An Edgebreaker-Based Efficient Compression Scheme for Regular Meshes

Andrzej Szymczak, Davis King and Jarek Rossignac

GVU Center,
Georgia Institute of Technology,
Atlanta, GA 30332, USA.

{andrzej,kingd,jarek}@cc.gatech.edu

June 27, 2000

Abstract

One of the most natural measures of regularity of a triangular mesh homeomorphic to the two-dimensional sphere is the fraction of its vertices having degree 6. We construct a linear-time connectivity compression scheme build upon Edgebreaker which explicitly takes advantage of regularity and prove rigorously that, for sufficiently large and regular meshes, it produces encodings not longer than 0.811 bits per triangle: 50% below the information-theoretic lower bound for the class of all meshes. Our method uses predictive techniques enabled by the Spirale Reversi decoding algorithm.

1 Introduction

Geometric data is typically represented by meshes, often triangular. Frequently, there is need to access such data via a network connection and, in such cases, bandwidth tends to become a serious obstacle to interactivity. An obvious way out of this problem is to use compressed representations.

The standard representation of a triangular mesh consists of two parts: connectivity and vertex coordinates and properties. If stored in uncompressed form, connectivity is typically more expensive. In this paper we are concerned with bit-efficient encodings of connectivity of triangular meshes which can be produced as well as decoded in linear time. There have been two interlaced threads of related research activity. One attempts to build compression algorithms with good compression ability for triangular meshes appearing in practice [13],[3],[12]. The goal of the other is to invent algorithms which have good worst-case characteristics, i.e. which guarantee encoding sizes of certain number of bits per triangle for simple (homeomorphic to the 2D sphere) meshes. Examples include [14],[9],[5],[4],[11],[10],[2]. The last three references exemplify the recent efforts to analyse the algorithms invented and tested for practical applications in terms of worst-case performance. Remarkably, the bound of 1.78 bits per triangle from [2] is the best worst-case bound for a linear compression scheme proved so far.

The ultimate limitation of the worst-case analysis is the information-theoretic lower bound on the compression ratio which follows from the enumeration results of [15]: $4-1.5\log_2 3 \approx 1.623$ bits per triangle. This is very close to the worst-case bounds for the algorithms already analysed (note that an $O(n \log n)$ algorithm able to approach the information-theoretic lower bound is known [4]) and, at the same time, much more than the experimentally measured performance of the state of art compression schemes (see e.g. [13],[3]), which usually produce compressed representations with sizes of about 0.5–1 bits per triangle. This raises a doubt whether worst-case analysis in the form exercised until now is an adequate tool for assessing connectivity compression algorithms and explaining their measured performance and whether it is able to provide correct cues helping to improve them.

The experimental measurements mentioned above show that, in practice, the best connectivity compression algorithms are able to take advantage of regularity of the input mesh to bring the compressed size much below the information-theoretical lower bound. In this paper we present the first (up to our knowledge) attempt to quantify rigorously the effect that regularity of a mesh has on the compression rate. It is a well-known consequence of the Euler's formula that the average degree of a vertex in a large simple mesh is close to 6. Thus, one can expect that a typical mesh has a lot of vertices of degree 6. This is indeed the case for many 3D models, including the ubiquitous 35947-vertex Stanford bunny model, whose 75.9% vertices have degree 6. This motivates treating the fraction of degree-6 vertices as a measure of regularity. We construct a compression scheme which explicitly takes advantage of regularity and prove that, for sufficiently large and regular meshes, it produces encodings of size not exceeding 0.811 bits per triangle: over 50% below the information-theoretical lower bound mentioned above. An important feature of our algorithm is that it is extremely simple to implement, since it is a combination of the Edgebreaker compression algorithm [11], Spirale Reversi decompression algorithm [7] and arithmetic coding [16].

Clearly, our choice of a measure of regularity is highly disputable: there are many other notions of regularity one can think of. Besides, our argument still does not fully explain why the state of art compression schemes perform so well in experimental tests (e.g. for models having 25% vertices of degrees different from 6, about as much as in the Stanford bunny model, we can only prove that our algorithms approach 1.75 bits per triangle, while the experiments in [12] led to compression rates below 1 bit per triangle). Investigation of other regularity measures which better model the structure of meshes encountered in practice and their impact on performance of various compression schemes is an interesting topic for future research. We also hope that our ideas can be turned into practice – this will require efforts to decrease the size of terms negligible in asymptotic analysis and organizing the information in a way helping general purpose compression algorithms to uncover regularity which has not been explicitly described yet.

status (state of 'discovered' flags)			action	
$T_{\mathrm{right}(h)}$	$T_{\mathrm{left}(h)}$	tip(h)	symbol output	half-edges pushed (in order)
FALSE	FALSE	FALSE	C	right(h)
FALSE	FALSE	TRUE	S	left(h), right(h)
TRUE	FALSE	TRUE	R	left(h)
FALSE	TRUE	TRUE	L	right(h)
TRUE	TRUE	TRUE	E	none

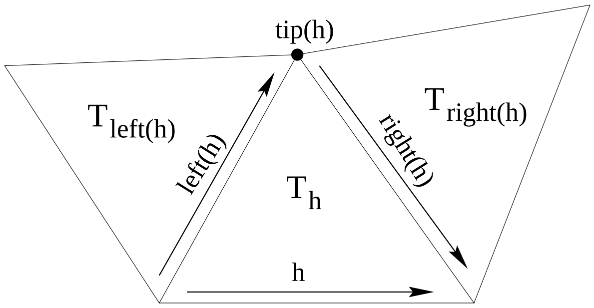

Figure 1: Edgebreaker encoding: actions for a given status and notation for a neighborhood of a half-edge of h

2 Edgebreaker and Spirale Reversi

In this section we briefly recall the principles of Edgebreaker [11] and Spirale Reversi style of reconstructing the encoded mesh from the encoding string [7]. We are going to assume that the meshes we are dealing with are *simple* i.e. are triangulations of the two-dimensional sphere. Equivalently, they are manifold triangle meshes with no boundary nor handles.

