Connectivity Coding: New Perspectives for Mesh Compression

Stefan Gumhold
WSI/GRIS, Universität Tübingen

Reprint

Abstract

Compact encodings of the connectivity of planar triangulations is a very important subject not only in graph theory but also in computer graphics. For triangle meshes used in computer graphics the topologically planar regions dominate by far. New results by Isenburg et. al [6] even show that the connectivity is sufficient to describe shape by itself. Most coding methods for planar triangulations can be extended to manifolds of bounded genus with the same upper and lower bounds on the bit rate.

In 1962 Tutte enumerated the number of different planar triangulations and his results show, that at least 3.245 bits per vertex are necessary to encode the connectivity graph of planar triangulations. In this article we improve the so far best upper bound [4] and show that the connectivity of a planar triangulation can be encoded with less than 3.525 bits per vertex, while ensuring a linear run time for encoding and decoding.

1 Introduction

This article improves the lowest upper bound for the encoding of planar triangulations with three border edges as defined by Tutte in [11]. Two planar triangulations are defined to be equal, if there exists a bijection between their connectivity graphs that maps all border vertices of the first triangulation to the border vertices of the second triangulation. Tutte enumerated all different planar triangulations and showed in this way that any encoding has to use at least 3.245 bits per vertex for sufficiently large triangulations. So far the best encoding schemes [1] and [10] consumed 4 bits. The latter – the Edge Breaker scheme – could be improved to 3.67 bits per vertex [9] and 3.58 bits per vertex by Gumhold [4].

Planar triangulations are a special case of closed manifold triangle meshes where the genus of the triangle mesh is zero. As most encoding schemes for pla-

nar triangulations can be extended to manifold triangle meshes with border, the schemes are also important in the representation of surface models and have been studied extensively (see [2] for an introduction). Latest work by Isenburg et al. [6] shows that the connectivity contains enough information to describe shape.

The algorithmic scheme of the Edge Breaker [10] is very simple and similar to the work of Itah [8] and to the Cut-Border Machine [5]. It visits the triangles of an edge-connected component of a triangle mesh in an order defined by the triangle connectivity itself. The same traversal is used for encoding and decoding[1]. The connectivity is translated triangle by triangle into one of the five operation symbols CLRSE. By the use of codebooks, the Edge Breaker allows to encode the connectivity of typical triangle meshes to an average of 2.2 bits per triangle, whereas the Cut-Border Machine achieves with arithmetic coding and conditional probabilities an average of 1.9 bits per vertex [3]. These results are only valid for regular meshes. For an arbitrary mesh the improved techniques cannot guarantee a good upper bound.

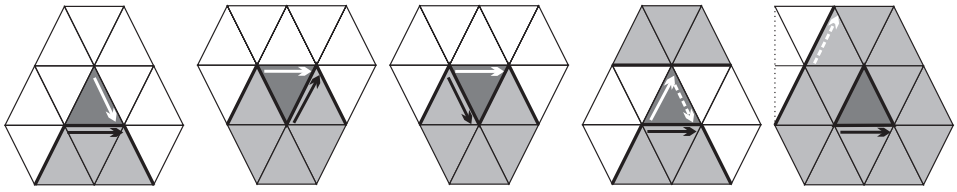
In the following we extend the techniques developed in [4] and give a linear runtime encoding and decoding scheme with an upper bound of 3.525 bits per vertex. Section 2 reviews the Edge Breaker encoding scheme with a small modification on the split operation. In section 3 we describe constraints on the symbol stream produced by the Edge Breaker encoding, that can be exploited to reduced the worst case bit rate. The next two sections 4 and 5 describe techniques to exploit the constraints on the symbol stream. Numerical issues are discussed in section 6. After the results in section 7 we give concluding remarks in section 8.

2 Edge Breaker Coding

The Edge Breaker translates the connectivity of a planar triangulation into a sequence of five different symbols. The encoding algorithm is a region growing algorithm, which stores at any time all vertices and edges of the planar triangulation, which divide the so far encoded triangles from the not yet encoded triangles. These vertices and edges form a set of closed loops, which is called the *cut-border*. Before encoding starts, the cut-border is initialized to one loop containing the external edges and vertices.

