DOI: 10.1111/j.1467-8659.2007.01026.x

COMPUTER GRAPHICS *forum*
Volume 26 (2007), number 4 pp. 813–823

Distance-Ranked Connectivity Compression of Triangle Meshes

P. Marais, J. Gain and D. Shreiner

Collaborative Visual Computing Laboratory, Computer Science Department, University of Cape Town, Private Bag, Rondebosch, 7701 South Africa
patrick,jgain@cs.uct.ac.za; shreiner@siggraph.org

Abstract

We present a new, single-rate method for compressing the connectivity information of a connected 2-manifold triangle mesh with or without boundary. Traditional compression schemes interleave geometry and connectivity coding, and are thus typically unable to utilize 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 that 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 fashion, and using global vertex information, we can predict the position of the vertex that completes the unprocessed triangle attached to a given edge. We then rank the vertices in the neighborhood 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 set of rank values thus possesses low entropy and compresses very well. The sequence of rank values is all that is required to represent the mesh connectivity—no special split or merge codes are necessary.

Results indicate improvements over traditional valence-based schemes for more regular triangulations. Highly irregular triangulations or those containing a large number of slivers are not well modelled by our current set of predictors and may yield poorer connectivity compression rates than those provided by the best valence-based schemes.

Keywords: geometric compression, connectivity, triangle prediction

ACM CCS: I.3.m Computer Graphics: *connectivity coding*

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 fueled research into techniques that compress both the geometry (vertex positions) and the connectivity information of triangle meshes.

The earliest approaches [Dee95] exploited the adjacency structure of a triangle mesh to avoid storing unnecessary connectivity information and applied simple quantization and delta encoding to represent vertex positions. In general, the issue of connectivity compression is deemed more pressing, since, while it is possible to quantize 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 [TR98,Ros99] thus focused on ways to reduce the cost of encoding the mesh topology. Great strides have been made in this area. Edgebreaker [Ros99] 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 [Ros01] have refined this bound. The next great breakthrough arrived with 'valence-based' schemes [TG98], which use

Submitted December 2005
Revised December 2006
Accepted January 2007

the number of edges attached to a vertex to derive a compact coding algorithm that, 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 [KADS01]. Valence-based schemes generate 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 [TG98,KADS01,LAD02]. 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 information 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 [DG00,MMG06], 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 that 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 closest neighboring 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 neighbors 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 (as in Figure 1(a)), 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—the interested reader is referred to [AG03] for a summary.

We confine our discussion to single-rate compression connectivity schemes, since our connectivity compression technique falls into this category. Such schemes generally come in two flavors: face-based and valence-based. Face schemes are usually derivatives of *Edgebreaker* [Ros99, AFSR03], while valence schemes are modifications or extensions of Touma and Gotsman's valence-based encoder [TG98,KADS01,KPRW05].

Our approach assumes *separate* encoding for vertex and connectivity information, and that the entire quantized 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 [CR04] 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, less regular meshes tend to generate a large number of incorrect guesses, which incurs a substantial bit code penalty. Valence encoders will perform much better in such cases.

Another approach which uses geometry to drive the connectivity encoding is *Angle Analyzer* [LAD02]. 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. The recently developed *Freelence* technique of Külberer *et al.* [KPRW05] uses a clever traversal scheme to further reduce the cost of valence coding. There is also an associated geometry compression component that yields very impressive results. Unfortunately, the paper only presents geometry results based on manually optimized quantization parameters, and the generality of the scheme can thus not be easily established.

Inspired by their earlier success with vertex encoding [DG00], Gandoin and Devillers [GD02] 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. Subsequent work by Peng and Kuo [PK05] improved on these results by introducing a clever traversal of a standard octree, and combining this with a compact progressive encoding of mesh connectivity. While the results they achieve are good in relation to other progressive schemes, they lag behind single-rate compression approaches.

