On the Non-Redundancy of Split Offsets in Degree Coding

Martin Isenburg
isenburg@cs.unc.edu

Jack Snoeyink
snoeyink@cs.unc.edu

University of North Carolina at Chapel Hill

Abstract

The connectivity coder by Touma and Gotsman encodes a planar triangulation through a sequence of vertex degrees and occasional "split" symbols that have an associated offset value. It has been speculated that the split offsets might be redundant information that can be derived from the degree sequence, especially as this is true for similar split offsets used in other coding schemes.

In this paper we show that the split offsets of the TG coder are in general not redundant. We give examples of degree sequences that have two different decodings if the split offsets are not specified. Surprisingly, such examples are rare and a large number of encodings remain unique. The few encodings that are not unique have only a small number of valid decodings so that their split offsets could be replaced by one or two bits that specify the actual triangulation among all possible ones.

We investigate how the omission of explicit split information affects the uniqueness of four alternate offset-less encodings. One stores split and end symbols, one only split symbols, one stores only the number of splits, and one stores no split information at all. As we find valid decodings through exhaustive search that reaches exponential complexity, our work is mainly of theoretical interest and does not lead to new practical encoding algorithms.

Key words: Connectivity coding, mesh compression, split offsets, TG coder.

1 Introduction

Recent years have seen a number of schemes that compactly encode triangle mesh connectivity by a sequence of symbols that specify how to grow a "compression boundary" enclosing an already encoded region, one triangle at a time. A popular scheme is the Triangle Mesh Compression method by Touma and Gotsman [14], or TG coder for short. For planar triangulations, the TG coder generates a sequence of vertex degrees that usually contains a few "split" symbols with associated offset values.

There has been speculation that it might be possible to modify the TG coder to operate without explicitly storing the offsets values that associated with each "split" symbol. Such speculations are motivated by the fact that Edgebreaker [12] manages to avoid storing such offsets, whereas the otherwise almost identical Cut-border Machine [3] explicitly includes them. Similarly, Face-Fixer [7] avoids storing offsets, whereas the Dual-Graph Method [10] includes them.

In an information theoretic sense, the split offsets used by the Cut-border Machine and the Dual-Graph Method are redundant because they are implied in their symbol sequence. Edgebreaker and Face Fixer recreate these offsets by decoding either in two passes [12, 13] or in reverse [7, 8]. In both schemes, each "split" symbol S has a corresponding "end" symbol E that signals the completion of a compression boundary. Given a traversal that always completes one boundary part before continuing on the "split off" parts, the symbols S and E form nested pairs, each of which encloses a subsequence of symbols. In Edgebreaker and Face-Fixer, such a symbol subsequence is a self-contained encoding of the portion of the mesh enclosed by that boundary part. Obviously, this also determines the length of that boundary part, which is exactly what is specified by the split offset.

It was less clear whether the split offsets used by the TG coder would also be redundant. For one thing, the symbol sequences produced by the TG coder do not contain explicit "end" symbols because the completion of a compression boundary is automatically detected. Therefore it is not easy to identify the subsequences of symbols that complete a particular boundary part. But simply adding "end" symbols does not make things much easier, because the split offsets of the TG coder cannot be derived from a subsequence of symbols alone.

Unlike the symbol subsequences of Edgebreaker and Face Fixer, the symbol subsequences of the TG coder are not self-contained encodings of some portion of a mesh. Decoding also depends on the state of the compression boundary at the beginning of the subsequence. This is illustrated in Figure 1 at the example of one degree subsequence that encodes two different submeshes following a "split". The TG coder stores significantly more state information on the compression boundary than Edgebreaker or Face Fixer. It maintains *slot counts* that

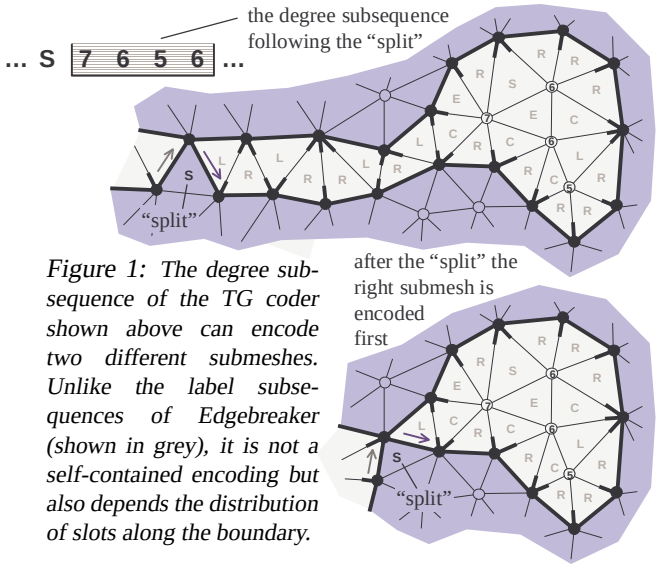


Figure 1: The degree subsequence of the TG coder shown above can encode two different submeshes. Unlike the label subsequences of Edgebreaker (shown in grey), it is not a self-contained encoding but also depends the distribution of slots along the boundary.

specify how many unprocessed edges are still incident to each boundary vertex. The split offsets also specify how to split these counts, not just the boundary vertices. The value of each count and their order around the boundary depends on all preceeding symbols. Therefore is it impossible to derive the split offsets with computations restricted to subsequences of symbols or by processing the symbols in reverse.

