EUROGRAPHICS 2001 / A. Chalmers and T.-M. Rhyne
(Guest Editors)

Volume 20 (*2001*), *Number 3*

Valence-Driven Connectivity Encoding for 3D Meshes

Pierre Alliez Mathieu Desbrun

USC

{alliez|desbrun}@usc.edu

Abstract

In this paper, we propose a valence-driven, single-resolution encoding technique for lossless compression of triangle mesh connectivity. Building upon a valence-based approach pioneered by Touma and Gotsman [22], we design a new valence-driven conquest for arbitrary meshes that always guarantees smaller compression rates than the original method. Furthermore, we provide a novel theoretical entropy study of our technique, hinting the optimality of the valence-driven approach. Finally, we demonstrate the practical efficiency of this approach (in agreement with the theoretical prediction) on a series of test meshes, resulting in the lowest compression ratios published so far, for both irregular and regular meshes, small or large.

1. Introduction

Efficient compression of triangle meshes is a timely research subject, as demonstrated by the fast-growing demand for fast transmission of 3D contents over the Internet [21]. Over the last five years, we have witnessed a rapid decrease in bit rates with the design of smart compression schemes. Yet, and contrary to the previous information media such as audio, images, or video, little is known about the theoretical entropy of 3D meshes.

Lossless connectivity encoding has been the focus of numerous research work [4, 20, 15, 22, 10], including proven bit rate upper bounds [15, 13, 16, 7, 18], and guaranteed asymptotic behavior [22, 18]. Although bit rate is the most theoretically challenging part of a connectivity encoder, authors have also added essential features from a telecommunication standpoint, such as progressivity [8, 19, 14, 3, 25, 5, 12, 1], error resilience [2] and computational cost [6, 16, 11] among several others.

While the importance of such features cannot be ignored, single-rate encoding (i.e., simple encoding of a mesh) remains challenging for two basic reasons. First, even progressive algorithms require the encoding of a *coarse base mesh* at the very beginning of the bitstream, the most critical part of a transmission. Second, the upper bound deduced from the census of planar triangulations from Tutte [23] has **not** yet been reached, and the theoretical study of the entropy of 3D meshes is still an open question. Guaranteeing upper bounds for single-rate encoding of connectivity is a very important research topic, since it is definitely interesting to be able to predict the duration required for transmission of 3D models, or the memory space required for storage. However, the actual compression rate of an encoder is even *more important in practice*.

In this paper, we propose a **single-rate compression technique to encode connectivity** that, compared to the results published so far, offers the *best compression ratios* on arbitrary meshes. We also prove that our valence-driven technique is pertinent since a pure valence encoding is *optimal* in terms of entropy.

1.1. Previous work

The *EdgeBreaker* technique [15] is currently the best encoding technique proposing both a rigorous theoretical analysis and an outstanding worst-case bound of the connectivity compression bit rate. The original method used at most 4 bit per vertex (denoted 4 b/v for simplicity), and has been followed by numerous improvements on its upper bounds [13, 16, 7], its asymptotic behavior [18] and its efficiency [16, 11]. In [18], Szymczak *et al.* describe a coding improvement that also detects and benefits from mesh regularity.

Besides *EdgeBreaker* and its derived methods, two powerful techniques turn the connectivity of a triangle mesh into a sequence of valence codes [22, 10] in order to automatically benefit from the low statistical dispersion around the average 6 when using entropy encoding. This is achieved through a deterministic conquest [22] or by a sequence of half edge collapses [10].

In [22], Touma and Gotsman pioneered the conquest approach and compress the connectivity down to less then 0.2 b/v for very regular meshes, and between 2 and 3.5 b/v otherwise, in practice. The so-called conquest consists in conquering the edges of successive pivot vertices in an orientation-consistent manner and generating valence codes for traversed vertices. Three additional codes: *dummy*, *merge* and *split* are required in order to encode boundaries, handles and conquest incidents respectively. The *dummy* code occurs

each time a boundary is encountered during the conquest; the number of *merge* codes is equal to the genus of the mesh being encoded. The *split* code frequency is mainly linked to the mesh irregularity. Intuitively, if one looks at the encoding process as a conquest along a spiraling vertex tree, the *split* codes thus indicate the presence of its branching nodes.

The *Mesh Collapse Compression* scheme from Isenburg and Snoeyink [10] performs a sequence of edge contractions until a single vertex remains in order to obtain bit rates of 1 to 4 b/v. For each possible edge to contract, the authors cut and open the mesh along the edge, which creates a *digon* - an outer face bounded by only two edges. Resulting digons are *simple*, *complex* or *trivial* according to the local edge configuration. For *simple* digons, an edge-contraction is applied and one valence code is output. For *complex* digons, a *start* symbol is output. For *trivial* digons, an *end* symbol is output in order to record a reversible operation.

1.2. Overview

In spite of the absence of linear worst-case bit-rate complexity, the two latter methods turn out to be very appealing since entropy coding of a valence sequence automatically benefits from the mesh regularity and produces very competitive compression ratios. In this paper, we demonstrate that, even if the technique from Touma and Gotsman [22] turns out to be almost optimal in the regular case (this has not been really challenged since 1998), some important degrees of freedom in the encoding algorithm remain. In particular, we achieve a reduction of the rate of 15% on average for coarse irregular meshes, which is extremely beneficial for encoding a base mesh in a progressive transmission [12]. Also, we propose for the first time an analysis of the maximum entropy that a valence-encoding method can achieve asymptotically, and we demonstrate a surprising upper bound that exactly equals Tutte's census [23], hinting the optimality of our method.

In the remainder of this paper, we first give some common definitions in section 2, before describing the encoding method we built upon in section 3, and then describe our valence-driven encoding strategy for connectivity compression in section 4. Comparative results on numerous meshes are given in section 5, while section 6 gives some concluding remarks and future work.