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

The area of point cloud compression has seen renewed activity with the increased interest in point-based

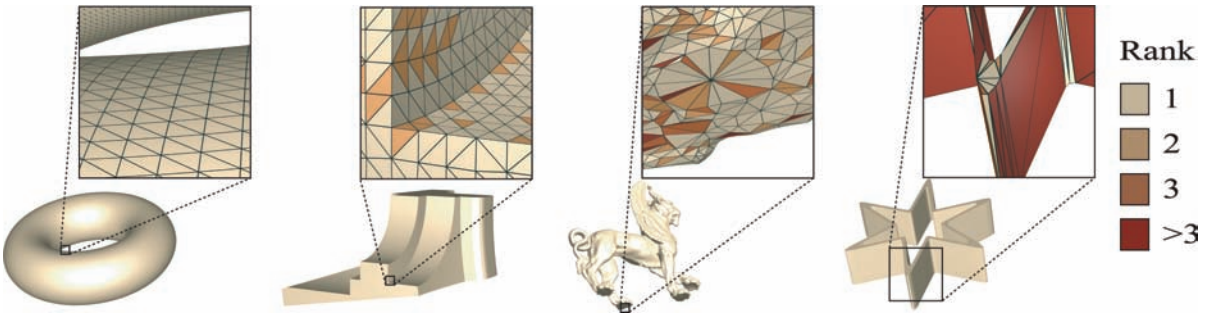

Figure 1: ***Distance-ranked compression***. *(a) highly regular, torus (0 bpv), (b) regular, fandisk (0.65 bpv) (c) irregular, feline (1.95 bpv)) (d) pathological, star (6.67 bpv). The colors indicate the distance rank code for each triangle.*

rendering. A number of point cloud compression schemes [LK00,OS04,FCOAS03] resample the input point set in some way,which renders them unsuitable for our purposes. The progressive scheme presented by [DG00] uses a space subdivision to achieve good lossless compression over a range of meshes. However, we prefer a single rate compressor, since we are only interested in lossless geometry representations. [TR98] introduced a single-rate vertex-spanning tree compressor. Unfortunately, this requires that partial connectivity information be available at decompression. Gumhold *et al.* [GKIS04] present a spanning tree representation for compressing points without connectivity information. A simple prediction scheme is used to reduce the size of the error correction term. Merry *et al.* [MMG06] also use a spanning tree but combine this with a more sophisticated prediction mechanism to achieve results comparable to triangle-based geometry predictors. Particularly good results are achieved for regularly sampled point sets such as those obtained from laser range scans. As an alternative to spanning tree techniques, Schnabel and Klein [SK06] have recently introduced an octree spatial decomposition that is similar in spirit to [PK05]. However, unlike the latter, this is a point compression scheme and does not use or require any connectivity information. While the results are rather sparse, they are encouraging and provide evidence that the compression of point geometry alone can be done effectively.

For those cases in which a set of points is triangulated using a specific algorithm, such as Delaunay triangulation, one can dispense with coding the connectivity altogether, provided one has a scheme to code the vertex locations. Similarly, if the triangulation deviates in only a small way from some canonical triangulation, one can encode the connectivity difference between the two. The work of Kim *et al.* [KPJC99] illustrates this. The authors encode the connectivity of a Triangular Irregular Network (a 2.5D surface mesh) using an average of 0.2 bpv. While such results are impressive, and outperform general triangle encoders, they are aimed at a very specific kind of mesh (which is very close to a Delaunay triangulation, in this case) and cannot be expected to perform well in general.

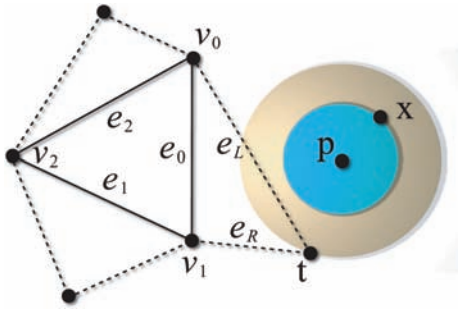

Figure 2: ***Distance-ranked prediction***. *Give the current processed triangle, (v_0, v_1, v_2), we estimate the closing vertex,* ***p***. *Vertex* ***t*** *is the point we are trying to find—this is the second closest vertex to* ***p***, *the closest being* ***x***. *We thus generate the code* ***2***,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

3. The Algorithm

Distance rank connectivity encoding involves only one fundamental operation: estimating the point that completes the triangle on the current edge. Figure 2 shows the steps involved in encoding an edge from a given base triangle. The encoding and decoding algorithms are presented in Table 1 and Table 2, respectively.