2.1 Edgebreaker encoding

The Edgebreaker encoding procedure transforms a given input mesh into a string of five symbols from the set {C,L,E,R,S}. The symbols are in one-to-one correspondence with the triangles of the mesh. The order of symbols is defined by a depth-first search traversal of the mesh. Roughly speaking, each of the symbols encodes whether certain mesh elements adjacent to its corresponding triangle have been discovered before that triangle is first visited during the traversal. Below we state the algorithm in full detail using the half-edge representation of the input mesh.

As an auxiliary data structure we use a stack of half edges. We also equip each of the vertices and triangles of the mesh with a binary flag indicating whether it has been discovered or not.

At startup, all flags are set to FALSE and the stack is initialized to hold one arbitrarily chosen half edge. Then, until the stack is not empty, we pop a half-edge h and, depending on the state of the 'discovered' flags of the triangles $T_{\mathrm{right}}(h)$, $T_{\mathrm{left}}(h)$ and the vertex tip(h) (see Figure 1) we output one of the symbols in $\{C, L, E, R, S\}$ and push one, two or none of the half-edges right(h), left(h) on top of the stack. The required actions are shown in Figure 1. Before closing the loop, we mark the triangle T_h and all of its vertices as discovered.

The reader is referred to [11] for details. Let us only indicate that, for the understanding of the encoding and decoding process it is important to see how the structure of the undiscovered portion of the mesh evolves during encoding. It should be clear that, at each stage of the algorithm, the union of all discovered triangles is a connected set. Since the mesh is a triangulation of a 2D sphere, the complement of that set is a finite union of two dimensional disks. A careful inspection of the algorithm reveals that:

- Each time an S symbol is produced, the number of components of the undiscovered part increases by one (more precisely, an S-type triangle 'splits' its component).

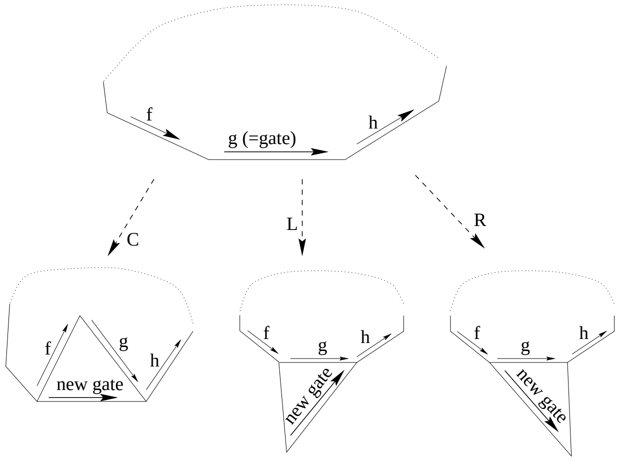

Figure 2: Changes to the mesh on stack caused by the C, L and R symbols

- With each E symbol the number of such components decreases by one. Namely, a component consisting of one triangle disappears.

- The statement 'for each connected component of the complement of the discovered part there is *exactly* one half-edge on its bounding loop which is also on the stack' is an invariant of the encoder's main loop.

2.2 Spirale Reversi decoding

The Spirale Reversi decoding process essentially follows the execution path of the Edgebreaker encoding procedure backwards. It scans the encoding string starting from the last symbol and, for each of the symbols read, performs an operation on a stack of meshes. The meshes on stack are triangulations of connected components of the complement of the discovered part of the mesh in the analogous moment of time during compression. Their order on stack corresponds to that of half-edges on stack while encoding the mesh. The operation associated with each of the symbols 'undoes' the effect that it has on the undiscovered part during compression. The details are given below.

At startup, the stack is empty and at termination it contains just one item - the reconstructed mesh identical to the encoded one. Each of the meshes on the stack has an associated special external (i.e. not having an opposite) half-edge. That special half-edge will be called its *gate*. Gates correspond to the half-edges placed on stack during compression. Symbols C,L,R generate operations on the mesh on top of stack and do not cause the number of items on stack to change. The effect of those operations is shown in Figure 2. Each of them adds a single triangle to that mesh but they differ in how that triangle is 'glued' to it or how

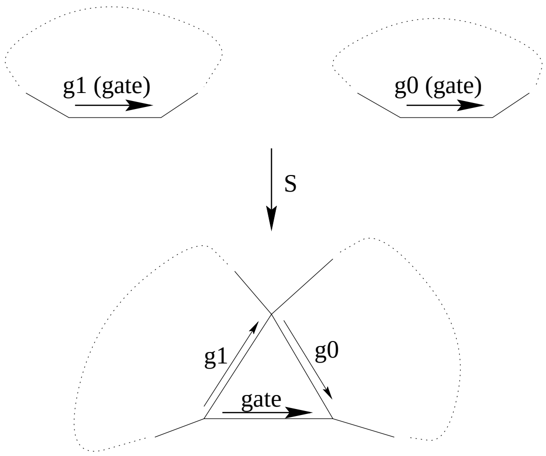

Figure 3: S operation: the mesh on top of stack (with gate $g0$) is glued to the right edge of the new triangle, and the second on top (with gate $g1$) - to its left edge; the bottom edge becames the gate of the resulting mesh which replaces the former two on stack

the new gate is assigned to the transformed mesh. An E symbol causes a new one-triangle mesh be pushed on stack. Therefore it increases the number of meshes on stack by one. An S symbol causes two meshes on top of stack to be popped and glued to form a larger mesh, which is then pushed on the stack (Figure 3).

