UNC Technical Report TR-02-048

Compressing Polygon Mesh Connectivity with Degree Duality Prediction

Martin Isenburg
University of North Carolina at Chapel Hill
isenburg@cs.unc.edu

Abstract

In this paper we present a coder for polygon mesh connectivity that delivers the best connectivity compression rates meshes reported so far. Our coder is an extension of the vertex-based coder for triangle mesh connectivity by Touma and Gotsman [26]. We code polygonal connectivity as a sequence of face and vertex degrees and exploit the correlation between them for mutual predictive compression. Because low-degree vertices are likely to be surrounded by high-degree faces and vice versa, we predict vertex degrees based on neighboring face degrees and face degrees based on neighboring vertex degrees.

Key words: Connectivity coding, graph coding, mesh compression, non-manifold meshes, degree duality.

1 Introduction

A polygon mesh is the most widely used primitive for representing three-dimensional geometric models. Such polygon meshes consists of mesh *geometry* and mesh *connectivity*, the first describing the positions in 3D space and the latter describing how to connect these positions together to form polygons that describe a surface. Typically there are also mesh *properties* such as texture coordinates, material attributes, etc. that describe the visual appearance of the mesh at rendering time.

The standard representation of a polygon mesh uses an array of floats to specify the positions and an array of integers containing indices into the position array to specify the polygons. A similar scheme is used to specify the various properties and how they are attached to the mesh. For large and detailed models this representation results in files of substantial size, which makes their storage expensive and their transmission slow.

The need for more compact representations has motivated researchers to develop efficient mesh compression techniques. Most of these efforts have focused on connectivity compression [4, 25, 26, 21, 9, 22, 18, 11, 3, 12, 19, 23, 1]. There are two reasons for this: First, this is where the largest gains are possible, and second, the connectivity coder is the core component of a compression engine and usually drives the compression of geometry [4, 25, 26, 15], of properties [4, 24, 3], and of how properties are attached to the mesh [24, 12, 13].

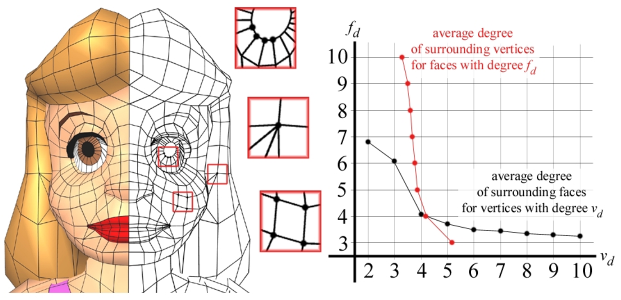


Figure 1: On the left is a close-up of the cupie mesh. Notice that low-degree vertices are likely to be surrounded by high-degree faces and vice-versa. On the right are two plots confirming this. They report the average degree of surrounding faces (vertices) for vertices (faces) of degree d for our example models.

In this paper we introduce a connectivity coder for polygon meshes that achieves the best compression rates reported so far. Our *Degree Duality coder* extends Touma and Gotsman's triangle mesh compression scheme [26] to polygon meshes and borrows ideas from a paper by Alliez and Desbrun [1] to improve the compression rates. The scheme by Touma and Gotsman codes the connectivity of triangle meshes as a sequence of vertex degrees. Our scheme codes the connectivity of polygon meshes as a sequence of vertex degrees *and* a sequence of face degrees. Furthermore it exploits the correlation between neighboring vertex and face degrees for mutual predictive compression. Low degree vertices are more likely to be surrounded by higher-degree faces and vice versa as illustrated in Figure 1. We predict vertex degrees based on the degree of neighboring faces and we predict face degrees based on the degree of neighboring vertices[1].

2 Coding Mesh Connectivity

Representing the connectivity of a mesh with a list of position indices requires at least $kn \log_2 n$ bits, where n is the total number of positions and k is the average number of times each position is indexed. For pure triangular

[1]A similar coder was developed independently and during the same time period by a group of researchers at CalTech and USC [17].

meshes, k tends to be around 6, while for polygon meshes is closer to 4. The problem with this representation is that the space requirement increases super-linearly with the number of positions, since $\log_2 n$ bits are needed to index a position in an array of n positions.

Efficiently encoding mesh connectivity has been subject of intense research and many techniques have been proposed. Initially most of these schemes were designed for fully triangulated meshes [4, 25, 26, 21, 9, 22, 11, 23, 1], but more recent approaches [18, 12, 19, 17] handle arbitrary polygonal input. These schemes do not attempt to code the position indices directly—instead they only code the *connectivity graph* of the mesh.

If a polygon mesh is *manifold*, then every face (i.e. every edge loop) of its connectivity graph corresponds either to a polygon or a hole. Furthermore, every vertex of the graph has a corresponding position in 3D space. This implies that for representing mesh connectivity, it is sufficient to specify (a) the connectivity graph of the mesh and (b) which of its faces are polygons/holes. The mapping from graph vertices to positions can be established with an order derived from the graph connectivity.

Hence, mesh connectivity is compressed by coding the connectivity graph (plus some additional information to distinguish polygons from holes) *and* by changing the order in which the positions are stored. They are arranged in the order in which their corresponding graph vertex is encountered during some deterministic graph traversal. Since encoding and decoding of the connectivity graph also requires a traversal, the positions are often reordered as dictated by this encoding/decoding process.

This basically reduces the number of bits needed for storing mesh connectivity to whatever is required to code the connectivity graph. This is good news: the connectivity graph of a polygon mesh with sphere topology is homeomorphic to a planar graph. It well known that such graphs can be coded with a constant number of bits per vertex [27] and exact enumerations exist [28, 29]. If a polygon meshes has handles (i.e. has non-zero genus) its connectivity graph is not planar. Coding such a graph adds a logarithmic number of bits per handle [22], but most meshes have only a very small number of handles.

Unfortunately only the connectivity of manifold polygon meshes can be coded this way. A mesh is *manifold* if all vertices of its connectivity graph have a neighborhood homeomorphic to a disk or a half-disk. Polygonal models that describe solid objects tend to have this property. However, when generated from other surface representations (e.g. trimmed NURBS) non-manifoldness is often introduced by mistake. Also hand-authored content is frequently non-manifold, especially if the artist tried to optimize the mesh (e.g. minimize its polygon count).

