Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

FELIX KÄLBERER, KONRAD POLTHIER, ULRICH REITEBUCH, MAX WARDETZKY

Compressing Triangle Meshes using Geometric Information

ZIB-Report 04-26 (July 2004)

FreeLence: Compressing Triangle Meshes using Geometric Information

Felix Kälberer* Zuse Institute Berlin
Konrad Polthier† Zuse Institute Berlin
Ulrich Reitebuch‡ Zuse Institute Berlin
Max Wardetzky§ Zuse Institute Berlin

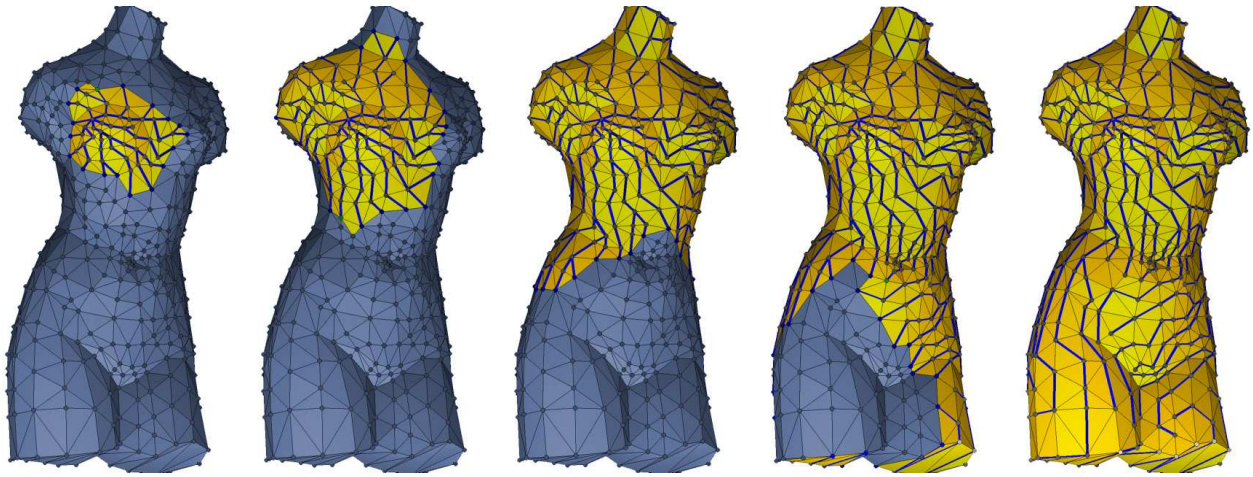

Figure 1: Several stages of the FreeLence encoding process of the Venus torso mesh. The blue vertex tree connects each vertex with its free valences, i.e. the number of incident unprocessed vertices.

Abstract

We introduce FreeLence, a lossless single-rate connectivity compression algorithm for triangle surface meshes. Based upon a geometry-driven traversal scheme we present two novel and simple concepts: free-valence connectivity encoding and entropy coding based on geometric context. Together these techniques yield significantly smaller rates for connectivity compression than current state of the art approaches - valence-based algorithms and Angle-Analyzer, with an average of 36% improvement over the former and an average of 18% over the latter on benchmark 3D models, combined with the ability to well adapt to the regularity of meshes. We also prove that our algorithm exhibits a smaller worst case entropy for a class of "well-behaved" triangle meshes than valence-driven connectivity encoding approaches.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations.

Keywords: Mesh Compression, Connectivity Encoding, Polygonal Meshes, Context-based Entropy Coding.

1 Introduction

Competitive compression of 3D geometric models stays an ever-growing demand for a wide range of contemporary industrial applications. While resolution and size of such models are constantly growing, it remains a challenge to reduce the amount of physical storage they absorb. For geometry streaming over the Internet, for online rendering and shared-network applications where bandwidth is at stake, such a reduction is crucial. At the same time, often no loss of the original high resolution data can be afforded. Hence reduction means the removal of redundant information while keeping the indispensable parts. This is where lossless connectivity encoding jumps in.

Over the last few years lossless connectivity encoding has been a very active area of research which led to major advances in compression technology. Here we present FreeLence, a new and simple to implement compression scheme, which leads to serious practical and theoretical advances. We improve the bit rates over those of the best connectivity coders known to date, namely Angle-Analyzer by Lee, Alliez and Desbrun [Lee et al. 2002] and valence-based techniques as pioneered by Touma and Gotsman [Touma and Gotsman 1998] and improved by many others with an average of 18% over the former and 36% over the latter. We combine the desirable features of these existing approaches while improving some of their

*e-mail:kaelberer@zib.de
†e-mail:polthier@zib.de
‡e-mail:reitebuch@zib.de
§e-mail:wardetzky@zib.de

drawbacks. Most notably, by using geometric information to predict combinatorial information we remove redundancies inherent to valence-based schemes, where at the same time our approach has the ability to adapt to highly regular meshes, making it suitable for high-resolution uniform models. Additionally, building on a theoretical analysis we provide evidence that valence-based techniques may not be 'near-optimal' by improving their worst case by 11%.

Our high compression rates are based on two main novel technical details:

1.) our encoding relies on the number of *free-valences* at the currently processed vertex: instead of storing the number of all valences of each vertex, as in valence-based encoding schemes, we only store the number of free valences, i.e. the number of unprocessed incident vertices of a vertex as it is encountered in the process of conquering the mesh. This significantly reduces the variance in the dispersion of stored numbers and hence the entropy of the code sequence.

2.) the use of *context-based* arithmetic coding with contexts based on geometric properties of the mesh. Passing *geometric hints* to the entropy coder increases the compression ratios by an average of 20% on our test models.