2.3 Binarization of the CLERS sequence

The CLERS sequence produced by Edgebreaker has to be converted into a binary string. Several ways of doing that, taking advantage of the symbol frequencies or dependencies have been proposed. The original method suggested in [11] is based on the observation that the frequency of C's is always equal to 50%. Thus, it is natural to use a Huffman code with 1 bit for a C and 3 bits for any of the other 4 symbols. This leads to encoding sizes of 2 bits per triangle. Improvements of that scheme can be found in [10] (1.83 bits per triangle) and [2] (1.78 bits per triangle).

3 Predicting C's

The effect that a C symbol has on the decoded part of mesh during decompression makes it possible to predict it with probability proportional to the regularity of the encoded mesh. It is clear from Figure 2 that a C causes the starting vertex of the gate to become an internal vertex of the mesh on top of stack. Therefore, no triangles incident to that vertex are added later during decoding process. Assume that the meshes to be encoded are expected to have a large fraction of degree-6 vertices. If this is the case, we can attempt to predict a C based on the number of triangles (in the already decoded part of the mesh) incident upon the starting point of the gate. Namely, if the number of such triangles is 5, a C is likely to occur; if it is not, we would rather expect some other symbol. This naturally leads to the idea of encoding the CLERS sequence as the LERS sequence obtained from it by skipping all C's (thus, of length $t/2$, t being the number of triangles) and a binary *hit/miss sequence* of length t whose entries indicate whether the prediction described above is correct or not. Of course, the hope is that the prediction it is correct most of the time so that the hit/miss sequence consists of mostly 1's and therefore can be greatly compressed using entropy coding. Below we argue that this is indeed the case.

A prediction attempt is based on the number of triangles in the decoded part of the mesh incident upon the starting vertex of the gate. Let us charge that vertex with a possible prediction failure. It is clear that vertices of degree 6 are not responsible for any prediction failures. Vertices of degree less than 6 are responsible for at most one failure. To see this, notice that we never predict a C when such a vertex is a starting point of a gate. Thus, a miss means that we have a C at such a vertex. As a result, it becomes an internal vertex and therefore is never a starting point of a gate afterwards. In particular, we can never fail at that vertex again. A similar argument shows that one cannot miss more than twice at a vertex of degree greater than 6. One miss at such a vertex may happen when it is the starting point of the gate and the number of its incident triangles is 5 (note that it does not have to happen for every high degree vertex). Just as for low degree vertices, the next miss can only be caused by a C and hence makes that vertex internal, allowing no further misses. To sum up, the number of misses cannot be greater than $n_{<6} + 2n_{>6}$, where $n_{>6}$ and $n_{<6}$ denote the numbers of vertices of degree greater and less than 6 (respectively). Asymptotically, the average degree of a vertex is equal to 6. Since all vertices have degree greater than 2, this means that each three vertices of degree greater than 6 require at least one vertex of degree less than 6 to balance their degree surplus. Thus, at least 25% of all vertices of degree different than 6 have degrees less than 6. As a result, the number of misses can be bounded by $1.75n_{\neq 6}$, where $n_{\neq 6}$ is the number of vertices of degree different from 6. The hit/miss sequence of a regular mesh can therefore be entropy encoded using close to

$$-0.875f \log_2(0.875f) - (1 - 0.875f)\log_2(1 - 0.875f).$$

bits per triangle, where $f = 2n_{\neq 6}/(t + 4) \approx 2n_{\neq 6}/t$ is the fraction of the vertices of degrees different than 6. The above statement holds if $0.875f \leq 0.5$ or $f \leq 4/7$. Otherwise, 1 bit per triangle may be required. The LERS sequence (recall that its length equals half the number of triangles) can be encoded with 1 bit per triangle by using a 2 bit code for each of the four symbols. We sum up the results of this section in the following theorem.

Theorem 3.1 *A large enough mesh with t triangles can be encoded using arbitrarily close to*

$$M(f) := \begin{cases} \begin{array}{c} 1 - 0.875f \log_2(0.875f) - \\ (1 - 0.875f)\log_2(1 - 0.875f) \end{array} & \text{if } f \leq 4/7 \\ 2 & \text{otherwise.} \end{cases}$$

bits per triangle, where f stands for the fraction of its vertices which have degree different from 6. For any $\epsilon > 0$, encoding and decoding procedures achieving $M(f) + \epsilon$ bits per triangle encoding sizes for large enough meshes can be implemented so that they run in linear time.

Linear-time implementations of the coding and decoding procedures can be obtained by employing arithmetic coding to compress the hit/miss sequence. The decoding procedure requires almost no change except for maintaining and testing a counter of incident triangles for each vertex. The coding procedure must evaluate whether the predictor hits or misses for each produced symbol. The brute force (but linear time) solution is to make the decoder a part of the encoder and simulate the decoding process after building the CLERS string. This requires two mesh traversals. A more elegant way requiring only one pass is to decide the success of the prediction based on the number of undiscovered triangles incident to the starting point of the half-edge popped from the stack in each iteration of the encoder's main loop.