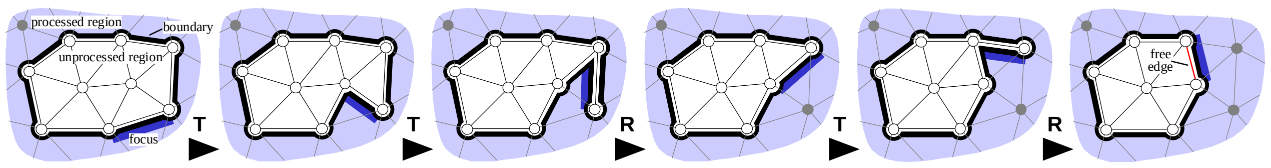
Optimally coding non-manifold graphs directly is hard and there are no efficient solutions yet. Most schemes either require the input mesh to be manifold or use a preprocessing step that cuts non-manifold meshes into manifold pieces [7]. A notable exception is the layering scheme proposed by Bajaj et al. [3], but this seems quite complicated to implement. Cutting a non-manifold mesh replicates all vertices that sit along a cut. Since it is generally not acceptable to modify a mesh during compression, the coder needs to describe how to stitch the mesh pieces back together. Guéziec et al. [6] report how to do this in an efficient manner. Our Degree Duality coder implements a much simpler stitching scheme at the expense of less compression. For typical meshes with few replicated vertices the use of a simpler scheme is sufficient.

3 Coding Manifold Connectivity Graphs

A planar graph with v vertices, f faces, and e edges can be partitioned into two dual spanning trees. One tree spans the vertices and has $v - 1$ edges, while its dual spans the faces and has $f - 1$ edges. Summing these edge counts results in Euler's relation $e = (v-1)+(f-1)$ for planar graphs. Turan [27] observed that this partition can be used to encode planar graphs. He gave an encoding that used 12 bits per vertex (bpv). Improving on Turan's work, Keeler and Westbrook report a 9.0 bpv encoding for planar graphs, which they can specialize to a 4.6 bpv encoding if the graph is fully triangulated [16].

Taubin and Rossignac were first to use these graph coding techniques for compressing the connectivity of triangle meshes. Their Topological Surgery [25] method run-length encodes both spanning trees and adds a few bits per handle for non-planar connectivity graphs, which results in bit-rates of around 4 bpv in practice.

All recent connectivity compression schemes [26, 9, 21, 22, 5, 12, 19] code this information by following the same *region growing* approach: An iterative process encodes edges/faces adjacent to the already processed region (one at a time) and produces a stream of symbols that describe (a) the degree of each processed face and (b) the adjacency relation between a processed edge/face to the processed region. These schemes maintain one or several boundary loops that separate a single processed region from all unprocessed regions. The edges and vertices on the boundary are called *boundary edges* and *boundary vertices* and they are considered *visited*. Each boundary encloses an unprocessed region. The edges, faces, and vertices of this region are called *unprocessed*. If the connectivity graph has handles, then boundaries can be nested, in which case an unprocessed region is enclosed by more than one boundary. Each boundary has a distinguished edge called the *focus*. The algorithm works on



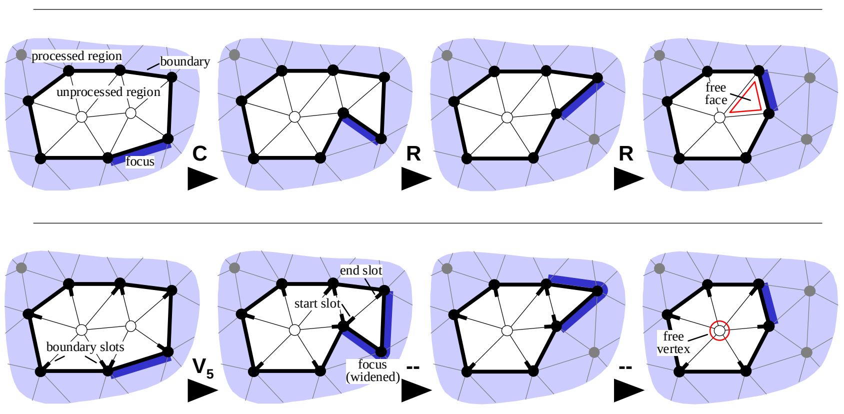


Figure 2: The three different approaches to connectivity coding by region growing: edge-based (top), face-based (middle), and vertex-based (bottom). Every iteration of the edge-based coder processes the *free edge* and describes its adjacency relation to the active boundary. Similarly every iteration of the face-based coder processes the *free face* and describes the adjacency relation. For both coders these descriptions specify how the active boundary is updated. When the vertex-based coder processes a face, it only needs to describe what happens at the *free vertex* of this face to specify the boundary update.

the focus of the *active boundary*, while all other boundaries are kept in a stack.

The adjacency relation between an edge or a face and the already processed region can be described in terms of its adjacency relation to the active boundary. For the case of non-zero genus meshes there will be one situation per handle in which this relation involves a boundary from the stack. The characterizing difference between the compression schemes mentioned earlier is: (a) how the boundaries are defined, (b) how processing an edge or a face updates the boundaries, and (c) with which graph elements the description of the update is associated. Depending on the latter the compression schemes can be classified as *edge-based*, *face-based*, and *vertex-based*.

We will now highlight the exact difference between these three classes of coders. First we do this for the case of pure triangular connectivity. Because all faces are triangles (i.e. have degree three) there is no need to record face degrees. Then we show how each class of coder extends to code arbitrary polygonal connectivity. To simplify this classification we temporarily assume a mesh of sphere topology without boundary, so that we can ignore how to deal with holes and handles.

Edge-based schemes [20, 10] describe all boundary updates per edge. The boundaries are loops of half-edges that separate the region of processed edges from the rest. Each iteration grows the processed region by the edge adjacent to the focus of the active boundary (see Figure 2). This *free edge* is either adjacent to an unprocessed triangle or to the active boundary (in one of four different ways). Triangle Fixer [10] describes the boundary updates corresponding to these five configurations using the symbols T, R, L, S, or E. The Dual Graph method [20] does the same but replaces each pair of symbols S and E by a symbol S_j. Its associated offset j represents a distance in number of edges along the boundary.

Face-based schemes [9, 22, 5] describe all boundary updates per face. The boundaries are loops of edges that separate the region of processed faces from the rest. Each iteration grows the processed region by the face adjacent to the focus of the active boundary. This *free face* can be adjacent to the active boundary in one of five ways. Edgebreaker [22] describes the boundary updates corresponding to these five configurations using the symbols C, R, L, S, or E. Again, each pair of symbols S and E can be replaced by a symbol S_j as done by the Cut-Border Machine [9]. The lowest guaranteed worst-case bounds for coding triangular connectivity have been established for face-based schemes [8].