Triangles are encoded at a specific cut-border edge, which is called the *gate*. Each time a triangle has been encoded at the gate location, the gate is set to another cut-border edge in a predetermined way such that all cut-border edges will be visited in the end. In the beginning the gate is set to one of the external edges. Fig. 1 shows the different operations and their symbols that we collect in the Edge

[1] where decoding is done in reverse direction



a) *"center"*: C b) *"left"*: L c) *"right"*: R d) *"split"*: S e) *"end"*: E

Figure 1: The five different Edge Breaker operations and the corresponding symbols.

Breaker alphabet $\mathcal{A} \stackrel{\text{def}}{=} \{C, L, R, S, E\}$. The so far encoded triangles are shaded in a light gray in the figure, the cut-border edges are bold black as is the gate before the operation, the currently encoded triangle is dark gray and the new gate after the operation is white. In case of the split operation, the cut-border splits into two loops and for each one a new gate is created. During coding the right gate is pushed on a stack and activated after the left cut-border loop has been completely encoded after an end operation. The encoding permutes the vertices into the order they are encountered during encoding in the center operations.

The only operation which cannot be decoded by simple repetition of the encoding operation is the split operation. Due to Isenburg [7] is the observation that the inverse operations of the Edge Breaker operations can be done without any further knowledge. The decompression is therefore done in reverse order. The symbol string and the vertices are scanned from back to front. The different operations are performed in reverse order. In Fig. 1 we would interpret the white triangles as the so far decoded part, the dark gray triangle is decoded next and the gray triangles have not been decoded yet. The white arrow(s) is/are the gate(s) before the inverse operation and the black arrow is the gate after the inverse operation. In the *"inverse center"* operations (C^{-1}) the newly decoded triangle connects the gate to the previous edge on the cut-border and is the mirror image of the L operation in forward encoding direction. The *"inverse left"* (L^{-1}) replaces the gate by the newly decoded triangle and the gate is set to the right edge of the newly introduced cut-border edge as in the forward C operation. Similarly, the *"inverse right"* (R^{-1}) adds a triangle but puts the gate on the left new edge.

The *"inverse end"* (E^{-1}) creates a new cut-border loop consisting of three cut-border edges. The current loop is pushed onto a loop stack for later use in the inverse split operation. Finally, the *"inverse split"* (S^{-1}) merges the current loop with the top loop on the loop stack. The gate is set to the only newly introduced cut-border edge. As all forward and inverse operations can be performed in constant time, we can state the following theorem:

Theorem 2.1 *The connectivity of a planar triangulation with v vertices and 3 external edges can be encoded with a unique string of length $2v$ over five different symbols in linear time in v. The original connectivity can be decoded also in linear time in v.*

Coding the C-symbol with one bit and the other four symbols with three bits, the Edge Breaker scheme allows to encode any planar triangulation with no more than four bits per vertex[2].

3 Constraints

The upper bound of four bits per vertex can be improved as not all possible symbol strings are allowed. In this section we gather the different constraints on the symbol strings, when read in reverse direction, such that only valid triangulations can be produced:

1. *"inverse center"* (C^{-1}): The inverse center operation removes the gate and the previous edge from the cut-border and adds a new edge connecting their far apart end points. This operation can cause two invalid configurations.
   (a) The removal of one cut-border edge can reduce the length of the current cut-border loop to less than the minimal number of three edges.
   (b) The newly added edge could have been part of the decoded connectivity before the inverse center operation. This is for example always the case if the inverse center operation is performed after an inverse left operation. Then the inverse center operation would add a third triangle to the interior edge introduced by the left operation. But in a planar triangulation an edge with more than two edges is not allowed.
2. *"split"* (S^{-1}): An inverse split operation is only allowed if there is a cut-border loop on the loop stack.
3. *"inverse left / right / end"* (L^{-1} / R^{-1} / E^{-1}): The inverse left, right and end operations all introduce new cut-border edges. Therefore, they are not allowed at the end of the decoding process when the remaining inverse split and center operations cannot reduce the number of cut-border edges to the number of external edges.

