EUROGRAPHICS 2002 / G. Drettakis and H.-P. Seidel
(Guest Editors)

Volume 21 (*2002*), *Number 3*

Angle-Analyzer: A Triangle-Quad Mesh Codec

Haeyoung Lee
USC

Pierre Alliez
INRIA / USC

Mathieu Desbrun
USC

Abstract

We present Angle-Analyzer, *a new single-rate compression algorithm for triangle-quad hybrid meshes. Using a carefully-designed geometry-driven mesh traversal and an efficient encoding of intrinsic mesh properties, Angle-Analyzer produces compression ratios 40% better in connectivity and 20% better in geometry than the leading Touma and Gotsman technique for the same level of geometric distortion. The simplicity and performance of this new technique is demonstrated, and we provide extensive comparative tests to contrast our results with the current state-of-the-art techniques.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Surface mesh compression, connectivity coding, geometry coding.

1. Introduction

While a picture is often said to be worth a thousand words, a 3D model could be said to be worth a thousand pictures. Therefore, efficient 3D mesh compression algorithms have been in high demand in the past few years to reduce the storage room needed for large, detailed 3D models and to consequently decrease transmission time over a network.

Single-rate compression of 3D meshes has been a very active area of research[4, 25, 20, 8, 24, 10, 12, 9, 2, 13, 11] over the last five years. While an encoder can have various interesting properties such as efficiency, resiliency, and a small memory footprint, the most sought-after and challenging feature is still low compression rate. Ever since the introduction of the current most efficient algorithm[25] by Touma and Gotsman for triangle meshes in 1998, improvements[2] and extensions to polygon meshes[13, 11] have been proposed to significantly improve the connectivity encoding. However, there has been no major improvement on the overall compression ratios because the geometry still dominates the global bit-rate, with the connectivity being typically one tenth of the size.

In this paper we propose to focus on triangle-quad hybrid meshes. We note that most polygon meshes have a high occurrence of triangles and quads and very few higher-order polygons (see Figure 1), and as a consequence, tailoring an algorithm to only these two cases does result in a both *simpler* and *more robust* implementation. Dealing with the remaining minority of higher-order polygons will not affect the bit rate significantly.

By introducing a geometry-driven mesh traversal and encoding only the intrinsic properties of a mesh, our Angle-Analyzer generates code sequences of lower entropy for both geometry and connectivity. As a result, order-0 or order-1 adaptive arithmetic encoding[26, 23] of the sequence of symbols generates better compression ratios than Touma and Gotsman's, i.e., 40% lower in connectivity and 20% lower in geometry with the same or better level of geometric distortion measured by the Metro tool[3]. Before detailing the algorithm itself, we first review basic knowledge and previous work on single-rate mesh encoding.

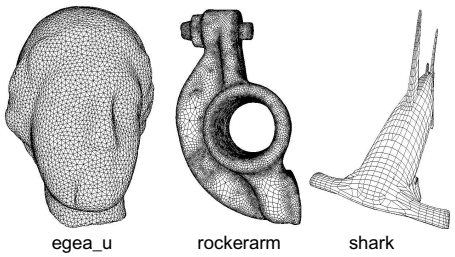

Figure 1: Mesh examples – *a uniform triangle mesh, a quad mesh, and a mesh mainly made of triangles and quads: three meshes that a single-rate compression technique must typically encode.*

1.1. Related Work

Quads and triangles are widely-used primitives for modeling the surface of 3D objects. Such primitives are arranged in the form of polygon meshes that consist of *connectivity* and *geometry*.

Connectivity Previous research on single-rate compression has been mostly dedicated to connectivity encoding. The innovative valence-based approach[25] for triangle meshes proposed to encode the *valence* of every vertex (i.e., its number of emanating edges) in a deterministic traversal going through successive pivot vertices. Similarly, the gate-based approaches, "Edgebreaker"[20, 21] and "Cut-Border Machine"[8, 7] were also initially developed for triangle meshes. They encode a mesh in a spiraling depth-first[20] or breadth-first[8] spanning-tree order traversal and generate one symbol per triangle. The first approach[25] outputs approximately V codes and naturally adapts to the mesh regularity, while the second[20] outputs $2V$ codes (one per face)

and has an easily-provable upper bound of 4 bits [20] for simple triangle meshes, along with a much simpler implementation — an attractive feature for commercial applications.

A first modification of the valence-based approach[25] proposed to change the deterministic mesh traversal by an *adaptive* traversal[2] in order to reduce the number of accident codes and to therefore improve the efficiency. Additionally, the valence approach has been extended recently to arbitrary polygon meshes[13, 11], and has been provided with a theoretical study of optimality[13].

In parallel to these developments, the gate-based technique have also been improved in various aspects[10, 12, 7, 16, 15, 22] that even increased the advantage of simplicity over a valence-based approach. However, the compression rates obtained by this family of techniques remain often worse than the best available[2, 13, 11].

Geometry The global quantization method associated with geometry prediction[25] is the most widely-used geometry compression technique by single rate encoders, and has not been seriously challenged for the past three years. This approach builds a regular 3D grid inside the mesh bounding box and snaps all the mesh vertices onto this grid. During the mesh traversal, every vertex position is predicted from its adjacent vertices using a linear prediction method (the so-called parallelogram rule). The resulting (integer) residuals are then compressed using entropy coding. The main reason of the success of global quantization (also used recently for progressive compression[1]) lies in the simplicity of its implementation (only one pass is required, and a linear predictor outputs an integer value ready for entropy encoding).

Recently, the parallelogram prediction was extended using prediction trees [17]. After assigning prediction error of two adjacent triangles to the weight of the associated edge in a graph of vertices and edges, a minimal spanning tree will be created and each vertex position is predicted from the preceding triangle in the tree.

Other than this prediction approach that depends significantly on the order of the mesh traversal, one finds the innovative "occurrence approach" [5] where a progressive vertex localization is obtained through transmission of vertex occurrences in smaller and smaller bins. The authors demonstrate a gain due to the bit-sharing coming from the transmitted occurrence until complete vertex isolation (i.e., once every bin contains only one vertex). Since this latter technique gets rid of the order over the vertices, it cannot benefit from a prediction coming from a mesh traversal. A "vector quantization approach" [18] was also introduced. It suggested to transform each vertex to the corresponding model space based on previous triangle. The resulting model space vector set and the correction vector set are quantized. Their resulting bit rates were better than Touma-Gotsman's technique for 8 bit quantization only, which is often not visually acceptable.