Vertex-based schemes [26] describe all boundary updates per vertex. The boundaries are loops of edges that separate the region of processed faces from the rest. Furthermore, they store for every boundary vertex the number of *free degrees* or *slots*, which are unprocessed edges incident to the respective vertex. Each iteration grows the processed region by the face adjacent to the *widened focus* of the active boundary. The focus often needs to be widened such that there is a *start slot* and an *end slot* for the face. Only if the processed face has a *free vertex* that is not part of the widened focus, the boundary update needs to be described. Two scenarios are possible: either the free vertex has not been visited, in which case its degree is recorded, or is has been visited, in which case a its distance in slots along the active boundary is recorded and the active boundary is split.

Under the assumption that no splits occur the resulting code sequence contains the degree of every vertex. A result by [2] that was published in [1] shows that the entropy of this sequence asymptotically approaches the number of bits needed to encode an arbitrary triangulated planar graph as found by Tutte's enumeration [28]. However, while it is possible to significantly reduce the number of splits using a sophisticated region growing strategy as proposed by [1], we will later show that such heuristics cannot guarantee to avoid splits completely.

The extension of edge-based schemes to polygon meshes is simple [21, 12]. Whenever the free edge is adjacent to an unprocessed face, its degree d is recorded. For the Face Fixer scheme [12] symbol T is simply replaced with symbol F_d. The extension of face-based schemes to polygon meshes is more complex [18, 19]. The number of possible configurations in which a face of degree d can be adjacent to the active boundary equals the Fibonacci number $F(2d-1)$ [18]. For a quadrilateral face, for example, there are 13 possible configurations. The lowest guaranteed worst-case bounds for coding pure quadrangular meshes have been established for face-based schemes by using a *splitting-rule* [18]. It splits each quadrilateral into two triangles such that the probability for the 13 possible Edgebreaker label combinations can be exploited for compression.

Previously proposed vertex-based schemes only handle triangular connectivity. In this paper we propose the extension to polygonal connectivity.

4 Coding with Face and Vertex Degrees

The vertex-based coder by Touma and Gotsman [26] codes the connectivity graph of a manifold triangle mesh as a sequence of vertex degrees. We now describe how to extend their approach to code the connectivity graph of a manifold *polygon* mesh using a sequence of vertex degrees *and* a sequence of face degrees. Like for triangle meshes, occasionally a split or a merge operation is needed instead of a vertex degree.

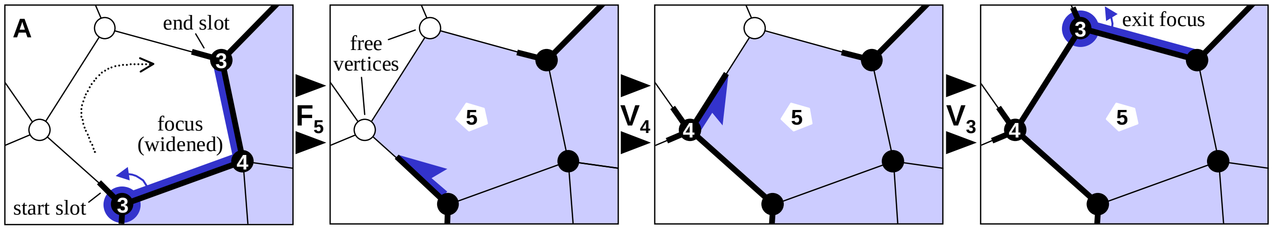
Encoding: Starting with a connectivity graph of v vertices and f faces, the encoder produces two symbol sequences: one is a sequence of $f-1-s+m$ face degree symbols F_d, the other is a sequence of $v+s+m$ symbols which consists of v vertex degree symbols V_d, s split operation symbols S_j with associated offset j, and m merge operation symbols $M_{i,k}$ with associated index i and offset k. The connectivity graph can be reconstructed by simultaneously processing both symbol sequences.

The coder maintains one or several loops of *boundary edges* that separate a single processed region from all unprocessed regions. Furthermore, it stores for every *boundary vertex* the number of *free degrees* or *slots*, which are unprocessed edges incident to the respective vertex. Each of these *boundaries* encloses an unprocessed region; its faces, vertices, and edges are called *unprocessed*. In the presence of handles one boundary can contain another, in which case they enclose the same unprocessed region. Each boundary has a distinguished boundary edge called *focus*. The algorithm works on the focus of the *active boundary*, while the other boundaries are kept in a stack.

The initial active boundary is defined counterclockwise around an arbitrary edge, one of its two boundary edges is defined to be the focus. Each iteration of the algorithm processes the face adjacent to the focus of the active boundary. This involves recording its degree and processing its *free vertices* as illustrated by the three examples in Figure 3. Since including a face consumes two boundary slots we sometimes need to widen the focus until there is a *start slot* and an *end slot* for the face. The number of *focus vertices* is called the *width* of the focus. In scenarios **A**, **B**, and **C** of Figure 3 the focus has a width of 3, 2, and 4 respectively. The *free vertices* are those that are not part of the widened focus.

The free vertices are then processed in counterclockwise order starting from the start slot. Three different cases can arise. According to the original reference [26] we call them *add*, *split*, and *merge* (see Figure 3). By far most the frequent case is *add*, which happens whenever the free vertex has not been previously visited. In this case we record the vertex degree d for which we will use the symbol V_d. However, when we encounter a free vertex that has already been visited we either have a *split* or a *merge*. The latter occurs only for meshes with handles (e.g. with non-zero genus). In this case the free vertex is on a stack boundary, which causes the active boundary to *merge* with the respective stack boundary. We record the index i of that boundary in the stack and the number of slots k between the focus of the stack boundary and the merge slot, denoted by symbol $M_{i,k}$. In the other case the free vertex is on the active boundary, which causes the active boundary to *split* into two. We push one part on the stack and record the number of slots j between the new stack focus and the split slot, denoted by symbol S_j.

