UNC Technical Report TR-01-03

Compressing the Property Mapping of Polygon Meshes

Martin Isenburg Jack Snoeyink
University of North Carolina at Chapel Hill
{isenburg | snoeyink}@cs.unc.edu

Abstract

Many polygon meshes have properties such as shading normals, colours, texture coordinates, and/or material attributes that are associated with the vertices, faces or corners of the mesh. While current research in mesh compression has focused on connectivity and geometry coding, the compression of properties has received less attention. There are two kinds of information to compress. One specifies each individual property—the property values. The other describes how the properties are attached to the mesh—the property mapping. In this paper, we introduce a predictive compression scheme for the property mapping that is 2 to 10 times more compact than previously reported methods.

1. Introduction

Polygonal meshes are commonly used to represent surfaces in 3D and serve as the de facto standard for fast interactive visualization. It can require a large number of polygons to accurately represent a detailed model. The fact that transmission bandwidth and storage capacity are a limited resource in many graphics applications has motivated researchers to find compact representations for such data.

Over the past few years an immense number of mesh compression schemes have been proposed [3, 19, 20, 18, 15, 5, 7, 2, 16, 12, 6, 13, 10, 8, 17, 14, 11, 1]. The majority of these contributions focus on only two aspects of mesh compression: coding the mesh connectivity, that is the incidence relation among the vertices, and coding the mesh geometry, that is the specific location of each individual vertex. While these two are undoubtedly the basic ingredients of any polygon mesh, a large number of polygonal datasets also contain mesh properties.

Typically, mesh properties are used to describe the visual appearance of the mesh for the moment it is rendered. Sometimes this information explicitly specifies the colour with which the polygons will be displayed. But often, especially for dynamic environments, it specifies how to compute the colour based on the lights in the scene. In this case shading normals and material attributes are attached to the polygon mesh and used to calculate the colour at run-time based on a lighting calculation. In addition, a polygon mesh can have texture coordinates that specify the mesh appearance through references into an image or a light map.

The compression of mesh properties involves two kinds of information: the *property values* and the *property mapping*. The property values specify each individual property, such as the r, g, and b components for a colour or the u and v components for a texture coordinate. The property mapping specifies how these properties are attached to the mesh. Deering [3] initiated research on compressing the property values and was followed by others [18, 2]. These works, however, pay less attention to the problem of compressing the property mapping. The few previously proposed techniques [18, 5, 8] are fairly basic and use between 1.5 to 6 bits per vertex (bpv). This is surprisingly many, especially given the abundance of papers on mesh connectivity coding [19, 20, 15, 5, 7, 2, 16, 12, 13, 8, 17, 14, 1] that scramble for improvements of around 0.2 bpv to 0.4 bpv on compression rates that are already as low as 2 to 3 bpv.

In this paper we show how a set of simple predictions can be used to efficiently compress the property mapping of polygon meshes. Our predictive compression scheme results in bit-rates between 0.1 and 2 bpv, which improves by a factor of 2 to 10 over previous methods [18, 5, 8].

In the next section, we give some basic definitions and review the problem of compressing polygon meshes. In Section 3 we characterize the property mapping of polygon meshes. In Section 4 we discuss previously proposed methods for its compression. Then, in Section 5, we describe our predictive approach and compare our results to those of other methods. The efficiency of our coder for the case of stripified triangle meshes is the topic of Section 6, before we discuss our contribution in the last section.

2. Preliminaries

A *polygon mesh* is a collection of polygonal *faces* that intersect only along shared *edges* and *vertices*. Any edge is shared by at most two faces; unshared edges are *bound-*

ary edges. Around each face we find a cycle of vertices and edges; around each vertex we find a cycle of edges and faces. Each appearance of a face in a vertex list or of a vertex in a face list is called a *corner*.

We distinguish between three mesh components: *connectivity*, which describes the incidences between vertices, edges, and faces, *geometry*, which refers to the locations of each vertex, and *properties* such as shading normals, colours, texture coordinates, and material attributes, which are attached to the vertices, faces or corners of the mesh.

Typically, polygon mesh compressions schemes first encode the mesh connectivity [19, 20, 5, 16, 8]. Starting from an arbitrary edge, the encoder traverses the vertices and the triangles of the mesh using some deterministic strategy (e.g. breadth or depth first search). During this traversal the encoder records a stream of symbols from which the corresponding decoder can reconstruct the connectivity.

The geometry and the properties of the mesh are usually compressed with simple delta coding [3] or more sophisticated predictive coding methods [19, 20, 2], but always based on local neighbourhood information that is available to the encoder *and* the decoder at that time. The availability of neighbourhood information is assured by using the same traversal for encoding and decoding; this way the encoder can keep track of what the decoder knows.