We choose an initial triangle on the surface and place the edges, using a consistent ordering, onto an edge queue. This queue is in fact double ended (a deque): although we always remove candidate edges from the front of the queue, we allow insertion on both ends. For each edge we wish to identify the vertex (from the set of all vertices) that completes the attached triangle. This is accomplished by using a prediction scheme based on parts of the mesh already visited *and* (potentially) the complete set of vertices. Given a predicted estimate for the closing vertex position, we rank the neighboring vertices based on their Euclidean distance from the predicted point and record the rank which

Table 1: *The encoding algorithm.*

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

Table 2: *The decoding algorithm.*

```
Build kd Tree //using encoded vertices
Q = empty
Read in start Triangle (V0, V1, V2)
Q.push_front: Edge(V0, V1), Edge(V1, V2), Edge(V2, V0)
Initialise edge counts
while Q ≠ empty
  E ← Q.pop_front
  (Vi, Vj) ← OrientedEdgeVertices(E)
  if (E is OPEN) // fetch an rank code
   K ← DECODE_SYMBOL
   if (K == 0) // boundary edge
     continue
   else
     P ← PredictPoint(V, ProcessedMesh)
     VTarrow KthClosestPoint(P, K, V)
     InsertTriangle(Vi, Vj, VT)
     Update edge counts
     if (Edge(Vi, VT) is OPEN)
       Q.push_front: Edge(Vi, VT) // Left
     if (Edge(VT, Vj) is OPEN)
       Q.push_back: Edge(VT, Vj) // Right
```

corresponds to the correct closing vertex. Identifying the closing triangle produces two new (consistently ordered) edges: *Left* and *Right*. Initially, we placed these new edges on the back of a regular queue, which yields a standard breadth-first (BF) traversal of the mesh triangles. However, a BF traversal tends to visit triangles in a nonlocal manner. Some preliminary experimentation showed that a better approach, which improves compression performance by 5.6% on average, is to try and process all the triangles attached to a local vertex before moving to more distant parts of the mesh. To this end, we adopt a traversal in which the *Left* edge is inserted at the front of the queue, while the *Right* edge is placed on the back of the queue.

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. Each edge has an associated edge count which indicates how many triangles are associated with that edge. Prior to encoding the mesh, we flag all boundary edges and set the edge counts to 0 on all edges except the starting three edges in the queue, which are assigned an edge count of 1. As each new triangle is added, the boundary flags and edge counts are used to ensure that only valid edges are placed on the queue. If a flagged (boundary) edge is added to the queue, it will not be expanded at a later stage—however, pushing such a redundant edge onto the queue allows us to simplify the encode/decode logic. The decode phase mirrors the encode phase—this is why we need to write out 0 codes for boundary edges: the decoder will not know which edges are simply 'open' (unprocessed) or boundary edges, so we have to provide this information in the encoded stream. It is important to emphasize that we only use information in the encoder which would be available to the decoder at that particular step.

By keeping track of the edge counts, we can also appropriately prune the edge traversal 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.

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 edge queue in the same way. For each edge, we compute the predicted point and search for the Kth nearest neighbor (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 edge 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 from future edge searches 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.

Figure 3 demonstrates the full algorithm for a small triangulation with boundary. The encoding and decoding phases are essentially mirror images of each other. The scheme