Figure 4 shows the graph of the bound on the encoding size for our scheme as a function of the mesh regularity. The graph also shows that in the limit of regularity, the compression rate approaches 1 bit per triangle – almost 40% less than the information-theoretical lower bound on the performance of any compression algorithm in the general case.

3.1 Example and discussion

The ubiquitous 35947-vertex Stanford bunny model available from the Stanford 3D Scanning Repository `http://graphics.stanford.EDU/data/3Dscanrep/` has about 75% vertices of degree 6. Let us consider a simple mesh having that percentage of vertices of degree 6, i.e. one with $f = 0.25$. Theorem 3.1 claims that our algorithm is able to encode such a mesh using less than $M(f) \approx 1.758$ bits per triangle. However, it holds for sufficiently large meshes and states that it does not really guarantee $M(f)$ bits per triangle but has some overhead ϵ which can be made arbitrarily small. Below we discuss the origin and magnitude of this asymptotically negligible term.

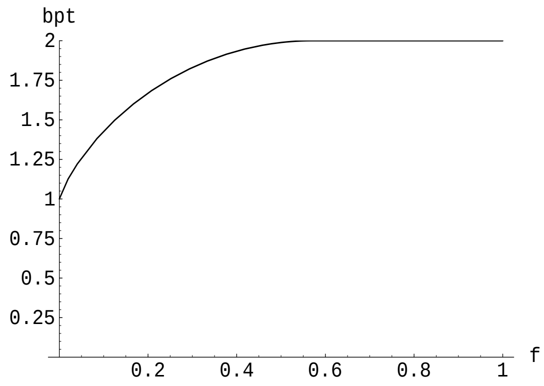

Figure 4: The encoding cost (in bits per triangle) which can be approached by the algorithm of Section 3 as a function of f

The need for ϵ arises from the fact that efficient implementations of arithmetic coding do not use exact rational arithmetics but rather perform calculations using integer arithmetics in some fixed range. Before stepping into details, let us recall that the 'idealized' arithmetic coding algorithm starts with the interval $I_0 = [0, 1)$ and builds a nested sequence of intervals based on the input string (in our case, binary). Assume that the probability of a '0' is $p \in [0, 1]$. Then, the probability of a '1' is $1-p$. For the j-th symbol of the input string, the interval I_{j-1} is split into two intervals, open on the right and closed on the left, with length ratio $p : 1-p$ and either the first or the second one (depending on whether the j-th symbol of the input string is 0 or 1) is chosen to be I_j. The final interval I_N, N being the number of symbols, has length $l(p, N) = (p^p(1-p)^{(1-p)})^N$ and therefore can be encoded using $L = \lceil -\log_2 l(p, N)\rceil$ bits as the numerator of a quotient $s/2^L$ belonging to I_N. Additional bits have to be used to encode the frequencies of symbols (e.g. as the pair of integers N and pN), but that extra cost is bounded by $3\lceil \log_2 N\rceil = o(N)$ bits. However, using exact arithmetic one cannot compute the interval I_N in linear time as the numbers used to represent the endpoints of I_j's grow in size with each step of the algorithm. A way out of this problem (see [6] or [16] for a discussion) is to approximate each of the intervals I_j with a subinterval of $[0, 1]$ whose endpoints are quotients with equal denominators being powers of 2 and numerators are B-bit integers differing by more than 2^{B-2}. This allows to compute I_j from I_{j-1} by means of a finite number of integer arithmetic operations on $O(B)$-bit unsigned integers, but, because of the necessity of rounding the split point causes the ratio of the lengths of I_{j+1} and I_j to be slightly different from p or $1-p$ (as in the idealistic variant), the difference being bounded by 2^{2-B}. A straighforward calculation being a special case of the results of [6] shows that the extra encoding cost caused by rounding can be bounded by $2^{2-B}(B-2) + 2^{3-B}$ bits per triangle. This means that the bound on the length of the entire encoding is

$$M(f) + 2^{2-B}(B-2) + 2^{3-B} + 3\lceil \log_2 t\rceil/t$$

bits per triangle (where t stands for the number of triangles). Thus, for realistic values of $B = 16$ and $t \geq 10000$, the encoding cost is bounded by $M(f) + 0.006$.

4 Better results for asymptotically regular meshes

In this section we show how to improve the asymptotic compression ratio for large and highly regular meshes. So far we have shown that, if the size of the mesh tends to infinity and the fraction of its vertices having degree 6 to 1, then the cost of encoding C's tends to 0 bits per triangle. This result was obtained using a predictor based on the degree of the starting vertex of the gate. In this section we will show how to use the degree information for the endpoint of the gate to further reduce the encoding size.

The idea is to split the encoding into three parts. The first one is the hit/miss sequence for the predictor described in the previous section, asymptotically free to encode. The second is a binary sequence distinguishing C,L,S from E,R (called the CLS/ER sequence later on). Finally, the third sequence is a binary string which resolves all the doubt left, i.e. allows to tell L from S or E from R (we shall call it LE/SR). Since C's are half of all symbols, the third string can be encoded using 0.5 bits per triangle. Below we describe an efficient way of encoding the CLS/ER string. It should be clear that, for the decoder, the CLS/ER string and the other two sequences do not carry disjoint information. The simplest reason is that any correct prediction for a C obviously has to be reflected in the CLS/ER string. Thus, an efficient scheme for encoding the CLS/ER sequence has to capture and encode the *difference* of the information provided by that sequence and the hit/miss and LE/SR strings. In order to construct such a scheme, we need to take a closer look on the dependencies between symbols within the CLS/ER sequence.