[2]Each symbol introduces one triangle. There are twice as many triangles as vertices. Each vertex corresponds to exactly one C symbol. This sums up to $v + 3v = 4v$ bits.

The next two sections describe how to account for the constraints 1a and 1b on the inverse center operations. We do not take into account the other two constraints because it can be shown that they do not influence the bit rate.

4 Conditional Unities

We use arithmetic coding [3] in order to achieve near optimal compression rates with the flexibility to avoid the constraints on the inverse center operations. Suppose $S = s_1 s_2 \dots s_t$ is the reversed symbol stream, where t is the number of operation symbols or as well the number of triangles. In the setting of arithmetic coding one can assign for each symbol from the alphabet a different probability. Different probabilities $p_{i,\alpha}$ can be assigned for each to be encoded symbol s_i depending on all previously encoded symbols $s_1, \dots, s_{i-1}$. An arithmetic coder allows us to encoded the symbol stream with

$$\mathcal{B}(S) \stackrel{\text{def}}{=} \sum_{i=1}^{t} \mathcal{B}(s_i)\text{bits} \qquad \text{where} \qquad \mathcal{B}(s_i) \stackrel{\text{def}}{=} \log_2 \frac{1}{p_{i,s_i}},$$

such that it makes sense to say that a specific symbol consumes for example 2.525 bits. The simplest encoding assigns the probabilities $p_C = \frac{1}{2}$ and $p_{L/R/E/S} = \frac{1}{8}$, corresponding to one and three bits. As in our case the number of triangles t is equal to $2v - 4$, where v is the number of vertices, and as there are $(v - 3)$ C operations, the consumed storage space sums up to $v - 3$ plus $3 \cdot (v - 1)$ equals $4v - 6$ bits, what is less than four bits per vertex.

As in sufficiently large connectivity graphs every second symbol is a C, we assign $p_C = \frac{1}{2}$ and for all other symbols $p_{L/R/E/S} = \tau$, which should be larger than $\frac{1}{8}$ in order to achieve a better bit rate per vertex. With the now introduced concept of *conditional unities* we can keep τ as the constant that tells us the total bit rate via the formula

$$b(S) \stackrel{\text{def}}{=} \frac{\mathcal{B}(S)}{v} = 1 + \log_2 \frac{1}{\tau}. \tag{1}$$

Without any knowledge of the decoding process, we know that the probabilities of the different symbols must sum up to one

$$1 = p_C + p_L + p_R + p_S + p_E \stackrel{?}{=} \frac{1}{2} + 4\tau.$$

If the equality on the right side would be true, we could compute τ to $\frac{1}{8}$ and would end up with a bit rate of four bits per vertex. But here we neglected the fact, that

[3] see [12] for an introduction to arithmetic coding

after we saw an E symbol in the reversed symbol string, no C symbol may follow because it would reduce the number of cut-border edges to two, what is not allowed. Thus the probabilities of the symbols that are allowed under the precondition that the previous symbol has been an E do not sum up to one. Instead they will sum up to a number smaller than one that we denote as the conditional unity ${}^{E}\mathbf{1}$ under precondition that an E symbol preceded the current symbol. The C symbol may neither follow a L symbol because it would destroy the planarity of the triangulation. We can now revise our first equation on the probabilities to three new ones:

$$\begin{aligned} \mathbf{1} &= \frac{1}{2} + \tau\left({}^{L}\mathbf{1} + \mathbf{1} + \mathbf{1} + {}^{E}\mathbf{1}\right) \\ {}^{E/L}\mathbf{1} &= \tau\left({}^{L}\mathbf{1} + \mathbf{1} + \mathbf{1} + {}^{E}\mathbf{1}\right) \\ {}^{E}\mathbf{1} &= \tau\left({}^{L}\mathbf{1} + \mathbf{1} + \mathbf{1} + {}^{E}\mathbf{1}\right). \end{aligned}$$