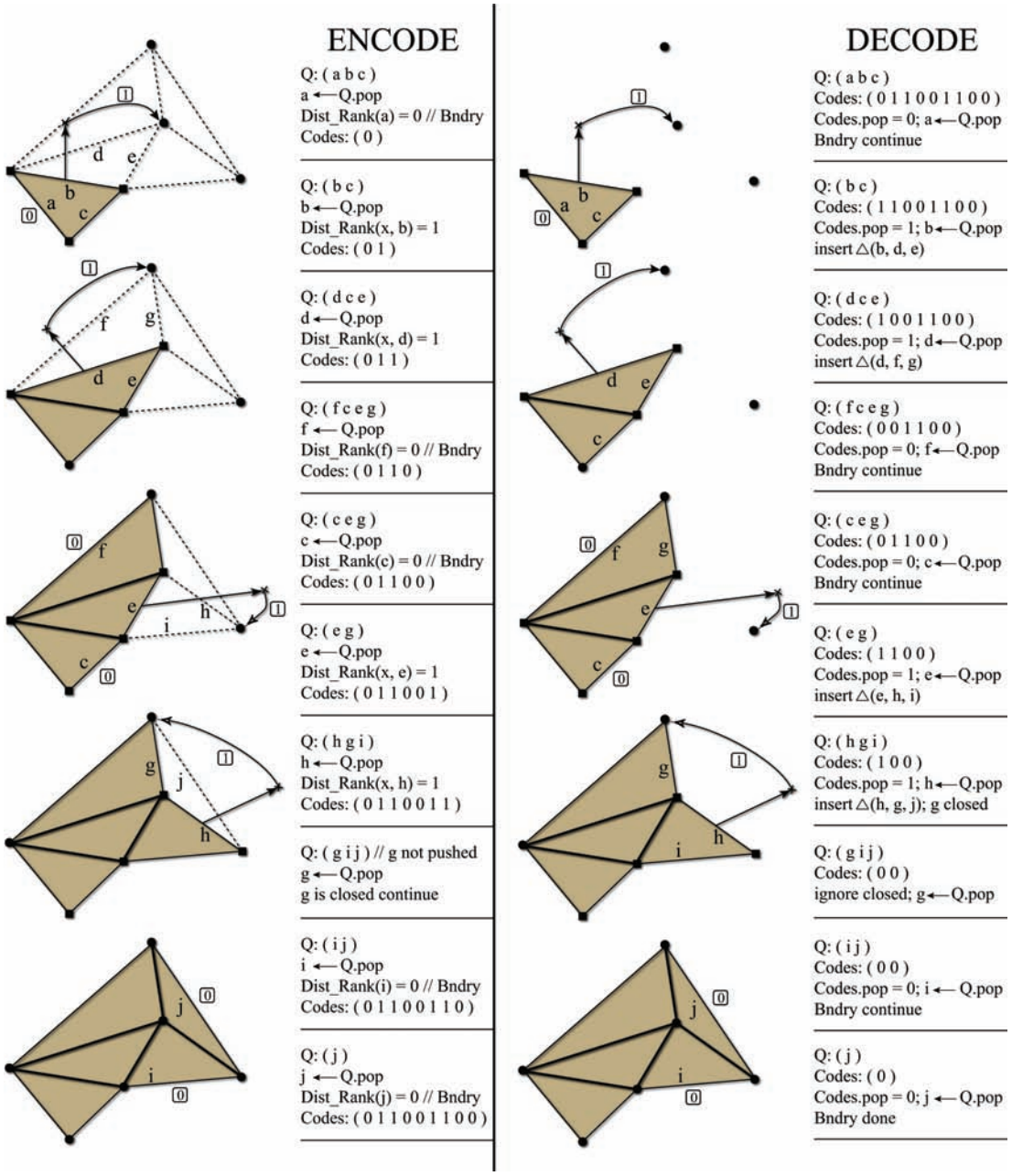

Figure 3: ***Encode****: We begin with the three edges a, b and c placed on the queue and proceed by always popping edges off the front of the queue. A vertex prediction is computed and the distance rank (the boxed number) is established. Each new closed triangle produces a left and right edge that are pushed onto the front and back, respectively, of the queue and edge counts are updated. A square (rather than circular) vertex indicates that the vertex will not be considered when calculating the distance rank. Boundary edges have a distance rank of 0. Edge counts are indicated by the thickness of the triangle line (thick lines have a count of 2) and the shaded triangles are those which are processed at each step.* ***Decode:*** *We start with the edges of the initial triangle, a, b and c. We then form a prediction and search for the Nth closest point, ignoring processed/flagged vertices, where N is given by the distance rank code. A closing vertex creates a new triangle which is inserted into the mesh, triggering updates of the appropriate edge counts. The codes are examined before the edge queue; if an edge is determined to be closed (either by a 0 code or because its edge count is 2), we proceed to the next edge immediately.*

automatically deals with holes and boundaries, but cannot deal with nonmanifold triangulations.

3.1. Numerical issues

We assume that the input mesh vertices are quantized in the standard way:

1. a bounding box (which encloses the vertex set) is computed;
2. a fixed number of bits (usually 12) is used to quantize each coordinate;
3. no two vertices share the same quantized coordinates. If this is not true, the quantization can either be refined, or the degenerate triangles or edges can be removed in a preprocess step.

When computing distance rankings, it is possible that several points may be located at the same distance from a given query point. To deal with this problem, we use the vertex *indices* to disambiguate our selection. We assume that the vertex compression scheme will preserve the vertex ordering, so that both the encoding and decoding steps will have consistently numbered vertices. If this is not the case, a simple lexicographical sort on the vertex coordinates can be used to reorder the data in a consistent manner. We first order the points that map to unique distances using their computed distance values. Then, we insert the points with duplicate distances into this sequence, at the appropriate distance value, ordered by their unique vertex indices. This has the effect of increasing the range of the rank values since there are more distance values to consider and the matching point position may be further down in the sequence. Fortunately, for a sensible choice of predictor (cf. Section 3.2), this does not happen very often. To avoid issues with floating point precision in our distance calculations, we perform all calculations using double-precision floating-point arithmetic and then cast the results to IEEE754 single precision floats before making comparisons.