The fact that encoder and decoder use the same traversal order is also used to establish the connection between the geometry/property data and the vertices, faces, or corners that they are associated with. Since the traversal defines an implicit ordering of these mesh elements, the encoder simply stores the compressed geometry/property data in the order in which the vertices, faces, or corners are encountered that they are attached to. The decoder performs the same traversal and re-assigns the data to the appropriate places.

A *one-pass* coder uses the traversal order induced by the connectivity encoder to process geometry and property data. A *multi-pass* coder, on the other hand, traverses the mesh two or more times during both encoding and decoding. The decoder first reconstructs the complete connectivity information before re-attaching the geometry and the properties. In the multi-pass case the mesh traversals used to compress and store the geometry and property data can be different from the traversal used for connectivity coding.

3. Characterizing the Property Mapping

In the literature a property mapping is often classified as either *per-vertex*, *per-face*, or *per-corner* with the first two being special cases of the last. In the per-vertex case the properties are attached to the vertices of the mesh; a common property is shared by all corners around a vertex. In the per-face case the properties are attached to the faces of the mesh; a common property is shared by all corners around a face. In the per-corner case the properties are attached to the corners of the mesh; although each corner could have a different property, typically a common property is shared by a set of adjacent corners.

For shading normals, colours and texture coordinates there is usually a one-to-one mapping between the properties and the mesh elements. A per-vertex mapping has as many properties as vertices and each property is used by one vertex. Similarly a per-face mapping has as many properties as faces and each property is used by one face. For a per-corner mapping there are at least as many properties as vertices and at most as many properties as corners. Here each property can be used by a set of adjacent corners if they lie around the same vertex.

An exception is the mapping of material attributes, which are usually attached to the faces of the mesh. They are mapped differently because each material attribute is used by many faces. All faces that represent the same real-world surface are given the same material attribute.

A per-vertex mapping for shading normals, colours, and texture coordinates results in a completely smooth shaded mesh when interpolated shading (e.g. Gouraud shading) is applied. It is smooth *along* each edge because the properties attached to the vertices at its ends are interpolated. It is smooth *across* each edge because the same properties are interpolated on both sides of the edge. Shading discontinuities may be introduced by cutting the mesh along a discontinuity and duplicating the affected vertices. These duplicates are given the same vertex location but different properties, which creates the shading discontinuity.

A per-face mapping of shading normals or colours gives the mesh a faceted appearance unless it is highly tessellated. An example for a face-based property assignment is a pre-computed radiosity solution. Each face is assigned a colour that expresses the amount of light it emits or transmits.

A per-corner mapping is a more elegant way to specify shading discontinuities in smooth shaded meshes. Each vertex in a smooth region of the mesh has a single property that is shared by all its corners. Vertices along a discontinuity, however, have multiple properties each of which is shared by a set of adjacent corners. This avoids having to cut the mesh and deal with multiple copies of vertices.

Following, a few definitions to characterize the different configurations that can arise for the property mapping:

Around every vertex is a cycle of corners and edges. In this paper we use a counter clockwise order to talk about a next/following and a previous/preceding edge or corner. A vertex can be visited through any of its edges. For each edge there is a unique traversal of the corners surrounding the vertex. The traversal starts with the corner following the respective edge and ends with the corner preceding it.

We say a corner is a *smooth corner* if it uses the same property as the previous corner, otherwise we have a *crease*

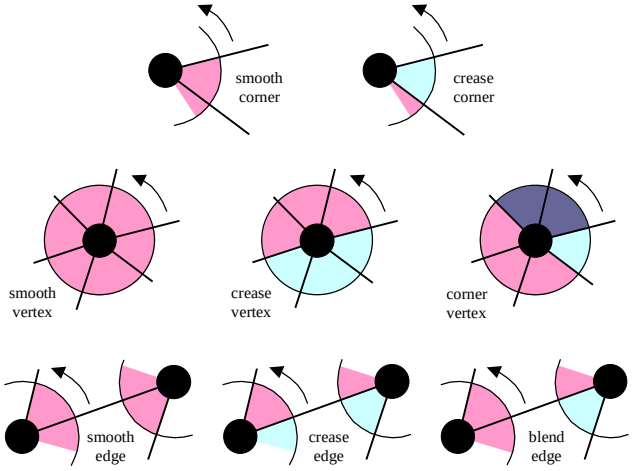

Figure 1. Different shaded corners have different properties associated. A smooth corner uses the same property as the previous corner, while a crease corner (a crease) uses a different one. Smooth vertices have no crease, crease vertices have two creases, and corner vertices have three or more creases. Smooth edges have no crease, crease edges have two creases, while blend edges have only one crease.

and consequently call it a *crease corner* (see also the illustrations in Figure 1). A *smooth vertex* uses the same property for all adjacent corners—it has no crease. A *crease vertex* uses two different properties each associated with a set of adjacent corners—it has two creases. A *corner vertex* uses three or more properties each associated with a set of adjacent corners—it has more than two creases. A *smooth edge* has no property discontinuity on either end. The two corners that are next around the respective vertices are both smooth corners—this edge has no crease. A *crease edge* has a property discontinuity on each end. The two corners that are next around the respective vertices are both crease corners—this edge has two creases. And finally, a *blend edge* has a property discontinuity on only one end—it has one crease.