1.1 Previous Work

Starting with Deering's pioneering work [Deering 1995] *geometry compression* for 3D meshes has undergone rapid and exciting developments. Early work on triangle mesh compression includes [Taubin and Rossignac 1998]. In general, geometry compression can be said to divide into two main directions - compressing the connectivity, or combinatorics, of the mesh and compressing its geometric information, for instance, through quantization. Each of these directions has several sub-branches. There are single-rate and progressive schemes, re-meshing and spectral decomposition approaches and geometry quantization techniques in position and angle space. A thorough survey of recent techniques and directions can be found in [Alliez and Gotsman 2003]. Without thriving for completeness, here we only sketch the work on single-rate connectivity encoding which is relevant to our approach. The techniques we describe here all rely on a common pattern: from a chosen starting point, the mesh is conquered by a growing disk. If in this process the disk touches upon itself its boundary *splits* in two. If later two of these boundaries meet again, they *merge*.

The *Edgebreaker* algorithm [Rossignac 1999] encodes the connectivity of a mesh using a fixed traversal scheme which conquers one triangle at a time by crossing edges, or *gates*. It has a provable worst case bit rate of 4 bits per vertex (bpv) for lossless storage of connectivity. Edgebreaker underwent several improvements as in [King and Rossignac 1999] where a bit rate of 3.67 bpv can be guaranteed. An interesting derivative of Edgebreaker is the Cut-Border-Machine [Gumhold and Strasser 1998] and its improvements [Gumhold 1999] with a provable worst case bit rate of 3.552 bpv.

A different branch of this development consists of *valence-based* techniques [Touma and Gotsman 1998] and its derivatives like [Alliez and Desbrun 2001]. Here the vertices of the mesh are conquered instead of the faces. The conquering scheme depends only on the combinatorics of the mesh, not its geometry. These techniques benefit from an increasing regularity of a mesh when the dispersion of valences concentrates more around the average of 6 edges per vertex. The original Touma-Gotsman coder practically achieves bit rates between 2 and 3.5 bpv. One of its attractive aspects is the asymptotic limit of 0 bpv for perfectly regular meshes. Besides the *valence part* of the code, special "split" symbols need to be stored, too. Whereas the valence part of the code can be shown to lie below the Tutte information limit of 3.2452, compare [Gotsman 2003], it is the "split" part which still escapes a thorough analysis and is in fact critical for the final code length.

The interplay between geometry and connectivity has been investigated by several authors. It is shown in [Isenburg et al. 2001] that a big amount of geometric information is contained in the connectivity of a mesh. But since the geometric information is also completely contained in the vertex positions that are saved along with the connectivity, some of this information is redundant. There are two natural ways to remove this redundancy: using connectivity information to improve geometry coding or using geometry information to improve connectivity coding. The first approach was investigated in [Sorkine et al. 2003] and [Ben-Chen and Gotsman 2003]. Signal processing techniques on the connectivity graph were used to achieve high compression ratios for vertex positions. The second approach was recently taken by [Coors and Rossignac 2003] who use a geometry-based connectivity prediction to obtain a serious improvement compared to the Edgebreaker encoder. It also gives Edgereaker the ability to adapt to mesh regularity, but on irregular meshes bit rates are above those of the best valence-based coders. In Angle-Analyzer, [Lee et al. 2002] use a significant amount of geometric information to improve connectivity compression ratios. Its symbol sequence is a derivative of the Edgebreaker "gate" approach. Angle-Analyzer gives the best bit rates known to date of about 2 to 2.5 bpv in practice. However, one of its drawbacks is that it cannot benefit as much from highly regular meshes as valence-based approaches.

Apart from the practical approaches there is a high theoretical interest in guaranteeing worst case bit rates, see [Gotsman 2003] or [Khodakovsky et al. 2002]. In [Poulalhon and Schaeffer 2003] a coder is introduced which works at the Tutte census [Tutte 1963], so that it is provably optimal in the theoretical sense. However, for practical purposes this is not the case as it does not adapt to mesh regularity - the code length is equal for *all* meshes with the same number of vertices.

1.2 Overview

In this paper we introduce the FreeLence coder which uses a geometry-driven traversal scheme and predicts *free valences* from local geometric information. Popular valence-based coders store the full number of valences at each vertex, with an average of 6 valences per vertex. This is equivalent to storing the combinatorial Gauß curvature of the edge graph of the mesh. In contrast, FreeLence only stores the number of free valences and thereby benefits from a *very low statistical dispersion of the free valences* being about 1 free valence per focus vertex.

Additionally, a context-based arithmetic entropy coder is used for storing the free valences. Into which context a concrete free valence enters, depends on the opening angle of the focus to which this free valence belongs. Opening angles are used to give the coder a hint what the most suitable context for storage might be. Entropy coders greatly benefit from such hints.

Also, we challenge the assumption that valence-based approaches are 'near-optimal' by proving that *free-valence* techniques yield better worst case bit rates for the valence part of the code, while at the same time having the ability to use exactly the same traversal scheme, and hence having the same occurrences of split symbols as the Touma-Gotsman algorithm. Note, that the actual number of splits appearing in FreeLence is significantly below the Touma-Gotsman rate because of the usage of a geometry-driven scheme.

The paper is organized as follows: Section 2 introduces data structures and explains the FreeLence coder. Section 3 goes into a detailed discussion and provides experimental evidence how our coder improves over other techniques. A theoretical analysis of worst case bit rates for our coder is given in the appendix of this paper.

2 The FreeLence Algorithm

We consider lossless compression of the combinatorial mesh of triangulated orientable topological 2-manifolds of arbitrary genus and possibly with boundary.