2. Definitions

In this paper, we consider 2-manifold meshes, where the local surface around each vertex is homeomorphic to a disk. We assume a consistent orientation of the faces, but no restriction on the genus and boundaries which can both be arbitrary. The encoding process we consider generates an ordered set of valences from the mesh vertices and a few additional accident codes while conquering edges according to a counter-clockwise orientation around successive pivot vertices. For the sake of clarity, we first define items and notions that both the original algorithm [22] and ours make heavy use:

- **conquest edge list:** a set of adjacent edges forming a closed path on the mesh. As we will see, the conquest only acts on the vertices of the active edge list, locally separating the surface in an inner conquered region and an outer free region;
- **pivot vertex:** a vertex chosen for conquering its adjacent edges in a counter-clockwise fashion. It is sometimes called a focus vertex;
- **full vertex:** a vertex with every edge already conquered;
- **dummy vertex:** a vertex temporary added to the mesh in order to virtually remove one hole or one boundary;
- **ghost vertex:** a unique ubiquitous vertex added in order to virtually remove all holes and boundaries. A ghost vertex is not manifold, thus it cannot be chosen to be a pivot;
- **state flags:** *free* or *conquered* are assigned to faces, vertices and edges according to whether they have been conquered or not.

3. Original connectivity encoding method [22]

This section provides a quick summary of the connectivity encoding technique [22]: we refer the reader to the original paper for a lenghtier presentation. The encoding process starts with a triangulated mesh, and generates a closed topology by adding and connecting one common *dummy* vertex to each boundary vertex. An arbitrary seed face f_{seed} is chosen, its vertices are added to an active (conquest) edge list and their three corresponding valences are output to the *code sequence*, which will eventually be compressed using conventional entropy encoders. The face f_{seed}, its vertices and its edges are flagged conquered. The first vertex of f_{seed} is chosen as the pivot vertex. The conquest can now begin.

3.1. Conquest of a next free edge

The current pivot vertex v_{pivot} tries to conquer its next free edge e in the counter-clockwise order. Let us call v the vertex belonging to e, and v_p the vertex preceding v_{pivot} in the active edge list. As depicted in Figure 1, the vertex v is inserted before the pivot in the active list, the edges e and $\{v, v_p\}$, and the face $\{v_p, v, v_{pivot}\}$ are flagged conquered. According to the current state of v, four cases may occur:

1. v has not been conquered yet. It is now flagged conquered, and its valence is output to the code sequence. The active edge list is thus expanded as shown in Figure 1(b);
2. v is a *dummy* vertex and has not been conquered yet. It is thus flagged conquered, and a code *dummy* and its valence are output to the code sequence. The active edge list is also expanded;
3. v has already been conquered and belongs to the active list. To lift the local ambiguity of connectivity, a *split* code and an associated forward offset (which indicates where v is located in the current active list) are generated (see Figure 2). The active list is then split in two: the (internal) sub-list is pushed to the stack for future treatment, while the conquest continues using the (external) list;

4. *v* has already been conquered and belongs to an inactive (sub-)list present on the stack. This case requires the localization of the inactive list in the stack and of *v* in that list. A *merge* code along with its associated list index in the stack and offset in the inactive list are generated. The two lists are merged, leading to an active list containing two occurrences of *v* (see Figure 3). The inactive list is discarded from the stack. The *merge* code can only occurs when the genus is greater than 0.

3.2. Full pivot removal

When (and while) the current pivot is full (i.e. with no free edges), it is removed from the list, thus making the size of the list decrease. If we call v_n the vertex following v_{pivot} in the active edge list, the face $\{v_p, v_n, v_{pivot}\}$ and the edge $\{v_p, v_n\}$ are flagged conquered (see Figure 4). Notice that an edge has been conquered at no cost. No code is output during this operation since the decoder, having previously received the original valence of the vertices on the active list, is able to decrease the number of free edges during the reconstruction process and is thus able to automatically detect full vertices. After a recursive call for removal of full vertices, and if the active list is non empty, the *next vertex* in the list is chosen as the pivot (remember this for the next section), and the edge conquest is resumed (see section 3.1). If the active list is empty, we discard it from the list stack, the next one is popped out and becomes active. The conquest is complete when the list stack is empty. Every vertex, face and edge are now conquered. The decoding process being straightforwardly deduced from the encoding process, we refer to the original paper [22] for more details.

3.3. Discussion

Although this technique results in very low bit rates for very regular meshes, the bit rate goes up to 2 or 3 b/v for irregular meshes such as those sent at the beginning of a progressive transmission. After experimenting with the Touma and Gotsman's encoder [22] in order to examine the code sequence, the number of *split* codes and the range of their offsets turn out to seriously impede the compression ratios when the models are somewhat irregular. Recall that a *split* code is followed

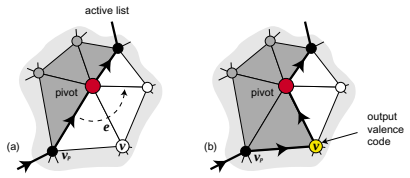

Figure 1: *Conquest atomic operation: (a) the current pivot conquers the next edge counter-clockwise; (b) corresponding vertex v is added before the pivot in the active list. Four items are flagged conquered, i.e. two edges, one face and the vertex v. The valence of v is output to the code sequence.*

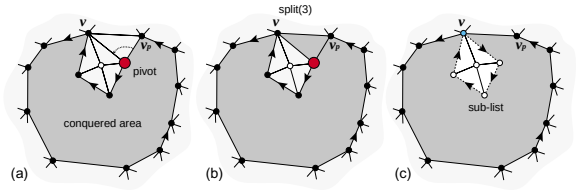

Figure 2: *Split code: (a) the current pivot vertex tries to conquer an edge linked to a vertex* already *inserted in the active list; (b) a split code and its associated forward offset are generated. (c) the active list is split; notice that the original counter-clockwise seed list has generated a clockwise sub-list, thus allowing the conquest of the enclosed region.*

by an index expressed in number of vertices in the active list (see Figure 2). This is a serious obstacle to guarantee a linear worst case bit-rate complexity, as pointed by [16]. In addition, there are presently no theoretical bounds on this valence-based approach. Our contribution, described in the remainder of this paper, is two-fold: we improve upon the original method while we provide a first analysis of the asymptotical worst-case bit rate.

4. New Valence-driven Encoding

In this section, we present our new mesh connectivity encoding method. Building upon the technique [22] presented in the previous section, we propose i) to replace the deterministic conquest by an adaptive one in order to minimize the number of *split* codes; ii) to decrease the range of the offsets associated with each split code; iii) to improve compression rates when encoding objects with numerous boundaries; and finally, iv) to provide the first upper bounds for any valence-based approach, confirming the pertinence of a valence-driven algorithm.