After processing all free vertices we exit the face and move to the next focus (see Section 6). This repeats until all faces have been processed. Notice that for each boundary we do not need to record the degree of its last face. At this point a boundary has no slots left and wraps around this face. Therefore the number of recorded face degrees F_d equals at most the number of faces f minus



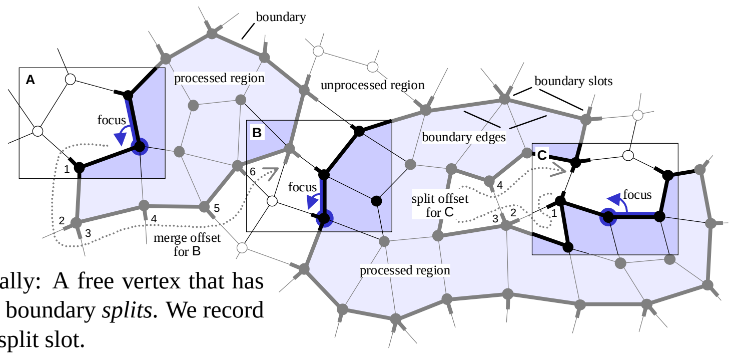
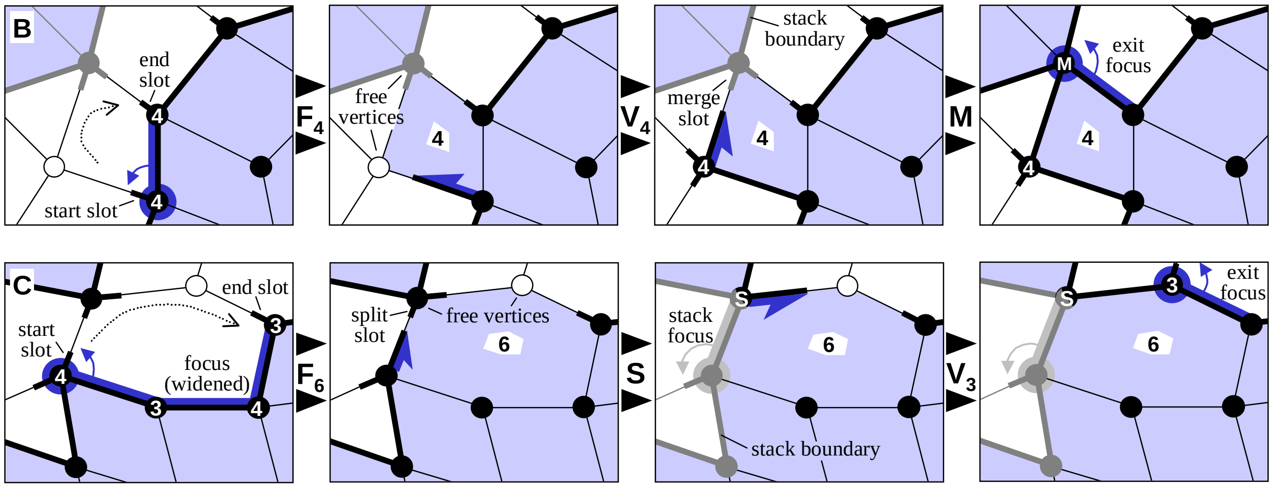


Figure 3: The three frame sequences **A**, **B**, and **C** illustrate different scenarios that can arise when processing a face. **A** is the most common one: The free vertices of the face have not been visited before, we *add* them to the boundary and record their degree. **B** only occurs for meshes with handles: A free vertex that has already been visited is on a boundary in the stack. The active boundary *merges* with this stack boundary. We record its stack index and the number of slots between the stack focus and the merge slot. **C** happens occasionally: A free vertex that has already been visited is on the active boundary. The active boundary *splits*. We record the number of slots between the new stack focus and the split slot.



one. Each split increases and each merge decreases the number of boundaries by one. Thus the exact number of face degrees recorded, given we have s split and m merge operations, is $f - 1 - s + m$ (see also Section 8).

Decoding: The decoder exactly replays what the encoder does by performing the boundary updates described by the two symbol sequences. A step by step example run of the decoding process is shown in Figure 4.

Complexity: We assume that the mesh genus is a small constant, so that there are only a constant number of merge operations. Each face is processed once. The cost of processing a face is proportional to its degree plus the cost for processing its free vertices. The sum of all face degrees is linear and each vertex is added once. This leaves us with the split operations. They are the critical ones, because they require to walk the offset along the boundary. Since we always know the length of the boundary we can always walk the shorter way. Thus, in the worst case the boundary consists of all v vertices and is recursively split into half, resulting in a time complexity of $O(v \log_2(v))$.

5 Compressing with Duality Prediction

The two symbol sequences are compressed into a bitstream using a adaptive arithmetic coding [30]. Given sufficiently long input, the compression rate of such a coder converges to the entropy of the input. The entropy for a sequence of n symbols is $-\sum_n \left(p_i \log_2(p_i)\right)$, where the ith symbol occurs with probability p_i.

Whenever a face is processed we need to specify if it represents a polygon or a hole in the mesh. Using the arithmetic coder we code this with two symbols. Similarly whenever a free vertex is processed we need to specify if an add, a split or a merge operation was used. We distinguish between the frequently occurring adds and the other two with three different symbols.

What remains is compressing the face degrees, the vertex degrees, and the offsets and indices associated with split and merge operations. The basic idea is to exploit the fact that high-degree faces tend to be surrounded by low-degree vertices, and vice versa, for predictive compression. For every vertex we know the degree of the face that introduces it. For every face we know the degrees of all vertices of the (widened) focus. We found that using four different predictions each way captures the correlation in the duality of vertex and face degrees quite well. Offsets and indices, on the other hand, are compressed with the minimal number of bits needed based on their known maximal range.

5.1 Compressing Face Degrees

When a face is processed the degrees of all vertices on the (widened) focus are known. The lower their average degree, the more likely this face has a high degree and vice-versa (see Figure 1). This can be exploited by using different contexts for entropy coding the face degrees, dependent on this vertex degree average. In practice the use of four such *face-degree contexts* seems to capture this correlation quite well. We have different contexts for an average vertex degree (a) below 3.3, (b) between 3.3 and 4.3, (c) between 4.3 and 4.9, and (d) above 4.9. These numbers were first chosen based on the plot in Figure 1 and then corrected slightly based on experimental results.

Each of the four face-degree contexts contains 4 entries: The first three entries represent face degrees 3, 4, and 5 and the last entry represents higher degree faces. These are subsequently compressed with a special *large-face-degree context*. This special context is also used for faces that correspond to holes in the mesh. All contexts are initialized with uniform probabilities that are adaptively updated. Four bits at the beginning of the code specify face degrees that do not occur in the mesh. Their representing entry is disabled in all contexts. For our set of example meshes, this predictive coding of face degrees improves the bit-rates on average by 12.2 %.