3.2. 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: *triangle regularity*, *surface curvature* and *triangle size*.

1. *Triangle Regularity* If the triangulation is very regular (triangles are similar), then a simple parallelogram rule, as used in [TG98], can be employed to predict the closing vertex, using the three 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 along a ray perpendicular to the midpoint of the base edge such that, initially at least, an equilateral triangle is formed. For a highly regular mesh, a prediction based on a parallelogram suffices, otherwise we use the midpoint predictor.

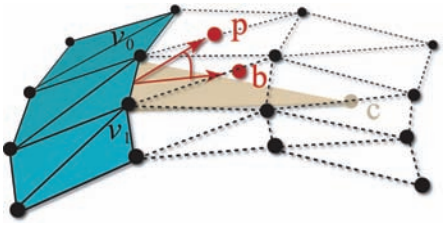

Figure 4: ***Correcting for surface curvature***. *Given the prediction edge* (v_0, v_1)*, we predict the closing vertex as* ***p***. *We search in a neighborhood about* ***p*** *to find a small number of mesh vertices within a fixed radius, and compute the centroid,* ***c****, of this set. In many cases, the centroid will lie close to the underlying mesh surface, and the set of points* (v_0, v_1, c) *thus describe a crude planar approximation the local triangle mesh surface. To compute the final predicted point,* ***b****, we rotate* ***p*** *about* (v_0, v_1) *onto the plane spanned by* (v_0, v_1, c).

2. *Surface Curvature* 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 an averaging plane for the upcoming surface, and then bending the prediction onto this plane, we can largely overcome this problem—see Figure 4. Of course, the position of the centroid will depend on the number of points chosen. In order to keep the prediction local we limit the neighborhood to a maximum of 10 points lying no further away than four times the current edge length.

   We also implemented higher order surface fitting along the same lines as [GA03], by fitting a quadric surface to the upcoming points and then intersecting it with a tangential circle centered on the current edge to arrive at a prediction, but the results were unpredictable (ranging from a 3.8% improvement to a 2.2% degradation, with an average improvement of only 0.32%) and did not justify the computational expense.
3. *Triangle Size* The final ingredient for a good prediction scheme is the ability to deal with rapidly changing triangle 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

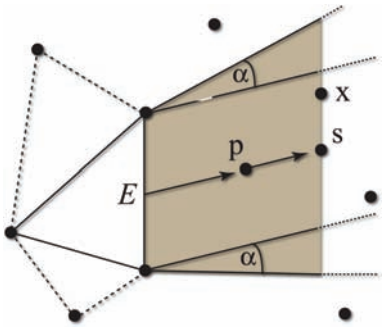

Figure 5: ***Triangle size changes.*** *We predict our closing vertex* ***p****. Then, we search for the first vertex that lies in the region ahead of E, bounded by two planes parallel to the prediction direction. In this case, we find the vertex* ***x****. We then scale the prediction in this direction 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.*

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' centered on the base prediction edge. (See Figure 5). While this is not always optimal, it tends to deal well with abrupt transitions in triangle size and has a significant impact on compression (with an average improvement of 50% and up to 70% on meshes with abrupt size changes).

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 behavior 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 toward small values (see Table 3). We use a standard *adaptive arithmetic coder* [MNW98] 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 three 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

To allow for comparison with other triangle mesh compression schemes, and since there is no standard test corpus, we have used the same models where possible (see Figure 6). Model sizes range from 300 to 170K vertices, and the test corpus contains CAD models, scanned models and simplified models. Only connectivity and geometry are compressed: we do not address the issue of compressing other mesh attributes. Model vertices are quantized to 12 bits per coordinate, the value that most other authors use.

