Compressing Polygon Mesh Geometry with Parallelogram Prediction

Martin Isenburg[*]
University of North Carolina
at Chapel Hill

Pierre Alliez[†]
INRIA
Sophia-Antipolis

Abstract

In this paper we present a generalization of the geometry coder by Touma and Gotsman [34] to polygon meshes. We let the polygon information dictate where to apply the parallelogram rule that they use to predict vertex positions. Since polygons tend to be fairly planar and fairly convex, it is beneficial to make predictions *within* a polygon rather than *across* polygons. This, for example, avoids poor predictions due to a crease angle between polygons. Up to 90 percent of the vertices can be predicted this way. Our strategy improves geometry compression by 10 to 40 percent depending on (a) how polygonal the mesh is and (b) on the quality (planarity/convexity) of the polygons.

I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—surface, solid, and object representations;

Keywords: Mesh compression, polygon meshes, geometry coding, linear prediction, parallelogram rule.

1 INTRODUCTION

The emerging demand for visualizing polygon meshes in networked environments has motivated research on efficient representations for such data. Since transmission bandwidth tends to be a limited resource, compressed polygon mesh formats are beneficial. The basic ingredients that need to be compressed are the mesh *connectivity*, that is the incidence relation among the vertices, and the mesh *geometry*, that is the specific location of each individual vertex. Optionally there are also mesh *properties* such as texture coordinates, shading normals, material attributes, etc.

Traditionally, mesh compression schemes have focused on pure triangular meshes [7, 33, 34]. The triangle is the basic rendering primitive for standard graphics hardware and any other surface representation can be converted into a triangle mesh. A popular compressor for triangle meshes was proposed by Touma and Gotsman [34]. Their connectivity coder tends to give the best bit-rates for compressing triangular connectivity and recent results have a posteriori confirmed the optimality of their approach [2]. Also their geometry coder delivers competitive compression rates. The simplicity of this coder allows a fast and robust implementation and is one reason for the popularity of their scheme. Yet the achieved bit-rates are so good that it continues to be the accepted benchmark coder for geometry compression.

However, there are a significant number of polygon meshes that are not completely triangular. The 3D models of the Viewpoint library [35] for example, a well-known source of high-quality meshes, contain only a small percentage of triangles. Likewise, few triangles are found in the output of many modeling packages. The dominating element in these meshes is the quadrilateral, but pentagons, hexagons and higher degree faces are also common.

[*]isenburg@cs.unc.edu http://www.cs.unc.edu/~isenburg/pmc
[†]pierre.alliez@sophia.inria.fr

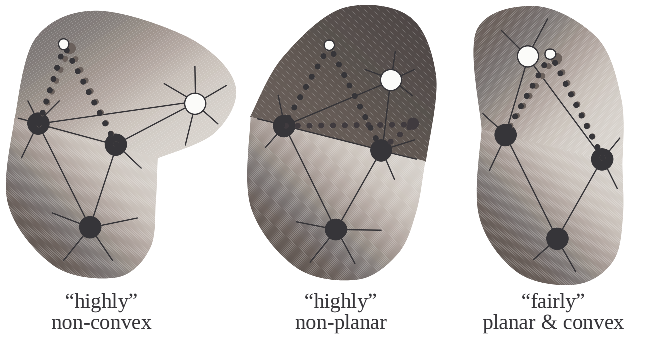


Figure 1: The parallelogram predictor performs poorly when adjacent triangles are highly non-convex or highly non-planar. Performing this prediction *within* a polygon is beneficial: polygons tend to be fairly convex and planar.

The first approaches to compress polygonal meshes performed a triangulation step prior to compression and then used a triangle mesh coder. In order to recover the original connectivity they had to store additional information that marked the edges added in the triangulation step. Inspired by [21] we showed in [13] that polygonal connectivity can be compressed more efficiently directly in its polygonal representation by avoiding the triangulation step.

More recently we have extended the connectivity coder of Touma and Gotsman [34] to the polygonal case and also confirmed its optimality [12, 18]. This approach gives the lowest reported compression rates for polygonal connectivity and can be seen as a generalization of Touma and Gotsman’s connectivity coder [34].