A mesh with a per-vertex mapping has no creases; all its corners are smooth corners, all its vertices are smooth vertices, and all its edges are smooth edges. Such meshes have a one-to-one mapping from vertices to properties. A mesh with a per-face mapping has only creases; all its corners are crease corners, all its vertices are crease or corner vertices, and all its edges are crease edges. Such meshes have a one-to-one mapping from faces to properties. A mesh with a per-corner mapping is somewhere between these two. Such meshes tend *not* to have a one-to-one mapping from corners to properties. However, there is a one-to-one mapping from smooth vertices *plus* crease corners to properties.

4. Compressing the Property Mapping

Neither a per-vertex nor a per-face property mapping need to be stored explicitly. The property values are simply stored in the order in which the vertices and faces that they are associated with are encountered during the traversal of the mesh. Every property is stored only once because of the one-to-one mapping between vertices/faces and properties. An exception is the mapping of material attributes. Since a material attribute is used by many faces it would be stored many times. In this case it is cheaper to store the property mapping from faces to materials explicitly but therefore each material attribute only once. This type of mapping can be efficiently encoded using the *super face* concept of the Face Fixer compression scheme [8]. Using super faces allows to compress structural information, like face groupings, together with the mesh connectivity at the expense of a few extra bits.

A per-corner property mapping also needs to be stored explicitly. If every property is to be stored only once, we must specify which corners share a property. So far three different methods for compressing per-corner mappings of properties have been proposed.

Gumhold and Strasser [5] describe how to include this mapping information into the bit-stream of their one-pass coder. During the traversal of the mesh, all edges are classified as either smooth edges or as crease/blend edges using one bit. The latter are distinguished using two additional bits that specify, for each end of the edge, whether it is a crease or not. Encoding three possible configurations (e.g. the case that both ends have no crease cannot occur) with two bits is slightly wasteful and could be improved. For triangle meshes this approach requires at least 3 bpv (e.g. all edges are smooth) and at most 9 bpv (e.g. no edge is smooth). In praxis bit-rates range between 3 and 5.5 bpv.

Taubin et al. [18] store a *discontinuity bit* at the moment their mesh traversal reaches a corner for the first time. This bit is “0” for a smooth corner and a “1” for a crease corner. The property data that is associated with crease corners is then stored in the order in which the corresponding corners marked with “1” are encountered. This approach requires exactly as many bits as the mesh has corners, which implies a bit-rate of 6 bpv for triangular meshes.

In [8] we proposed a simple but effective improvement on the work by Taubin et al. [18]. Based on the observation that meshes often have a significant fraction of smooth vertices, we proposed a scheme that uses *vertex bits* and *corner bits*. We use one bit per vertex to distinguish smooth vertices (“1”) from crease and corner vertices (“0”). The corners around a crease or corner vertex are then marked as before, while the corners around a smooth vertex need no further treatment. Again, the property data associated with smooth vertices and crease corners is stored in the same or-

der as the corresponding "1" bits appear in the bit sequence. For triangle meshes this approach requires at least 1 bpv (e.g. all vertices are smooth) and at most 7 bpv (e.g. no vertex is smooth). The performance gain/loss of this approach over [18] depends on the fraction of smooth vertices. For polygon meshes with an average vertex degree of d the break-even point is reached when this fraction is approximately $1/d$ with the gain increasing as the fraction gets larger. If we can afford an initial pass over the mesh we could always choose the better of the two methods. Otherwise, the bit-rate is at most 1 bpv above, but potentially 5 bpv below the bit-rates of Taubin et al. [18].

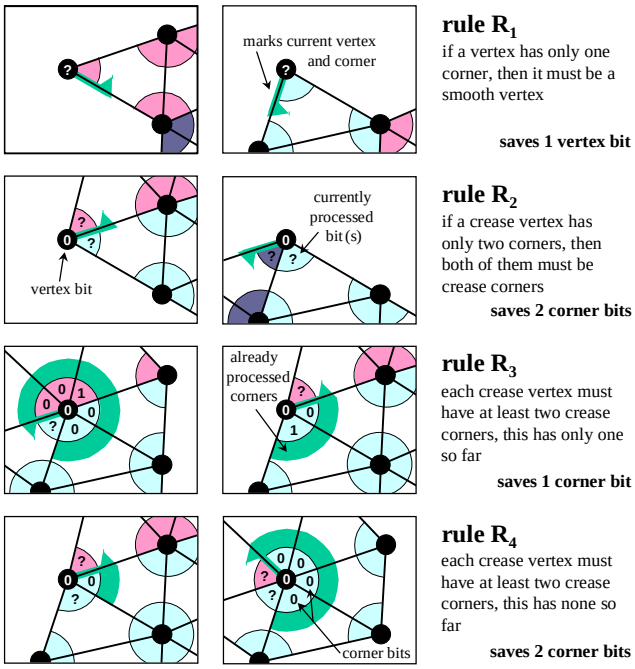

Figure 2. Simple rules to save vertex and corner bits. Using rule R_1 avoids unnecessary vertex bits, rules R_2, R_3, and R_4 avoid unnecessary corner bits.