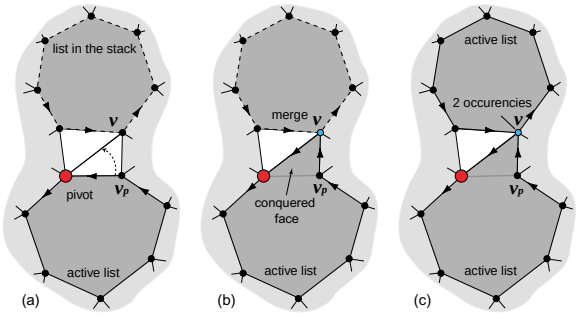

Figure 3: *Merge code. (a) the current pivot vertex from the active list tries to conquer a vertex already conquered, belonging to an inactive list previously pushed to the list stack; (b) the vertex belonging to the next edge around the current pivot is inserted before the pivot in the active list. One face and two edges are conquered, a merge code is generated, with its two associated offset codes (one for a localization in the list stack and another one for the localization within the list). (c) the two lists are merged.*

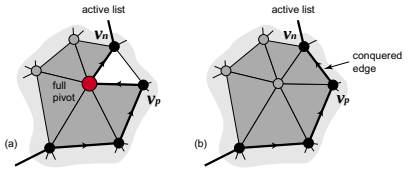

Figure 4: *Full pivot removal: (a) the current pivot is full, with no free edges; (b) it is removed from the list, and one face and one edge are flagged conquered. No code is output.*

4.1. Valence-driven edge conquest

As we previously mentioned, a conquest on an irregular mesh generates many split codes, along with costly offsets coding. Figure 5(a) illustrates a typical example of the creation of a cavity, leading to a split of the active edge list. It is therefore interesting to adapt the conquest to avoid the creation of such cavities. We now present a simple heuristic using the number of free edges for each vertex. In [22], the next pivot is always selected as the next vertex on the active edge list. However, both the encoder and the decoder always have the whole active edge list to choose from, and picking a specific vertex can dramatically reduce the occurrences of splits later on in the conquest.

We first tried a very simple heuristic, consisting in choosing the vertex with the *minimum free edges* as the next pivot, i.e., the minimum number of neighbors not visited yet (see Figure 5(c)). The number of free edges for each treated vertex is known without any additional computations, since it is simply the vertex's transmitted valence decreased by the number of neighbors already processed. As sketched in Figure 5(b), this will favor vertices inside cavities over vertices that may create splits. This simplistic modification already decreases the average number of splits by 50%. Notice that all the full vertices will disappear first, since they correspond to zero free edges. Further tests have shown however that more refined heuristics behave even better.

Since the previous heuristic can lead to many vertices with the same minimum number of free edges, it sometimes fails to pick a vertex really in a cavity, thus not really helping in reducing the number of splits. An averaged count of free edges, depending on a larger neighborhood, would be more accurate in such a case. Therefore, we use instead an adaptive *mean* number of free edges to select the next pivot vertex. If the first heuristic fails to select a single vertex, we now compute a mean number of free edges *using a vertex and it two immediate neighbors on the active edge list*. If, again, this test does not lead to a single choice, we will use the four closest neighbors (two on each side) to enlarge the averaging support, etc. For a given support size s, the mean number of free edges m of a vertex of index v on the active edge list is given by:

$$m = \frac{1}{2s+1} \sum_{i=-s}^{s} (1 - |\frac{i}{s}|) n_{v+i}^{fe}, \qquad (1)$$

where n_v^{fe} is the number of free edges of the vertex v.

As noted before, the conquest strategy is driven using previously decoded data so that the decoder can follow this exact same heuristic at no additional bit cost. The conquest action being limited to the fore front of the conquest, i.e. the vertices of the active list, we thus restrict the search to a small ratio of the vertex count. For irregular coarse meshes, the number of split codes is now reduced by 70% in average (see Table 1). Figure 6 illustrates an example of the resulting valence-driven conquest. Note that our new strategy corresponds to adding a sort of "surface tension" to the contour of the conquered region: we grow it only where it will minimize the number of contour edges for the same number of interior points. Now that the number of split codes is reduced, we can further seek for a reduction of the range of the offsets associated with the remaining split codes.

4.2. Reduction of offset range

In [22], the offset codes resulting from a *split* or a *merge* code are expressed in number of vertices in a list, leading to a possibly large range. From an information theory point of view, the smaller the range, the better the compression rate, obviously. We thus propose to replace the offset codes by a unsigned index in the list *once sorted in the decreasing order of the Euclidean distance from each vertex to the pivot*. We also exclude the vertices incident to the pivot since they cannot be candidates to the split. This drastically reduces the range of the indices during splits. In practice, this offset be-

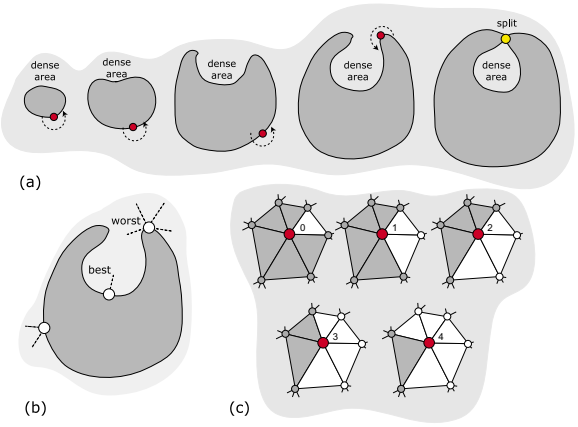

Figure 5: *(a) The original conquest [22] is simulated in the neighborhood of a highly variable density mesh. Each new pivot is chosen in a deterministic way, i.e. the vertex succeeding the previous pivot in the list. A split code is far more frequent in regions with highly variable density and appears after a cavity has been formed by the active list. (b) Intuitively, the key idea of our adaptive conquest consists of choosing the best pivot candidate so that it tends to minimize the number of split codes. Such a choice is driven by the number of free edges for each vertex on the list. (c) Several configurations of the conquest around a valence-6 pivot vertex, with associated number of free edges.*