There is another small improvement possible: The minimal degree of the face equals the width of the focus. If the focus is wider than 3 we can improve compression further by disabling those entries of the chosen context that represents *impossible* degrees. Although this improves the compression rates by only 1 or 2 percent, it was simple to integrate into the arithmetic coder.

5.2 Compressing Vertex Degrees

When a free vertex is processed, the degree of the respective face is known. The lower its degree, the more likely this vertex has a high degree and vice-versa. Again we exploit this for better compression by using four different contexts. We switch the *vertex-degree context* depending if the face is a triangle, a quadrangle, a pentagon, or a higher degree face.

Each of the four vertex-degree contexts contains 9 entries: The first eight entries represent vertex degrees 2 to 9 and the last entry represents higher degree vertices. These are subsequently compressed with a special *large-vertex-degree context*. All contexts are initialized with uniform probabilities that are adaptively updated. Nine bits at the beginning of the code specify vertex degrees that do not occur in the mesh. Their representing entry is disabled in all contexts.

For our set of example meshes, this predictive coding of vertex degrees improves the bit-rates on average by 6.4 %. Predictive coding of vertex degrees does not improve the compression rates as much as predictive coding of face degrees, because we use less information for each prediction. While each face degrees is predicted using an average of two or more vertex degrees, each vertex degree is only predicted by a single face degree.

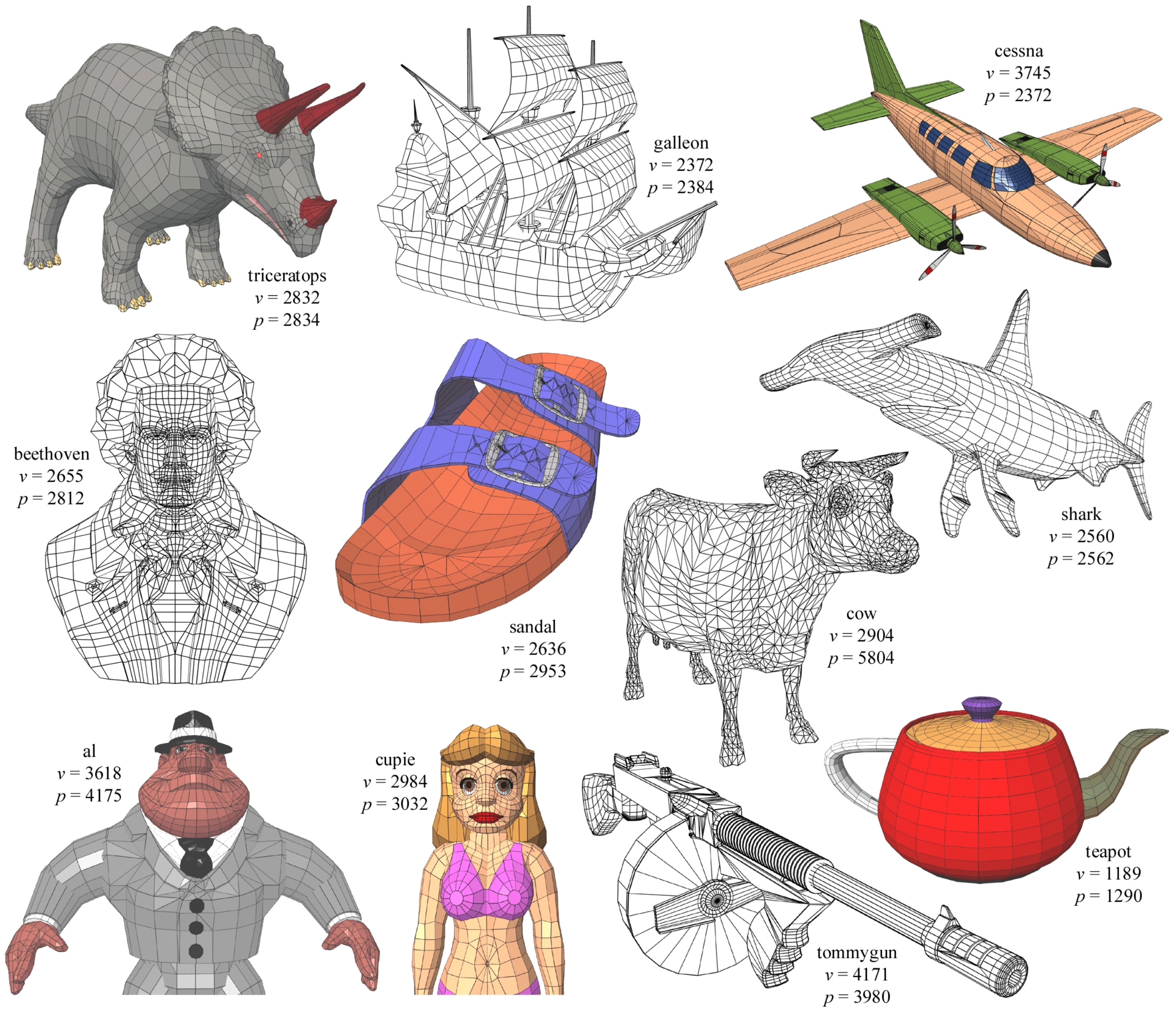
5.3 Compressing Offsets and Indices

An integer number that is known to be between 0 and n can be encoded with exactly $\log_2(n+1)$ bits. We use this for compressing the offsets and indices associated with the split and the merge operation. Whenever a split offset j, a merge index i, or a merge offset k is encoded or decoded, the maximal value of this number is known. For the split offset j it equals the number of slots on the active boundary, for the merge index i it equals the size of the stack, and for the merge offset k it equals the number of slots on the indexed boundary in the stack.

6 Reducing the Number of Splits

After processing a face, we could continue with the exit focus as the next focus. This is the strategy of the original vertex-based coder for triangle meshes proposed by Touma and Gotsman [26]. However, Alliez and Desbrun [1] propose a more sophisticated strategy for picking the next focus that significantly reduces the number of splits. This is beneficial, because split operations are expensive to code: On one hand we need to specify where in the sequence of vertex degrees they occur and on the other hand we need to record their associated split offset.