Four simple rules, which are illustrated in Figure 2, can further reduce the number of vertex and corner bits needed:

rule R_1 Vertices that have only one corner do not need a vertex bit. Such vertices are always smooth vertices.

rule R_2 Crease vertices that have only two corners do not need corner bits. The vertex bit already determines whether it is a smooth vertex and both corners are smooth corners, or a crease vertex and both corners are crease corners.

rule R_3 If all but one corner of a vertex have been marked and there has been only one crease corner, then there is no need for the last corner bit. Because corner bits are only used for crease/corner vertices and such vertices have at least two crease corners, the last corner must be a crease corner.

rule R_4 Similarly, if all but two corners of a vertex have been marked and there has been no crease corner, then there is no need for the last two corner bits.

The rules R_1 and R_2 rarely apply for meshes without holes or boundary, since usually only vertices on the boundary have as few as one or two corners. However, these rules are very effective for compressing the property mapping of stripified triangle meshes, which is described in Section 6.

5. Predictive Compression

Generally mesh compression techniques make heavy use of predictive coding. To compress the location of a vertex Taubin and Rossignac [19] predict it with a linear combination of previously decoded vertices and then store only a corrective vector. Touma and Gotsman [20] do the same, but propose a more efficient prediction method. Connectivity coders also use predictive methods. Szymczak et al. [17] report a predictive compression scheme for decoding Edgebreaker [16] encoded meshes with the Spirale Reversi decoder [9]. They exploit the fact that triangle meshes usually have a large number of vertices with degree 6 to predict the next Edgebreaker label.

	corners		edges			vertices		
name								
button	66	34	66	34	—	—	100	—
dragknob	66	34	66	34	—	—	100	—
handle	60	40	60	40	—	—	70	30
handle1	66	34	66	34	—	—	100	—
handle2	98	2	98	2	—	92	6	—
part1	66	34	66	34	—	—	98	2
part4	83	17	83	17	—	50	50	—
part5	66	34	66	33	<1	1	94	3
spool	81	19	81	19	—	43	57	—
rotor	83	17	83	17	—	49	51	—
oilfilter	77	23	77	23	—	33	61	6
galleon	70	30	70	30	—	48	38	14
sandal	76	24	76	23	<1	56	33	11

Table 1. Statistics on the normal mapping for our example meshes. Reported are the percentages of smooth and crease corners, and of smooth, crease, and blend edges, and also of smooth, crease, and corner vertices.

Predictive coding can also be used to compress the property mapping. We use neighbourhood information to predict vertex bits and corner bits. We base our predictions on two simple observations that are also reflected in the statistics on the per-corner shading normal mapping given in Table 1 for the set of example meshes shown in Figure 5:

- Most edges are either smooth edges or crease edges.
- Most vertices are either smooth vertices or crease vertices.

For predictive compression we establish a set of eight simple predictions. For the vertex bits we have two predictions P_1 and P_2, for the corner bits we have six prediction P_3 to P_8. We use an adaptive arithmetic coder [21] that learns and exploits the probability of the predictions being correct.

Two example configurations for each of the vertex bit predictions are illustrated in Figure 3. They are as follows:

prediction P_1 If the vertex connects to any processed vertex at a crease, we predict this vertex to be a crease or corner vertex, because we assume the connecting edge is a crease edge.

prediction P_2 In all other cases we make no assumption.

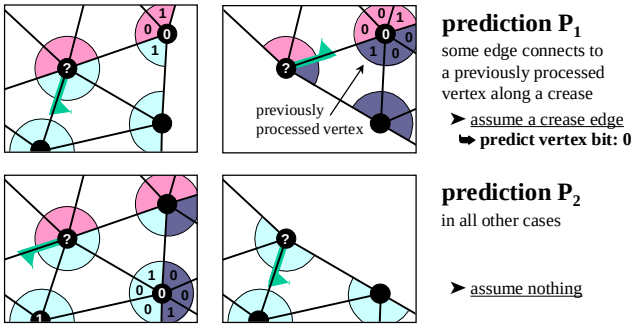

Figure 3. Two example scenarios for each of the two cases used to predict vertex bits.