1.2. Overview

Improvement on Connectivity Encoding A first reading of the theoretical analysis in [13] may make one consider that it is impossible to significantly improve the bit-rates from the valence/degree approach[13, 11]. This seems especially true since any modification of the conquest order will *not* change the distribution of symbols that still has the same valence dispersion, and therefore the same entropy, if one considers order-0 arithmetic encoding only. However, the analysis demonstrates the near-optimality only for worst-case meshes, and there could still be large margin for improvement on any given mesh with a completely different approach.

Our initial approach for connectivity is to *mix* the simplicity of the Edgebreaker/Cut-border approaches and the efficiency of the valence-based approach. As we will see in detail, this can be done by revisiting the definition of the Edgebreaker's five descriptors (plus the additional two for higher genus and holes) and the design of the traversal to minimize the entropy of the resulting sequence. The connectivity encoding is therefore performed by going back to a gate-based approach along with a novel *cooperation* between connectivity and geometry, in order to perform an efficient adaptive mesh traversal driven by both criteria. The basic item required for the mesh traversal is a gate, i.e., an oriented edge (see Figure 2). For each gate processed one symbol is output, in order to indicate how to stitch its front face with the current encoded/decoded part of the mesh. Gates are organized in ordered lists, and a stack of gate lists will be needed to handle special symbols.

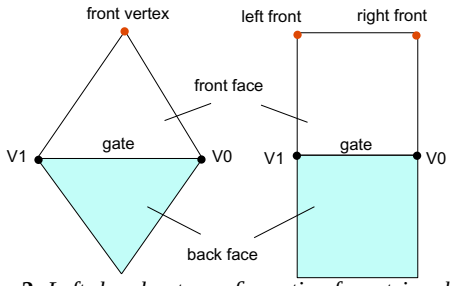

Figure 2: *Left: local gate configuration for a triangle mesh. Right: local gate configuration for a quad mesh. A gate is an oriented edge from $V0$ to $V1$. The back face is the one already visited and the front face is the next visited during the traversal. Both hybrid configurations (triangle to quad, and quad to triangle) can also happen.*

Improvement on Geometry Encoding We have extensively experimented with several geometry encoding methods and measured the resulting rates and distortions [3]. The best results have been obtained by encoding the *intrinsic* properties of a mesh, i.e., the dihedral and internal angles between or inside triangles. These angles have a naturally bounded range and obvious peaks in symbol distribution so a simple, one-pass linear quantization followed by arithmetic encoding is very efficient. We have also experimented

with linear or non-linear prediction, and even entropy-driven quantization. Additional results are detailed in[19].

Pseudo-code The resulting compression algorithm interleaves connectivity and geometry encoding until there is no more gate to process, as follows:

repeat
init
pick the next uncoded connected component
1. pick a seed face (degree 3 or 4)
2. store its gates in ordered list
3. put the list on top of the stack of lists
mesh traversal
repeat
pop the first ordered list off the stack
repeat
1. pick the best gate in the ordered list
2. if the front face is unprocessed,
– process it (triangle or quad)
– store the resulting gates in ordered list
– if new front vertex, encode geometry
3. remove processed gate from the list
until the list is empty
until the stack is empty
Until no more connected component

This very simple pseudo-code, very similar to the associated decompression algorithm, will result in approximately 40% better rates than previous published results for connectivity and 20% better rates for geometry with equal or less distortion. A complete description of our algorithm is detailed in Section 2 for connectivity and Section 3 for geometry coding. Results and conclusions are given in Section 4.

2. Connectivity coding

We first notice that the counterclockwise traversal of edges[25] or faces[13] around successive pivots manages to deduce more local information than an Edgebreaker-type algorithm. Indeed, by turning around each vertex of a gate in clockwise or counterclockwise order, we can identify relationships between the gate and front vertices and generate symbols accordingly, potentially saving unnecessary symbols. Moreover, local geometric information can be used to determine the adaptive traversal to minimize the occurrences of special symbols. We now present our encoding approach, first for triangles, then for quads, and finally, for hybrid quad/triangle meshes.

2.1. Triangle Meshes

Contrary to the seven descriptors used in the original Edgebreaker or the six operations in the improved Cut-border Machine algorithm, we define only *five* relationships and their associated symbols between a front vertex and an active gate for an arbitrary-genus triangle mesh: C (*create*), CW (*mesh clockwise*), CCW (*mesh counterclockwise*), S (*skip*), and J (*join*).

Encoding If the front vertex has not been visited yet, a symbol C will be generated. Two new gates will replace the current gate in the ordered gate list, just like in the original Edgebreaker algorithm (see Figure 3). If the front vertex has been previously visited, we can locate the front vertex by turning either clockwise around V1 or counterclockwise around V0 (Figure 3). A symbol CCW or CW will therefore be generated accordingly. A new gate will replace both the current gate and the next gate in the list. If the active gate is on the mesh boundary, there is no front face and a symbol S (*skip*) will be generated (Figure 3).

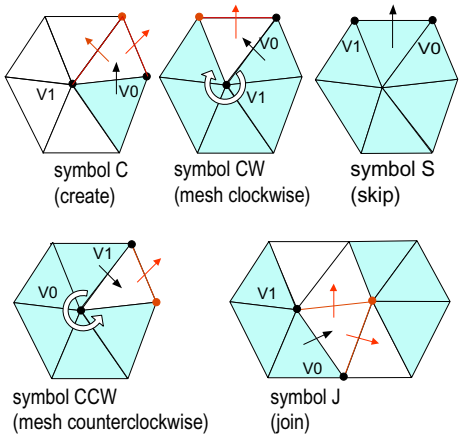

Figure 3: *Set of symbols used for a triangle mesh: the red vertices are front ones. The red gates are new gates to be inserted into the gate list to continue the conquest.*

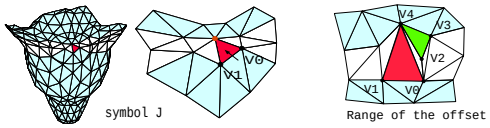

Figure 4: *Symbol J: When the gate list merges, a symbol J occurs. The range of the offset is restricted to # (neighbors - visited neighbors) of V1. If there were other vertex like V2 not connected to V1 having smaller distance to V1, no J (red) but C(green) would be occurred due to minimum angle based choice of the gate (not v0-v1 but v3-v4 would have been chosen).*