The general scheme of FreeLence follows that of other connectivity coders: it starts at an arbitrarily chosen inner vertex of the mesh and keeps growing bigger and bigger disks around it until the mesh is fully conquered. If a disk touches itself, its boundary *splits* into two components. If later two boundaries meet again, they *merge*.

At the heart of FreeLence is the idea to *exploit geometric information* about the mesh in a *twofold* way: first of all, it governs the way of *how to grow disks*, see Section 2.2, and secondly, it governs the way *to store the number of free valences* of each focus in separate tables depending on the opening angle of the focus, see Section 2.3.

2.1 Definitions and Related Data Structures

Here we introduce relevant terminology and data structures, sticking to common definitions as closely as possible.

Active List (or cut border): A closed and oriented edge path in the mesh. It is represented as a doubly linked list of vertices. It separates the mesh locally in an inner region of conquered edges and an outer part containing free edges. There may be several active lists at a time; however, we only use a single *priority queue* to store all vertices belonging to some active list.

Opening Angle: Assigned to each vertex in an active list. Except for special cases, "joins", its value is $360^\circ - \sum \angle(e_i, e_{i+1})$, the sum being taken over all angles between neighboring conquered edges incident to the vertex, measured in degrees. The opening angles constitute the *keys* of the priority queue of active vertices.

Focus (or pivot): The vertex in an active list which is currently processed by the encoder or decoder. The focus is chosen to be the minimum in the priority queue of active vertices.

Free Vertex: A vertex of the mesh which has not been previously visited by the encoder.

Free Edge: An edge not previously conquered.

Free Valence: The number of free vertices directly connected to the current focus vertex by an edge.

2.2 Geometry-driven coding with free valences

Conquering the mesh is based on increasingly growing disks. For this growing process, FreeLence uses a simple *geometric rule*: at each step the algorithm defines a *focus* as the vertex with the smallest opening angle in the active lists. The number of *free valences* of the focus is stored in the code sequence as an intermediate stage of mesh compression. Then the free triangles around the focus are conquered (thereby growing the disk), and a new focus is determined until all vertices of the mesh have been processed. For triangle meshes, since for each free valence exactly one vertex is created, the sum of all stored numbers in this code sequence equals the number of vertices of the mesh (up to "join" events).

2.3 Context-based arithmetic coding - predicting combinatorics from geometry

The free valences of the code sequence are stored in separate tables. Each table corresponds to an interval of a dissection of angle space. Which table to choose from is governed by the opening angle of the focus. Small angles usually indicate a smaller number of free edges, and bigger angles a bigger number. Therefore, the numbers in a single table exhibit only a small variation - a strong advantage for an entropy coder. On average, for each $i = 1..6$, an opening angle of $i \cdot 60^\circ$ corresponds to $(i-1)$ free edges. However, a concrete mesh may heavily deviate from this pattern, see Figure 2.3.

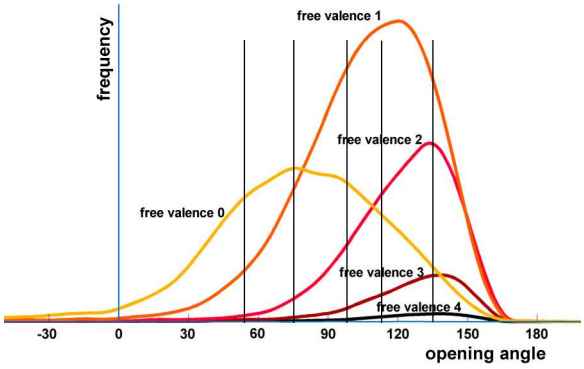

Figure 2: Distribution of opening angles for an encoding run of the David model. Each free valence has its own distribution. Note how the curves for different values of free valences overlap due to the irregularity of the mesh. Negative opening angles can occur at vertices with negative Gauß curvature and at boundary vertices. The vertical lines show the optimized thresholds by which the free valence codes are grouped into contexts.

Therefore, we further adjust each context table *according to the geometry of a concrete mesh*. That means, the encoder first stores for each value of free valences the distribution of the opening angles over this value. By using a greedy heuristic, the dissection of angle space is then optimized in such a way, that the sum over all entropies in the tables becomes minimal. The vertical lines in Figure 2.3 show such a dissection for the David mesh. Each table is now compressed separately with an order-0 entropy coder [Witten et al. 1987].

In other words, one can look at the categorization of free valences into different tables as an ordering according to the discrete geometric Gauß curvature. As valences correspond to combinatorial Gauß curvature this can be understood as prediction of combinatorics from geometry.

2.4 Detailed Description

The FreeLence algorithm operates on two main data structures - a single *priority queue* where all active vertices are sorted by their

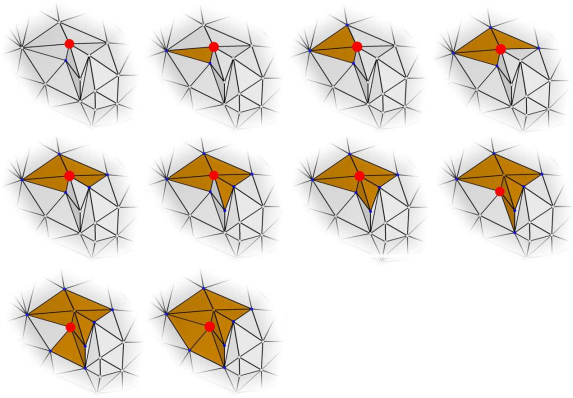

Figure 3: A sample run of the first steps of the encoder. A starting vertex is chosen with 6 free valences. The number 6 is stored in the code and all incident triangles to the vertex are conquered. Then a new focus is chosen according to the minimal opening angle in the active list. This focus has 2 free valences, so a 2 is written and all free triangles incident to the focus are conquered.

opening angles, and one ore more doubly linked lists of these vertices, the *active lists*. . The algorithm uses a finite set of *symbols*, **P** "move to previous", **N** "move to next", **J** "join", **D** "dummy", and a set of *numbers* which specify "free valence" and "distance". The algorithm works as a finite state machine.