Two example configurations for each of the corner bit predictions are illustrated in Figure 4. They are as follows:

prediction P_3 If the current edge connects to a processed vertex at a crease, we predict the next corner to be a crease corner, because we assume the edge is a crease edge.

prediction P_4 If the current edge connects to a processed vertex but not at a crease, we predict the next corner to be a smooth corner, because we assume the edge is a smooth edge.

prediction P_5 If there have been already two (or more) crease corners, we predict the next corner to be a smooth corner because we assume this vertex is a crease vertex.

prediction P_6 If there has been already one crease corner and since then there have been less than $\lfloor (degree-1)/2 \rfloor$ smooth corners, we predict the next corner to be a smooth corner because we assume this vertex is a crease vertex.

prediction P_7 If there have been $\lfloor (degree-1)/2 \rfloor$ or more smooth corners, we predict the next corner to be a crease corner because we assume this vertex is a crease vertex.

prediction P_8 In all other cases we make no assumption.

To effectively use predictive methods for single bits we need to utilize a coder that can exploit the probability of correct predictions. Arithmetic coders [21] are well suited for this. We use different probability tables for each of the eight predictions P_1 to P_8. Every probability table has two entries, one predicting the likelihood of a "0" bit and the other predicting the likelihood of a "1" bit. Initially we set these probabilities to roughly express our predictions. We use an adaptive version of an arithmetic coder that updates the respective probability table after every prediction.

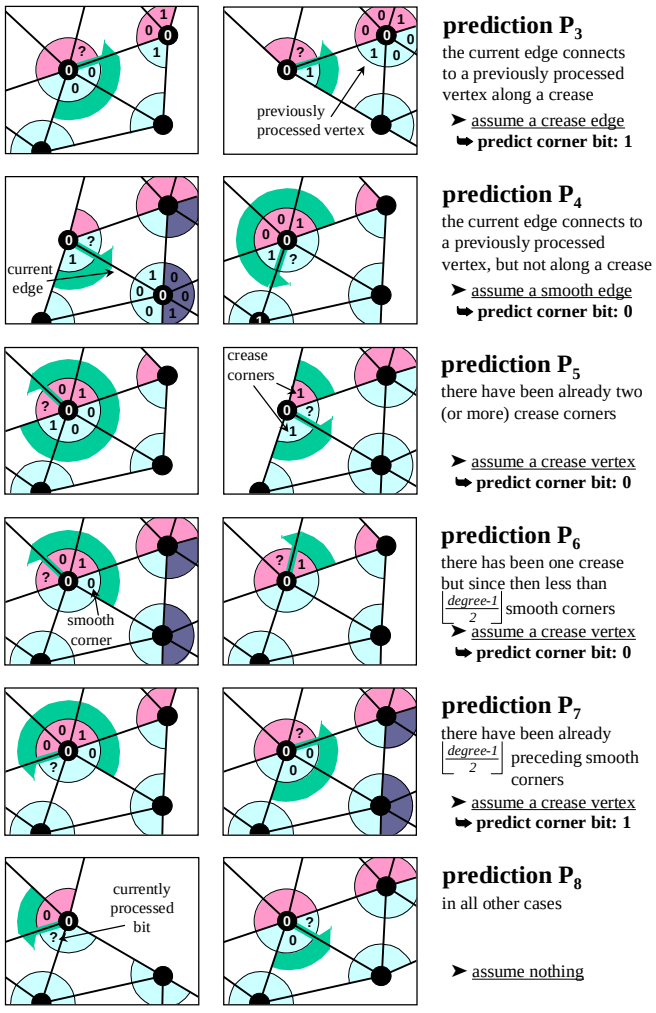

Figure 4. Two example scenarios for each of the six cases used to predict corner bits.

Typical bit-rates for compressing the property mapping using our approach are reported in Table 2. For comparison we also give the bit-rates achieved using discontinuity bits as proposed by Taubin et al. [18], using crease edge bits as proposed by Gumhold and Strasser [5], and using vertex and corner bits as proposed by Isenburg and Snoeyink [8].

One can argue that the comparison between a coder that simply stores a bit sequence and one that applies arithmetic coding to a bit sequence is unfair. This is true. The amount of compression that can be achieved by applying an arithmetic coder to a bit sequence depends on the entropy of the bit sequence. Given a sufficiently long input, the com-