When the decoder has no means to identify the location of a previously-visited front vertex, a symbol J followed by an offset will be generated. J occurs when the gate list merges as described in Figure 4. The offset enables the decoder to locate the front vertex within the array of visited vertices sorted by the Euclidean distance to V1 of the gate. The offset will always be between 0 and the number of neighboring vertices decreased by the number of visited neighboring vertices. There should be at least four visited neighboring vertices for V1 for a J to occur, and we use 12 symbols in total for triangles and quads. (if there were other vertices not connected to V1 having a smaller Euclidean distance to V1, J would not occur, but a C instead, due to the minimum angle based choice of the gate as shown in Figure 4). As a result, the offset will only take up *one symbol* for any vertex V1 of valence up to 16. If the offset is greater than 12 we use

the last symbol of the table (i.e., 11) as a reserved unfolding symbol.

In this J mode, splitting or merging of gate lists should be performed. This plays a similar role to the Split and Merge codes of the valence-based approach [25, 2], or the connect and union of the gate-based approach [7]. If the front vertex belongs to the current gate list, the current gate list should be split into two gate lists around the front vertex. If the front vertex belongs to another gate list in the stack, the current gate list and this particular gate list should be merged to a single list[19]. An example of a symbol sequence generated during a mesh traversal is depicted in Figure5.

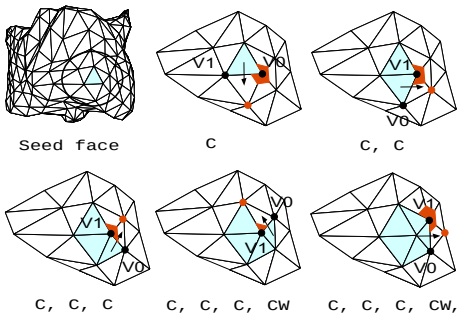

Figure 5: *Example of symbol sequence: in red are the new front vertices. The red wedge shows the minimum angle between two consecutive gates.*

Decoding Initially, three vertices will be decoded and used as the seed face. Three initial gates are then added to the gate list and the reconstruction starts by always applying the same rule over and over again: the next best gate is chosen according to a simple test detailed in Section 2.4 and the decoding traversal proceeds to the next symbol. For the symbol C, a new front face will be created, along with a new front vertex. Two new gates will be added to the gate list. For the symbol CW and CCW, we localize the front vertex by a clockwise or counterclockwise rotation around the appropriate gate vertex until we find the next gate vertex. The new front face is then created and a new gate will be added to the gate list as well. For the symbol S, nothing has to be done. For the symbol J, the front vertex will be located by using the decoded offset value as an index to the sorted array of visited vertices.

2.2. Quad Meshes

Due to the presence of two front vertices facing a gate in a quadrilateral, eight relationships and their associated symbols are defined to encode the connectivity of quad meshes: C2 (*create 2 vertices*), CL (*create left*), CR (*create right*), M (*just mesh*), DCW (*double-clockwise turn*), DCCW (*double-counterclockwise turn*), JQ (*join for quad*), and S (*the same skip as in the previous section*).

Encoding If both front vertices have not been visited, a symbol C2 will be generated. Three new gates will replace the current gate in the ordered gate list (Figure 6). If the right front vertex can be located by turning a counterclockwise around V0 and the left front vertex has not been visited, a

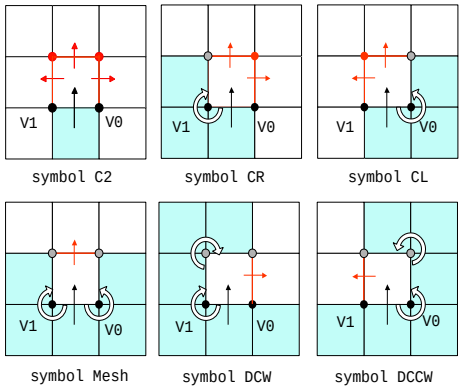

Figure 6: *Set of symbols for a quad mesh: the red vertices are new front vertices. The grey vertices are already-visited front vertices. The red gates are new gates to be inserted into the ordered gate list.*

symbol CL will be generated. If this is true for the opposite direction, a symbol CR will be encoded with two new gates replacing the two consecutive gates found as described by Figure 6. There are three cases where we can find both front vertices by turning around the gate vertices: by a counterclockwise turn around V0 and a clockwise turn around the V1 (symbol M); by double counterclockwise turn around V0 (symbol DCCW); by double clockwise turn around V1 (symbol DCW). One new gate replaces three consecutive gates in the ordered gate list (Figure 6). The symbol S is for the gate without the front face, the same one as for triangle meshes. If one or both of the front vertices have already been visited, the symbol JQ will be generated as described by Figure 7. It will be used in different cases: when both front vertices have been visited, when one of the front vertices has not been visited, and when one of them can be found by turning around a pivot. A single symbol is sufficient to handle all these cases since the decoder will be able to deduce the exact case by local exploration. For each case, the symbol JQ and two offsets will be generated. The presence of a new front vertex to create will be determined by a negative unitary offset. The offset for the left or right front vertex is the index of the array of vertices on the boundary sorted by the distance to V0 or V1 accordingly and also restricted to neighboring vertices of V0 or V1 as for the offset of J in triangle meshes. Splitting/Merging of the gate lists should be processed in the same way as for triangle meshes: if the front vertices belong to the current gate list, we must split the current gate list; if not, merging should occur.

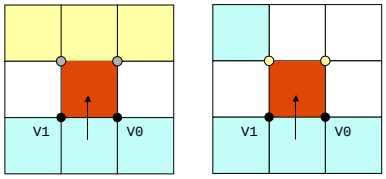

Figure 7: *Detailed cases of the extra symbol JQ: splitting or merging of the gate lists occurs. Some of the yellow quads or yellow vertices have not been visited yet.*

Decoding The initial four vertices will be decoded and form the seed quad face. Four initial gates are pushed to the gate list. The best next gate is then selected from this list of gates. Based on the decoded symbol, a new front face will be created with newly/previously visited front vertices and new gates will be added to the gate list like the decoding process for triangle meshes. We will discuss the gate selection process as for the triangle meshes in Section 2.4.