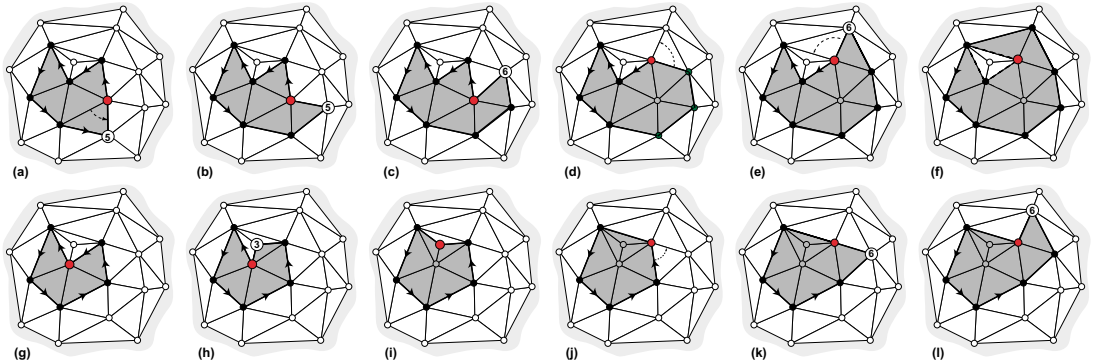

Figure 6: *Top line, the original coder from [22]: (a) The next counter-clockwise edge is conquered from the active pivot. A valence code 5 is output. (b) code 5. (c) code 6. (d) the pivot, full, is removed from the list at null cost. The next vertex in the active list is chosen as pivot. (e) code 6. (f) code split with an offset of 2. The code sequence is {5,5,6,6,split(2)}. Bottom line, our technique: (g) the best pivot candidate is searched into the active list. One unique vertex has only one free edge, it is thus chosen as a pivot. (h) code 3. (i) the full pivot is removed. The best pivot candidate has 0 free edge. (j) the full pivot is removed. The next best pivot candidate has 2 free edges. (k) code 6. (l) code 6. The pivot is now full. The code sequence is {3,6,6}.*

comes zero in most of the cases. Therefore, we always add 6 to this offset so that our arithmetic encoder can compress this offset just as if it was a valence, optimizing the compression rate (see details in Section 4.4).

Notice that, although such reordering turns out to be very efficient from a bit-rate standpoint, it requires a decoding of the geometry interleaved with the connectivity. This dependence to the geometry is sometimes not desirable, if for instance a progressive geometry transmission is used. In that case, we recommend to keep the original offset with addition of a sign bit in order to encode the backward split cases in a more efficient manner. Indeed, the active edge list being closed, it decreases the range of the offsets in most of the cases (as mentioned in [7]).

4.3. Encoding of multiple boundaries

In the technique summarized in section 3, a *dummy* vertex being added and connected to each boundary, objects with numerous boundaries may lead to numerous *dummy* codes, followed by their associated high valence. When an object with many boundaries (or several connected components containing various boundaries) is encoded, the bit rate will seriously increase unnecessarily. We have opted for a more efficient handling of the boundaries, by inserting *only one* ubiquitous *ghost* vertex connected to every boundaries. We insert at the beginning of the bit stream one bit to indicate the presence or absence of boundaries, and the ghost vertex's total valence, i.e. the total number of vertices lying on boundaries. During the conquest, we generate a *ghost* code each time it is encountered. The *ghost* vertex being most probably not manifold, it is never chosen as a pivot during the conquest. In order to allow the conquest to terminate, an active list containing only the *ghost* vertex is simply removed from the stack, even when not full. If the *ghost* vertex is not full and the stack is empty, the decoder concludes that more vertices are to be treated, and pushes a new active list on the stack with the next three valence codes. This simple modification allows us to encode efficiently meshes with several boundaries, when the previous method was suffering from a rapid bit rate increase for these sort of meshes. Notice that this is particularly interesting for coarse meshes with a lot of small holes, where the number of holes is not negligible compared to the total number of vertices.

4.4. Efficient Encoding of the Code Sequence

Contrary to [22], we encode the sequences of codes generated during the conquest through the *range encoder* [17], an adaptive arithmetic encoder [24]. This has two main advantages: first, this encoder is distributed for free with GNU general public license, and second, the bit rate compression analysis will be easier, and will allow us to find asymptotical upper bounds (see section 4.7) which would have been difficult using the Huffman coding mixed with RLE proposed by [22].

Roughly speaking, the range encoder exploits the probability distribution of the symbols to optimize their individual size. However, it doesn't require the transmission of a table of occurrences like a Huffman encoder would: instead, the probability of occurrence of each symbol is assumed equiprobable first, and then is updated *while the symbol sequence is sent/received*. If the number of distinct symbols generated is known, along with their probability of occurrence in the sequence, we can explicitly compute the average number of bits per symbol the encoding will require. In practice, as soon as the size of the sequence is larger than a thousand, the prediction and the actual bit rate observed only differ by less than a few percent.

4.5. Initialization of the Adaptive Entropy Encoder

As mentioned in the previous paragraph, the range encoder [17] assumes an equiprobable distribution of codes at the beginning, by lack of a better guess. When the mesh to

encode is large, the range encoder will adapt quickly to the actual distribution. However, if the mesh is relatively small (and remember than single-resolution encoders are mainly used for small, irregular meshes), this initial distribution could be easily optimized. Our tests have shown that most meshes have a relatively similar valence distribution, like depicted in Figure 9: such a distribution is therefore a more educated guess than constant probability.

In practice, we reserve the first three symbols $\{0;1;2\}$ for the *accident* codes: $\{ghost;split;merge\}$, and feed the range coder [17]. We initialize the probability distribution with null frequencies for the three accident codes and a fitted Gaussian distribution centered on the average 6 value elsewhere. This turns out to improve the bit-rates, especially for low complexity meshes as mentioned before. The best σ parameter found from the Gaussian curve is quantized using 4 bits and stored at the beginning of the bit stream.

4.6. A Simplified Solution: Getting Rid of the Splits

During the development of the technique detailed above, we came across a slightly different solution, significantly simpler to program. However, in practice, the bit rates obtained are not as good as the solution presented above (and sometimes even not as good as [22]). If a quick implementation of small meshes is sought, this solution is definitely attractive: we therefore describe it roughly here.