We have implemented a decoding scheme that "tries out" all possible split offsets in a brute-force manner, backtracking as soon as it notices that it cannot complete a triangulation. This is slow due to the exponentially increasing search space and therefore impractical as a decoding algorithm. But this approach does find counter-examples that establish the non-redundancy of the offsets—namely, symbol sequences that lead to more than one valid decoding if different split offsets are used.

In practice, our implementation finds that only few sets of split offsets actually lead to valid decodings. For example, of the $290,898$ possible encodings of the well-known horse model (see Table 1), $290,889$ are unique when both split and end symbols are stored, and the remaining 9 have only two valid decodings each. For those we could replace all the split offsets with a single bit that specifies which one of the two decodings it is. In fact, we explore how often a unique triangulation is obtained under four different encodings without explicit offsets: The first stores both split and end symbols, the second stores split symbols only, the third stores only the number of splits, and the fourth stores vertex degrees alone. As expected, we find that omitting more split information results in more possible decodings, but it is surprising how many still have unique triangulations.

Unfortunately we do not have efficient algorithms for finding all possible valid decodings. We currently achieve this by exhaustive search through all potential splits and the complexity of this search increases quickly. For the encodings that store split and end symbols we can significantly prune the search tree and solve even relatively large triangulations in reasonable time. But for encodings that do not store end symbols and especially for those encodings that do not even store split symbols we have to limit our experiments to very small triangulations. Hence, our work is mainly of theoretical interest and does not lead to new practical encoding algorithms.

The remainder of this paper is organized as follows: The next section reviews how the TG coder encodes a triangulation and reports several invariants of the encoding process. Section 3 describes the four different offset-less encodings and gives counterexamples that (a) prove that offsets are not redundant and (b) show that the encodings have more possible decodings as less split information is retained. Section 4 describes the algorithms that search for all valid decodings. Experimental results are reported in Section 5 for a set of well known triangle meshes and a set of randomly generated triangulations. We conclude with a discussion of related issues.

We must mention that there are several incentives to include explicit split offsets in the encoding. They allow decoding in a single forward pass over the symbol sequence, which makes it possible to decompress in a streaming fashion [5]. They give the freedom to choose a mesh traversal that is not recursive. The Cut-Border Machine and the TG coder can easily be modified to operate in a breadth-first manner, which leads to more coherent mesh layouts [6]. However, note that for a breadth-first traversal order even the offsets of the Cut-Border Machine are no longer redundant. Finally, explicit split offsets often result in better overall compression rates because they allow to incorporate heuristics for predictive compression of the vertex degrees [9].

2 Connectivity coding with the TG Coder

A triangulation, for the purpose of this paper, is a maximal, three-connected, planar graph in which every face is a triangle. To encode a triangulation, the TG coder [14] performs a region-growing process by maintaining one or more compression boundaries into which it includes triangle after triangle. It usually includes the triangle that is adjacent to the *gate* edge, which advances in clockwise order around the *focus* vertex as illustrated in Figure 2. However, it will immediately include any triangle that shares two edges with the compression boundary, even if neither of these two edges is the gate.

The encoder starts with an initial compression boundary of length two that is defined around an arbitrary edge of the triangulation. One of the two boundary edges be-