Storing free valences. We give a description for encoding a triangle mesh without boundary (the handling of boundaries is described later in this section). The algorithm starts by picking an arbitrary vertex v, the first *focus*. All edges of v are free so v gets assigned an outer angle of 360°, and the number of free edges is stored in the connectivity code. Then the triangles in the vertex star of v are closed in clockwise order, and the vertices of the boundary of the vertex star of v constitute the first active list. All the vertices in an active list are doubly linked to each other according to their previous and next neighbors. Also, to each vertex in the active list an *opening angle* is assigned, equaling 360° minus its conquered inner angles.

A new focus is repeatedly chosen according to the minimal opening angle among all vertices in active lists. As all vertices in active lists reside in a single priority queue, the focus may jump among different lists. The number of free valences of the focus is then stored and the open triangles around it are closed. Now the focus is considered processed, and is removed from the active list and the priority queue. We keep repeating this scheme until we hit a focus which has a free edge to a previously visited vertex; we then need to store a special *symbol*.

Local and global splits - P, N and J. For a spherical geometry a previously visited vertex which is connected to the focus by a free edge, must lie in the same active list as the focus itself. This event is called a *split*. Splits are a critical part for an entropy coder as they destroy the uniformity of the sequence. In practice, splits take up 0.1% to 5% of all symbols. Most of these splits happen *locally*, namely when the successor or predecessor vertex of the current focus has no free edges. In this case, a single triangle gets inserted without adding new vertices. We only store the symbol **P** or **N** depending on whether the predecessor or successor to the focus has no free edges left. The previous or next vertex can then be conquered without storing another number, because the number of free edges around it is known to be zero. The queue of active vertices is then updated according to the new opening angles.

For each focus we keep conquering its free edges in clockwise order. If a free edge is hit which connects the focus to a previously visited vertex, and this vertex is not the previous or next to the focus, we have encountered a *join*, see Figure 2.4. Most joins correspond to splits, i.e. connect the focus to a vertex in the same active list. For manifolds with handles there is another kind of joins - where the focus connects to a vertex in another active list. Those are commonly called *merges*. The number of merges equals the number of handles of the mesh. When a join is encountered, a symbol **J** together with the number of free edges in clockwise order to the join vertex is stored. A third number is needed by the decoder to identify the join vertex in the queue. Therefore, the vertices in the priority queue are sorted according to their Euclidian distance from the focus. The position in this ordering of the join vertex to the focus is stored as a third number in a separately encoded *offset list*.

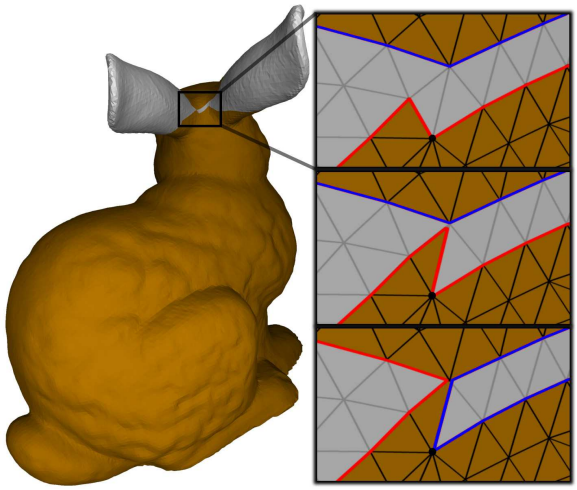

Figure 4: A sample run for Bunny. The zoom sequence to the right shows the occurrence of a *split*. The current focus is emphasized. The upper picture shows the active list right before the split. The middle picture is where the split occurs, and in the lower picture the active list is split in two new lists.

Connecting the focus to the join vertex either splits an active list or merges two different lists, depending on whether the join vertex is in the same active list as the focus or a different one. In either case, the join vertex is duplicated and the previous and next pointers of the active lists are updated.

Special symbols like *previous*, *next*, *join*, and *dummy* are represented as negative integers: -1, -2, -3 and -4 to distinguish them from free valences.

Boundary vertices. FreeLence handles boundaries similar to the coder of Touma and Gotsman: The vertices of each boundary are connected to a new dummy-vertex, so that the mesh is closed. When the encoder hits a dummy-vertex for the first time, a special boundary code **D** is written to provide a hint for the decoder to remove the dummy vertices after decoding the mesh.

As for joins, an index is needed along with the boundary symbol to indicate which of the free edges of the focus is connected to the dummy vertex. However, this index can be omitted if there are no free valences other than the dummy vertex. This way, we need to store only one boundary symbol per hole. In the case that the boundary is hit by more than one focus independently, a split occurs, with the dummy vertex being the join vertex. We use a negative offset to indicate that the join is a dummy vertex. To prevent dummy vertices from becoming a focus, we provide them with

```
Encode (TriangleMesh M) {
 init
   v0 = pickInnerVertex()
   addToActiveList(v0)
   addToQueue(v0, 360)                  // outer angle 360°
 repeat
   v = getNewFocus()                    // new focus
   if 0==freeEdges(v.prev)              // move to previous
     print(P)
   elseif 0==freeEdges(v.next)          // move to next
     print(N)
   elseif isJoinVertex(v)               // join
     first = getFirstJoinEdge(v)
     fillToJoinEdge(v, first)
     join(v, first)
     print(J, first, distance(v, first))
   else                                 // no joins
     nFree = getNumFreeValences(v)
     makeVertexFull(v)
     print(nFree)
   removeFullVertices()
   updateQueue()
 until queueIsEmpty() }
```

Table 1: Pseudocode of the FreeLence encoder.

infinitely large keys in the priority queue.