The split code has been designed to solve an *incident of conquest*, i.e., when the current pivot vertex tries to conquer an edge linked to a vertex already transmitted (see section 3.1). Thus, we have to *deal with the problems* when they are encountered during the conquest. Instead of treating the problems, *delaying* them until they completely disappear turns out to be much simpler. This can be achieved by generating a *skip* code without any parameter instead of a *split*, and not splitting the list but rather resuming the conquest somewhere else (see Figure 7). The current pivot is then flagged *skipped* and is not candidate to pivot anymore, except as a full pivot in order to guarantee the termination of the conquest. For genus-0 objects, all the problems disappear without any further actions, while meshes with higher genus end up with a single triangle stripset where each border vertex is flagged *skipped*. A fixed number of additional codes ($2 \cdot genus - 1$ face indices to be exact) are then required to terminate the conquest by simply *zipping* the triangle stripset (see Figure 8) without need for additional bits.

This simple code substitution makes the conquest straightforward. Indeed, it does not require any stack since the active list is never split. We thus recommend it for a simpler implementation of the original coder [22]. Again, this is of course to the detriment of the bit rate.

4.7. Theoretical Bit-Rate Bounds of our Encoder

The principal critique of [22] found in the bibliography is the absence of clear upper bounds on the number of bits per vertex. Although a theoretical upper bound is less important

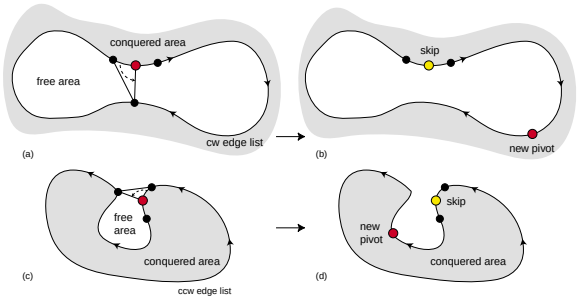

Figure 7: *(a) The pivot of the clockwise oriented list intends to conquer an edge linked to an already transmitted vertex. (b) the pivot is flagged* skipped*, a* skip *code is output, and another (best) pivot candidate is sought, not taking into account skipped vertices. (c,d) same as upper with a counter-clockwise oriented active list.*

than the rate-distortion behavior in practice, it is fundamental for a number of applications to guarantee a good worst-case bit rate to know, for instance, the maximum expected time for transmission. Moreover, if some prior knowledge of the mesh is known, we would like to exploit it in order to find tighter guaranteed bounds: very regular meshes (valence 6 almost everywhere) results obviously in much smaller bit rates.

A quick analysis of the bit rate for [22] seems to lead to a non linear worst-case bit rate. Indeed, if we bound the number of splits by V (the number of vertices in the mesh), and each offset values by V also, we obtain a size of $V \cdot log_2(V)$ bits, resulting in $log_2(V)$ bits per vertex. However, this bound is far from being reasonable. Roughly speaking, if there is really a large number of splits (order of V), then they cannot lead to lists of size V, but rather of lists with a constant low vertex count. Inversely, if the list during a split is of size V, then there cannot be a large number of other splits, since we have already visited the entire mesh. This explains why the

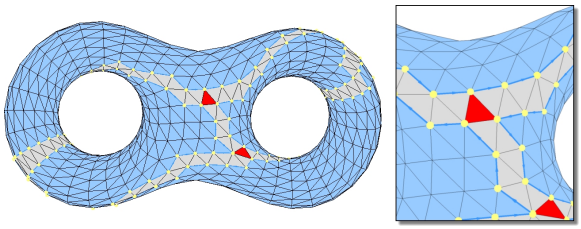

Figure 8: *Conquest on a genus-2 mesh. Only one active list has been required. Conquered area is colored in blue, while free area is colored in grey. Using* skip *codes builds a set of connected triangle strips, where each vertex is flagged* skipped *(colored in light-yellow). The transmission of the faces located at the cross of three strips (colored in red) is sufficient for the decoder to* zip *remaining strips at no cost.*

observed bit rate for our method is always between 0 and 3.4 b/v for any mesh with more than 100 vertices. Note that even the mesh given in [16] as being a worst-case example for the approach in [22] turns out to be not conclusive: their "snowflake" example does *not* create any splits when encoded. It does create splits on 50% of the vertices *only if* the recursive full pivot removal (detailed in section 3.2) is not used (this point is not obvious in the original paper). Therefore, both our approach and [22] performs optimally on such an Hamiltonian mesh. A pathological case for the number of splits remains elusive; our approach consisting in decreasing the number of splits is therefore particularly crucial in this context.

Nevertheless, we can *guarantee* theoretical upper bounds if the number of splits is supposed negligible. In the appendix, we show that maximizing the entropy of a mesh, under the constraint of the Euler's formula, leads to the *optimal* 3.24 bits per vertex required in theory [23]. This is the first time that Tutte's constant ($\log_2(256/27) = 3.24\ldots$) appears in the theoretical bounds of a triangle encoder, proving *a posteriori* the pertinence of the valence coding.

Additionally, we are able to bound the asymptotical worst bit-rate for meshes with known regularity: given for instance the percentage of valence-6 vertices in a mesh (regularity criterion defined by [18]), we obtain an explicit formula which ensures a bound decreasing to zero for regular meshes (see Figure 10). Notice that the lowest bit-rate known for the regular case was 0.811 b/v [18]. All the details about these bounds and their proofs are postponed until the Appendix of this paper. We believe that having such bounds on the bit rate is a very desirable feature of a mesh compression scheme. Notice the number of splits is very small for any object: we conjecture that the number of splits is sublinear. If this is true, then the number of splits along with the number of bits used to code the split index will *not change the optimality* of our technique.