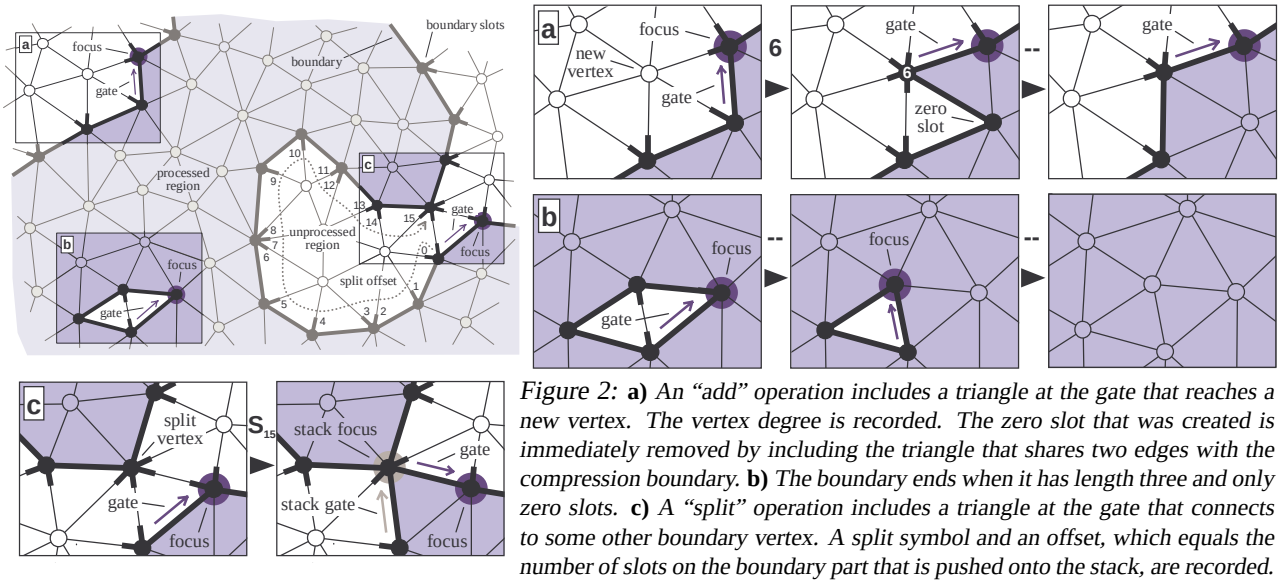


Figure 2: **a)** *An "add" operation includes a triangle at the gate that reaches a new vertex. The vertex degree is recorded. The zero slot that was created is immediately removed by including the triangle that shares two edges with the compression boundary.* **b)** *The boundary ends when it has length three and only zero slots.* **c)** *A "split" operation includes a triangle at the gate that connects to some other boundary vertex. A split symbol and an offset, which equals the number of slots on the boundary part that is pushed onto the stack, are recorded.*

comes the *gate* and its target vertex becomes the *focus*. The encoder records the degree of the two initial boundary vertices in an agreed-upon order and sets their *slot count* to be one less than their degree. Throughout the encoding process these slot counts are constantly updated and always reflect the number of unprocessed edges incident to the respective boundary vertex.

Whenever all boundary vertices have a slot count of one or higher, the triangle adjacent to the gate shares only one edge with the compression boundary. In this moment the encoder will perform either an "add" or a "split" operation and record the corresponding symbols. Usually the third vertex of the triangle at the gate is a previously unprocessed vertex. In this case the encoder simply "adds" this vertex as a new boundary vertex and records its degree, as illustrated in Figure 2a. Occasionally, however, the third vertex of this triangle is already on the boundary. In this case the encoder splits the compression boundary into two loops, temporarily stores one on a stack, and continues encoding on the other. Then the encoder records a split symbol together with the number of slots that are on the boundary part that is stored on the stack. This *split offset* specifies how many slots are clockwise along the boundary between the gate and the split vertex, as illustrated in Figure 2c.

Whenever the slot count of a boundary vertex drops to zero, there is a triangle that shares two edges with the compression boundary. The encoder immediately includes such triangles into the compression boundary without recording anything. The special case that all boundary vertices have a slot count of zero indicates the completion of this boundary[1]. This happens only for a boundary of length three as illustrated in Figure 2b. Then the encoder either continues on the boundary that was most recently stored on the stack or terminates if the stack is empty. There is no other scenario in which two or more adjacent boundary vertices can have a slot count of zero.

It is important to note that there are two pieces of information contained in the split offset. It specifies both the length of the boundary part that is pushed onto the stack and how to divide the remaining slots at the split vertex among the two boundary parts. In comparison, the split offsets used by the Cut-Border Machine only contain one piece of information, namely the the length of the boundary part that is pushed onto the stack.

Counts and Invariants

There are several invariants that are maintained during a valid decoding; these can help us recognize early when we have attempted a wrong choice of split or split offset. If b is the number of boundary edges and s is the number of slots, we can count the number of unprocessed edges by their contributions to vertex and face degrees to obtain:

$$\sum_{\mathrm{f}_k \subset \mathcal{U}} deg(\mathrm{f}_k) - b = \sum_{\mathrm{v}_k \subset \mathcal{U}} deg(\mathrm{v}_k) + s, \qquad (1)$$

where $\subset \mathcal{U}$ means unprocessed. In particular, this invariant individually holds for the edge and slot counts of each compression boundary together with the face and vertex degree counts of the enclosed mesh region [4]. Since all our faces are triangles we can re-write equation 1 as $3t - b = d + s$ where t is the number of unprocessed triangles and d is a shortcut for the sum of degrees of

[1]The end of a compression boundary can also be detected as the moment in which all boundary vertices have a slot count of one. This way the last vertex degree of each boundary part can also be omitted [9].

the unprocessed vertices. Euler's relation for a triangulation with t triangles, v vertices and a single boundary of length b tells us that $t + b = 2v - 2$. Substituting v with $b + i$ where i is the number of internal vertices and solving for t gives us $t = 2i + b - 2$. Substituting the t of the rewritten equation 1 results in a new invariant

$$6i + 2b = d + s + 6 \tag{2}$$

that relates the length b and slot count s of an individual compression boundary and the number of unprocessed vertices i and their total degree d.