In a similar spirit we show in this paper how to extend their geometry coder to the polygonal case. Thereby we confirm our claim from [13] that polygonal information can be used to improve compression of geometry. Our polygonal geometry coder is just as simple as the original Touma and Gotsman (TG) coder [34] but significantly improves the compression rates on polygonal meshes.

The TG coder compresses the geometry of triangle meshes with the *parallelogram rule*. The position of a vertex is predicted to complete the parallelogram formed by the vertices of a neighboring triangle and only a corrective vector is stored. The sequence of correctors, which tend to spread around the zero vector, can be compressed more compactly than the sequence of positions.

However, the parallelogram predictor performs poorly when adjacent triangles are highly non-convex or highly non-planar as illustrated in Figure 1. Since polygons tend to be fairly convex and fairly planar we can use them to make better parallelogram predictions. We (a) predict *within* rather than *across* polygons and (b) traverse the polygons such that we can do this as often as possible.

2 MESH COMPRESSION

Recently we have seen a number of novel and innovative approaches to the compression of polygonal meshes. There are spectral methods [16, 17] that perform a global frequency decomposition of the surface, there are space-dividing methods [8, 9] that specify the mesh connectivity relative to a geometric triangulation of connectivity-less coded positions, there are remeshing methods [20, 19] that compress a regularly parameterized version instead of the original mesh, there are angle-based methods [25] that encode the dihedral and internal angles of the mesh triangles, there are model-space methods [24] that compress prediction errors in a local coordinate frame using vector quantization, there are feature-based methods [30] that find repeated geometric features in a model, and finally there are progressive methods [32, 11, 28, 6, 3, 1] that allow to incrementally reconstruct the mesh at multiple resolutions.

All of the above schemes are fairly complex: Some perform expensive global operations on the mesh [16, 17], some require complex geometric algorithms [8, 9], some modify the mesh prior to compression [20, 19], some involve heavy trigonometric computations [25], some make use of complex quantization schemes [24], some perform expensive searches for matching shapes [30], and some are meant to fulfill a more complex task [32, 11, 28, 6, 3, 1]. We do not attempt to improve on these schemes. Instead we generalize a simple and popular triangle mesh compressor [34] to achieve better compression performance on polygonal meshes.

Traditionally, the compression of mesh connectivity and the compression of mesh geometry are done by clearly separated (but often interwoven) techniques. Most efforts have focused on connectivity compression [7, 33, 34, 26, 10, 29, 21, 4, 13, 22, 31, 2, 12, 18]. There are two reasons for this: First, this is where the largest gains are possible, and second, the connectivity coder is the core component of a compression engine and usually drives the compression of geometry [7, 33, 34] and of properties [14].

Most connectivity compression schemes use the concept of region growing [12]. Faces adjacent to an already processed region of the mesh are processed one after the other until the entire mesh is completed. Most geometry compression schemes use the traversal order this induces on the vertices to compress their associated positions with a predictive coding scheme. Instead of specifying each position individually, previously decoded information is used to predict a position and only a corrective vector is stored. All popular predictive coding schemes use simple linear predictors [7, 33, 34].

The reasons for the popularity of linear prediction schemes is that (a) they are simple to implement robustly, (b) compression or at least decompression is fast, and (c) they deliver good compression rates. For nearly five years already the simple parallelogram predictor by Touma and Gotsman [34] is the accepted benchmark that many recent approaches compare themselves with. Although better compression rates have been reported, in practice it is often questionable whether these gains are justified given the sometimes immense increase in algorithmic and asymptotic complexity of the coding scheme. Furthermore these improvements are often specific to a certain type of mesh. Some methods achieve significant gains only on models with sharp features, while others are only applicable to smooth and sufficiently dense sampled meshes.

Predictive geometry compression schemes work as follows: First the floating-point positions are uniformly quantized using a user-defined precision of for example 8, 10, 12, or 16 bits per coordinate. This introduces a quantization error as some of the floating-point precision is lost. Then a prediction rule is applied that uses previously decoded positions to predict the next position and only an offset vector is stored, which corrects the predicted position to the actual position. The values of the resulting corrective vectors tend to spread around zero. This reduces the variation and thereby the entropy of the sequence of numbers, which means they can be efficiently compressed with, for example, an arithmetic coder [37].