Since the decoding process has to follow this strategy, the quest for this better focus can only use information that is available to the decoder. Alliez and Desbrun [1] suggest to move the focus to the boundary vertex with the lowest number of slots. In case there is more than one such vertex, they choose the least dense region by averaging over a wider and wider neighborhood. This strategy makes keeping track of the next candidate an expensive



mesh	vertex degree distribution									face degree distribution				holes /		bpv		coding
name	2	3	4	5	6	7	8	9	>9	3	4	5	>5	handles		ff	dd	gain
triceratops	–	8	2816	8	–	–	–	–	–	346	2266	140	82	–	–	2.115	**1.189**	43.8 %
galleon	7	430	1595	270	66	4	1	–	–	336	1947	40	61	–	–	2.595	**2.093**	19.3 %
cessna	8	642	2470	384	178	41	18	1	3	900	2797	180	50	–	–	2.841	**2.543**	10.5 %
beethoven	21	279	1925	295	99	20	14	–	2	680	2078	44	10	10	–	2.890	**2.102**	27.3 %
sandal	–	280	1857	329	95	18	7	12	38	961	1985	7	–	14	12	2.602	**2.115**	18.7 %
shark	–	–	2560	–	–	–	–	–	–	188	2253	83	38	–	–	1.670	**0.756**	54.7 %
al	2	538	1999	720	268	69	15	1	6	1579	2505	44	47	–	–	2.926	**2.429**	17.0 %
cupie	16	272	2405	234	37	12	8	–	–	384	2506	114	28	–	–	2.307	**1.640**	28.9 %
tommygun	–	1557	2002	395	152	21	18	8	18	992	2785	84	119	–	6	2.611	**2.258**	13.5 %
cow	–	7	87	514	1796	364	98	23	15	5804	–	–	–	–	–	2.213	**1.781**	19.5 %
teapot	2	14	1022	125	18	5	1	–	2	215	1070	3	2	–	1	1.669	**1.127**	32.5 %

Table 1: The vertex count v and the polygon count p for the example models is reported above. The table also gives the vertex and face degree distribution for each of these models. We compare the connectivity compression rates (*bpv*) of the Face Fixer coder (*ff*) to those of the proposed Degree Duality coder (*dd*) and report the improvement in percent.

operation. Using a dedicated priority queue, for example, would require $O(\log(b))$ per boundary update, where b is the number of vertices on the active boundary.

The obvious question is whether it is possible to avoid the split operations all together. We can prove that splits cannot be avoided by using a strategy that only uses the already encoded/decoded part of the mesh. Given any such strategy we can always construct a mesh that is guaranteed to result in a split.

First of all, the connectivity of a non-zero genus mesh will require at least as many splits as the mesh has handles. But also for meshes without handles a split operation can occur: Imagine your favorite mesh of torus topology. The encoder eventually has to use the merge operation to code the handle. Every merge operation is preceded by a split operation. In the moment this split operation is performed, we stop the encoding process, perform an edge cut in the unprocessed region such that it opens the handle, insert two large polygons or holes into the cut, and continue the encoding process on the mesh (which now has sphere topology). The coder did not notice what happened, because the edge cut was performed in the region it has not yet seen. But now the coder has produced a split for a mesh of genus zero.

Nevertheless, to reduce the number of splits is especially important in the polygonal case, because here a split operation can pinch off parts of the boundary that do not enclose any unprocessed vertices and that can be as small as a single unprocessed face. This does not happen in the pure triangular case. Inspired by Alliez and Desbrun [1] we suggest a similar, but simpler heuristic to pick the next focus. Most importantly, our strategy does not affect the asymptotic complexity of the decoder.

The focus is moved to the boundary vertex with the smallest number of slots in counterclockwise direction as seen from the current focus. This current focus is usually the exit focus of the face processed last or the stack focus if a new boundary was just popped of the stack. However, we only move the focus if the smallest number of slots is 0 or 1, otherwise the focus remains where it is. Table 2 reports the success of this strategy in reducing the number of splits and the bit-rate.

Starting a brute-force search along the boundary for the vertex with the smallest number of slots would mean a worst-case time complexity of O(n^2). Instead we keep track of the next 0 and the next 1 slot by organizing all of them into two cyclic linked lists. In both lists we always point to the slot that is closest in counterclockwise direction and perform the necessary updates as the boundary changes. This data structure can be maintained without affecting the asymptotic complexity of the decoder.

mesh	current		0 or 1 slot		coding
name	# splits	bpv	# splits	bpv	gain
triceratops	53	1.311	**25**	**1.189**	9.3 %
galleon	78	2.309	**18**	**2.093**	9.4 %
cessna	172	2.882	**28**	**2.543**	11.8 %
beethoven	99	2.431	**15**	**2.102**	13.5 %
sandal	85	2.295	**25**	**2.115**	7.8 %
shark	24	0.818	**13**	**0.756**	7.6 %
al	92	2.616	**14**	**2.429**	7.1 %
cupie	56	1.786	**15**	**1.640**	8.2 %
tommygun	131	2.449	**32**	**2.258**	7.8 %
cow	154	2.313	**13**	**1.781**	23.0 %
teapot	10	1.167	**3**	**1.127**	3.4 %

Table 2: The number of splits and the resulting bit-rate using the *current* focus compared to moving the focus to the next 0 *or* 1 *slot* and the coding improvement in percent.

7 Coding Non-Manifold Meshes

Compared to Guéziec et al. [6] our Degree Duality coder implements a much simpler stitching scheme to recover non-manifold connectivity, that allows a robust, minimal-effort implementation at the expense of less efficiency. However, the number of non-manifold vertices is typically small, which justifies the use of a simpler scheme.

Whenever a free vertex is processed by an add operation we simply specify if this indeed is a new position or not using arithmetic coding. If it is a new position we increment the position counter. Otherwise it is an old position and its index needs to be compressed as well. We can do this with $\log_2(n)$ bits where n is the number of positions already encoded/decoded.