While there is currently no point-cloud compressor that provides universally superior results, we have opted to use the compressor of Merry *et al.* [MMG06] since this performs well on meshes without a large number of slivers or widely varying triangle sizes. Such triangulations are generally avoided in computer graphics since they can lead to bad shading artifacts. It is likely that other point-cloud compressors (e.g., [GKIS04,SK06]) would yield better results on irregular meshes. However, since this paper primarily addresses connectivity compression, and the issue of point-cloud compression is addressed by others, we provide only a small set of illustrative geometry compression results (Table 5). This shows that the overall compression results for regular models compare well with baseline valence schemes. Given the growing interest in point-based representations, we anticipate a significant amount of future research in the area of point-cloud compression. Since we have decoupled the connectivity and geometry encoding, our technique will benefit immediately from any future improvements in this area.

Tests were run on a P4 3 GHz machine with 512 MB of memory and all meshes were processed in memory. A kd-tree [MA97] is used to accelerate nearest neighbor queries. This kd-tree implementation does not support incremental queries. Since each new point query is independent of its predecessors, the system run times are significantly higher than a properly optimized implementation would allow.

Table 4 presents our results for connectivity encoding.

The encode and decode times differ by only a few percentage; we have thus presented the average encode/decode time

Table 3: ***Histogram of arithmetic compression codes.*** *This table shows the distribution of symbols submitted for arithmetic compression. A 0 code is only output on open edges and the only open models are bunny and nefertiti. Note the significant clustering of first and second closest point symbols (in the 1 and 2 columns). Symbols with an occurrence less than 0.001 have been omited.*

Model	0	1	2	3	4	5	6	7	8	9	10
armadillo		0.932	0.066	0.001							
blob		0.872	0.114	0.009	0.003	0.001	0.001				
bunny	0.003	0.956	0.039	0.001							
dinosaur		0.913	0.070	0.011	0.003	0.001	0.001				
fandisk		0.943	0.056	0.001							
feline		0.804	0.145	0.031	0.010	0.005	0.002	0.001	0.001	0.001	
horse		0.947	0.048	0.003	0.001						
horse-lres		0.970	0.026	0.002	0.001						
nefertiti	0.057	0.861	0.056	0.019		0.003	0.003				0.002
rabbit		0.928	0.068	0.003	0.001						
venus		0.828	0.124	0.029	0.010	0.004	0.002	0.001	0.001		
venus-lres		0.772	0.146	0.045	0.020	0.008	0.004	0.002	0.002		

Table 4: ***Connectivity compression results.*** *Results for a number of popular connectivity coders are presented where possible—AA [LAD02], FL [KPRW05], Nopt [KADS01], TG [TG98] and VDr [AD01]. For Nopt/VDr we choose the better of the two listed results; if one result is missing, we list the available 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 (as a percentage) and average encode/decode times are presented in the last two columns.*

Model	$\|V\|$	Nopt VDr	TG	FL	AA	DR	Gain %	Time s
armadillo	172974	1.65	1.83			**0.74**	55	18.9
blob	8036		1.70			**1.28**	25	0.8
bunny	34834	1.07	1.29			**0.59**	45	3.4
dinosaur	14070	2.25	2.39	1.44	1.69	**1.04**	28	1.4
fandisk	6475	0.93	1.08	0.74		**0.65**	12	0.5
feline	49864	2.20	2.38	1.23	1.50	1.95	−58	5.1
horse	48485	1.33	1.51			**0.70**	47	5.2
horse-lres	19851	**2.25**	2.34	0.96	1.35	**0.44**	54	1.8
nefertiti	299	**2.37**	2.83	2.42		**1.66**	30	0.0
rabbit	67039	1.47	1.66			**0.80**	45	6.3
venus	50002	2.05	2.20	1.73		1.79	−3	5.2
venus-lres	8268	**2.71**	2.82		1.95	2.30	−17	0.8

for each mesh to avoid further clutter in Table 4. As expected, the (de)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 three integers are required to specify the starting triangle.

The predictor selection scheme outlined in Section 3.2 was used to automate the selection of the appropriate predictor rule for each triangle. The 'midpoint' predictor was used most frequently, which demonstrates that a simple parallelogram rule is often a poor predictor of mesh geometry.

A quick perusal of the results shows that the scheme performs very well on meshes that display regularity (Figure 1a, b), generally reducing the connectivity information to less than 1 bpv. 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.

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 arbitrarily 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 this case, models such as these are not very common since most triangulations of surfaces tend to yield triangulations with some degree of regularity.

There is no useful theoretical bound on the compression performance of the scheme, unless one makes very restrictive assumptions about the underlying mesh structure.