Splitting a boundary of length b with slot count s creates two boundaries of length b_1 and b_2 with slot counts s_1 and s_2 such that $b+1 = b_1+b_2$ and $s-4 = s_1+s_2$. At the same time the i unprocessed vertices with degree total d are seperated into i_1 and i_2 vertices with corresponding degree totals d_1 and d_2 where $i = i_1+i_2$ and $d = d_1+d_2$. Hence, after the split we have to replace the old invariants with two new invariants $6i_1 + 2b_1 = d_1 + s_1 + 4$ and $6i_2 + 2b_2 = d_2 + s_2 + 4$. These invariants are related so that if the old invariant was true and one of the new invariants is true, the other new invariant is also true.

There also exists a useful inequality between the number of unprocessed triangles t that are enclosed by the compression boundary and the current length b of this boundary. The fact that completing a boundary of length b requires a minimum of b triangles gives us the following invariant

$$t \geq b \tag{3}$$

which turns into an equality in the moment the boundary encloses only a single unprocessed vertex.

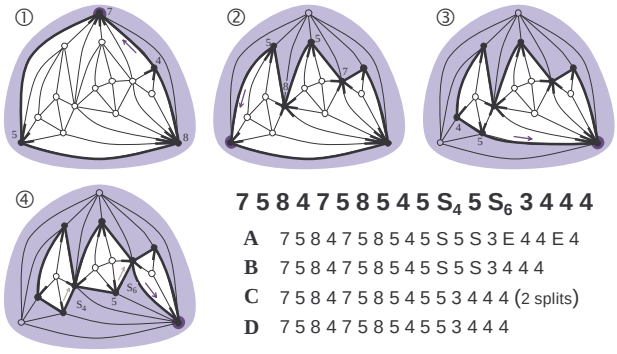


Figure 3: The original TG encoding of this triangulation with 15 vertices has two splits with offsets 4 and 6. In the following we investigate the uniqueness of four offset-less encodings: **A)** *with split and end symbols,* **B)** *with split symbols,* **C)** *with the number of splits, and* **D)** *without any split information.*

3 Encodings without explicit offsets

This work originated with the intention to find small counterexample that would show that – unlike the offset's used by the Cut-Border Machine – the offsets used by the TG coder are not redundant. Several coffee refills later, after having produced a pile of scrap paper covered with larger and larger examples of carefully constructed triangulations, we realized that finding a counterexample was more difficult than anticipated. However, an implementation of an exhaustive search through randomly generated triangulations was more successful.

Theorem. *For the TG coder without split offsets, but with end symbols, there exist two different triangulations that have the same offset-less encoding.*

In the following we actually consider four alternate offset-less encodings that include less and less information about the split operations that occur during the encoding process. Figure 3 shows an example of a standard TG encoding (with explicit split offsets) and the four offset-less variations (A-D). In this example, all four offset-less encoding are unique. Since there are examples for which the strongest encoding (A) is not unique, there are examples for which all four are not unique. We include all four, nevertheless, because it is interesting to see how the small counterexamples get smaller and the likelihood that a particular offset-less encoding is non-unique increases as we omit more and more split information.

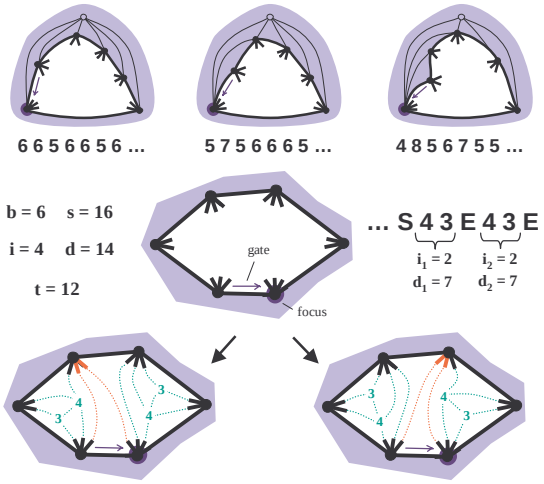


Figure 4: The smallest scenario where an offset-less encoding with split and end symbols is not unique. This scenario can occur in triangulations having as few as 11 vertices. Note, however, that the top-right example has only one valid decoding that results in a three-connected triangulation, despite the compression boundary having reached the same state.