8 Counts and Invariants

The sum of all v vertex degrees and the sum of all f face degrees both equal twice the number of edges e. That means the two sums are equal.

$$\sum_{k=1}^{f} deg(\mathrm{f}_k) = \sum_{k=1}^{v} deg(\mathrm{v}_k) = 2e \qquad (1)$$

If we know all vertex degrees and all face degrees but one we can compute it as the one completing the equality.

$$\mathrm{f}_f = \sum_{k=1}^{v} deg(\mathrm{v}_k) - \sum_{k=1}^{f-1} deg(\mathrm{f}_k) \qquad (2)$$

Furthermore we have the following invariants: The sum of degrees of all unprocessed faces minus the number of all boundary edges b equals twice the number of unprocessed edges u. And also the sum of degrees of all unprocessed vertices plus the number of all boundary slots s equals twice the number of unprocessed edges u.

$$\sum_{\mathrm{f}_k \subset \mathcal{B}} deg(\mathrm{f}_k) - b = \sum_{\mathrm{v}_k \subset \mathcal{B}} deg(\mathrm{v}_k) + s = 2u \quad (3)$$

where $\subset \mathcal{B}$ means unprocessed (or enclosed by some boundary). Furthermore, this invariant is true for the face and vertex degree count of every unprocessed regions together with edge and slot count of the respective boundaries that enclose it. In the moment a split occurs, one such equation $\mathcal{E}$ is split into two new equations $\mathcal{E}'$ and $\mathcal{E}''$ that are related with $s = s' + s''$, $b = b' + b''$, $u = u' + u''$, and the correspondingly *split* sums of unprocessed face and vertex degrees. The offset associated with the split operation specifies s'' and b''. Together with the two degree sequences they specify implicitly when each boundary end. This explains why we can omit one face degree for every split operation—it creates a new equation just like (2) that can be solved for a single face degree.

9 Summary and Discussion

We have described coder for polygon mesh connectivity that delivers the best connectivity compression rates meshes reported so far. On our example models the compression rates improve between 10 % to 55 % over those of the Face Fixer coder [12] with an average improvement of 26 %. Furthermore, we provide a web page [14] containing a prototype implementation of our coder in pure java that proves the bit-rates reported in Table 1.

Our main contribution is (a) the extension of vertex-based coding to polygonal connectivity using a sequences of vertex degrees *and* a sequence of face degrees (b) the observation that the correlation in the duality of the degrees can be used for mutual predictive compression.

Khodakovsky et al. [17] extend a result by [2] that was published in [1] to show that summed entropies of the face degree sequence and of the vertex degree sequence converges to Tutte's bound on the enumeration of planar graphs [29]. This seems to suggest that degree coding is optimal in the sense that it uses not more bits to encode a connectivity than needed to distinguish it among all possible connectivities with the same number of vertices.

This does not mean that degree coding always outperforms other coders. We can construct pathologic examples where other coders perform better. Using the cow model from Table 1 we generated a triangle mesh and a quadrangle mesh that demonstrate this. We generated the triangle mesh by placing a new vertex into every triangle of the original mesh and by connecting it to its three vertices. All new vertices have degree three, while the degree of every vertex of the original mesh doubles. This connectivity compresses to 0.988 bpv using Edgebreaker [22], whereas the Degree Duality coder needs 1.569 bpv. Similarly we generated the quadrangle mesh by placing a new vertex into every original triangle and by connecting it to the three new vertices that are placed on every original edge. All new vertices have either degree three or degree four, while the degree of the original vertices remains unchanged. This connectivity compresses to 1.376 bpv using Face Fixer [12], whereas the Degree Duality coder needs 1.721 bpv. However, such pathological cases rarely occur in practice.

The author thanks Jack Snoeyink for helpful comments and thorough reviews of the manuscript.

10 References

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

[2] P. Alliez, M. Desbrun, S. Gumhold, M. Isenburg, and C. Gotsman. *personal communication*, March 2001.

[3] C. Bajaj, V. Pascucci, and G. Zhuang. Single resolution compression of arbitrary triangular meshes with properties. In *Data Compression Conference'99 Conference Proc.*, pages 247–256, 1999.

[4] M. Deering. Geometry compression. In *SIGGRAPH'95 Conference Proceedings*, pages 13–20, 1995.

[5] L. de Floriani, P. Magillo, and E. Puppo. A simple and efficient sequential encoding for triangle meshes. In *Proc. of 15th European Workshop on Computational Geometry*, pages 129–133, 1999.

[6] A. Guéziec, F. Bossen, G. Taubin, and C. Silva. Efficient compression of non-manifold polygonal meshes. In *Visualization'99 Conference Proceedings*, pages 73–80, 1999.

[7] A. Guéziec, G. Taubin, F. Lazarus, and W.P. Horn. Converting sets of polygons to manifold surfaces by cutting and stitching. In *Visualization'98 Conference Proceedings*, pages 383–390, 1998.

[8] S. Gumhold. New bounds on the encoding of planar triangulations. Technical Report WSI-2000-1, Tübingen, March 2000.

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

[10] M. Isenburg. Triangle Fixer: Edge-based connectivity compression. In *Proceedings of 16th European Workshop on Computational Geometry*, pages 18–23, 2000.

[11] M. Isenburg and J. Snoeyink. Mesh collapse compression. In *SIBGRAPI'99 Conference Proceedings*, pages 27–28, 1999.

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

[13] M. Isenburg and J. Snoeyink. Compressing the property mapping of polygon meshes. In *Pacific Graphics'01*, pages 4–11, 2001.

[14] http://www.cs.unc.edu/˜isenburg/pmc/

[15] Z. Karni and C. Gotsman. Spectral compression of mesh geometry. In *SIGGRAPH'00 Conference Proc.*, pages 279–286, 2000.

[16] K. Keeler and J. Westbrook. Short encodings of planar graphs and maps. In *Discrete Applied Mathematics*, pages 239–252, 1995.

[17] A. Khodakovsky, P. Alliez, M. Desbrun, and P. Schroeder. Near-optimal connectivity encoding of 2-manifold polygon meshes. *to appear in Graphic Models*, 2002.

[18] D. King, J. Rossignac, and A. Szymczak. Connectivity compression for irregular quadrilateral meshes. Georgia Tech, TR–99–36.

[19] B. Kronrod and C. Gotsman. Efficient coding of non-triangular meshes. In *Proc. of Pacific Graphics*, pages 235–242, 2000.

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

[21] J. Li, C. C. Kuo, and H. Chen. Mesh connectivity coding by dual graph approach. Technical report, March 1998.