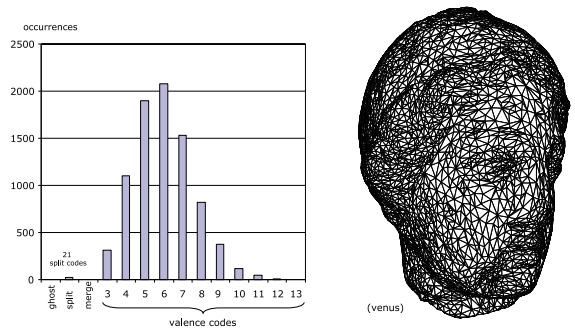

Figure 9: *Distribution of the code sequence generated during the traversal of the* venus *mesh. Entropy encoding benefits from the low dispersion of data around the average valence 6.*

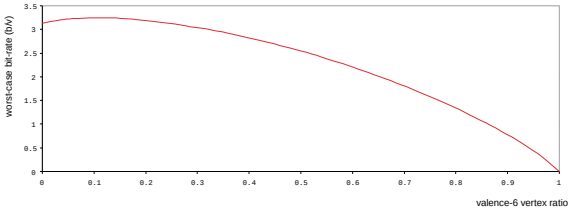

Figure 10: *Asymptotic behavior, function of the proportion of valence-6 vertices.*

4.8. Discussion

We have described a new connectivity encoding scheme, with both good bit rates in practice and good theoretical bounds. In essence, the proposed optimized conquest makes the behavior of the algorithm *adaptive* to the previously decoded data. We believe that it would also be applicable for several conquest-type algorithms. Similarly to [22], the decoding process is easily deduced from our encoding scheme. We also offer a theoretical analysis of our algorithm, showing that encoding only valences is optimal.

5. Results

Table 1 gives a comparison on several meshes shown in Figure 11, along with the associated numbers of *split* and *merge* codes. The connectivity of the *triceratops*, commonly used for comparison, compresses down to *1.88 b/v* with our technique while [20], [9], and [22] result in 4.3, 2.52, and 2.2 b/v respectively. This mesh is representative of the gain obtained on coarse irregular meshes. Figure 12 shows an example of a conquest on the David's head. It is also interesting to mention that: a) our technique is barely sensitive to the seed face, therefore a random face can be selected; b) running gzip or other standard file compression tools on our compressed data actually increases their size; c) use of higher order arithmetic encoding does not reduce the bit rate further. These three facts are a good sign that we are close to the actual entropy of the mesh connectivity. The current implementation of our encoding/decoding technique processes on average 20,000 faces/s on a regular PIII PC. Finally, the percentage of splits in our different examples never exceeds 0.9% for small meshes, and decreases with the number of vertices as expected.

6. Conclusions and future work

In this paper, we described a lossless, single-resolution connectivity encoding technique which results in the best compression ratios published so far. We also demonstrate that under the assumption of a negligible number of splits, we reach the optimal upper bound (3.24 b/v) for the bit rate per vertex for large, arbitrary meshes. Our method uses an adaptive conquest over the mesh to transform the mesh into a sequence of valences and few additional codes. This sequence is then processed by an adaptive arithmetic encoder that will use the low dispersion present in these valences to compress

Mesh	#F	#V	TG[22] b/v [#s;#m]	Ours b/v [#s;#m]
david 1	586	328	3.58 (2)	2.96 (1)
david 2	2924	1512	3.15 (21)	2.88 (7)
david 3	11820	6035	2.92 (87)	2.70 (43)
david 4	47753	24085	2.69 (255)	2.52 (138)
venus	16532	8268	2.82 (43)	2.71 (21)
foot	20028	10016	2.33 (14)	2.20 (3)
knot	7680	3887	0.035 (1;1)	0.024 (1;1)
body	1396	711	2.62 (3)	2.35 (0)
feline	99732	49864	2.38 (147;2)	2.27 (57;2)
mann. 1	839	428	2.97 (6)	2.51 (1)
mann. 2	23402	11704	0.46 (10)	0.37 (6)
dinosaur	28136	14070	2.39 (59)	2.25 (17)
nefertiti	562	299	2.83 (3)	2.37 (0)
fandisk	12946	6475	1.08 (6)	1.02 (0)
bunny	29783	15000	2.064 (11)	1.98 (5)
triceratops	5660	2832	2.20 (29)	1.88 (5)
femur	7798	3897	3.00 (73;2)	2.71 (38;2)

Table 1: *Compression results (in bit per vertex).* s *and* m *denotes the number of split and merge codes respectively.*

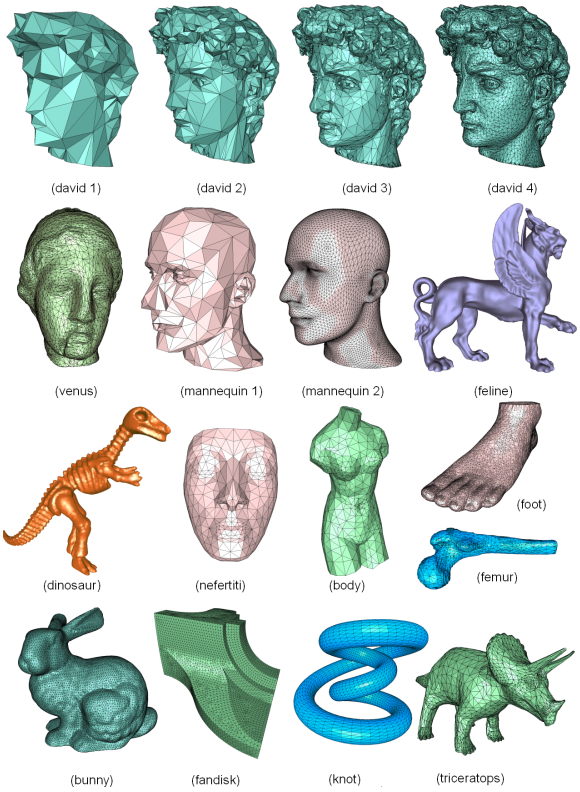

Figure 11: *3D meshes used for our bit-rate measurements.*

it. On large meshes like the David's head, we obtain compression rates of 188 to 1 compared to the connectivity part of a VRML ascii file.

We did not work on geometry compression at all in this paper. Current progressive techniques such as [12] seems to performed very well on uniform meshes, but more work may have to be done for irregular meshes if lossless geometry compression is sought. *Progressive* encoding, still in the context of lossless transmission, is also pertinent to optimize the rate-distortion behavior, as explained in [3, 14, 1]. Finally, genus reduction, error-resilience, polygonal meshes, non-manifold, even polygon soup encoding are also obvious future work. An executable code implementing our method can be found on the web at: `http://www-grail.usc.edu/SingleRateEncoder/`.