2.3. Hybrids of Triangle/Quad meshes

Based on the front face type, one of the two schemes previously mentioned can be applied. Overall, 12 symbols are generated for hybrid meshes (S, 4 symbols for a triangle front face, and 7 symbols for a quad front face). With these 12 symbols, there is no ambiguity for the decoder at any time.

2.4. Geometry-Driven Adaptive Traversal

The gate located at the most inner part of the concave shaped gate list should be selected as the next best gate to avoid J or JQ, which causes splitting or merging. We tried two methods to choose the next best gate:

- choose the best gate based on the number of polygons already visited around V0 of the gate,
- choose the best gate having the minimum angle with the following gate along the active border.

The first method is similar to the pivot selection inspired by the valence-driven edge conquest described in [2]. We noticed that there still could be a better choice for the next best gate. By choosing the gate having the *minimum angle* between two consecutive gates in the current gate list, the gate at the inner part can always be selected to better localize concave parts of the conquered region. This geometry-driven traversal removes or dramatically decreases the occurrences of J as shown in Table 1 and moreover generates low-entropy code sequence as shown in Table 2 and Figure 8. An order-1 adaptive arithmetic coder [26] is used to benefit from this distribution of symbols.

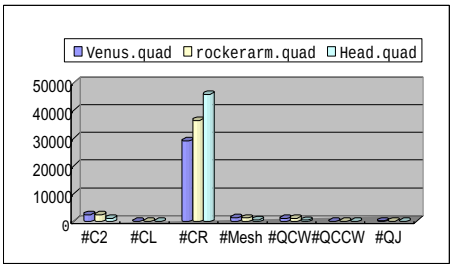

Figure 8: *Example of symbol distribution for quad meshes. Notice the large majority of CR and the low occurrence of other symbols.*

2.5. Discussion

The compression ratios of our connectivity are listed in Table 3 for the triangle meshes shown in Figure 9 and in Table 4 for the quad meshes shown in Figure 10.

model	#V	AD01 (b/v)	J in AD01	AA (b/v)	J in AA
feline	49864	2.37	1951	1.50	106
max	2545	2.36	808	1.45	13
horse	19851	2.38	694	1.35	20
tf2	14169	1.70	106	1.00	18
dino	14070	2.47	571	1.69	59
nefer_u	10413	1.52	1	0.65	0
foot	10016	2.38	301	1.56	14
venus	8268	2.96	727	1.95	149
egea_u	5315	1.71	4	0.82	0
body	711	3.14	11	2.12	4
nefer	299	3.37	7	2.27	0

Table 1: *Comparisons between two adaptive traversals on triangle meshes for connectivity compression: the third and the fifth columns show the compression ratios in b/v of the first method (Alliez-Desbrun 2001) and the second method accordingly. Notice the dramatically decreased occurrences of J symbols when using the minimum-angle gate. nefer_u and egea_u are uniform meshes.*

model	#V	C	CW	CCW	J	S
feline	49864	49861	49653	111	106	0
max	25445	25442	25316	29	13	87
horse	19851	19848	19817	12	20	0
tf2	14169	14166	13898	21	18	232
dino	14070	14067	13962	47	59	0
nefer_u	10413	10410	10011	0	0	402
foot	10016	10013	9982	18	14	0
venus	8268	8265	7950	167	149	0
egea_u	5315	5312	5060	0	0	255
body	711	708	677	6	4	24

Table 2: *Symbol distribution for triangle meshes: the second column shows the number of vertices. The others show the number of the occurrences for each symbol. Notice very high occurrences of C/CW. There is no CCW and no J for uniform meshes, nefer_u and egea_u, explaining the extremely low bit-rate.*

Our Angle-Analyzer is especially good for meshes with irregular or uniform triangulation. For uniform meshes, the number of symbols generated are further reduced to 3: C, CW, and S as shown in Table 2. On average, for triangular meshes, our Angle-Analyzer uses 40% less bits than the Touman-Gotsman's valence-driven approach in deterministic traversal, and 35% less than Alliez-Desbrun's valence-driven approach in adaptive traversal — the two current best compression ratios on connectivity. But for very regular meshes, the valence-driven approach will generate at most 3 symbol types (i.e., 6, Split and Merge) so the bit rate will be usually lower than ours.

Even though the bounds of the bit rates of our connectivity should be investigated further for general cases, a

model	TG	AD	AA	vs TG	vs AD
feline	2.38	2.27	1.50	37%	34%
max	2.31	2.22	1.45	37%	35%
horse	2.34	2.24	1.35	42%	40%
dino	2.39	2.27	1.69	29%	26%
nefer_u	1.59	1.44	0.65	59%	55%
foot	2.33	2.2	1.56	33%	30%
venus	2.82	2.74	1.95	31%	29%
egea_u	1.73	1.63	0.82	53%	50%
body	2.62	2.38	1.96	25%	18%

Table 3: *Bit-rate (in b/v) comparisons for triangle meshes: the second column shows bit rates from Touma-Gotsman's and the third shows bit rates from Alliez-Desbrun's. The fourth column shows bit rates of our Angle-Analyzer (AA). The fifth shows ratio of AA over TG while the sixth shows ratio of AA over AD. AA produced more than 50% lower bit rates than TG and AD for uniform meshes, nefer_u and egea_u.*

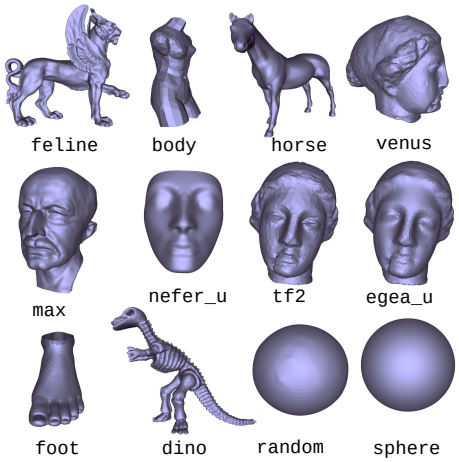

Figure 9: *Triangle meshes used: feline, body, horse, venus, max, nefer_u, tf2, egea_u, foot, dino, random, and sphere. random is an irregular non-uniformly meshed sphere and sphere is a regular mesh. Feline is a genus 2 mesh. body, max, tf2, venus, foot, nefer_u, and egea_u have a boundary.*

quick analysis for a 2-manifold triangle mesh without a boundary can be sketched as follows. For such a mesh, there will be only 4 codes generated: C, CW, CCW, and J, which will take 2 bits to encode. The number of occurrences of C is $(|V|-3)$, where $|V|$ is the number of vertices of the mesh. The sum of occurrences of CW, CCW, and J will be up to $|V|+|J|$ where $|J|$ is the number of symbols for offsets in J. Since Table2 exhibits a negligible number of symbol J in practice, the sum of occurrences of CW, CCW, and J can be assumed to be $|V|$. Therefore, the expected (yet, unguaranteed) maximum bit rate for arbitrary triangle meshes with handles and no holes will be $2*2*|V|/|V|=4$ b/v. In practice, the usual bit rate is around 1.5 b/v for triangle meshes, and 0.8 b/v for quad meshes.