The smallest examples for which all four offset-less encodings are non-unique can be found in triangulations with 11 vertices as illustrated in Figure 4. There are two possible ways of splitting the boundary of length 6 that has its 16 slots distributed in this particular configuration and where both resulting boundary parts still enclose

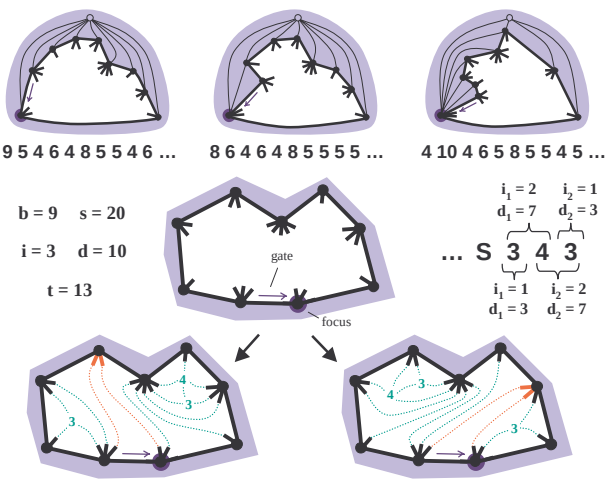


Figure 5: The smallest scenario where an offset-less encoding in no longer unique without end symbols. Triangulations need to have a minimum of 13 vertices for this scenario to occur. Again, the top-right example has only one valid decoding.

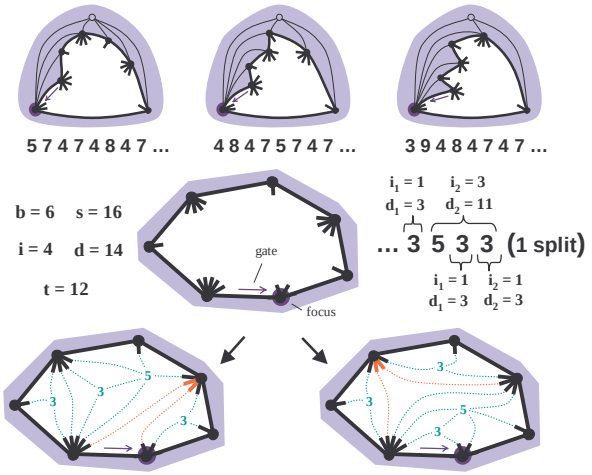


Figure 6: The smallest scenario for which an offset-less encoding becomes non-unique after dropping the split symbols while retaining their total number. The smallest triangulation in which this occurs has 12 vertices. The top-right encoding is unique despite an identical boundary state because of the non-boundary edges that connect the focus to other boundary vertices.

two unprocessed vertices of degree 3 and 4. However, there are not always two valid decodings in this scenario. Sometimes the focus is already connected to other vertices of the boundary through previously decoded triangles. This places additional constraints on the possibilities for splitting the boundary. The top-right triangulation from Figure 4, for example, has only one valid decoding.

In the following we demonstrate that the offset-less encodings are less likely to be unique as we omit more split information. Initially, we thought that already dropping the end symbols would make it so unlikely for encodings to be unique that it would be easy to find encodings that are unique with end symbols but non-unique without them. Another coffee shop session with pencil and paper taught us that finding such an example was again much harder than anticipated. However, an implementation of a backtracking search quickly finds the examples shown in Figure 5. Omitting the end symbol makes these encoding non-unique because of the added flexibility in having different numbers of vertices inside each boundary part.

The next step was to drop the split symbols and retain only the information about the total number of split operations. Having learned the futility of the paper and pencil approach we turned directly to a search algorithm. The anticipated example encodings that are unique with split symbols but become non-unique without are shown in Figure 6. Omitting the split symbol makes these encoding non-unique because of the added flexibility in performing the splits at an earlier or at a later moment.

In the last step we also omitted the information on the number of split operations and as expected this makes the encodings even less likely to be unique. In this case a triangulation needs only 10 vertices for its encoding to be non-unique as shown in Figure 7. The encoding, which is now simply a sequence of vertex degrees, can be decoded without a split operation or with a single split operation.

4 Searching for valid decodings

In order to find all valid decodings of an offset-less encoding we search through all possibilities of performing split operations. The more split information is contained in the encoding the fewer possibilities need to be searched. For the offset-less encodings **A** and **B** that explicitly store split symbols our decoder iteratively tries to split the current boundary in every possible way whenever it reaches a split symbol. For encodings **C** and **D** our decoder does the same whenever the boundary allows a split. For each attempt it recursively starts to decode the first boundary part and in case this is successful does the same for the second. Only if both recursions are successful it returns a success, otherwise it tries out the next possibility or returns a failure if there are none left. A few observations help us to immediately eliminate some splits from further consideration.