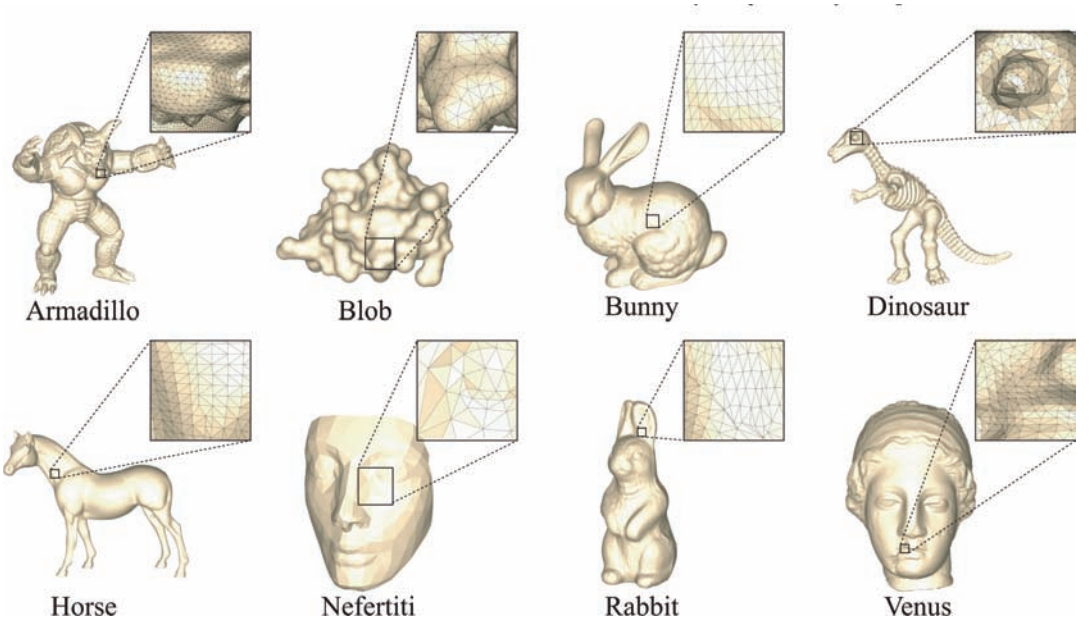

Figure 6: ***Model test set.*** *The models depicted here are commonly used in benchmarking compression performance. Some models are ommitted: fandisk and feline appear in Figure 1 and venus-lres and horse-lres are simply lower-resolution versions of the venus and horse models.*

Table 5: ***Combined compression results.*** *We have used the point cloud compressor of Merry et al. [MMG06] to produce a combined geometry and connectivity bit rate. This compressor is designed for point clouds with regular structure and thus works well for regularly distributed triangle mesh vertices. For comparison, we have presented the standard [TG98] valence encoder with results for both connectivity and geometry. It can be seen that the proposed scheme performs well for regular models, easily outperforming the reference scheme. All geometry was quantized to 12-bit accuracy.*

Model	TG Conn	TG Geom	TG Tot	DR Conn	DR Geom	DR Tot
armadillo	1.83	12.25	14.08	0.74	11.84	12.58
bunny	1.29	13.62	14.91	0.59	11.68	12.27
fandisk	1.08	14.84	15.92	0.65	12.79	13.44
horse	1.51	12.63	14.14	0.70	11.09	11.79
rabbit	1.66	12.46	14.12	0.80	9.86	10.66

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 dataset prior to encoding and decoding of mesh connectivity. The availability of this global information allows us to construct good predictors that 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 irregular triangles produce results that are on par with most valence-based schemes, although they fall short of the best reported results.

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. The current mesh traversal strategy does not utilize information from the processed mesh regions; a better strategy may be to traverse the mesh according to some fitness metric that explores regular regions first, deferring the processing of irregular regions until later.

The algorithm can readily be generalized 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.

Acknowledgements

This research was supported by a National Research Foundation Grant (No: 2053416). We hereby acknowledge the various institutions and research groups which hold copyright over the test corpus we used.

References

[AD01] ALLIEZ P., DESBRUN M.: Valence-driven connectivity encoding for 3D meshes. In *EG 2001 Proceedings*, CHALMERS A., RHYNE T.-M., (Eds.), *20*, 3. Blackwell Publishing, 2001, 480–489.

[AFSR03] ATTENE M., FALCIDIENO B., SPAGNUOLO M., ROSSIGNAC J.: Swingwrapper: Retiling triangle meshes for better edgebreaker compression, 2003.