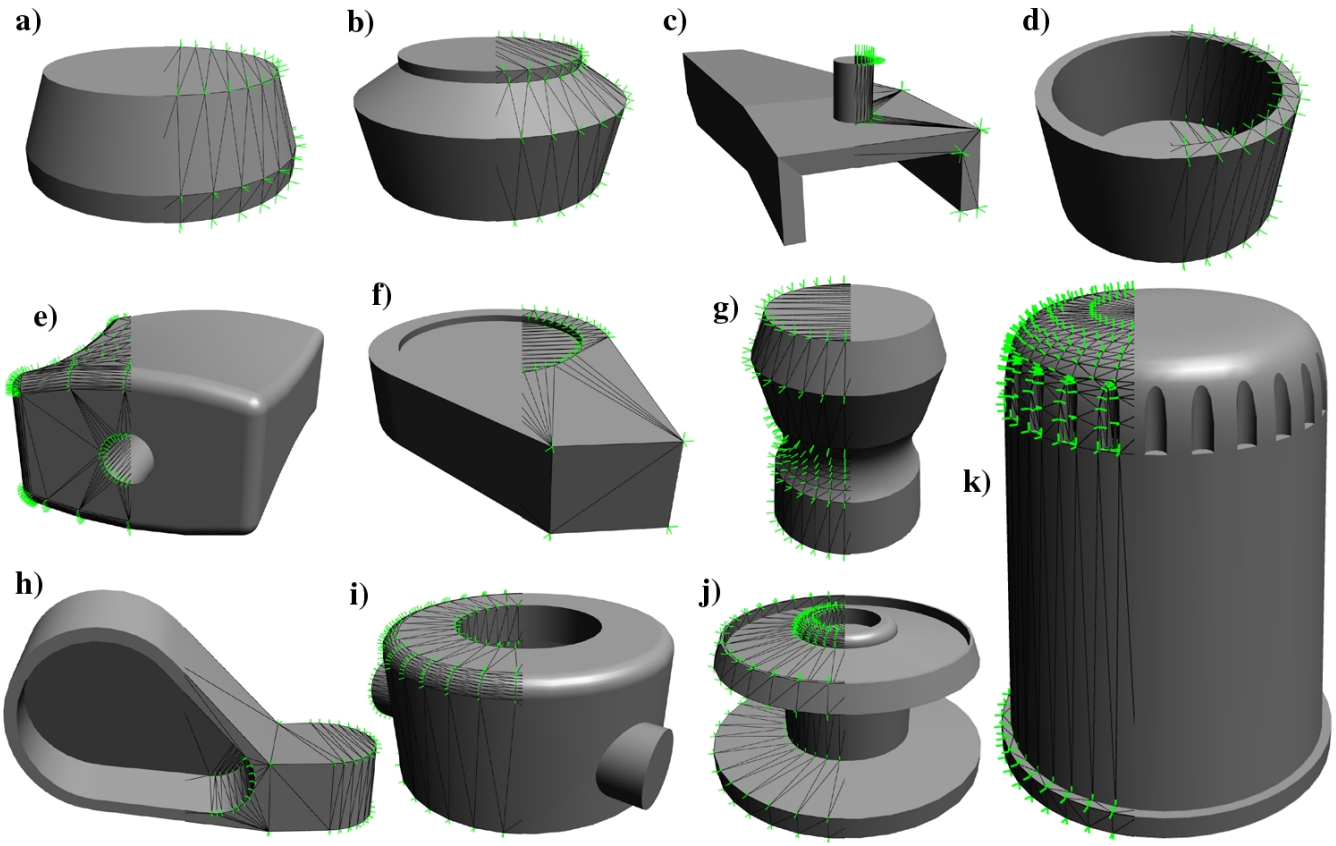

Figure 5. The example meshes used in this paper are courtesy of Engineering Animation Inc. All except the last are part of a fishing reel assembly: a) button, b) dragknob, c) handle, d) handle1, e) handle2, f) part1, g) part4, h) part5, i) rotor, j) spool, and k) oilfilter. Not shown are the galleon and the sandal mesh courtesy of Viewpoint Datalabs.

pression rate of such an entropy coder converges to the entropy of the input. The entropy for a sequence of n bits is $-n(p_0 \log_2(p_0) + p_1 \log_2(p_1))$, where p_0 and p_1 denote the probabilities for bit "0" and "1" to occur.

Correlation among subsequent bits can be exploited using a memory-sensitive coding scheme. An adaptive order-k arithmetic coder uses a different probability table for each combination of the preceding k bits and approximates the order-k entropy. For a sequence of n bits this is

$$-n \sum_{k\text{-bit strings } \alpha} \left(p_{0|\alpha} \log_2(p_{0|\alpha}) + p_{1|\alpha} \log_2(p_{1|\alpha})\right),$$

when n is large and where $p_{0|\alpha}$ and $p_{1|\alpha}$ denote the probability of finding bit "0" and "1" after the k-bit string α.