We see at once that the conditional unities ${}^{L}\mathbf{1}$ and ${}^{E}\mathbf{1}$ are the same resulting in only two equations that can easily be solved for ${}^{L}\mathbf{1} = \frac{1}{2}$ and $\tau = \frac{1}{6}$. Thus the bit rate would be $1 + \log_2 6 < 3.585$ bits per vertex. The corresponding arithmetic coder has two different lists of probabilities $\left(p_{1/2,\alpha}\right)$ – if no precondition holds $\left(\frac{1}{2}, \frac{1}{12}, \frac{1}{6}, \frac{1}{6}, \frac{1}{12}\right)$ and under precondition of an E or L $\left(0, \frac{1}{12}, \frac{1}{6}, \frac{1}{6}, \frac{1}{12}\right)$.

5 The State Machine

With the tool of conditional unities we can exploit the constraints 1a and 1b on page 4.

Constraint 1a does not allow a C symbol if the length of the current cut-border loop is three. Every time when an inverse end operation generates a new loop, we know that the length of the current loop is three afterwards. To remember this knowledge we introduce the conditional probabilities $\mathbf{1}_l$, where l specifies the known length of the current loop. Each inverse left and right operations increment the current cut-border loop by one, whereas the inverse C operation decrements it by one. After the inverse split operation we don't know anything about the length of the current loop as we didn't remember the length of the loop on the stack. Considering only the length of the current loop, the equations for the conditional unities are

$$\begin{aligned} \mathbf{1} &= \frac{1}{2} + \tau\left(3 \cdot \mathbf{1} + \mathbf{1}_3\right) \\ \mathbf{1}_3 &= \tau\left(2 \cdot \mathbf{1}_4 + \mathbf{1} + \mathbf{1}_3\right) \end{aligned}$$

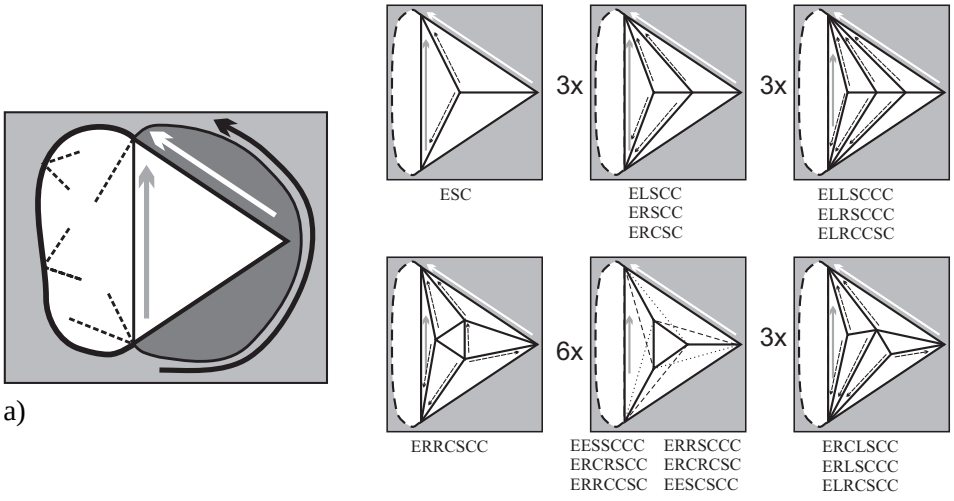


Figure 2: a) Illustration of constraint 1b on page 4. b) Different triangulations inside a triangle with one, two or three interior vertices.

$$\mathbf{1}_l \;=\; \frac{1}{2}\mathbf{1}_{l-1} + \tau\left(2\cdot\mathbf{1}_{l+1} + \mathbf{1} + \mathbf{1}_3\right) \quad \forall l > 3.$$