Since the dummy vertex has no geometric position, we define the angles of triangles incident to the dummy vertex to be 90 degrees.

2.5 FreeLence Pseudocode

The kernel of our algorithm is based on a single main loop: we keep adding free vertices and removing full vertices until there are no vertices left. In each step we update the priority queue of vertices in active lists according to their opening angles. To highlight the main ideas we include the pseudocode for encoding triangulated meshes of arbitrary genus but without boundary.

Decoding. The code for decompression is equally simple and works along the same lines; the "print" command is replaced by a "read" command, and the connectivity is restored from the information it reads.

Geometry quantization. In test implementations we first quantized the vertex positions to a desired accuracy of usually 8-12 bits, and then used the *parallelogram prediction* [Touma and Gotsman 1998] to predict vertex postitions during the mesh traversal. For higher face degrees we use its extension [Isenburg and Alliez 2002].

3 Analysis and Discussion

FreeLence yields the best bit rates for lossless compression of triangle meshes known so far. It achieves an average of 36% better rates over popular valence-based schemes like the one of Alliez-Desbrun and an average of 18% better rates over Angle- Analyzer for benchmark models, see table 2.

Adaption to regularity is one of the important features of our algorithm, making it attractive for high-resolution subdivision models. The ability to adapt to very regular models gives FreeLence a further advantage compared to Angle-Analyzer. The stronger the correlation between opening angles and free valences the better we can predict the latter number from the former. With an increasing regularity of a mesh our bit rates asymptotically tend to zero, a fact which is also true for the original Touma-Gotsman algorithm. Table 3 shows the bit rates for $\sqrt{3}$-subdivision of the "body" model, see also Figure 3.

model	vertices	VD	AA	FL	vs VD	vs AA
body	711	2.38	1.96	1.87	22%	5%
femur	3,897	2.71		2.09	23%	
egea	5,315	1.63	0.82	0.54	67%	34%
fandisk	6,475	1.02		0.72	30%	
venus	8,268	2.71	1.95	1.73	36%	11%
foot	10,016	2.20	1.56	1.35	39%	14%
mannequin	11,706	0.37		0.37	0%	
dino	14,070	2.25	1.69	1.44	36%	15%
horse	19,851	2.25	1.35	0.95	58%	30%
David	24,085	2.52		1.97	22%	
Max	25,445	2.22	1.45	1.19	47%	18%
feline	49,864	2.38	1.50	1.21	49%	19%
average		2.05	1.54	1.29	36%	18%

Table 2: Comparison of our FreeLence (FL) algorithm with the best known coders on several 3D benchmark triangle meshes: Valence Driven coder (VD) by Alliez-Desbrun and Angle Analyzer (AA) by Lee-Alliez-Desbrun. The given numbers are bits per vertex (bpv). The last two columns show the improvement ratios of FL compared to VD and AA. The last row shows the average improvement rate of FL over the other methods. Missing entries indicate that no comparison value is available.

verts	711	2,131	6,319	18,955	56,647	169,936
bpv	1.868	1.059	0.486	0.237	0.099	0.062

Table 3: Bit rates in bits per vertex (bpv) for recursive $\sqrt{3}$-subdivision of the body model, illustrating the ability of FreeLence to adapt to the regularity of a mesh.

Entropy bound and optimality. Under the assumption of no join symbols FreeLence guarantees bit rates of 2.8999 bits per vertex (bpv), a proof is given in the appendix. This is the best-known upper bound, improving a previous upper bound of 3.2364 bpv [Gotsman 2003] by 11%. This is especially interesting as one could choose the traversal scheme of our algorithm such that it equals that of Touma-Gotsman in which case the occurrences of joins (global splits) would be *identical*. Also, the numbers of stored symbols are asymptotically identical for both schemes, namely #vertices + #joins (see appendix). Note that in practice we have even less global splits than in the Touma-Gotsman case because of the geometry-driven choice of the next focus. Because *free-valence coding* improves valence-based coding by 11%, the latter might not be optimal.

Complexity. Since we can bound the maximum length of the active list as well as the number of joins by the number of edges, which is roughly three times the number of vertices. As the number of elements that are enqued is #joins + #vertices, we have $O(V)$ enqueueing steps, resulting in a theoretical run-time complexity of $O(V \log(V))$ for maintaining the priority queue. Additionally we have an extra penalty for handling joins. However, because the number of joins is low, the processing time for joins was neglectible for all models that we have tested.