The split vertex must have a minimal slot count of two and this minimal slot count increases for every boundary vertex with slot count one that is either directly to the left or to the right of the gate. This is because each of these *one-slots* becomes a *zero-slot* after the split and including each of the corresponding triangles consumes an additional slot at the split vertex. This can drastically reduce the number of boundary vertices that are potential

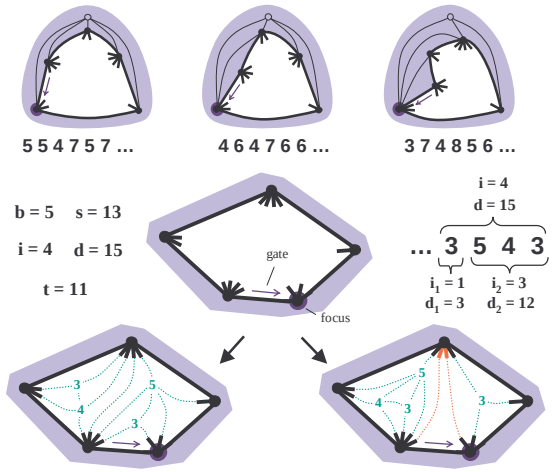


Figure 7: The smallest scenario where an offset-less encoding without any split information is not unique. It only needs triangulations of 10 *vertices for this to occur. The top-right encoding is unique since the split would lead to an invalid triangulation.*

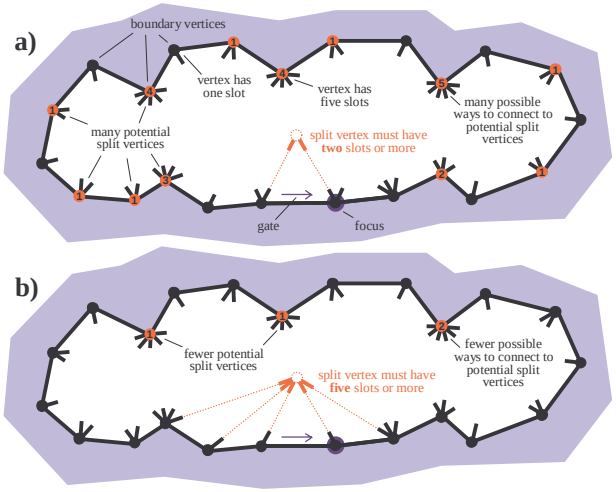


Figure 8: **a)** *The minumum slot count of the split vertex is two. Twelve boundary vertices satisfy this criteria resulting in twenty-five potential split configurations.* **b)** *The three vertices with a slot count of one that surround the gate increase the minimum slot count to five. In this case only three boundary vertices qualify with a total of four potential split configurations.*

candidates for the split vertex as illustrated in Figure 8.

It also decreases the number of combinations in which we can connect to a particular boundary vertex. Every slot such a boundary vertex has more than the minimal slot count provides another way for distributing the remaining slots between the two resulting boundaries.

For the offset-less encodings **A** with end symbols we have additional constraints that allow our backtracking decoder to further prune the search. Since the symbol subsequences that correspond to each boundary part can be identified exactly, we can compute the numbers of vertices i_1 and i_2 that each boundary part contains as well as their total degree d_1 and d_2. We can then eliminate all those splits from further consideration that have a combination of b_1 and s_1 that violates the invariant $6i_1 + 2b_1 = d_1 + s_1 + 4$. This constraint significantly reduces the number of potential splits as shown in Figure 9.

Once the current choice for a split satisfies all applicable contraints the decoder starts a recursion to decode the first boundary part. There are several invalid states this decoding process may encounter in which case it returns a failure: (a) the boundary is completed before the end symbol is reached, (b) the end symbol is reached before the boundary is completed, (c) the slot count of two neighboring boundary vertices drops to zero, (d) a split symbol is encountered but all possible ways of splitting the boundary fail, (e) the boundary can not be completed with the remaining triangle budget.

The last of these failures is based on equation 3. If we know either the exact or the maximal number of unprocessed triangles that a compression boundary encloses, we have an additional invariant to detect failures of the decoding process earlier. This early failure is most effective for the offset-less encodings **A** because end symbols allow us to compute the exact number of triangles.

Once we made the choice how to split the current boundary into loops of length b_1 and b_2 we can compute the number of triangles t that each boundary loop encloses as $2i_1+b_1-2$ and $2i_2+b_2-2$ because the number of unprocessed vertices i_1 and i_2 that each loop contains is known. We then give this value t as an additional parameter to the recursive decoding calls and decrement it for every decoded triangle. Should this value ever become smaller than the current length of the boundary the decoding process returns an early failure.

The absence of explicit end symbols means that the decoder can no longer identify the subsequences that correspond to each boundary part. This makes the search for the split offsets in several ways more expensive. First, we can no longer use invariant 2 to immediately eliminate some splits from further consideration, which results in more recursive calls. Second, we can no longer fail because an end symbol is reached too early or too late, which makes each recursive call more expensive as it takes longer for the decoding process to reach an invalid state. Third, we can not compute the exact number of triangles t that the boundary loops enclose, which significantly weakens the efficiency of our second invariant—especially for the recursion on the first boundary part.