Acknowledgements

Many thanks to Michael Schindler for providing invaluable help about the *Range Encoder* [17] library, to Mark Meyer for reviewing the paper, and to Zoë Wood for hints on graph theory. The authors want to thank Peter Schröder and Wim Sweldens for initial discussions, and to Stephan Gumhold, Martin Isenburg, and Craig Gotsman for helping us improving this paper tremendously. The David's head mesh is courtesy of Marc Levoy and the Digital Michelangelo Project (Igor Guskov provided us with the 4 resolutions), while the other meshes are courtesy of Caltech, Stanford, Hugues Hoppe, and Renato Pajarola. This work has been partially supported by the Integrated Media Systems Center, a NSF Engineering Research Center, cooperative agreement number EEC-9529152.

References

1. P. Alliez and M. Desbrun. Progressive Encoding for Lossless Transmission of 3D Meshes. *Siggraph 2001 Conference Proceedings*, 2001.

2. C. Bajaj, S. Cutchin, V. Pascucci, and G. Zhuang. Error Resilient Streaming of Compressed VRML. Technical Report 98-25, Center for Computational Visualization, CS dept. and TICAM. University of Texas at Austin, 1999.

3. D. Cohen-Or, D. Levin, and O. Remez. Progressive Compression of Arbitrary Triangular Meshes. *Visualization 99 Conference Proceedings*, pages 67–72, 1999.

4. M. Deering. Geometry Compression. *Siggraph 95 Conference Proceedings*, pages 13–20, 1995.

5. O. Devillers and P-M. Gandoin. Geometric Compression for Interactive Transmission. *Visualization 2000 Conference Proceedings*, pages 319–326, 2000.

6. L. De Floriani, P. Magillo, and E. Puppo. A Simple and Efficient Sequential Encoding for Triangle Meshes. *15th European Workshop on Computational Geometry*, march 1999.

7. S. Gumhold. New Bounds on the Encoding of Planar Triangulations. Technical Report WSI-2000-1, Wilhelm Schickard Institute for Computer Science, Tubingen, march 2000.

8. H. Hoppe. Progressive Meshes. *Siggraph 96 Conference Proceedings*, pages 99–108, 1996.

9. M. Isenburg. Triangle Strip Compression. *Proceedings of Graphics Interface 2000*, pages 197–204, 2000.

10. M. Isenburg and J. Snoeyink. Mesh Collapse Compression. *Proceedings of ACM Symposium on Computational Geometry*, pages 419–420, 1999.

11. M. Isenburg and J. Snoeyink. Spirale Reversi: Reverse Decoding of EdgeBreaker Encoding. *12th Canadian Conference on Computational Geometry*, pages 247–256, 2000.

12. A. Khodakovsky, P. Schröder, and W. Sweldens. Progressive Geometry Compression. *Siggraph 2000 Conference Proceedings*, pages 271–278, 2000.

13. D. King and J. Rossignac. Guaranteed 3.67v bit encoding of planar triangle graphs. *11th Canadian Conference on Computational Geometry*, pages 146–149, 1999.

14. R. Pajarola and J. Rossignac. Compressed Progressive Meshes. *IEEE Transactions on Visualization and Computer Graphics*, 6:1:79–93, 2000.

15. J. Rossignac. EdgeBreaker : Connectivity Compression for Triangle Meshes. *IEEE Transactions on Visualization and Computer Graphics*, pages 47–61, 1999.

16. J. Rossignac and A. Szymczak. Wrap&zip decompression of the connectivity of triangle meshes compressed with EdgeBreaker. *Journal of Computational Geometry, Theory and Applications*, 14:119–135, november 1999.

17. M. Schindler. A Fast Renormalization for Arithmetic Coding. *Proceedings of IEEE Data Compression Conference, Snowbird, UT*, page 572, 1998. http://www.compressconsult.com/rangecoder/.

18. A. Szymczak, D. King, and J. Rossignac. An EdgeBreaker-based efficient compression scheme for regular meshes, 2000.

19. G. Taubin, A. Guéziec, W. Horn, and F. Lazarus. Progressive Forest Split Compression. *Siggraph 98 Conference Proceedings*, pages 123–132, 1998.

20. G. Taubin, W. Horn, F. Lazarus, and J. Rossignac. Geometry coding and VRML. *Proceedings of the IEEE*, 86(6):1228–1243, 1998.

21. G. Taubin and J. Rossignac. 3D Geometry Compression, 1999-2000. ACM Siggraph conference course notes.

22. C. Touma and C. Gotsman. Triangle Mesh Compression. *Graphics Interface 98 Conference Proceedings*, pages 26–34, june 1998.

23. W. Tutte. A Census of Planar Triangulations. *Canadian Journal of Mathematics*, 14:21–38, 1962.

24. I.H. Witten, R.M. Neal, and J.G. Cleary. Arithmetic Coding for Data Compression. *Communications of the ACM*, 30(6):520–540, june 1987.

25. Y. Yemez and F. Schmitt. Progressive Multilevel Meshes from Octree Particles. *Proceedings of 2nd International Conference on 3-D Imaging and Modeling*, pages 290–299, 1999.

Appendix: Theoretical Results

We consider a 2-manifold mesh with V vertices, F faces and E edges, and the following usual assumptions:

- the mesh has no boundary and no holes, i.e. every edge has two adjacent faces,
- the mesh is topologically equivalent to a sphere, i.e., has a genus of zero.

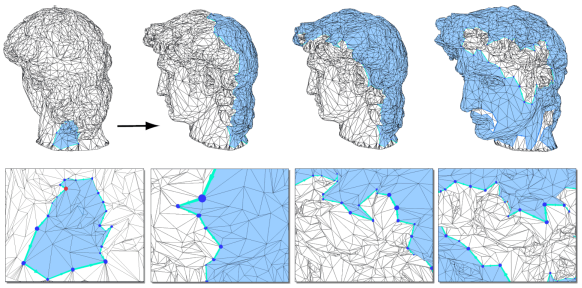

Figure 12: *Conquest behavior on the David mesh.*

Entropy of a code sequence

The entropy denotes the minimal number of bits required per symbol for lossless encoding of a given code sequence. It is both a function of the number N of *distinct* symbols and to their respective *probability* p_i in this particular sequence:

$$entropy = \sum_{i=1}^{N} p_i log_2 \frac{1}{p_i} \qquad (2)$$

This formula indicates that the final rate of our connectivity encoding technique is intricately related to the range and the dispersion of valences, i.e., their statistical distribution.