In Table 3 we give compression rates that are the result of applying adaptive arithmetic coding of orders 0 to 5 to the sequence of discontinuity bits generated by the method of Taubin et al. [18] and compare them with the compression rates achieved by our predictive coder. There are sudden improvements in the compression rates of the discontinuity bit sequence. The biggest jumps occur when the coder increases its memory to either 2 or 3 bits. The order-2 coder has learned the likelihood of two smooth corners being followed by a crease corner around a crease vertex. The order-3 coder has learned the likelihood of three smooth corners being followed by another smooth corner around a smooth vertex. However, our predictive scheme always outperforms any arithmetic order-k coding of the discontinuity bit sequence. First of all, an order-k coder has no means to learn predictions P_1 and P_2 from the discontinuity bit sequence, because they involve neighbouring vertices. Moreover, this coder is constantly mislead. It processes a continuous sequence of discontinuity bits and does not know whether consecutive bits are from corners of the same or of different vertices. This causes it to predict and to learn from

mesh characteristics			bits per vertex			
name	vertices	normals	T+	GS	IS	pred
button	99	198	6.0	4.9	6.6	1.2
dragknob	161	322	6.0	5.0	6.8	1.3
handle	100	236	6.0	5.3	6.3	2.1
handle1	128	256	6.0	5.0	6.6	1.5
handle2	1165	1235	6.0	3.1	1.3	0.1
part1	166	336	6.0	5.0	6.4	1.6
part4	330	495	6.0	4.0	3.8	0.9
part5	175	355	6.0	5.0	6.5	1.9
rotor	600	905	6.0	4.0	4.0	1.0
spool	649	1018	6.0	4.1	3.8	1.1
oilfilter	860	1484	6.0	4.4	4.7	1.5
galleon	2372	3974	4.0	3.2	2.8	1.0
sandal	2636	4096	4.1	3.0	2.7	0.9

Table 2. Compression results for the property mapping using discontinuity bits as proposed by Taubin et al. (*T+*), using crease edge bits as proposed by Gumhold and Strasser (*GS*), using vertex and corner bits as proposed by Isenburg and Snoeyink (*IS*), and our proposed predictive version of the latter (*pred*).

mesh	bits per vertex							
name	T+	aac0	aac1	aac2	aac3	aac4	aac5	pred
button	6.0	5.5	4.9	2.7	2.6	2.6	2.6	1.2
dragknob	6.0	5.5	4.6	2.5	2.5	2.5	2.5	1.3
handle	6.0	5.7	5.3	5.0	4.6	4.6	4.5	2.1
handle1	6.0	5.5	4.8	2.6	2.6	2.6	2.6	1.5
handle2	6.0	0.9	0.9	0.8	0.4	0.3	0.3	0.1
part1	6.0	5.5	5.0	3.5	3.3	3.3	3.3	1.6
part4	6.0	3.9	3.8	3.7	2.1	1.9	1.9	0.9
part5	6.0	5.5	4.7	4.1	4.1	4.1	4.1	1.9
rotor	6.0	4.2	4.0	3.6	1.3	1.1	1.1	1.0
spool	6.0	4.0	3.8	3.8	2.3	2.2	2.1	1.0
oilfilter	6.0	4.7	4.6	4.6	4.0	3.5	3.4	1.5
galleon	4.0	3.5	3.4	2.5	2.3	2.3	2.2	1.0
sandal	4.1	3.3	3.3	2.4	2.2	2.2	2.2	0.9

Table 3. The discontinuity bit sequence for coding the property mapping (*T+*) compressed with adaptive arithmetic coding of orders 0 to 5 (*aac0* to *aac1*) in comparison to our predictive coder (*pred*).

k-bit strings that contain corner bits from different vertices and therefore have little or no correlation.

6. Stripified Triangle Meshes

For interactive visualization the speed at which a mesh can be displayed is important. The use of triangle strips [22] reduces the workload required to render a mesh and thereby reduces the time needed to display it. Generating good triangle strips [4, 23] is a difficult and time consuming task. This motivated us to propose a compression technique that allows to include pre-computed triangle strip information with little overhead [6]. In this section we show that if such triangle strip information is available, it can be used to further improve the coding of the property mapping.

Triangle strips arrange the mesh triangles into long runs of adjacent triangles. When rendering a triangle strip, the vertices shared among subsequent triangles are stored on the graphics board. This way two vertices from a previous triangle are re-used for all but the first triangle of every strip. Compared to rendering each triangle individually this can potentially reduce the number of vertex repetitions by a factor of three.

Re-using a vertex not only includes its coordinates, but also shading normals, colours, or texture coordinates. However, not every pair of triangles shares these properties across the common edge. When triangles are connected along a crease or blend edge they uses a different property at both or one of their common vertices. The triangle strips generator must ensure that strips do not run across such discontinuities. All corners of a vertex that are adjacent in a triangle strip must share all properties attached to them.

The above complicates the process of creating triangle strips, but it can be exploited for reducing the number of bits needed to compress the property mapping. Since a property will always be shared by all corners of a vertex that are adjacent in a triangle strip, the property mapping can be thought of as per *strip corner* rather than per corner. The number of different corners for a mesh with t triangles is $3t$. However, decomposed into s strips we need to distinguish only $t + 2s$ strip corners for the mapping from properties to corners.