model	#V	AD01	AA	rate
feline.quad	205210	1.29	0.65	50%
head.quad	48099	0.73	0.36	51%
rockerarm.quad	41312	1.27	0.74	42%
venus.quad	34104	1.56	0.89	43%
david.quad	24599	1.70	1.08	36%
genus3.quad	6796	0.75	0.44	41%
body.quad	2891	1.28	0.64	50%
uglybox.quad	1432	2.12	1.36	36%
tiger.quad	1254	1.24	0.94	24%
nefertiti.quad	1191	1.5	0.81	46%

Table 4: *Bit rates (in b/v) of quad meshes: The third column shows the bit rates of [2]'s adaptive traversal and the fourth for the bit rates of our angle-based traversal. The fifth shows the improvement ratio of our Angle-Analyzer over [2].*

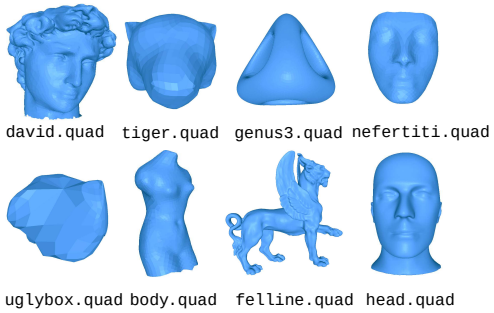

Figure 10: *Quad meshes used: david.quad, tiger.quad, genus3.quad, nefertiti.quad, uglybox.quad. body.quad, feline.quad(genus2), head.quad. rockerarm.quad(genus1) is shown in Figure 1. venus.quad is the same geometry as the triangulated venus, remeshed using quads. david, nefertiti, body, and head have a boundary.*

3. Geometry encoding

Geometry represents the dominant part of the overall bit rate of a compression algorithm. We therefore attempted to find a novel method to better encode the geometry of arbitrary meshes. In this section, we propose two different techniques, resulting in similar compression rates. We also introduce a novel entropy-driven geometry compression algorithm that can further reduce the bit-rate and offers an increased flexibility. Since a quad can be considered as two triangles glued together as shown in the right of Figure11, the following detailed description will be restricted to a triangle mesh: for a quad mesh, a similar method will simply be applied twice in a row to encode the position of each pair of front vertices.

3.1. Local Coordinate Based Geometry Encoding

The first approach we tried is a simple extension of the most widely used geometry quantization.

Global vs. Local Quantization This global quantization method originally used in[25] divides the bounding box of the mesh into a 3D regular grid for linear quantization. However, this simple approximation fails to accurately reproduce flat surfaces that are not parallel to one of the grid axes, adding

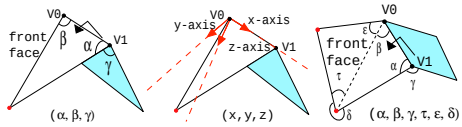

Figure 11: *Left: encoding three angles is sufficient to locate the front vertex across a gate. Middle: the front vertex coordinates can also be expressed in a local coordinate frame based on the gate and the back face. Right: encoding six angles for a quad.*

an important amount of distortion even for such a simple geometry. On the other hand, previous work on progressive geometry compression has demonstrated the interest of local coordinate frames to better encode positions[14, 1], due to the natural separation between geometry and parameterization. We have therefore modified the single-rate geometry encoding technique to include this simple idea.

A local coordinate system can be defined at each front face as shown in Figure 11. The active gate will define the local *x*-axis. V0 is assumed to be the origin. For the local *y*-axis, the unit vector on the local *x*-axis is rotated by $-\pi/2$ around the normal vector of the back face. The local *z*-axis is then obtained by computing the cross product of the local *x* and *y* axes or by reversing the normal of the back face. Since the range of each local coordinate value is not predefined, one initial pass is necessary to find the range of displacements between V0 and the front vertex expressed in these local frames. Storing the range values in the header file will allow the decoder to perform the exact same computation and therefore to decode the mesh properly.

Encoding Local Coordinates The vector (V0 - front vertex) is projected onto the unit vector on each local coordinate. These local coordinates are then quantized linearly and encoded. A decoding simulation is then needed to infer the decoded position of the front vertex. The encoder will then use this updated front vertex position in the following computations to synchronize with the decoder. Due to this decoding simulation, there is no need to encode correction vectors separately in our geometry encoding, which was the main factor for higher bit rates in the vector quantization method [18].

Decoding Local Coordinates The decoding is simply the reverse process. After finding the unit vectors on the local coordinate axis, the decoded local coordinates should be transformed to the global coordinate values to finally set the new position for the front vertex.

3.2. Angle-based Geometry Encoding

We then experimented with an alternative geometry compression algorithm that resulted in roughly similar bit-rates with, however, an easier implementation. The initial idea was to substitute angles for positions, therefore introducing a non-linear quantization in the encoding. To define a triangle in 3D space adjacent to a given back face, three angle values around that common gate are sufficient: two "intrinsic" angles (α,β) and an "extrinsic" angle (γ) between the neighboring faces as shown in Figure 11. The ranges of angles are naturally bounded regardless of the size of the triangles: $[0,\pi]$ for α and β, and $[-\pi,\pi]$ for γ. In addition, the level of quantization for each angle can be properly adapted: for instance, since γ, related to the mean curvature, has a wider range than α and β, a finer quantization of the γ range is recommended for a highly curved mesh. The distribution of α and β is expected to be mainly concentrated around $\pi/3$ for fairly uniform meshes and around 0 for γ for smooth surfaces. Due to the usual distribution of angles shown in Figure 12, an arithmetic coder will be particularly efficient at compressing this series of angles.