Higher face degrees. Once FreeLence is mastered for triangle meshes, it is straightforward to move on to meshes with mixed

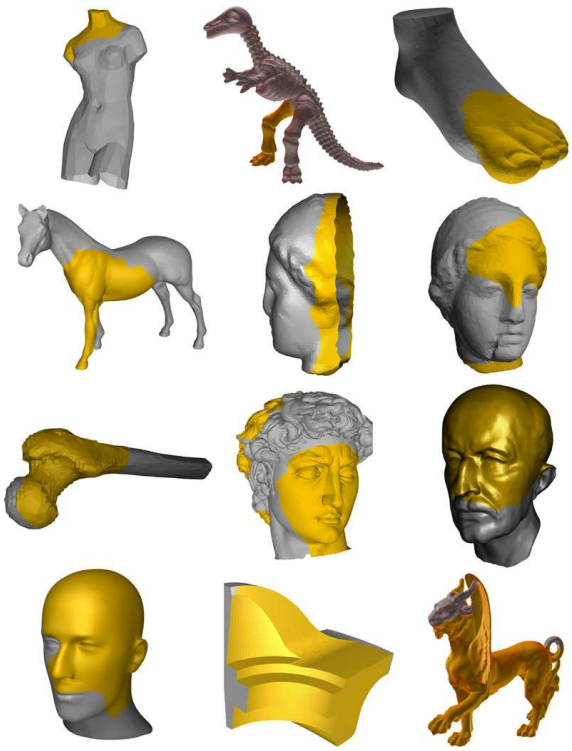

Figure 5: Models used in the comparison of FreeLence with other coder.

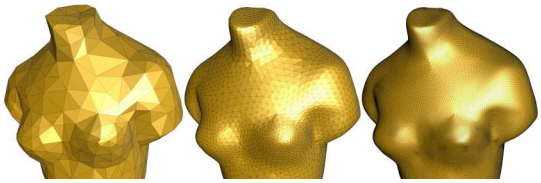

Figure 6: More regular models yield even higher compression rates.

face degrees. However, we do not go into much detail here as the straightforward approach, albeit simple, is not yet suited to compete with bit rates of other techniques such as valence-based [Isenburg 2002] or gate-based [Isenburg and Snoeyink 2000] schemes. Still, because of its simplicity we briefly sketch it here.

The major difference to the pure triangular case is that for each face its degree needs to be stored in a separate *degree table*. As for triangle meshes the focus is chosen according to the minimal opening angle in the priority queue of active lists and all faces of degree n with $n-2$ free vertices are conquered. For each face its degree n is stored in the degree table and the number of free valences to the focus is stored in the context table. During this process it may happen that a face of degree n is encountered which has less than $n-2$ free vertices. Then a split occurs within this face. To perform this split, a virtual edge is introduced which splits the face into a k-gon (to the left of the virtual edge) and a $(n-k+2)$-gon (to the right of the virtual edge). The decoder must get a hint to remove this edge, so instead of storing the degree k we store its negative degree $-k$ in the degree table. The storage of negative degrees is the equivalent to join symbols. Note, that by using an entropy coder the extra storage of the face degrees produces no noticeable overhead, even if all faces of the mesh are triangular.

4 Conclusion

We introduced FreeLence, a single-rate connectivity coder for triangle meshes which gives an average of 36% better bit rates for connectivity encoding than the best valence-based coders known so far, and an average of 18% over Angle-Analyzer. FreeLence combines the advantages of the most popular existing techniques, valence-based encoding and Angle-Analyzer. At the same time it overcomes some disadvantages of each of these approaches.

FreeLence demonstrates the use of geometric information to increase the regularity of symbol sequences. This increases the compression ratio much more than the use of order-1 arithmetic coder. In fact, combining order-1 entropy coding with our technique slightly worsens the bit rate on most of the tested models.

We also provide evidence that valence-based coding is not near-optimal by improving the worst case by 11%. FreeLence reduces the redundancy stored by valence-driven coders by replacing combinatorial curvature by free valence information.

For the future we plan to improve the handling of mixed face degrees in FreeLence.

5 Acknowledgements

We thank Pierre Alliez and Haeyoung Lee for making their collection of 3D benchmark models available to us. Thanks to Hans Lamecker for the pelvic bone. Bunny model is from Stanford.

References

ALLIEZ, P., AND DESBRUN, M. 2001. Valence-driven connectivity encoding of 3d meshes. In *Eurographics 2001 Conference Proceedings*, 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*.

BEN-CHEN, M., AND GOTSMAN, C., 2003. On the optimality of spectral compression of mesh data. preprint.

BENDER, E. A., AND WORMALD, N. C. 1985. The number of loopless planar graphs. *Discrete Mathematics 54*, 235–237.

COORS, V., AND ROSSIGNAC, J., 2003. Delphi: Geometry-based connectivity prediction in triangle mesh compression. preprint.

DEERING, M. 1995. Geometry compression. In *Siggraph 95 Conference Proceedings*, 13–20.

GOTSMAN, C. 2003. On the optimality of valence-based connectivity coding. *Computer Graphics Forum 22*, 1, 99–102.

GUMHOLD, S., AND STRASSER, W. 1998. Real time compression of triangle mesh connectivity. In *SIGGRAPH'98*, 133–140.

GUMHOLD, S. 1999. Improved cut-border machine for triangle mesh compression. In *Proceedings of Erlangen Workshop '99 on Vision, Modeling and Visualization*.

ISENBURG, M., AND ALLIEZ, P. 2002. Compressing polygon mesh geometry with parallelogram prediction. In *Visualization conference proceedings*, 141–146.

ISENBURG, M., AND SNOEYINK, J. 2000. Face fixer: Compressing polygonal meshes with properties. In *Proceedings of SIGGRAPH*, 263–270.

ISENBURG, M., GUMHOLD, S., AND GOTSMAN, C. 2001. Connectivity shapes. In *Visualization'01 Conference Proceedings*, 135–142.

ISENBURG, M. 2002. Compressing polygon mesh connectivity with degree duality prediction. In *Graphics Interface Conference Proceedings*, 161–170.

KHODAKOVSKY, A., ALLIEZ, P., DESBRUN, M., AND SCHRÖDER, P. 2002. Near-optimal connectivity encoding of 2-mainfold polygonal meshes. *Graphics Models, special isuue*.