The simplest prediction method that predicts the next position as the last position was suggested by Deering [7]. While this technique, which is also known as delta coding, makes as a systematic prediction error, it can easily be implemented in hardware. A more sophisticated scheme is the spanning tree predictor by Taubin and Rossignac [33]. A weighted linear combination of two, three, or more parent vertices in a vertex spanning tree is used for prediction. The weights used in this computation can be optimized for a particular mesh but need to be stored as well. Although this optimization step is computationally expensive, it is only performed by the encoder and not by the decoder. By far the most popular scheme is the parallelogram predictor introduced by Touma and Gotsman [34]. A position is predicted to complete the parallelogram that is spanned by the three previously processed vertices of a neighboring triangle.

Good predictions are those that predict a position close to its actual position. In the triangle mesh case the parallelogram rule gives good predictions if used across triangles that are in a fairly planar and convex position. Consequently, the parallelogram rule gives bad predictions if used across triangles that are in a highly non-planar and/or non-convex position (see Figure 1). Two approaches [23, 5] have recently been proposed to increase the number of good parallelogram predictions.

Kronrod and Gotsman [23] first locate good triangle pairs for parallelogram prediction and then try to use a maximal number of them. They construct a *prediction tree* that directs the traversal to good predictions. They propose a scheme that traverses this tree and simultaneously encodes the mesh connectivity. Especially on meshes with many sharp features, such as CAD models, they achieve significant improvements in geometry compression. However, this scheme is considerably more complex than the original method [34] in both algorithmic and asymptotic complexity.

Instead of using a single parallelogram prediction, Cohen-Or et al. [5] propose to average over multiple predictions. They define the *prediction degree* of a vertex to be the number of triangles that can be used to predict its position with the parallelogram rule. For typical meshes the average prediction degree of a vertex is two. In order to have as many multi-way predictions as possible, their geometry coder traverses the mesh vertices using a simple heuristic that always tries to pick a vertex with a prediction degree of two or higher. This approach improves slightly on the geometry compression rates reported in [34], but at the same time increases the complexity of the encoding and the decoding algorithm, since connectivity and geometry need to be processed in two separate passes.

Our geometry coder on the other hand can be seen as a generalization of the simple TG coder [34] to the polygonal case. The increase in complexity is only an extra *if ... else ...* statement and a second set of arithmetic probability tables.

3 GOOD “POLYGONAL” PREDICTIONS

We compress the geometry of a polygon mesh as follows: The vertex positions are uniformly quantized with a user-defined number of bits. In an order dictated by our connectivity coder [12] the vertex positions are predicted using the parallelogram rule [34]. This converts the sequence of quantized positions into a sequence of corrective vectors. This sequence of integer-valued correction vectors is then compressed using arithmetic coding [37]. Good predictions result in a corrector sequence with a small spread around the zero vector. Smaller symbol dispersion means lower entropy and leads to better arithmetic compression. We use the polygonal information to generate a sequence of mostly *good* predictions and only few *bad* predictions. The mesh traversal of our connectivity coder [12] naturally favors the occurrence of good predictions. Furthermore, we code good predictions with a different arithmetic context than bad predictions. This makes sure that bad predictions do not “spoil” the symbol distribution of the good predictions.

3.1 Quantizing of Vertex Positions

First the minimal and maximal x, y, and z coordinates of all positions are computed. They define a bounding box whose longest side is uniformly quantized with a user-defined number of k bits. The floating-point coordinates along this side are mapped into an integer number between 0 and $2^k - 2$. The floating-point coordinates along the other two sides of the bounding box are mapped into an integer range proportional to their length.

3.2 Predictive Coding of Quantized Positions

The first vertex position of each mesh component has no obvious predictor. We simply predict it as the center of the bounding box. There will be only one such *center* prediction per mesh component. The second and the third vertex positions cannot yet be predicted with the parallelogram rule since at least three vertices are needed for this. We predict them as a previously decoded position to which they are connected by an edge. This is simple delta coding and makes a systematic prediction error, but there will be only two such *last* predictions per mesh component. All following vertex positions use the parallelogram predictor. We distinguish two cases: a *within* and an *across* prediction (see Figure 2).