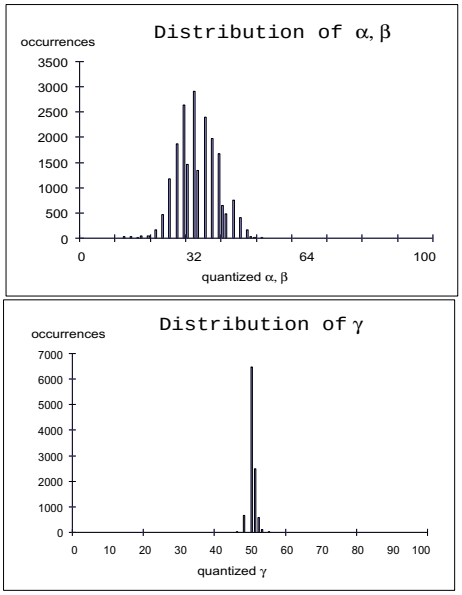

Figure 12: *Angle distribution for the uniform nefer_u mesh. Codes for α and β are concentrated around 32, which is 1.01 radians (57.6 degree) in the interval [0,π] quantized between 0 and 100. The codes for γ are concentrated around 50, corresponding to 0 in the interval [-π,π] quantized also between 0 and 100.*

Encoding Angles During the traversal of the gates, whenever a front vertex is visited for the first time, the appropriate angles needed to infer the position of the front vertex should be quantized and encoded as follows:

- Find the angles, α, β, and γ,
- Quantize them linearly to integer values, qα, qβ, and qγ with given numbers of quantization for each angle,
- Simulate the decoding process to infer the decoded position of the front vertex. Use this updated front vertex position in the following computations to synchronize with the decoder.

Decoding Angles The geometry decoding to recover the front vertex position proceeds as follows:

- Convert the integer values qα, qβ, and qγ to the actual real values rα, rβ, and rγ

- Calculate the normal vector Qnormal of the back face using the previously decoded vertices,
- Rotate the vector Left(V0-V1) around Qnormal by rα and normalize it,
- Calculate the length of the edge(V0-front vertex) using the sine law: edgeLength = $sin(r\alpha)/sin(\pi - r\alpha - r\beta)$ * length of Left,
- Find the decoded front vertex position by adding the vector edgeLength*normalizedLeft to V0.

As we will demonstrate later, both the local coordinate approach and the angle approach give similar bit-rates. The choice of one over the other may be very arbitrary. In our experience, the angle approach was actually slightly easier to implement, and requires only one pass.

3.3. Entropy-driven Encoding

We take advantage of the lossy nature of geometry compression to further decrease the bit rate. When a front vertex position is encoded, one could easily adjust the position of the encoded vertex without adding much distortion. This (small) degree of freedom can therefore be used to drive the encoding in order to locally minimize the number of bits used to encode this position.

Error Margin in Position For each vertex, we can determine a "safe" region within which this vertex can be arbitrarily placed without significantly impeding the visual appearance or the distortion. Quadrics like in [6] can be used to define these regions for every vertex depending on the local geometry; for instance, a locally planar mesh could allow the vertex to be moved in the plane a bit without introducing substantial geometric error. In our trials, we only experimented with identical cubic regions around each vertex to allow for a fair comparison with previous encoding techniques. More complex regions may be highly desirable in practice.

Entropy-driven Selection From all the possible vertices in the safe region, we now want to pick one that minimizes the number of bits needed to encode it, saving few extra bits without degrading the geometry dramatically. This can be done relatively easily by trying all possible quantized values (be they angles or local coordinates) that lie in the safe region. For each of them, we compute the number of bits it would take to encode it. We then pick the best one.

Entropy Evaluation If we denote by E_0 the entropy before the candidate integer code is added to the code sequence, and by E_1 the entropy after the candidate is added, we can directly compute the number of bits used to encode this particular vertex since:

- $E_0 = \sum n_i/N \log_2(N/n_i)$,
- $E_1 = \sum n'_i/(N+1) * \log_2((N+1)/n'_i)$

where N is the total number of codes encoded until now, n_i is the number of i code used until now, and n'_i the updated number after the new vertex is encoded. Obviously, only one of the n_i (refer it as i_0) will have its value incremented by one, meaning that $n'_{i_0} = n_{i_0} + 1$ and for all other i, $n'_i = n_i$. Now, the additional number of bits needed to encode this particular vertex position is equal to:

$$(N+1)\,E_1 - N\,E_0 = \sum_{i \neq i_0} n_i\,(\log_2(N+1)/N) + n_{i_0}\log_2(n_{i_0}/N) + (n_{i_0}+1)\log_2((N+1)/(n_{i_0}+1))$$

Since the first term is the constant, the last two terms are to be compared. The candidate with minimum entropy difference is chosen to ensure the locally lowest bit-rate. Notice that this entropy-driven encoding does not guarantee a lower global bit-rate, but only a local lowest bit-rate. However, we have only experienced a decrease in bit-rate in practice as shown in Figure 14, 15, and 16. Again, a better definition for the safe region could potentially save additional bits. However, to make the comparison with previous single-rate methods fair, we stuck to a fixed size.

3.4. Discussion

In the two geometry encoding methods we presented, we experimented with further refinements, including prediction rules (using average local γ, or predicting α and β from neighborhood values) and non-linear quantizations. We executed extensive comparisons on eight different methods for geometry encoding and compared them with 12 and 11 bit global quantization methods [25] using the Metro tool[3]. Due to doubled ranges, more dispersed distributions, or penalty cases of predictions, the net gain was null. More detailed explanations for the extensive study can be found in[19]. As a result, and surprisingly, the best rates came from simple encoding techniques: no-entropy driven, linear quantization without any prediction rule for both methods. We also experimented with single or multiple arithmetic coders for encoding generated sequences, also detailed in [19]. Separate coders for each angle value or coordinate value produced the best compression ratios. On average, the bit rates of our two techniques are 20% better than the global quantization as shown in Table 5. Even for the very randomized, small model tf2 with 0.49X0.69X0.99 as the bounding box and 0.0001196 ($0.49/2^{12}$) as the unit grid of x in global quantization method, our method can decrease the bit rate 5% smaller.

model	#V	TG12 (b/v)	Local (b/v)	Angle (b/v)
sphere	10242	6.96	5.15	4.91
random	4338	15.64	11.66	11.20
nefertiti_u	10413	13.43	9.17	9.47
max	25445	16.43	11.93	12.69
horse	19851	15.17	12.68	12.88
feline	49864	14.17	12.84	13.17
egea_u	5315	16.50	14.90	15.45
egea_tf2	14169	14.90	14.62	14.18

Table 5: Geometry compression rates for triangle meshes with 12-bit global quantization, and our results for the same $\mathcal{L}^2$ geometric distortion.