4.1 Structure of the CLS/ER sequence

The CLS/ER sequence is obtained from the CLERS sequence by replacing each C,L and S symbol with a '0' and each E and R with a '1'. To justify distinguishing C,L,S from E or R, observe that symbols in different groups have different effects on the end vertex of the gate during decompression. Namely, a C,L or S do not change it while E and R move the gate away from that vertex. Below we take a closer look at the influence that long CLS substrings have on the degree distribution of vertices of the reconstructed mesh.

The decompression procedure terminates with the stack holding one mesh with a gate. Let v be a vertex of the encoded mesh which does not bound that final gate. At a particular moment during decompression there may be several vertices in the reconstructed part of the mesh which correspond to v. Any such vertex will be called an *instance of* v. By k_v we shall denote the number of all C,L or S operations executed when an instance of v is the end vertex of the gate of the mesh on top of stack (such operations will be called *significant* later on). We shall show that the degree of v in the final mesh is at least $k_v + 4$.

Let us follow the increase of the total number N of all triangles incident upon all instances of v. Just before the first significant C,L or S operation is executed, N is at least 1 and each such operation increases N by 1. Thus, just after the last significant operation is performed, $N \geq k_v + 1$. Let v_* be the endpoint of the gate at this moment. The next symbol must be an R or E. If it is an R, N increases by 1 and v_* becomes an external vertex of the mesh on top of stack which does not bound the gate. If it is an E, v_* becomes a vertex of the mesh on top of stack as a result of some S operation executed later on. Notice (Figure 3: v_* is the end vertex of the gate of the mesh on the left) that this makes v_* an external vertex which does not bound the gate and increases its degree by at least two. To sum up, at some moment after all the significant operations are executed, $N \geq k_v + 2$ and there is an instance v'_* of v which is an external vertex of the mesh on top of stack which does not bound the gate. Since v does not bound the final gate, it has to be made internal by some C operation executed when its instance is the starting point of the gate. Such a C increases N by 1. So does the preceding operation (since v'_* did not bound the gate, this operation has not been counted yet). Thus, in the end, $N \geq k_v + 4$.

Let us define the *degree excess* of the mesh as the sum of degrees of all vertices of degree greater than 6 decreased by six times the number of such vertices. Since the sum of degrees of all vertices is equal to $6v - 12$, a degree excess of x requires at least $(x + 12)/3$ vertices of degree less than 6 to balance it. Since the number of vertices of degree different from 6 is assumed to be $o(t)$, so is the degree excess. The argument given in the previous paragraph proves that any maximal substring of the CLS/ER string consisting of $k > 2$ zeroes increases the degree excess by $k - 2$ (since all C,L and S operations which induce it are significant relative to the same vertex). Thus, the total length of long (i.e. of length ≥ 3) maximal strings of zeroes is $o(t)$. This property will play the key role in our encoding scheme for the CLS/ER sequence. To make the above analysis fully precise we have to deal with long sequences of zeros corresponding to CLS subsequences executed when the gate's endpoint is one of the endpoints of the final gate. However, this is not a problem since their total length is bounded by the contribution of these two vertices to the degree excess plus $O(1)$ and thus is $o(t)$.

4.2 Binarization of the CLS/ER string

We shall gradually simplify and remove correlations of the CLS/ER string with the hit/miss and LE/SR sequences. In the end we shall obtain a string with a short encoding which encapsulates the difference of information carried by the CLS/ER string and the hit/miss and LE/SR strings.

4.2.1 Long sequences of zeros

As we have already proved, the total length of all maximal long substrings of zeros is $l = o(t)$. Thus, locations of all zeros which occur in such substrings can be encoded using $o(1)$ bits per triangle as a separate entropy-coded binary sequence. We shall therefore skip all such zeros, obtaining the *simplified CLS/ER sequence* with no long zero substrings. Note that the length of this sequence is $t' = t - l = (1 - o(1))t$ and it still has $z' = t/2 - l = (1 - o(1))t/2$ zeros in positions corresponding to positions of C's in the CLERS sequence.

4.2.2 Skipping *CC's

The decoder can deduce every C from the already encoded part of mesh and the hit/miss sequence. Each CC combination which is not a part of a longer CLS substring induces a 00 combination in the simplified CLS/ER sequence. Such a combination, in turn, has to be preceded by a '1' (it is also followed by a '1', but it is not clear how the decoder can take advantage of it since it proceeds backwards). All three bits are redundant, so we shall omit them. This results in a binary sequence of length $t'' = t' - 3p$ having at least $z'' = z' - 2p$ zeros corresponding to C's in the CLERS sequence (but typically having other zeros as well), where p is the number of omitted 3-bit substrings.

