blob	8036		1.70			**1.45**	15	0.8
nefertiti	299	**2.37**	2.83	2.42		**1.64**	20	0.0

Table 2: **Connectivity compression results.** Results for a number of popular connectivity coders are presented where possible — AA [Lee et al. 2002], EB [Rossignac 1999], Nopt [Khodakovsky et al. 2001], TG [Touma and Gotsman 1998] and VDr [Alliez and Desbrun 2001]. For Nopt/VDr we choose the best listed result; if one result is missing, we list the avilable data in bold print. The DR columns present our results. Results in bold font indicate where our scheme improves performance over the best reported results. The compression gain and average encode times are presented in the last two column.

A quick perusal of the results shows that the scheme performs very well on meshes which display regularity (figure 1a), 1b)), decreasing the code size by up to 65%. This is quite remarkable, given that we are generating *twice* the number of codes compared to valence schemes. It should be noted that the valences in these meshes may be arbitrary — it is the regularity of the triangulation that matters. In all cases the results are much lower than the valence entropy. The *mannequin* mesh has a relatively low valence entropy and is thus ideally suited to a valence-based coding.

For irregular meshes (those with a wide mix of different triangles types) the gains are somewhat smaller, and some of the other schemes perform better — figure 1c). This is particularly true when there are a large number of slivers in the mesh, since the algorithm is most likely to predict vertices in the region ahead of the current edge. This is the case for the *venus* and *feline* meshes which appear to have been randomly triangulated. The current predictors do not perform optimally for such meshes although they remain competitive with a number of other valence-based schemes. It should be noted, however, that one could develop a predictor tailored to meshes with a high proportion of slivers. In fact, one strength of our method is that one can continue to add predictors to deal with a host of different mesh types. The predictor specification then requires an additional integer in the header.

There are some "pathological" meshes (figure 1d) for example) on which the two predictors fare poorly. These meshes are distinguished by having a large number of thin triangles (slivers), which are closed by vertices lying some distance to either side of the base

edge. While the results are poor in such a case, it is worth noting that the usual interleaved geometry predictors will yield correspondingly poor vertex compression results. Consequently, the *joint* compression results for interleaved compression will also be poor. In contrast, a global vertex compression scheme such as [Devillers and Gandoin 2000] will be less dramatically affected. We would thus expect the joint compression results for the separate coder to be on a par with the interleaved approach in this case.

There is no useful theoretical bound on the compression performance of the scheme, unless one makes very restrictive assumptions about the underlying mesh structure. In fact, given the nature of the prediction scheme, it is possible, although highly unlikely, to generate a ranking code equal to the size of the vertex set. Of course, as vertices are culled, this maximum value will shrink. The lack of a theoretical bound does not, however, detract from the utility of this approach, as illustrated by the results.

5 Conclusion and Future Work

We have presented a simple geometry-driven approach for encoding the connectivity information of a triangle mesh. The technique is based on a prediction operation which establishes a distance ranking to connect each new triangle into the mesh during a breadth-first surface traversal. Our approach departs from the traditional interleaving of vertex and connectivity compression: we have access to the entire vertex data set prior to encoding and decoding of mesh connectivity. The availability of this global information allows us to construct good predictors which yield small ranking values and produce a connectivity code with very low entropy. Although face-based, the low entropies arising from the prediction scheme ensure that the technique remains competitive with valence-based schemes. For meshes with a regular structure, we consistently achieve results of less than 1 bpv and generally outperform the best results reported in the literature. Meshes with a highly irregular structure produce results that are on par with valence-based schemes, unless they are pathological cases.

There are a number of ways in which the scheme can be extended. Our predictors make limited use of global vertex data and a single predictor is chosen based on a global estimate. Some experimentation shows that a more compact code results if one can choose the predictor on a per face basis. One possibility is to use a learning technique to switch predictors as more of the mesh is processed. While we use a breadth-first surface traversal, this is simply a matter of convenience. A better strategy is to traverse the mesh according to some fitness metric which explores regular regions first, deferring the processing of irregular regions until later.

The algorithm can readily be generalised to handle tetrahedral meshes. Furthermore, the notion of distance rank coding can also be applied to general polygonal meshes, although in this case one requires a degree code for each face, in addition to the ranking codes required to insert each vertex.

References

ALLIEZ, P., AND DESBRUN, M. 2001. Valence-driven connectivity encoding for 3D meshes. In *EG 2001 Proceedings*, A. Chalmers and T.-M. Rhyne, Eds., vol. 20(3). Blackwell Publishing, 480–489.

ALLIEZ, P., AND GOTSMAN, C. 2003. Recent advances in compression of 3d meshes. In *Proceedings of the Symposium on Multiresolution in Geometric Modeling*.

ATTENE, M., FALCIDIENO, B., SPAGNUOLO, M., AND ROSSIGNAC, J., 2003. Swingwrapper: Retiling triangle meshes for better edgebreaker compression.

COORS, V., AND ROSSIGNAC, J. 2004. Delphi: geometry-based connectivity prediction in triangle mesh compression. *The Visual Computer*, 20, 1–14.

DEERING, M. 1995. Geometry compression. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, ACM Press, 13–20.

DEVILLERS, O., AND GANDOIN, P.-M. 2000. Geometric compression for interactive transmission. In *Proceedings of the conference on Visualization '00*, IEEE Computer Society Press, 319–326.

GANDOIN, P.-M., AND DEVILLERS, O. 2002. Progressive lossless compression of arbitrary simplicial complexes. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, 372–379.

KHODAKOVSKY, A., ALLIEZ, P., DESBRUN, M., AND SCHROEDER, P., 2001. Near-optimal connectivity encoding of 2-manifold polygon meshes.

LEE, E.-S., AND KO, H.-S. 2000. Vertex data compression for triangular meshes. In *Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, 225.

LEE, H., ALLIEZ, P., AND DESBRUN, M., 2002. Angle-analyzer: A triangle-quad mesh codec.

MOFFAT, A., NEAL, R. M., AND WITTEN, I. H. 1998. Arithmetic coding revisited. *ACM Trans. Inf. Syst. 16*, 3, 256–294.

MOUNT, D., AND ARYA, S., 1997. Ann: A library for approximate nearest neighbor searching.

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

ROSSIGNAC, J. 2001. 3d compression made simple: Edgebreaker with zip&wrap on a corner-table. In *Proceedings of the International Conference on Shape Modeling & Applications*, IEEE Computer Society, 278.

TAUBIN, G., AND ROSSIGNAC, J. 1998. Geometric compression through topological surgery. *ACM Transactions on Graphics 17*, 2, 84–115.

TOUMA, C., AND GOTSMAN, C. 1998. Triangle mesh compression. In *Proceedings of Graphics Interface*.