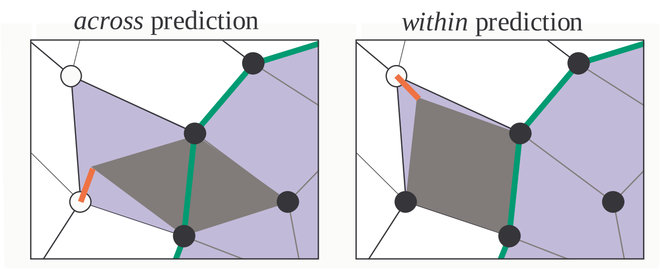


Figure 2: The parallelogram used for prediction is shown in dark grey and the corrective vectors are shown in red. Imagine the green shaded edges symbolize a sharp crease in the model: then *across* predictions will perform a lot worse than *within* predictions.

Polygonal faces tend to be fairly planar and convex. Although they are usually not perfectly planar, major discontinuities are improbable to occur across them—otherwise they would likely have been triangulated when the model was designed. Furthermore, a quadrilateral, for example, is usually convex while two adjacent triangles can potentially form a non-convex shape. Therefore predicting *within* a polygon is preferred over predicting *across* polygons.

A good mesh traversal order maximizes the number of *within* predictions. At least three vertices of a (non-triangular) polygon must be known before a *within* prediction is possible. A simple greedy strategy should grow the already processed mesh region as follows: (a) ideally continue with a polygon that shares three or more vertices with the processed region and (b) otherwise continue with a polygon that creates (a) for the next iteration.

Luckily the mesh traversal performed by our connectivity coder already gives us such a traversal (see also Figure 3). The mesh is traversed with a heuristic that aims at reducing the number of *split* operations, which are expensive to code. This idea of an *adaptive conquest* was originally proposed by Alliez and Desbrun [2] to improve the connectivity compression rates of the TG coder. Our strategy favors polygons that complete the boundary vertices of the processed region: ideally it continues with a polygon that completes a boundary vertex, which gives us (a). Otherwise it continues with a polygon that brings a boundary vertex closer to completion, which gives us (b). The results in Table 1 illustrate the success of this strategy: on average 84 % of the vertices are *within*-predicted.

mesh	predicted					% of	bpv	
name	total	within	across	last	center	within	within	other
triceratops	2832	2557	272	2	1	90	14.1	20.5
galleon	2372	2007	329	24	12	85	16.9	26.8
cessna	3745	3091	621	22	11	83	11.0	19.8
beethoven	2655	2305	326	16	8	87	21.0	24.2
sandal	2636	2084	525	18	9	79	14.1	22.8
shark	2560	2348	209	2	1	92	9.8	18.7
al	3618	2672	883	42	21	74	18.6	23.6
cupie	2984	2623	343	12	6	88	17.0	21.5
tommygun	4171	3376	678	78	39	81	10.9	19.5
cow	2904	0	2901	2	1	0	–	20.6
cow_poly	2904	2391	510	2	1	82	18.0	21.6
teapot	1189	1016	170	2	1	85	14.9	22.7
average						84	15.1	22.0

Table 1: This table reports how many vertices are predicted which way and the ratio between *within* prediction and other predictions. The bit rates in bits per vertex (bpv) for *within*-predicted versus otherwise predicted vertices are given for a precision of 12 bits.

3.3 Arithmetic Compression of Corrective Vectors

The parallelogram prediction produces a sequence of correctors that has less variation than the sequence of positions. Namely, the corrective vectors are expected to spread around the zero vector. This sequence of correctors has a lower information entropy than the original sequence of positions and can therefore be compressed more compactly. The entropy for a sequence of n symbols is $-\sum_n \big(p_i \log_2(p_i)\big)$, where the ith symbol occurs with probability p_i. Given sufficiently long input an adaptive arithmetic coder [37] converges to the entropy of the input. We use such a coder to compress the correctors into a compact bit-stream.