4.2.3 Skipping C's and encoding the reminder as a UVW sequence

The sequence σ which we are left with does not contain any triple of consecutive zeros. Therefore, it is a concatenation of *elementary strings* '1', '01' and '001' and can be treated as a string of three symbols, U,V,W, corresponding to those elementary strings. The sequence σ' which is obtained from σ by skipping zeros corresponding to C's in the original CLERS sequence (notice that the decoder can deduce the positions of such

C's using the hit/miss sequence and the decoded part of the mesh) also has the same property. Since each zero of σ corresponding to a C is in a different maximal substring of zeros (recall that we skipped CC's in the previous stage), the decomposition of σ' into elementary strings has at least as many '1' and '01' terms as there are C's inducing zeros in σ, i.e. at least $z' - 2p$. Thus, if u, v and w are the numbers of U,V and W symbols in the UVW encoding of σ' then $u+v \geq z'-2p = t/2-l-2p$. Another useful constraint on u,v and w follows from the fact that the length of σ' is $t'' - z'' = t - l - 3p - (t/2 - l - 2p) = t/2 - p$. Therefore, $u + 2v + 3w = t/2 - p$. Combining the two constraints yields one which is easier to work with: $u + 3v + 6w \leq t/2 + l$

4.2.4 Binarization of the UVW sequence

We shall turn the UVW sequence into a binary string using entropy coding. The encoding size is (with $o(t)$ terms left out),

$$-u\log_2 u - v\log_2 v - w\log_2 w + (u+v+w)\log_2(u+v+w),$$

where u, v and w have to satisfy the constraint $u + 3v + 6w \leq t/2 + l$. Now, letting $\bar{u} = u/t$, $\bar{v} = v/t$ and $\bar{w} = w/t$ we see that the number of bits per triangle required to encode the UVW sequence which a fixed precision arithmetic coder can approach arbitrarily close can be obtained by solving the following maximization problem (since $l = o(t)$, the objective function is continuous and we are interested in the asymptotic behavior of the solution, the term l/t can be omitted).

$$\begin{array}{l} \text{maximize:} \\ \quad S(\bar{u},\bar{v},\bar{w}) = (\bar{u}+\bar{v}+\bar{w})\log_2(\bar{u}+\bar{v}+\bar{w}) - \\ \qquad\qquad \bar{u}\log_2\bar{u} - \bar{v}\log_2\bar{v} - \bar{w}\log_2\bar{w} \\ \text{subject to:} \\ \quad \bar{u}+3\bar{v}+6\bar{w} \leq 1/2 \\ \quad \bar{u},\bar{v},\bar{w} \geq 0 \end{array} \tag{1}$$

Numerical exploration with *Mathematica* made us suspect that the solution is close to $(\bar{u},\bar{v},\bar{w}) \approx (0.168717, 0.0712798, 0.0195739) =: (\bar{u}_0,\bar{v}_0,\bar{w}_0)$, which yields an upper bound on the objective function in the feasible region of about 0.310761. This indicates that the UVW string can be compressed using less than 0.311 bits per triangle, but is not a rigorous statement yet. Below we make it rigorous.

It is easy to see that S is strictly increasing in directions parallel to any of the $\bar{u}$,$\bar{v}$ and $\bar{w}$ axes (since all its partial derivatives are positive). It follows that the solution to (1) lies on the plane p given by $\bar{u}+3\bar{v}+6\bar{w} = 1/2$ (if it would not, we could move it slightly along one of the axes to get a feasible point with a larger value).

The Hessian of S, i.e. the matrix

$$\begin{bmatrix} \frac{\partial^2 S}{\partial \bar{u}^2} & \frac{\partial^2 S}{\partial \bar{u}\partial \bar{v}} & \frac{\partial^2 S}{\partial \bar{u}\partial \bar{w}} \\ \frac{\partial^2 S}{\partial \bar{u}\partial \bar{v}} & \frac{\partial^2 S}{\partial \bar{v}^2} & \frac{\partial^2 S}{\partial \bar{v}\partial \bar{w}} \\ \frac{\partial^2 S}{\partial \bar{u}\partial \bar{w}} & \frac{\partial^2 S}{\partial \bar{v}\partial \bar{w}} & \frac{\partial^2 S}{\partial \bar{w}^2} \end{bmatrix} =$$

$$= \frac{1}{\ln 2} D \begin{bmatrix} -(\bar{v}+\bar{w}) & \sqrt{\bar{u}\bar{v}} & \sqrt{\bar{u}\bar{w}} \\ \sqrt{\bar{u}\bar{v}} & -(\bar{u}+\bar{w}) & \sqrt{\bar{v}\bar{w}} \\ \sqrt{\bar{u}\bar{w}} & \sqrt{\bar{v}\bar{w}} & -(\bar{u}+\bar{v}) \end{bmatrix} D^T$$

where $\bar{s} = \bar{u}+\bar{v}+\bar{w}$ and $D = \mathrm{diag}[\frac{1}{\sqrt{\bar{u}\bar{s}}}, \frac{1}{\sqrt{\bar{v}\bar{s}}}, \frac{1}{\sqrt{\bar{w}\bar{s}}}]$, is nonpositive definite for all $\bar{u},\bar{v},\bar{w} > 0$. Hence S is concave and therefore, in particular, the value of S at any local maximum on p maximizes S on p. We shall prove that there is such local maximum very close to the numerical solution $(\bar{u}_0,\bar{v}_0,\bar{w}_0)$.