However, we do not completely loose this invariant. Given the maximal number of triangles t that the current boundary encloses (i.e. before the split operation) and the

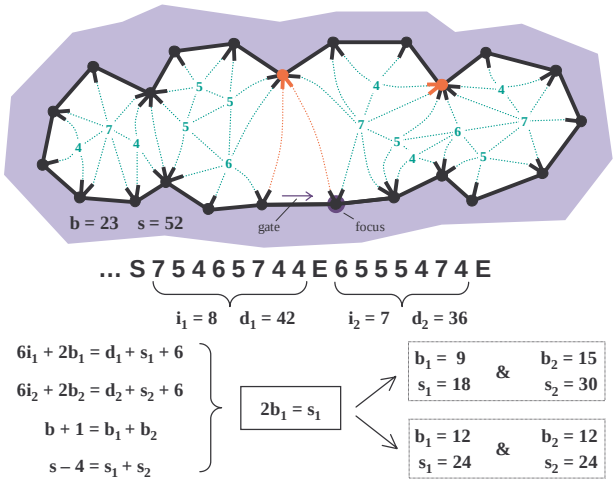


Figure 9: End symbols tell the decoder which vertices are enclosed by each boundary part. This leads to an additional constraint that leaves only two choices for splitting the boundary.

length b_1 and b_2 of the two loops that it is split into, we know that the first boundary can at most enclose $t-b_2-1$ triangles. Admittingly, this is a rather weak invariant. But should the first recursion complete with success it will tell us the actual number of triangles t_1 that were decoded. For recursivly decoding the second boundary part we then know that there are at most $t-t_1-1$ triangles left.

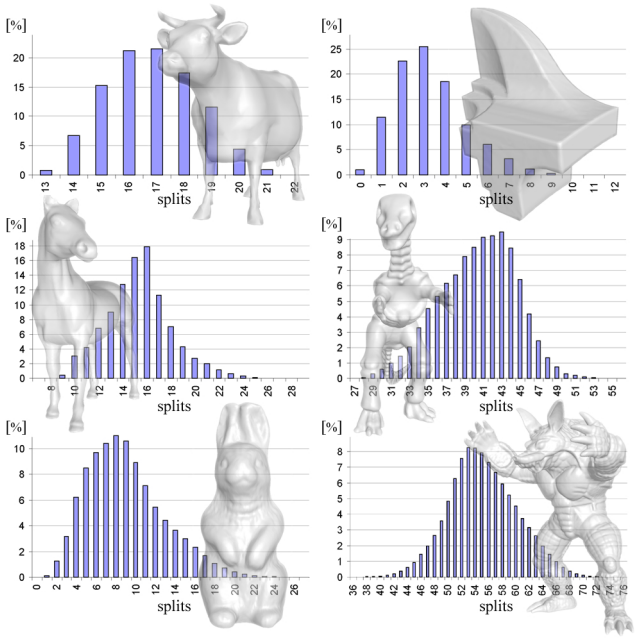
Omitting split symbols really increases the search space. Whenever decoding is about to perform an "add" operation and there are remaining splits to perform, it first tries whether it is possible to perform a "split" operation instead. This requires a minimal boundary length of five and a minimal slot count of ten. The boundary is then split every possible way and the corresponding recursive calls are tested for success. If we have more than one "split" operation left, we need to check each possibility multiple times—once for each way of distributing the remaining split operations onto the two recursions.

For the offset-less encodings **D** the decoder does not even know the total number of splits. It starts out assuming that the degree sequence can be decoded without any splits. If this fails we increase the number of splits and try again. At this point the search space is so immense that is only completes for a small number of splits

5 Experimental results

Initial experiments seemed to indicate that the split offsets of the TG coder might in fact be redundant. On our standard set of example meshes the search for split offsets would find the correct answer every run we tried. Table 1 shows that non-unique encoding are surprisingly rare.

We encoded six well-known models in every possible way (i.e. starting from every half-edge) and checked the



meshes			splits			non-unique
name	vertices	encodings	min	max	avg	encodings
cow	2,904	17,412	13	22	16.8	0
fandisk	6,475	38,838	0	12	3.3	0
horse	48,485	290,898	7	29	15.4	9
dinosaur	56,194	337,152	27	56	40.4	10
rabbit	67,039	402,222	0	27	9.0	56
armadillo	172,974	1,037,832	36	76	55.2	146

Table 1: The table lists for each mesh the number of vertices and the number of different encodings. The illustrations show which percentage of encodings has what number of splits. The minimum, the maximum, and the average number of splits are given in the table. Most importantly, we report the number of offset-less encodings of type **A** *that are not unique.*

resulting encodings for uniqueness when both split and end symbols are used. Table 1 shows that all these encodings are unique for the smallest two meshes and that there are only a few non-unique encodings for the larger models. For the fandisk, we also verified that all encodings **B** that only use split symbols are unique. Each of the non-unique encoding has only two valid decodings, so while the split offsets of those encodings are not redundant they could be replaced by a single bit.