Extensive Comparisons for each model by Metro The results of extensive comparisons on two meshes using Metro are depicted in Figure 13. Our methods offer more aggressive and adaptive quantizations than the global quantization. Additionally, the user have much more freedom in the bit budget, while the global quantization offers only the choice between 9- to 12-bit quantization. As a result, a better visual appearance of decompressed meshes for very lossy cases can be obtained even with smaller bit rates as shown in Figure 14, 15, and 16. Overall, we recommend the use of the angle-based geometry encoding for the following reasons:

- negligible difference of bit rates compared to the local frame encoding scheme,
- better bit rates on very randomized meshes,
- better visual look of decompressed meshes for very lossy cases (i.e.,9 or 10 bit global quantization),
- as simple to implement as the previous methods,
- more efficient processing as a one-pass algorithm.

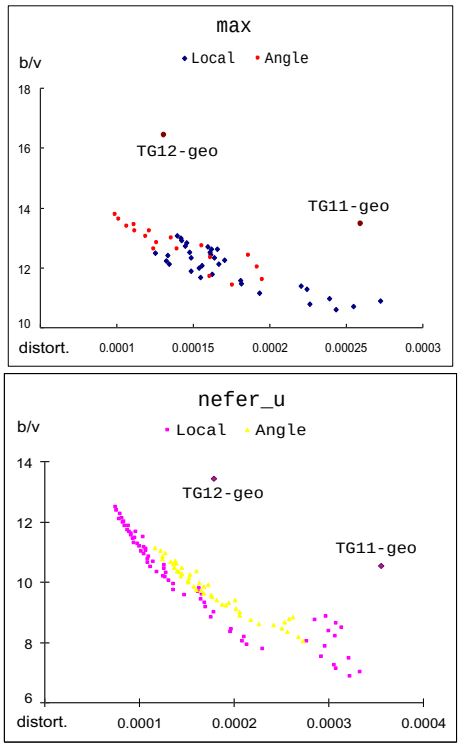

Figure 13: *Rate-distortion curves. The $\mathcal{L}^2$ distortion was computed using the Metro tool[3]. The horizontal axis represents distortion, while the vertical axis represents bit rates (b/v). We varied the degrees of quantization of the angles and/or the local coordinates to obtain the different bit-rates on these curves.*

4. Results and Conclusions

The compression ratios of our Angle-Analyzer on a series of surface meshes are listed in Table 6. Our geometry-driven connectivity encoding behaves on average 40% better than Touma and Gotsman's [25]. If the models are uniformly remeshed, the bit rates of our connectivity encoding will be reduced by more than 50%. The geometry encoding uses on average 19% less bits to encode surfaces with comparable distortion to the Touma-Gotsman's algorithm[25] with 12-bit quantization. The sphere model is very regular, therefore our bit rate for the connectivity is higher than Touma-Gotsman but due to the dominance of the geometry, the overall compression ratios of our AA is 26% better. The total bit rates are, on average, 20% better. Our Angle-Analyzer technique is therefore both more *flexible* and more *efficient* than previous single-rate encoding methods.

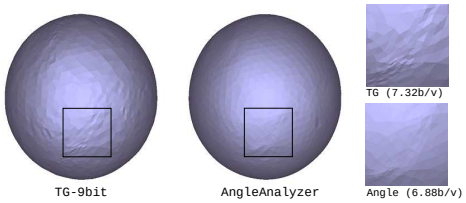

Figure 14: *Geometry compression: Decoded random sphere, (a) using the 9-bit global quantization of [25] with a resulting bit rate of 7.32 b/v and (b) using our entropy-driven local coordinate encoding with a resulting bit rate of 6.88 b/v.*

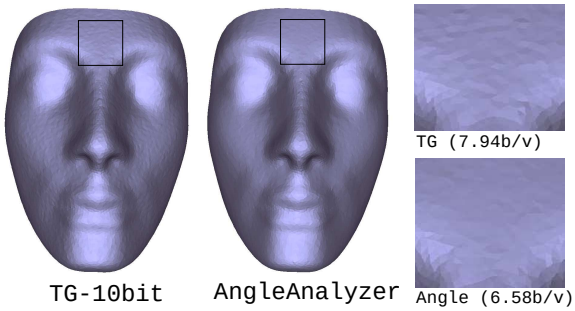

Figure 15: *Geometry compression: Decoded nefer_u, (a) using the 10-bit global quantization of [25] resulting in 7.942 b/v, and (b) using our entropy-driven angle encoding resulting in a bit rate of 6.58 b/v.*

In summary, we introduced a novel single-rate compression algorithm for 3D triangle-quad hybrid meshes that performs a simple geometry-driven mesh traversal, along with an encoding of the extrinsic and intrinsic angles of the mesh. We show that an angle-based adaptive selection of gates dur-

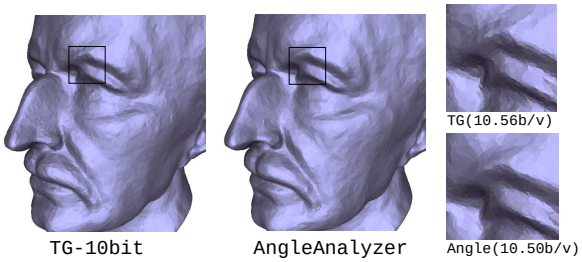

Figure 16: *Geometry compression: Decoded Max, (a) using the 10-bit global quantization of [25] resulting in 10.558 b/v, and (b) using our entropy-driven angle encoding resulting in 10.498 b/v.*

model	TG12 (b/v)	$\mathcal{L}^2$d-error in TG12	AA (b/v)	$\mathcal{L}^2$d-error in AA	rate
sphere	6.98	0.000104	5.14	0.000089	26%
random	16.07	0.000104	11.77	0.000102	27%
nefer_u	15.02	0.000178	10.21	0.000178	32%
max	18.74	0.000131	14.17	0.000127	24%
horse	17.51	0.000221	14.25	0.000215	19%
feline	16.55	0.000055	14.68	0.000052	11%
egea_u	18.23	0.000040	16.26	0.000039	11%
tf2	16.53	0.000040	15.19	0.000039	8%