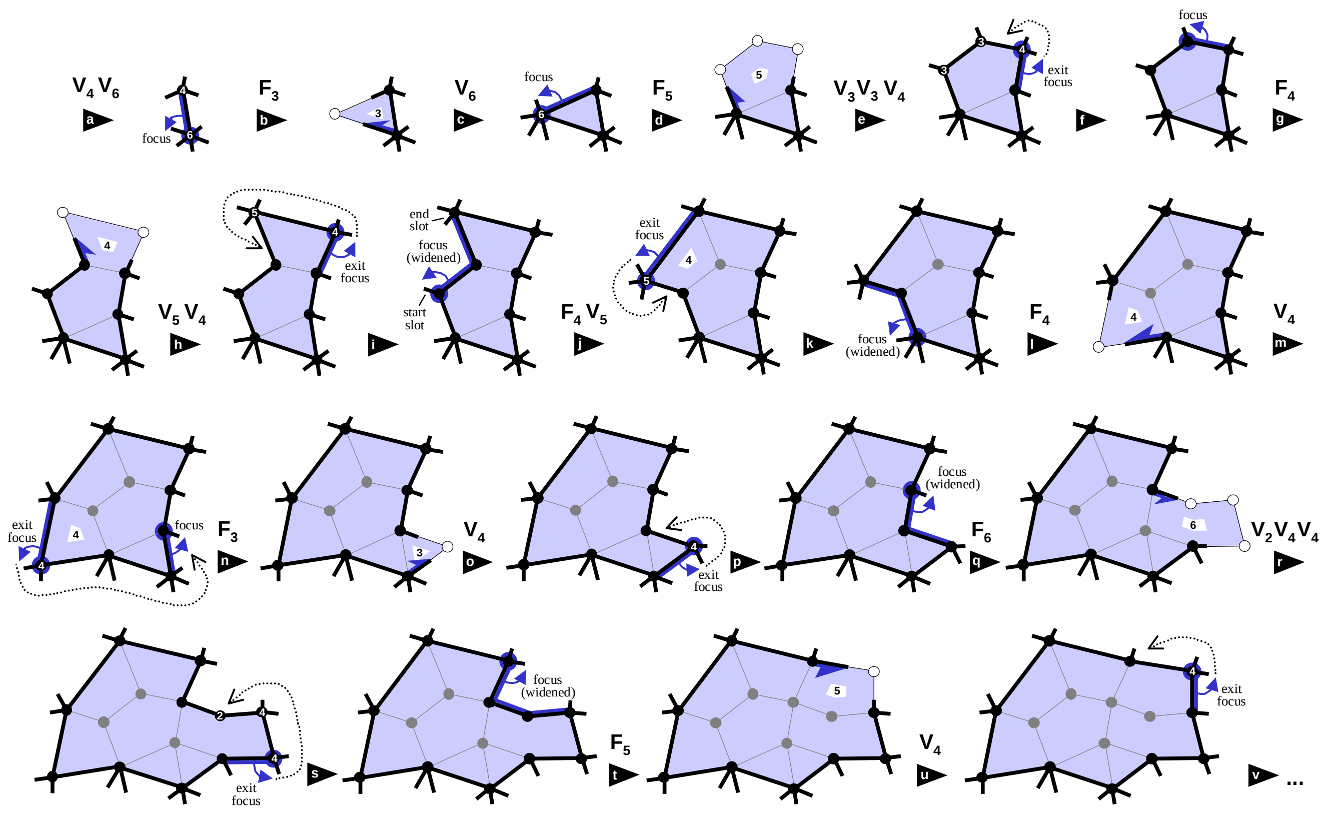


Figure 4: This is an example run of the decoding algorithm. It is an exact replay of the boundary updates performed during encoding: **(a)** The decoder creates the initial boundary by uncompressing the first two vertex degrees. Encoder and decoder use their order to agree on the initial focus. **(b)** Uncompress the first face degree. The average focus vertex degree of 5.0 determines which *face-degree context* the arithmetic decoder uses. **(c)** Uncompress the degree of the free vertex. The face degree of 3 determines which *vertex-degree context* the arithmetic decoder uses. The focus remains at the exit focus, because there is no vertex on the active boundary that has 0 or 1 slots. **(d)** Uncompress the next face degree. The average focus vertex degree that determines the face-degree context is again 5.0. **(e)** Uncompress the degrees of the three free vertices. Now the face degree that determines the vertex-degree context is 5. **(f)** The focus moves in counterclockwise direction along the boundary to the next boundary vertex with the lowest number of slots, which is 1 in this case. **(g)** Uncompress the next face degree. Use average focus vertex degree of 3.5 to determine the face-degree context (e.g. $fdc = 3.5$). **(h)** Uncompress the degrees of the two free vertices. Use face degree 4 to determine the vertex-degree context (e.g. $vdc = 4$). **(i)** The focus is moved in counterclockwise direction to the lowest number of slots, which is 0 in this case. Then the focus is widened such that there is a start slot and an end slot for the next face to process. **(j)** Uncompress the next face degree ($fdc = 3.6$) and uncompress the degree of its free vertex ($vdc = 4$). **(k)** Move the focus in counterclockwise direction to the vertex with 0 slots and widen the focus. **(l)** Uncompress the next face degree ($fdc = 4.6$). **(m)** Uncompress the degree of its free vertex ($vdc = 4$) and move the focus. **(n)** Uncompress the next face degree ($fdc = 5.0$). **(o)** Uncompress the degree of its free vertex ($vdc = 3$). **(p)** Move and widen the focus. **(q)** Uncompress the next face degree ($fdc = 4.0$). **(r)** Uncompress the degree of its three free vertices ($vdc = 6$). **(s)** Move and widen the focus. **(t)** Uncompress the next face degree ($fdc = 3.5$). The focus has a width of 4, therefore the face degree is also at least 4. We disable the entry of the chosen context that represents the *impossible* degree 3. **(u)** Uncompress the degree of its free vertex ($vdc = 5$). **(v)** And so on ...

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

[23] A. Szymczak, D. King, and J. Rossignac. An Edgebreaker-based efficient compression scheme for connectivity of regular meshes. In *Proc. of 12th Cnd. Conf. Comp. Geom.*, pages 257–264, 2000.

[24] G. Taubin, W.P. Horn, F. Lazarus, and J. Rossignac. Geometry coding and VRML. *Proc. of the IEEE*, 86(6):1228–1243, 1998.

[25] G. Taubin and J. Rossignac. Geometric compression through topological surgery. *ACM Trans. on Graph.*, 17(2):84–115, 1998.

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

[27] G. Turan. Succinct representations of graphs. *Discrete Applied Mathematics*, 8:289–294, 1984.

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

[29] W.T. Tutte. A census of planar maps. *Canadian Journal of Mathematics*, 15:249–271, 1963.

[30] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Comm. of the ACM*, 30(6):520–540, 1987.