[AG03] ALLIEZ P., GOTSMAN C.: Recent advances in compression of 3D meshes. In *Proceedings of the Symposium on Multiresolution in Geometric Modeling* (2003).

[CR04] COORS V., ROSSIGNAC J.: Delphi: geometry-based connectivity prediction in triangle mesh compression. *The Visual Computer 20* (2004), 1–14.

[Dee95] DEERING M.: Geometry compression. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (1995), ACM Press, 13–20.

[DG00] DEVILLERS O., GANDOIN P.-M.: Geometric compression for interactive transmission. In *Proceedings of the conference on Visualization '00* (2000), IEEE Computer Society Press, 319–326.

[FCOAS03] FLEISHMAN S., COHEN-OR D., ALEXA M., SILVA C. T.: Progressive point set surfaces. *ACM Trans. Graph. 22*, 4 (2003), 997–1011.

[GA03] GUMHOLD S., AMJOUN R.: Higher order prediction for geometry compression. In *Shape Modeling International* (2003), IEEE Press, 59–66.

[GD02] GANDOIN P.-M., DEVILLERS O.: Progressive lossless compression of arbitrary simplicial complexes. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), ACM Press, 372–379.

[GKIS04] GUMHOLD S., KARNI Z., ISENBURG M., SEIDEL H.-P.: Predictive point-cloud compression. In *ACM SIGGRAPH Conference Abstracts and Applications* (2004).

[KADS01] KHODAKOVSKY A., ALLIEZ P., DESBRUN M., SCHROEDER P.: Near-optimal connectivity encoding of 2-manifold polygon meshes, 2001.

[KPJC99] KIM Y.-S., PARK D.-G., JUNG H.-Y., CHO H.-G.: An improved tin compression using delaunay triangulation. In *PG '99: Proceedings of the 7th Pacific Conference on Computer Graphics and Applications*. Washington, DC (1999), IEEE Computer Society, 118.

[KPRW05] KAELBERER F., POLTHIER K., REITEBUCH U., WARDETZKY M.: Freelence—coding with free valences. In *Eurographics* (2005).

[LAD02] LEE H., ALLIEZ P., DESBRUN M.: Angle-analyzer: A triangle-quad mesh codec, 2002.

[LK00] LEE E.-S., KO H.-S.: Vertex data compression for triangular meshes. In *Proceedings of the 8th Pacific Conference on Computer Graphics and Applications* (2000), IEEE Computer Society, 225.

[MA97] MOUNT D., ARYA S.: Ann: A library for approximate nearest neighbor searching, 1997.

[MMG06] MERRY B., MARAIS P., GAIN J.: Compression of dense and regular point clouds. In *Proceedings of the 4th international conference on Computer Graphics, Virtual Reality, Vizualisation and Interaction in Africa* (2006), ACM Press.

[MNW98] MOFFAT A., NEAL R. M., WITTEN I. H.: Arithmetic coding revisited. *ACM Trans. Inf. Syst. 16*, 3 (1998), 256–294.

[OS04] OCHOTTA T., SAUPE D.: Compression of point-based 3D models by shape-adaptive wavelet coding of multi-height fields. In *Proceedings of Symposium on Point-Based Graphics 2004* (Zürich, June 2004), Eurographics.

[PK05] PENG J., KUO C.-C. J.: Geometry-guided progressive lossless 3d mesh coding with octree (ot) decomposition. *ACM Trans. Graph. 24*, 3 (2005), 609–616.

[Ros99] ROSSIGNAC J.: Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics 5*, 1 (1999), 47–61.

[Ros01] ROSSIGNAC J.: 3D compression made simple: Edgebreaker with zip&wrap on a corner-table. In *Proceedings of the International Conference on Shape Modeling & Applications* (2001), IEEE Computer Society, 278.

[SK06] SCHNABEL R., KLEIN R.: Octree-based point-cloud compression. In *Proceedings of Symposium on Point-Based Graphics 2006* (July 2006), BOTSCH M., CHEN B., (Eds.), Eurographics.

[TG98] TOUMA C., GOTSMAN C.: Triangle mesh compression. In *Proceedings of Graphics Interface* (1998).

[TR98] TAUBIN G., ROSSIGNAC J.: Geometric compression through topological surgery. *ACM Transactions on Graphics 17*, 2 (1998), 84–115.