Valence distribution for arbitrary meshes

In a triangular mesh without boundary, the possible range of valences is $[3;+\infty]$. Obviously, the number of *distinct* valences in the mesh is bounded by the vertex count. Using the Euler formula: $F+V-E=2$, we can derive the expression for the sum of valences. If we count three edges for each face of the mesh, we get $3F$. Since each edge is shared by exactly two faces, this counts the edges twice, we thus obtain $3F=2E$. If one substitutes this equality in the Euler formula, it yields: $V-E+2E/3=2 \Longleftrightarrow E=3V-6$. The sum of valence being equivalent to twice the number of edges, we obtain:

$$\sum valence = 2E = 6V - 12. \qquad (3)$$

The average valence for an arbitrary mesh is therefore 6. Equation 3 indeed implies that numerous vertices are of low valence (3 to 6), balancing for the valences higher than the average valence 6 for triangular meshes. Intuitively, a vertex with valence > 6 must have one or several associated low valence vertices to compensate. The minimum valence being equal to 3, the presence of high valences vertices implies that the probability of valence < 6 vertices increases dramatically, promoting efficient entropy encoding. As an example, one vertex with a high valence of $V-1$ constrains an average valence about 5 for the $V-1$ remaining vertices, and two vertices with valence $V-2$ constraints an average valence of 4 elsewhere.

Worst Asymptotical Bit Rate vs. Tutte's enumeration

From Equation 2, it is easily seen (and well known) that the worst compression ratio is encountered for both the maximum number of distinct symbols and an equal probability

for each of them. This would lead to a mesh where each vertex has a distinct valence and thus appears only once in the distribution. Fortunately, this case is impossible as previously explained, due to the average-6 valence.

Let's have a look at the maximum entropy of a mesh. For now, we will consider that our conquest happens *without* any skip or split codes. We call p_i the percentage of vertices of valence i in the worst-case mesh, for i from 3 to ∞. The set of p_i therefore represents the distribution of valence that leads to the worst bit rate through mesh encoding using our technique.

Since we are dealing with a mesh, we must satisfy two basic constraints: all the percentages must sum to one, and the valence average must be 6 (see above). It yields:

$$\sum_3^\infty p_i = 1, \quad \sum_3^\infty i \cdot p_i = 6. \tag{4}$$

Now, we must find what valence distribution p leads to the worst bit rate: we have to maximize the following infinite sum:

$$\sum_3^\infty p_i \cdot log_2 \frac{1}{p_i} \tag{5}$$

under the previously mentioned two constraints. After rewriting this optimization under constraints using Lagrange multipliers, p must maximize the following function f:

$$f(p_3, p_4, ..., \lambda, \mu) = \sum_3^\infty p_i log_2 \frac{1}{p_i} + \lambda(\sum_3^\infty p_i - 1) + \mu(\sum_3^\infty i \cdot p_i - 6) \tag{6}$$

where λ and μ are two Lagrange multipliers. Derivatives of f with respect to each of the unknown percentages p_i must be zero. It thus implies, for each i, that: $\log_2(p_i) = \lambda - 1 + \mu * i$. Therefore, we can rewrite p_i as:

$$p_i = \alpha e^{-\beta i}. \tag{7}$$

We then rewrite the constraints given by equation 8 as:

$$\sum_3^\infty p_i = \alpha \sum_3^\infty e^{-\beta i} = 1, \quad \sum_3^\infty i \cdot p_i = \alpha \sum_3^\infty i \cdot e^{-\beta i} = 6. \tag{8}$$

We now solve for α and β, knowing the two following closed form solutions (when β is strictly positive):

$$\sum_3^\infty e^{-\beta i} = \frac{e^{(-2\beta)}}{e^\beta - 1} \tag{9}$$

$$\sum_3^\infty i e^{-\beta i} = \frac{e^{(-2\beta)}(-3e^{(2\beta)} + 5e^\beta - 2)}{(e^\beta - 1)(-e^{(2\beta)} + 2e^\beta - 1)}. \tag{10}$$

The unique solution is: $\alpha = 16/27$ and $\beta = -ln(3/4)$.

If we now compute the maximum entropy e induced by the above worst-case distribution:

$$\begin{aligned} e &= \sum_3^\infty p_i log_2(\frac{1}{p_i}) = -\sum_3^\infty \alpha e^{-\beta i} log_2(\alpha e^{-\beta i}) \\ &= -log_2(\alpha) \sum_3^\infty \alpha e^{-\beta i} + \frac{\beta}{ln(2)} \sum_3^\infty i \alpha e^{-\beta i} \\ &= -log_2(\alpha) + \frac{6\beta}{ln(2)}. \end{aligned} \tag{11}$$

Substituting α and β in this last equation leads to an entropy of:

$$e = log_2(\frac{256}{27}) \simeq 3.24511249784. \tag{12}$$

This is exactly the number of bits needed to encode each vertex of an arbitrary planar mesh as found by Tutte [23] by a pure enumeration of all possible planar graphs. In the case where no split/skip codes occur, we thus have proved that **our valence-driven encoding is asymptotically optimal**, with a rate of 3.2452 b/v guaranteed. This also proves, a posteriori, that [22] was indeed optimal under the same assumption of no split codes.

Regular Meshes

As [18] proposed, we can also measure the regularity of a mesh using its fraction of valence-6 vertices, denoted S. By simply adding a constraint on p_6 in the previous proof, we now find β unchanged and

$$\alpha = \frac{4096}{6183}(1 - S) \tag{13}$$

Figure 10 illustrates the worst-case asymptotic behavior, with an expected zero entropy when the mesh is perfectly regular.

This has led us to use, in practice, a similar bit rate prediction formula, where we simply consider the number of split codes to be zero (negligible): from Equation 2 and knowing the exact valence distribution of a mesh, it is now straightforward to predict the bit-rate of the connectivity (see table 2) rather accurately. This works very well for complex 3D meshes, since the range encoder converges to the final entropy for a sufficiently large number of encoded symbols, as demonstrated in the following table. In all our tests, we always found the prediction to be in agreement with the measured bit rate within less than 2%.

Mesh	#V	Predicted	Measured	Diff.
venus	8268	2.67 b/v	2.71 b/v	1.49 %
feline	49864	2.25	2.27	0.88 %
david 3	6035	2.64	2.70	2.27 %
david 4	24085	2.48	2.52	1.61 %
dinosaur	14070	2.22	2.25	1.35 %

Table 2: *Prediction of bit-rates from valence distribution.*