We can extend this approach by also remembering the length p of the loop on top of the loop stack by introducing the conditional unities $\mathbf{1}_l^p$. This will allow to determine the loop length after an inverse split operation. Then the current loop will have the length $l + p - 1$. The second new rule is that after an inverse end operation the current loop length is stored in the length of the loop on top of the stack resulting in the equations

$$\begin{aligned}
\mathbf{1} &= \frac{1}{2} + \tau\left(3\cdot\mathbf{1} + \mathbf{1}_3\right) \\
\mathbf{1}_3 &= \tau\left(2\cdot\mathbf{1}_4 + \mathbf{1} + \mathbf{1}_3^3\right) \\
\mathbf{1}_l &= \frac{1}{2}\mathbf{1}_{l-1} + \tau\left(2\cdot\mathbf{1}_{l+1} + \mathbf{1} + \mathbf{1}_3^l\right) \quad \forall l > 3 \\
\mathbf{1}_3^p &= \tau\left(2\cdot\mathbf{1}_4^p + \mathbf{1}_{2+p} + \mathbf{1}_3^3\right) \quad \forall p \geq 3 \\
\mathbf{1}_l^p &= \frac{1}{2}\mathbf{1}_{l-1}^p + \tau\left(2\cdot\mathbf{1}_{l+1}^p + \mathbf{1}_{l+p-1} + \mathbf{1}_3^l\right) \quad \forall l > 3 \forall p \geq 3.
\end{aligned}$$

Finally, we introduce the conditional unities ${}_s\mathbf{1}_l^p$ that can also remember the length s of the loop on the second highest position on the loop stack. The equations are extended in the same way.

The second constraint 1b on page 4 says that the edge introduced by the inverse center operation is not allowed to be present in the so far decoded part. Fig. 2 a)

illustrates the second constraint on the C symbol. The so far decoded part is shown in white, the not yet decoded part in gray. The gate before the inverse center operation is the white arrow, the gate after the operation the black arrow. The dark gray shaded triangle with the bent edge is the currently decoded triangle. The bent edge introduced by the inverse center operation is the same as the edge cutting the so far decoded part into a triangle on the right and an arbitrary part on the left. Thus the newly added edge would coincide with the interior edge. This situation can arise after an inverse left operation. Suppose in Fig. 2 a) that the gate has been the gray arrow before the inverse center operation and then the white triangle has been encoded by an inverse left operation. Thus every time an inverse left operation has been performed an inverse center operation is not allowed to follow.

But that is just the simplest scenario for constraint 1b. Inside the white triangle in Fig. 2 a) can be further vertices forming a more complicated triangulation. Fig. 2 b) illustrates all different triangulations with up to three interior vertices, together with the reversed sequence of operation symbols that will produce this situation. We did not draw all 17 cases but wrote the number of similar case to the left of one representative. For representatives with three cases, the other two cases result from rotations of 120° and 240°. In the representative we drew the gate locations during forward encoding are shown for the first symbol string under the drawing. If we add the two one-symbol constraints E and L, we end up with 19 constraints. Let $\mathcal{C}$ be the set of all constraints

$$\begin{aligned}\mathcal{C} \quad \stackrel{\text{def}}{=} \{ \quad & E, L, ESC, ELSCC, ERSCC, ERCSC, ELLSCCC, \\ & ELRSCCC, ELRCCSC, ERRCSCC, EESSCCC, \\ & ERCRSCC, ERRCCSC, ERRSCCC, ERCRCSC, \\ & EESCSCC, ERCLSCC, ERLCSSS, ELRCSCC \quad \}.\end{aligned}$$

Next we define two predicates on Edge Breaker symbol strings

$$\begin{aligned}\forall S \in \mathcal{A}^* : \quad & \text{isConstraint}(S) \quad \Leftrightarrow S \in \mathcal{C} \\ \forall S \in \mathcal{A}^* : \quad & \text{isPrefix}(S) \quad \Leftrightarrow \exists H \in \mathcal{C} | S = H_{1..|S|},\end{aligned}$$

where $H_{1..l}$ denotes the string composed of the first l symbols of the string H. To ensure constraint 1b we define conditional unities of the form ${}_s^H\mathbf{1}_l^p$, where H is a string that specifies the history of symbols decoded before. For example the conditional unities ${}_4^{ESC}\mathbf{1}_5^3$ remembers the knowledge that the last encoded symbol was a C, the one before a S and the one before an E. Further more it remembers that the current loop length is 5, the length of the loop on top of the loop stack is 3 and the length of the next lower loop on the stack is 4. The knowledge that the previous symbols have been ESC allows us to discard the C symbol from the allowed symbols for this conditional unity.