KING, D., AND ROSSIGNAC, J. 1999. Guaranteed 3.67v bit encoding of planar triangle graphs. In *Proceedings of 11th Canadian Conference on Comp. Geometry*, 146–149.

LEE, H., ALLIEZ, P., AND DESBRUN, M. 2002. Angle-analyzer: A triangle-quad mesh codec. In *Eurographics conference proceedings*, vol. 21, 383–392.

POULALHON, D., AND SCHAEFFER, G. 2003. Optimal coding and sampling of triangulations. *30th international colloquium on automata, languages and programming (ICALP'03)*..

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

SORKINE, O., COHEN-OR, D., AND TOLEDO, S. 2003. High-pass quantization for mesh encoding. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*, Eurographics Association, 42–51.

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 *Graphics Interface Conference Proceedings*, 26–34.

TUTTE, W. 1963. A census of planar maps. *Canadian Journal of Mathematics 15*, 249–271.

WITTEN, I. H., NEAL, R. M., AND CLEARY, J. G. 1987. Arithmetic coding for data compression. *Communications of the ACM 30*, 6, 520–540.

Appendix: Entropy of free valence sequences

For triangle meshes we show that asymptotically the worst case bit rates for free valence codes is strictly less than the worst case bit rates for full valences codes. In fact we even show a stronger result, namely that the bit rates for free valence codes is 11% below the entropy of the code of average valences with the average being taken over all triangle meshes, see [Gotsman 2003] for precise distinctions between worst case and average entropies. Note that all of these bit rate estimates do not take "split" symbols into account but are only concerned with the valence part of the code sequence. Our bit rate estimate has a major consequence: full valence codes for triangle meshes contain redundant information which is not contained in free valence codes. In fact, one could replace the traversal scheme of FreeLence by the traversal scheme of [Touma and Gotsman 1998] so that the number of split occurrence would be identical for both codes. But since the valence part of the code for free valences is on average less then the entropy for full valence codes, this implies the non-optimality of (full) valence-based codes. The question about optimality is hence open once again.

Worst case entropy for free valence codes. We assume the setting of a triangulated orientable 2-manifold with V vertices and without boundary. We also assume that this manifold is topologically equivalent to a sphere. Recall that the entropy of a sequence of symbols is given by

$$E = -\sum_i p_i \cdot \log_2 p_i, \tag{1}$$

where p_i is the frequency of the i^{th} symbol. In our case we have the symbols **N** for *move to next*, **P** for *move tp previous* and **i** for *i free valences*. Their corresponding frequencies in the sequence are denoted by p_N, p_P and p_i. Under the assumption of no splits, the total number of these symbols is exactly $V-2$ because all vertices in the mesh, except those in the last triangle, correspond to exactly one of the symbols in the code, and for the last triangle only a single symbol, 0, is needed. This and the fact that each free valence corresponds to exactly one free vertex (except for the very first focus) imply that

$$\sum_{i=0}^{\infty} p_i + p_N + p_P = 1 \qquad \sum_{i=1}^{\infty} i \cdot p_i = 1. \tag{2}$$

We must now maximize the worst case free valence entropy

$$E_{\text{free\_val\_worst}} = -\sum_{i=0}^{\infty} p_i \cdot \log_2 p_i + p_N \cdot \log_2 p_N + p_P \cdot \log_2 p_P \tag{3}$$

under the assumptions (2). We take the standard way of employing Lagrange multipliers, so we let

$$f(p_N, p_P, p_0, p_1, ..., \lambda, \mu) =$$
$$E_{\text{free\_val\_worst}} + \lambda(\sum_{i=0}^{\infty} p_i + p_N + p_P - 1) + \mu(\sum_{i=1}^{\infty} i \cdot p_i - 1).$$

The partial derivatives of f with respect to p_i, p_N and p_P must then vanish so that $\log_2 p_i = (\lambda - 1) + i \cdot \mu$ and $\log_2 p_N = \log_2 p_P = (\lambda - 1)$. We can now set $\alpha := 2^{\lambda - 1}$ and $\beta := 2^{\mu}$ and get

$$p_i = \alpha \cdot \beta^i \quad \text{and} \quad p_N = p_P = \alpha. \tag{4}$$

Plugging back into (2) gives

$$\alpha \cdot (\sum_{i=1}^{\infty} \beta^i + 3) = 1 \quad \text{and} \quad \alpha \cdot (\sum_{i=1}^{\infty} i \cdot \beta^i) = 1. \tag{5}$$

Collecting terms in the infinite sums then gives

$$\frac{\alpha \cdot \beta}{(1-\beta)} + 3\alpha = 1 \quad \text{and} \quad \frac{\alpha \cdot \beta}{(1-\beta)^2} = 1. \tag{6}$$

Solving these last two equations for α and β yields

$$\alpha = \frac{1}{2} - \frac{1}{\sqrt{12}} \quad \text{and} \quad \beta = \frac{3}{2} - \frac{3}{\sqrt{12}} \tag{7}$$

Combining equations (3) and (4) then gives the worst case entropy for free valence codes

$$\boxed{E_{\text{free\_val\_worst}} = -\log_2(1 - \frac{\sqrt{3}}{2}) = 2.8999... \text{ bpv.}}$$

Free-valence-based code optimality. In [Gotsman 2003] Gotsman has remarked that the entropy of *all* triangle mesh valence distributions, $E_{\text{all\_val}}$ is strictly less than then *Tutte entropy*. This analysis is

based on the results of [Bender and Wormald 1985] concerning the relative frequency of valences over *all* triangle meshes,

$$E_{\text{all\_val}} = 3.2364... \text{ bpv}$$
$$E_{\text{Tutte}} = \log_2(\frac{256}{27}) = 3.2452... \text{ bpv}.$$