Table 6: *Results of the overall compression ratios of our Angle-Analyzer (AA) using angle-based geometry with Touma and Gotsman's 12bit quantization. The distortion is measured in $\mathcal{L}^2$ norm using the Metro tool*[3].

ing the connectivity traversal also helps in suppressing the occurrences of special symbols and generates low-entropy code sequences. Angles between edges and faces allow the design of an efficient one-pass geometry encoding scheme. As a result, our Angle-Analyzer technique demonstrates competitive compression ratios compared with other single-rate encoding schemes, with a significantly-increased versatility in the choice of the compression rates. These two contributions, for connectivity and geometry, can also be independently used in existing compression algorithms as a better and simpler substitute.

As for future work, we plan to design better prediction rules, as well as to adapt the current technique to naturally handle non-manifold meshes. In order to improve the compression ratios even further, we also start to think about a hierarchical approach (yet still for single-rate encoding) in the spirit of [5].

Acknowledgements Many thanks to Yiying Tong for his help. Also many thanks to Weonjoon Choi for his inspirational support. The work reported here was supported in part by IMSC NSF Engineering Research Center (EEC-9529152), and by a NSF CAREER award (CCR-0133983).

References

1. ALLIEZ, P., AND DESBRUN, M. Progressive Encoding for Lossless Transmission of 3D Meshes. *ACM Siggraph Conference Proceedings* (2001), pp.198–205.
2. ALLIEZ, P., AND DESBRUN, M. Valence-Driven Connectivity Encoding of 3D Meshes. *Eurographics Conference Proceedings* (2001), pp.480–489.
3. CIGNONI, P., ROCCHINI, C., AND SCOPIGNO, R. Metro: Measuring Error on Simplified Surfaces. *Computer Graphics Forum 17(2)* (1998), pp.167–174.
4. DEERING, M. Geometry Compression. *ACM Siggraph Conference Proceedings* (1995), pp.13–20.
5. DEVILLERS, O., AND GANDOIN, P.-M. Geometric Compression for Interactive Transmission. *Visualization 2000 Conference Proceedings* (2000), pp.319–326.
6. GARLAND, M., AND HECKBERT, P. Surface Simplification Using Quadric Error Metrics. *ACM Siggraph Conference Proceedings* (1997), pp.209–216.
7. GUMHOLD, S. Improved cut-border machine for triangle mesh compression. *Erlangen Workshop'99 on Vision, Modeling and Visualization* (1999).
8. GUMHOLD, S., AND STRASSER, W. Real Time Compression of Triangle Mesh Connectivity. *ACM Siggraph Conference Proceedings* (1998), pp.133–140.
9. HOPPE, H. Progressive Meshes. *ACM Siggraph Conference Proceedings* (1996), pp.99–108.
10. ISENBURG, M. Triangle Strip Compression. *Proceedings of Graphics Interface 2000* (2000), pp.197–204.
11. ISENBURG, M. Compressing polygon mesh connectivity with degree duality prediction. *Graphics Interface 2002* (2002), pp.161–170.
12. ISENBURG, M., AND SNOEYINK, J. Face fixer: Compressing polygon meshes with properties. In *ACM SIGGRAPH 2000 Conference Proceedings* (2000), pp.263–270.
13. KHODAKOVSKY, A., ALLIEZ, P., DESBRUN, M., AND SCHRÖDER, P. Near-Optimal Connectivity Encoding of 2-Manifold Polygon Meshes. *The Journal of Graphical Models/special issue* (2002).
14. KHODAKOVSKY, A., SCHRÖDER, P., AND SWELDENS, W. Progressive Geometry Compression. *ACM Siggraph Conference Proceedings* (2000), pp.271–278.
15. KING, D., AND ROSSIGNAC, J. Guaranteed 3.67v bit encoding of planar triangle graphs. *11th Canadian Conference on Computational Geometry* (1999), pp.146–149.
16. KING, D., ROSSIGNAC, J., AND SZMCZAK, A. Connectivity compression for irregular quadrilateral meshes. Tech. Rep. TR–99–36, GVU, Georgia Tech, 1999.
17. KRONROD, B., AND GOTSMAN, C. Optimized triangle mesh compression using prediction trees. *Proceedings of 8th Pacific Graphics 2000 Conference* (2000).
18. LEE, E.-S., AND KO, H.-S. Vertex data compression for triangle meshes. *Proceedings of the 8th Pacific Graphics Conference on Computer Graphics and Application* (2000), pp.225–234.
19. LEE, H. Thesis proposal on 3d mesh single-rate compression. *http://www-scf.usc.edu/ leeh/qual/qual.pdf* (2002).
20. ROSSIGNAC, J. EdgeBreaker : Connectivity Compression for Triangle Meshes. *IEEE Transactions on Visualization and Computer Graphics* (1999).
21. ROSSIGNAC, J., SAFONOVA, A., AND SZYMCZAK, A. 3d Compression Made Simple: Edgebreaker on a Corner-Table. *Shape Modeling International Conference* (2001).
22. ROSSIGNAC, J., AND SZYMCZAK, A. Wrap& zip decompression of the connectivity of triangle meshes compressed with EdgeBreaker. *Journal of Computational Geometry, Theory and Applications 14* (1999).
23. SCHINDLER, M. A Fast Renormalization for Arithmetic Coding. *Proceedings of IEEE Data Compression Conference, Snowbird, UT* (1998), p. 572.
24. TAUBIN, G., AND ROSSIGNAC, J. 3D Geometry Compression, 1999-2000. ACM Siggraph conference course notes.
25. TOUMA, C., AND GOTSMAN, C. Triangle Mesh Compression. *Graphics Interface 98 Conference Proceedings* (1998), pp.26–34.
26. WITTEN, I., NEAL, R., AND CLEARY, J. Arithmetic Coding for Data Compression. *Communications of the ACM 30(6)* (june 1987), pp.520–540.