For experiments on the uniqueness of the other offset-less encodings we were forced to significantly lower the complexity of our example models. The search space virtually explodes as the number of splits increases and often we had to abort a seemingly endless computation. We used Poulalhon and Schaeffer's technique [11] and to generate one million triangulations of n vertices for which starting from a random edge produced at least one

split. In Table 2 we list how many offset-less encodings become non-unique because they have two/multiple decodings as we omit more and more split information.

n	splits			non-unique encodings			
	min	max	avg	**A**	**B**	**C**	**D**
15	1	3	1.0	310/-	344/-		
20	1	4	1.1	358/-	459/1		
30	1	6	1.5	468/-	654/-		
40	1	7	2.0	520/-	855/-		
50	1	8	2.6	682/-	1,003/1	to	
60	1	9	3.3	848/-	1,257/-	be	
70	1	11	4.0	947/2	1,388/4	completed	
80	1	12	4.7	1,057/1	1,542/2		
90	1	14	5.4	1,175/1	1,765/4		
100	1	14	6.0	1,368/-	1,990/5		
200	2	24	12.9	2,600/-	n.a.	n.a.	n.a.
500	15	52	33.4	6,672/26	n.a.	n.a.	n.a.

Table 2: This table reports how many offset-less encodings **A**, **B**, **C**, *and* **D** *have two/multiple valid decodings for the same set of one million encodings of random triangulations with n vertices that start at a random edge and have at least one split.*

6 Discussion

There have been attempts to establish a guaranteed bound on the coding costs of the TG coder. However, the infrequently occuring "split" symbols made this a difficult task. Alliez and Desbrun [1] suggested an adaptive traversal heuristic that lowered the number of split operations and the remaining number of "splits" seemed negligibly small. Therefore the authors restricted their worst case analysis to the vertex degrees. Surprisingly, the maximal entropy of a distribution of n degrees that sum up to $6n - 18$ coincides with the information theoretic minimum of 3.24 bits per vertex for planar triangulations that is due to Tutte's enumeration work [15].

However, Gotsman [2] has shown subsequently that the entropy analysis of Alliez and Desbrun includes many degree distribution that do not correspond to actual triangulations. He incorporates additional constraints on the distribution that lowers its worst-case entropy below Tutte's bound. This means that there are fewer valid permutations of degrees than triangulations and that additional information is necessary to distinguish between. So the split information does contribute a small but necessary fraction to the encoding and is therefore not negligible. Our results also show that the split information recorded by the TG coder is not redundant and are therefore supportive of Gotsman's findings.

References

[1] P. Alliez and M. Desbrun. Valence-driven connectivity encoding for 3D meshes. In *Eurographics'01 Conference Proceedings*, pages 480–489, 2001.

[2] C. Gotsman. On the optimality of valence-based connectivity coding. *Computer Graphics Forum*, 22(1):99–102, 2003.

[3] S. Gumhold and W. Strasser. Real time compression of triangle mesh connectivity. In *SIGGRAPH'98 Conference Proceedings*, pages 133–140, 1998.

[4] M. Isenburg. Compressing polygon mesh connectivity with degree duality prediction. In *Graphics Interface'02*, pages 161–170, 2002.

[5] M. Isenburg and S. Gumhold. Out-of-core compression for gigantic polygon meshes. In *SIGGRAPH'03 Conference Proceedings*, pages 935–942, 2003.

[6] M. Isenburg and P. Lindstrom. Streaming meshes. In *manuscript*, pages 1–8, 2004.

[7] M. Isenburg and J. Snoeyink. Face Fixer: Compressing polygon meshes with properties. In *SIGGRAPH'00 Conference Proceedings*, pages 263–270, 2000.

[8] M. Isenburg and J. Snoeyink. Spirale reversi: Reverse decoding of the Edgebreaker encoding. In *Proceedings of 12th Canadian Conference on Computational Geometry*, pages 247–256, 2000.

[9] M. Isenburg and J. Snoeyink. Early-split coding of triangle mesh connectivity. In *manuscript*, pages 1–8, 2005.

[10] J. Li and C. C. Kuo. A dual graph approach to 3D triangular mesh compression. In *Proceedings of ICIP'98*, pages 891–894, 1998.

[11] D. Poulalhon and G. Schaeffer. Optimal coding and sampling of triangulations. In *30th International Colloquium on Automata, Languages and Programming (ICAZLP)*, pages 1080–1094, 2003.

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

[13] J. Rossignac and A. Szymczak. Wrap&zip: Linear decoding of planar triangle graphs. *The Journal of Computational Geometry, Theory and Applications*, 1999.

[14] C. Touma and C. Gotsman. Triangle mesh compression. In *Graphics Interface'98*, pages 26–34, 1998.

[15] W.T. Tutte. A census of planar triangulations. *Canadian Journal of Mathematics*, 14:21–38, 1962.