The plane p can be parametrized with $\bar{v}$ and $\bar{w}$. This results in the following formulas for the restriction of S to p (denoted by S_p and treated as a function of $\bar{v}$ and $\bar{w}$) and its derivatives.

$$\begin{aligned} S_p(\bar{v},\bar{w}) = {} & (0.5-2\bar{v}-5\bar{w})\log_2(0.5-2\bar{v}-5\bar{w}) - \\ & \bar{v}\log_2\bar{v} - \bar{w}\log_2\bar{w} - \\ & (0.5-3\bar{v}-6\bar{w})\log_2(0.5-3\bar{v}-6\bar{w}) \\ \frac{\partial S_p}{\partial \bar{v}} = {} & 3\log_2(0.5-3\bar{v}-6\bar{w}) - \\ & 2\log_2(0.5-2\bar{v}-5\bar{w}) - \log_2\bar{v} \\ \frac{\partial S_p}{\partial \bar{w}} = {} & 6\log_2(0.5-3\bar{v}-6\bar{w}) - \\ & 5\log_2(0.5-2\bar{v}-5\bar{w}) - \log_2\bar{w} \\ \frac{\partial^2 S_p}{\partial \bar{v}^2} = {} & \tfrac{1}{\ln 2}\left(\tfrac{4}{0.5-2\bar{v}-5\bar{w}} - \tfrac{9}{0.5-3\bar{v}-6\bar{w}} - \tfrac{1}{\bar{v}}\right) \\ \frac{\partial^2 S_p}{\partial \bar{w}^2} = {} & \tfrac{1}{\ln 2}\left(\tfrac{25}{0.5-2\bar{v}-5\bar{w}} - \tfrac{36}{0.5-3\bar{v}-6\bar{w}} - \tfrac{1}{\bar{w}}\right) \\ \frac{\partial^2 S_p}{\partial \bar{v}\partial \bar{w}} = {} & \tfrac{1}{\ln 2}\left(\tfrac{10}{0.5-2\bar{v}-5\bar{w}} - \tfrac{18}{0.5-3\bar{v}-6\bar{w}}\right) \end{aligned}$$

One can check on a calculator that:

$$\begin{aligned} \log_2\bar{v}_0 \in {} & [-3.81037, -3.81036], \\ \log_2\bar{w}_0 \in {} & [-5.67493, -5.67492], \\ \log_2(0.5-3\bar{v}_0-6\bar{w}_0) \in {} & [-2.56733, -2.56732], \\ \log_2(0.5-2\bar{v}_0-5\bar{w}_0) \in {} & [-1.9458, -1.945799], \end{aligned}$$

Starting from the above and using interval arithmetic [1] one can show that

$$\frac{\partial S_p}{\partial \bar{v}}(\bar{v}_0,\bar{w}_0) \in [-0.000032, 0.00001]$$

and

$$\frac{\partial S_p}{\partial \bar{w}}(\bar{v}_0,\bar{w}_0) \in [-0.000065, 0.00001].$$

Hence $|\mathrm{grad}\ S_p(\bar{v}_0,\bar{w}_0)| \leq 0.0001$. Similarly, the Hessian of S_p multiplied by $\ln 2$ is a symmetric matrix with entries in the intervals shown below at any point $(\bar{v}, \bar{w})$ belonging to the square $Q = [\bar{v}_0 - 0.0001, \bar{v}_0 + 0.0001] \times [\bar{w}_0 - 0.0001, \bar{w}_0 + 0.0001]$.

$$\begin{bmatrix} [-53,-51] & [-69,-67] \\ [-69,-67] & [-170,-166] \end{bmatrix}$$

The characteristic polynomial of the above matrix is $P(\lambda) = \lambda^2 + B\lambda + C$, where $B \in [217, 223]$ and $C \in [3705, 4521]$. It follows that its both roots are less than -15 (e.g. since $P(-15) > 0$ and $P(-100) < 0$). Thus, all eigenvalues of the Hessian are to the left of $-15/\ln 2 < -20$. As a result,

$$D^2 S_p(\bar{v}, \bar{w}) x^2 < -20|x|^2$$

for any vector x and $(\bar{v}, \bar{w}) \in Q$.

Making use of the above observations and the Taylor's Formula (see e.g.[8]) we see that whenever $(\bar{v}, \bar{w}) \in Q$,

$$\begin{aligned} & S_p(\bar{v}, \bar{w}) - S_p(\bar{v}_0, \bar{w}_0) = \\ & \quad = DS_p(\bar{v}_0, \bar{w}_0)(\bar{v} - \bar{v}_0, \bar{w} - \bar{w}_0) + \\ & \qquad +0.5 D^2 S_p(\bar{v}_1, \bar{w}_1)(\bar{v} - \bar{v}_0, \bar{w} - \bar{w}_0)^2 \\ & \quad \leq |\mathrm{grad} S_p(\bar{v}_0, \bar{w}_0)||(\bar{v} - \bar{v}_0, \bar{w} - \bar{w}_0)| - \\ & \qquad 10|(\bar{v} - \bar{v}_0, \bar{w} - \bar{w}_0)|^2 \\ & \quad \leq (0.0001 - 10|(\bar{v} - \bar{v}_0, \bar{w} - \bar{w}_0)|) * \\ & \qquad *|(\bar{v} - \bar{v}_0, \bar{w} - \bar{w}_0)|, \end{aligned}$$

where $(\bar{v}_1, \bar{w}_1)$ belongs to the interval joining $(\bar{v}, \bar{w})$ and $(\bar{v}_0, \bar{w}_0)$ therefore also to Q. Hence $S_p(\bar{v}, \bar{w}) \leq S_p(\bar{v}_0, \bar{w}_0)$ on the circle of radius $r = 0.00001$ centered at $(\bar{v}_0, \bar{w}_0)$ which proves that S_p has a local maximum at some point inside that circle. Wherever this maximum is, the monotonicity of S along the directions parallel to the $\bar{u}, \bar{v}, \bar{w}$ coordinate axes implies that its value is bounded by $S(0.5 - 3(\bar{v}_0 - r) - 6(\bar{w}_0 - r), \bar{v}_0 + r, \bar{w}_0 + r)$ which can be confirmed to be less than 0.311 using a calculator. This result is clearly not optimal - better bounds can be obtained by increasing the accuracy of calculations. We are quite sure that the *Mathematica* solution of 0.310761 can be approached much closer in this way.