The bit-saving rules R_1 to R_4 are now applicable to strip corners instead of to corners. Since the number of strip corners per vertex is much lower than the number of corners, these rules apply much more often and save many more bits. Nothing changes for the predictive technique. We continue to predict vertex bits using P_1 and P_2 and corner bits (but only for strip corners) using P_3 to P_8. In Table 4 we list the compression rates achieved using a bit sequence of vertex and strip corner bits (i.e. without predictive compression) and the compression rates after applying the predictions.

The results in Table 4 show that the availability of triangle strip information makes the compression of the property mapping a lot cheaper. However, this information is only available if it was encoded with the mesh in such a way that it can be decoded prior to decoding the property mapping. The Triangle Strip Compression scheme [6] compresses the triangle strips with the mesh connectivity. In most cases this makes encoding the connectivity more expensive. However, the savings we get from the improved compression of the property mapping is often sufficient to offset this expense.

mesh characteristics				bits per vertex	
name	t	s	$(t+2s)/3t$	IS	pred
button	194	4	0.35	1.1	0.2
dragknob	318	6	0.35	1.1	0.2
handle	196	24	0.42	2.0	0.5
handle1	252	5	0.35	1.1	0.2
handle2	2326	125	0.37	1.0	0.1
part1	328	6	0.35	1.1	0.2
part4	656	34	0.37	1.1	0.3
part5	346	9	0.35	1.2	0.2
rotor	1200	41	0.36	1.1	0.3
spool	1294	24	0.35	1.1	0.2
oilfilter	1716	135	0.39	1.4	0.5

Table 4. Compressing the property mapping of stripified triangle meshes using a bit sequence of vertex and strip corner bits (*IS*) and the proposed predictive version (*pred*). Also reported are the number of triangles t and strips s for each mesh and the ratio $(t+2s)/3t$ between strips corners and corners.

7. Discussion

We have introduced a predictive method for compressing the mapping from corners to properties. Our compression rates improve by a factor of 2 to 10 on previously reported methods. We have also described how our scheme can exploit triangle strip information for compressing the property mapping. The improvement in compression is so significant that it can potentially offset the expense of including the triangle strip information with the mesh.

We have also tested simpler versions of our predictive scheme. The predictions P_6 and P_7 require access to the degree of a vertex for two integer operations. If simplicity and speed of the implementation are crucial, it is possible to replace them with simpler predictions. We experimented with predictions based on absolute counts of preceding smooth and crease corners. The increase in bit-rates was less than 5 percent. This is not surprising, since most of our coder's efficiency results from predictions P_1 to P_5.

Currently our coder uses a simple depth-first traversal order. It might be possible to improve compression by directing the traversal to places where we expect the most correct predictions. A traversal order that follows the discontinuities on the mesh, for example, should make sure that most vertex bits are predicted correctly. For connectivity coding Alliez and Desbrun [1] used such an approach to improve the compression scheme by Touma and Gotsman [20].

References

[1] P. Alliez and M. Desbrun. Valence-driven connectivity encoding for 3D meshes. In *Eurographics'01*, to appear.

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

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

[4] F. Evans, S. S. Skiena, and A. Varshney. Optimizing triangle strips for fast rendering. In *Visualization'96 Conference Proceedings*, pages 319–326, 1996.

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

[6] M. Isenburg. Triangle Strip Compression. In *Graphics Interface'00 Conference Proceedings*, pages 197–204, 2000.

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

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

[9] M. Isenburg and J. Snoeyink. Spirale Reversi: Reverse decoding of the Edgebreaker encoding. In *Proc. of 12th Canad. Conf. on Computational Geometry*, pages 247–256, 2000.

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

[11] Z. Karni and C. Gotsman. 3D mesh compression using fixed spectral bases. In *Graphics Interface'01*, pages 1–8, 2001.

[12] D. King and J. Rossignac. Guaranteed 3.67v bit encoding of planar triangle graphs. In *Proc. of 11th Canad. Conf. on Computational Geometry*, pages 146–149, 1999.

[13] D. King, J. Rossignac, and A. Szymczak. Connectivity compression for irregular quadrilateral meshes. Technical Report TR–99–36, GVU Center, Georgia Tech, Nov. 1999.

[14] B. Kronrod and C. Gotsman. Efficient coding of non-triangular meshes. In *Proceedings of Pacific Graphics'00*, pages 235–242, 2000.

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

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

[17] A. Szymczak, D. King, and J. Rossignac. An Edgebreaker-based efficient compression scheme for connectivity of regular meshes. In *Proceedings of 12th Canadian Conference on Computational Geometry*, pages 257–264, 2000.

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

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

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

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

[22] M. Woo, J. Neider, and T. Davis. *Open GL Programming Guide*. Addison Wesley, 1996.

[23] X. Xiang, M. Held, and J. Mitchell. Fast and efficient stripification of polygonal surface models. In *Proceedings of Interactive 3D Graphics*, pages 71–78, 1999.