Only strings H, for which "isPrefix(H)" holds, are useful to be remembered in the conditional unities. But for the loop lengths there is no natural limit such that we have to introduce artificial limits $l_{\max}$, $p_{\max}$ and $s_{\max}$. With these limits we can produce all the conditional unities by following from no precondition, i.e. **1**, all possible encoding paths.

6 Numerical Solution

Let us collect for a given number of constraints and given maximum lengths $l_{\max}$, $p_{\max}$ and $s_{\max}$ all the different conditional unities in the vector $\vec{\mathbf{1}}$, such that the first component $\vec{\mathbf{1}}_1$ equals to **1**. Let n be the dimension of $\vec{\mathbf{1}}$. Then the equations defining the conditional unities are of the form

$$\vec{\mathbf{1}} = M(\tau)\vec{\mathbf{1}} \qquad M(\tau) \in I\!R^{n\times n},$$

with a sparse square matrix M, that depends on τ. This equation is of the form of an eigenvalue problem. The difference is that the eigenvalue is known to be one, but the matrix depends on the parameter τ. We are looking for a τ within the range $[\tau_{\min}, \tau_{\max}] = [3.245, 4]$ bits per vertex. Once we have found the correct τ_1, the matrix $M(\tau_1)$ has an eigenvalue of 1. Furthermore all entries auf $M(\tau_1)$ are less or equal to 1, which implies that 1 is the largest eigenvalue of $M(\tau_1)$. If we choose a τ_+ larger than τ_1, we underestimate the bit rate and all entries of the matrix $M(\tau_+)$ either increase or stay the same, such that also the largest eigenvalue has to increase or stay the same. Similarly, for a τ_- smaller than τ_1 the eigenvalue is less or equal to 1. Thus the largest eigenvalue of $M(\tau)$ is a monotonous function of τ. This enables us to apply a simple interval subdivision to find τ_1 starting with the interval $[\tau_{\min}, \tau_{\max}]$. For each τ the vector of conditional unities and the largest eigenvalue can be found with a power iteration.

Before we use the found conditional probabilities to define the state machine of the arithmetic coder, we clean them up by throwing away conditional unities that correspond to the same state, i.e. the same probability and the same defining equation.

7 Results

In order to analyze the effect of the constraints versus the different cut-border loop lengths, we also restrict the number of exploited constraints to $c_{\max} \in \{0, \ldots, 19\}$. We call the numbers $l_{\max}$, $p_{\max}$, $s_{\max}$ and $c_{\max}$ the *limit parameters*. To find the best values for the limit parameters we implemented an optimization method, that