Our calculation of the worst case entropy for *free valence codes* shows that $E_{\text{all\_free\_val}}$ - the entropy of the distribution of free valences in *all free valence codes in the class of all triangle meshes* is not only strictly less than the Tutte entropy but also strictly less than $E_{\text{all\_val}}$, the entropy of the distribution of full valences in all the free valence codes in the class of all triangle meshes:

$$\boxed{E_{\text{all\_free\_val}} \leq E_{\text{free\_val\_worst}} < E_{\text{all\_val}} < E_{\text{Tutte}}}$$

This implies that valence-based coding may in fact not be optimal but can be improved by free-valence-based coding because the **dispersion of symbols in free-valences-based codes is less than in valence-based codes**. Intuitively this is based on the fact that the average number of free valences is 1 free edge per vertex whereas the average number of full valences in a triangle mesh is 6 edges per vertex. Together with the fact that the traversal scheme for free-valence-based coding can be made the same as for the Touma-Gotsman algorithm (so that the occurrence of split symbols becomes identical in both of them) implies that *free-valence-based coding breaks the 'optimality' for valence-based schemes in practice.*

Counting symbols. We now assume an orientable triangulated 2-manifold mesh with V vertices and g handles but without boundary. We prove that the number of stored symbols of a free-valence coder is asymptotically equal to $V+J$, the number of vertices of the mesh plus the number join symbols (splits + merges) in the code. This is the same number as for valence-based coders so that both techniques have identical code lengths. We first give a count of the split symbols: each split increases the number of active lists by 1. Each active list will finally 'die' in a single triangle which is closed. Right before such a triangle is closed its boundary will make up an active list itself. Let N_0 be this number of 'dying' triangles in the conquering process. Also, to each handle there corresponds exactly one split and exactly one merge operation which together leave the number of active lists invariant. Hence we have for the number of joins

$$J = N_0 + 2 \cdot g - 1.$$

We can now count the total number of symbols N of a free valence coder: each vertex contributes one of the symbols **N** for *move to next*, **P** for *move to previous* or **i** for *i free valences*. Each join operation duplicates the focus and the corresponding join vertex and adds an extra **J** itself (there is an extra list where the distances of the join vertices to the focus are stored; those are not counted here). So the number of joins increase the number of symbols by $3 \cdot J$. However, each occurrence of a 'dying triangle', a triangle whose boundary becomes itself an active list, requires only a single 0 for one of its vertices, the other two do not require an extra symbol. Hence, the total number of symbols is num_stored_symbols $= V + 3 \cdot J - 2 \cdot N_0$. Together with the last equation this implies

$$\boxed{\text{num\_stored\_symbols} = V + J + 4 \cdot g - 2}$$

.

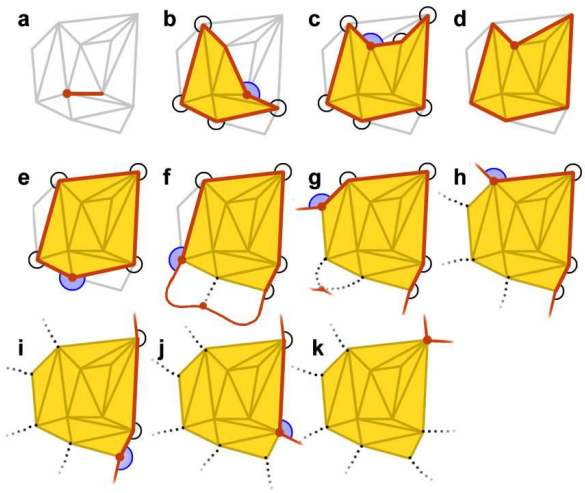

Figure 7: A complete run of FreeLence. (**a**) Pick an initial vertex, it has valence 6, write **6**. (**b**) Close the triangles incident to the initial vertex and find a new focus according to minimal opening angle; the new focus has 2 free valences, write **2**. (**c**) Close the triangles incident to the focus and find a new focus according to minimal opening angle; the vertex previous to the focus has no free edges, write **P** and close previous. (**d**) Focus has 0 free valences, write **0**. (**e**) Focus is boundary vertex with 1 free edge preceding the boundary in clockwise order, write **D 1**. Connect a virtual edge (dashed line) to the dummy vertex. Decrease opening angles of the focus' neighbors by 90°. The total count of free valences of the focus is 1 write **1**. (**f**) New focus has 1 free valence, write **1**. (**g**) to (**j**). Focus has 0 free valences; write **0**. (**k**) Active list has length two. Stop. The full seqence is **6 2 P 0 D 1 1 1 0 0 0 0**.

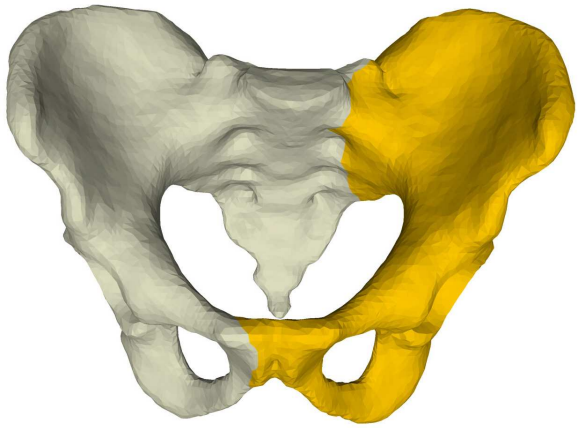

Figure 8: Intermediate state of the encoding of a pelvic bone. Note how circular the active list remains during the processing.