4.3 Final result

We can state the results of this section in the form of the following theorem.

Theorem 4.1 *The compression and decompression algorithms described in this section run in linear-time and achieve 0.811 bits per triangle for sufficiently large and regular meshes.*

4.4 Example

To illustrate the ideas of this section, let us look at an Edgebreaker encoding of the four-triangle mesh representing the boundary of a tetrahedron after two steps of 1-to-4 subdivision. A possible CLERS sequence for that mesh is

CCCCCRRCCRCCRCCRCCRRCCRCRCCRCRCC
RCRCCRRCCRCRCRLCRCRSERCRSCRERSEE

It is not easy to see what the hit/miss sequence is, but, according to our code, it is

11111111111111111111111111111101
1111111111110111111111110111111

The CLS/ER and LE/SR sequences are

00000110010010010011001010010100
10100110010101001010110100111011

and

111111111111111110111011101100

It is clearly visible that the entropy of the hit/miss sequence is low. We also note that the same applies to the LE/SR sequence: this paper leaves the question of symbol frequencies within the LE/ER sequence open, but there might be room for improvement here. Our way of encoding the CLS/ER sequence is to encode its long zero subsequences separately (in this example, there is only one such subsequence, the 00000 prefix: this should not be a surprise since the encoded mesh has zero degree excess). The simplified CLS/ER sequence is

11001001001001100101001010010100
110010101001010110100111011

By skipping the pairs of zeros corresponding to CC's and their preceding 1's we obtain a shorter sequence 11101010110101001010110100111011. Skipping zeros corresponding to C's yields 11111111101101110111011, a sequence whose UVW encoding is UUUUUUUUU-VUVUUVUUVU. That UVW sequence together with the hit/miss and LE/SR sequences and the encoding of the long zero subsequences (in this case, only one) comprise the encoding of the input mesh. All the three sequences can be arithmetically encoded and, since their entropies are small, the sizes of their encodings are small as well.

5 Conclusion

We described an Edgebreaker-based linear-time compression scheme for the connectivity of simple triangular meshes which has a provable worst-case bound of 0.811 bits per triangle for sufficiently large and regular meshes. This is 50% less than anyone will ever be able to prove about a compression scheme for the class of all triangular meshes. The algorithms are simple to implement and future research will show whether the ideas presented in this paper can lead to compression algorithms achieving better results for meshes encountered in practice. Another topic for further study are other measures of regularity which better reflect the structure of 'typical' meshes and their impact on performance of various compression schemes.

References

[1] O.Aberth, Precise Numerical Analysis, William C.Brown Publishers, Dubuque, Iowa, 1988.

[2] S.Gumhold, *New Bounds on The Encoding of Planar Triangulations*, preprint 2000.

[3] S.Gumhold and W.Strasser, *Real Time Compression of Triangle Mesh Connectivity*, Computer Graphics (Proc. SIGGRAPH), pp.133-140, July 1998.

[4] X.He, M.-Y.Kao and H.-I. Lu, *A Fast General Methodology for Information-Theoretically Optimal Encodings of Graphs*, in: Jaroslav Nesetril (Ed.): Algorithms - ESA '99, Proc. 7th Annual European Symposium, Lecture Notes in Computer Science, Vol. 1643, Springer, 1999.

[5] X.He, M.-Y.Kao and H.-I. Lu, *Linear-Time Succinct Encodings of Planar Graphs via Canonical Orderings*, SIAM Journal on Discrete Mathematics 12(3), August 1999.

[6] P.G.Howard and J.S.Vitter. *Analysis of Arithmetic Coding for Data Compression*, Information Processing and Management, 28(6), 1992, 749-763.

[7] M.Isenburg and J.Snoeyink, *Spirale Reversi: Reverse decoding of Edgebreaker encoding*, Technical Report TR-99-08, Department of Computer Science, University of British Columbia, September 1999.

[8] J.Jost, Postmodern Analysis, Universitext, Springer-Verlag Berlin Heidelberg 1998.

[9] K.Keeler and J.Westbrook, *Short Encodings of Planar Graphs and Maps*, Discrete Applied Mathematics, No. 58, pp.239-252, 1995.

[10] D.King and J.Rossignac, *Guaranteed 3.67V Bit Encoding of Planar Triangle Graphs*, 11th Canadian Conference on Computational Geometry.Vancouver, BC, September 1999.

[11] J.Rossignac, *Edgebreaker: Compressing the connectivity of triangle meshes*, GVU Technical Report GIT-GVU-98-17, Georgia Institute of Technology, IEEE Transactions on Visualization and Computer Graphics vol.5(1), 1999.

[12] J.Rossignac and A.Szymczak, *Edgebreaker compression and Wrap&Zip decoding of the connectivity of triangle meshes*, Computational Geometry: Theory and Applications, 1999.

[13] C.Touma and C.Gotsman, *Triangle Mesh Compression*, Proceedings Graphics Interface 1998, pp. 26-34.

[14] G.Turan, *Succinct representations of graphs*, Discrete Applied Mathematics 8, pp. 289-294, 1984.

[15] W.T.Tutte, *A census of planar triangulatiuons*, Canadian Journal of Mathematics 14 (1962), pp. 21-38.

[16] I.H.Witten, R.M.Neal and J.G.Cleary, *Arithmetic Coding for Data Compression*, Comm.ACM 30 (1987), pp. 520-540.