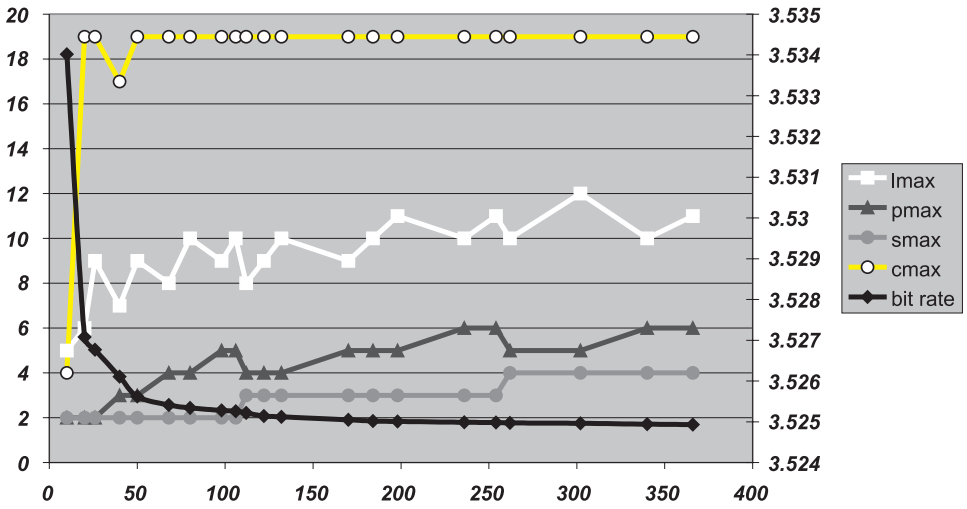


Figure 3: For given maximum number of variables the smallest bit rates plotted together with the set of limit parameters

finds for a given maximum number of states the limit parameters that produced the smallest bit rate. Figure 3 plots the best bit rate and the limit parameters over the number of necessary states. The scale of the best bit rate is shown on the right. The rate drops off in an exponential manner converging to something below 3.525 bits per vertex.

More interesting is that the limit parameters for the loop lengths grow linearly, whereas the number of used constraints nearly instantly hits the maximum of 19. This result suggest that the highest potential for reducing the bit rate further lays in the number of considered constraints. With the limit parameters $l_{\max} = 10$, $p_{\max} = 6$, $s_{\max} = 3$ and $c_{\max} = 19$ one can achieve a bit rate of less than 3.525 bits per vertex with 236 conditional unities. Thus we can refine our coding theorem to:

Theorem 7.1 *The connectivity of a planar triangulation with v vertices and 3 external edges can be encoded with less than 3.525 bits per vertex in linear time in v. The original connectivity can be decoded also in linear time in v.*

8 Conclusion

In this article we showed how to encode a planar triangulation with no more than 3.525 bits. This is currently the best-known result. First we analyzed the con-

straints on the Edge Breaker code strings. Then we introduced a new coding technique – the conditional unities in combination with an arithmetic coder based on a state machine – which allows to exploit the constraints. We developed fast numerical methods to solve the resulting modified eigenvalue problem in order to build these state machine based arithmetic coders for given parameters that limit the extend in which the constraints are exploited. This allows us to trade off between the number of states in the arithmetic coder and the achieved bit rate. To exploit this flexibility in full depth we designed a search function that finds the parameters, which achieve the smallest bit rates, for a given maximum number of states. A plot of these parameters showed that there is so far unexploited potential in the constraint that ensures that no more than two triangles are incident to one edge. In future work we will design an automatic method to fully exploit this constraint.

References

[1] R. C. Chuang, A. Garg, X. He, and M. Kao. Compact encodings of planar graphs via canonical orderings and multiple parentheses. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, pages 118–129, 1998.

[2] C. Gotsman, S. Gumhold, and L. Kobbelt. Simplification and compression of 3d meshes.

[3] S. Gumhold. Improved cut-border machine for triangle mesh compression. In *Erlangen Workshop '99 on Vision, Modeling and Visualization*, Erlangen, Germany, November 1999. IEEE Signal Processing Society.

[4] S. Gumhold. New bounds on the encoding of planar triangulations. Technical Report WSI–2000–1, Wilhelm-Schickard-Institut für Informatik, University of Tübingen, Germany, January 2000.

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

[6] M. Isenburg, S. Gumhold, and C. Gotsman. Connectivity shapes. In *Proceedings of the IEEE Visualization Conference*, page to appear. IEEE Computer Society, October 2001.

[7] M. Isenburg and J. Snoeyink. Spirale reversi: Reverse decoding of the edgebreaker encoding. Technical Report TR-99-08, Department of Computer Science, University of British Columbia, October 4 1999. Mon, 04 Oct 1999 17:52:00 GMT.

[8] A. Itai and M. Rodeh. Representation of graphs. *Acta Informatica*, 17(2):215–219, June 1982.

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

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

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

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