The correctors produced by *within* predictions tend to be smaller than those produced by *across*, *last*, and *center* predictions. That implies that the entropy of the *within* correctors will be lower than that of the others. For entropy coding it is beneficial not to spoil the lower entropy of the *within* correctors with the higher entropy of the other correctors. Therefore we use two different arithmetic contexts [37] depending on whether a corrector is the result of a *within* prediction or not. The results in Table 1 confirm the benefit of this approach: the arithmetic coder compresses the correctors of *within* predictions on average 30 percent better than the others.

For quantization with k bits of precision we map the position coordinates to a number between 0 and $2^k - 2$. We chose this instead of the more intuitive 0 and $2^k - 1$ range because it simplifies efficient arithmetic coding of the correctors. The chosen range allows us to express each corrective coordinate as number between $-2^{k-1} - 1$ and $+2^{k-1} - 1$, which can be specified using a sign bit and $k - 1$ value bits. The values of the correctors are expected to be spread around zero without preference for either sign. The value bits give a symmetric and sign-independent measure of the prediction error. The high-order bits are more likely to be zero than the low-order bits. The highest order bit, for example, will only be set if the prediction error is half the extend of the bounding box or more. It benefits from almost all predictions. The lowest order bit, on the other hand, can be set whenever there is a prediction error. It benefits only from the few exact predictions.

For memory-efficient arithmetic compression we break the sequence of $k - 1$ value bits into smaller sequences. This prevents the probability tables from becoming too large. For a precision of $k = 12$ bits, for example, we break the sequence of 11 value bits into sequences of 5, 3, 2, and 1 bits (plus the sign bit). Currently we initialize the arithmetic tables with uniform probabilities and use an adaptive coder that learns the actual distribution. An additional coding gain could be achieved by initializing the table with the expected distribution.

4 IMPLEMENTATION AND RESULTS

Most geometry compression schemes that are actually used in industry-strength triangle mesh coders are those with simple and robust implementations. For example Deering's delta coder [7] is used for geometry compression in Java3D [15], Taubin and Rossignac's spanning tree predictor [33] will be used in the upcoming MPEG-4 standard [27], and Touma and Gotsman's parallelogram predictor [34] is used by Virtue3D's compression engine [36], which is licensed by several others companies.

The proposed geometry compression scheme can be seen as a natural extension of the Touma and Gotsman [34] predictor to the polygonal case. Aside from an additional switch statement and a second set of probability tables, our algorithm has the same simple implementation. However, the results listed in Table 3 show that our generalization to polygon meshes gives an immediate improvement of more than 20 percent in the geometry compression rates. Note that for a purely triangular model (e.g. the cow model) we get roughly the same bit-rates as the Touma and Gotsman coder. This validates that our improvement really comes from using the polygonal information.

Our approach improves on the Touma and Gotsman [34] similarly at 8, 10, and 12 bits of precision. This demonstrates that the coding gains are independent from the chosen level of quantization. However, the relative percentage of compression achieved by a geometry coder is strongly dependent on the number of precision bits. This is clearly demonstrated in Table 2, which reports geometry compression gains at different quantization levels: with increasing precision the achieved compression ratio decreases.

This means that predictive compression does not scale linearly with different levels of precision. Such techniques mainly predict away the high-order bits. If more precision (= low bits) is added the compression ratio (e.g. the compressed size in proportion to the uncompressed size) decreases. In order to make a meaningful statement about the average compression rates of a geometry coder it is necessary to clarify at which quantization they were achieved. In Table 2 we report the performance of our geometry compression scheme at commonly used levels of precision.

mesh	8 bit		10 bit		12 bit		14 bit		16 bit	
name	bpv	gain	bpv	gain	bpv	gain	bpv	gain	bpv	gain
triceratops	6.7	72	10.7	64	14.8	59	19.0	55	23.0	52
galleon	8.6	64	13.2	56	18.4	49	23.8	44	28.9	40
cessna	4.8	80	8.2	73	12.5	65	17.3	59	22.3	54
beethoven	9.8	59	15.4	49	21.4	41	27.4	35	33.4	30
sandal	7.2	70	11.2	63	15.9	56	21.4	49	26.9	44
shark	4.7	80	7.3	76	10.6	71	13.9	67	17.3	64
al	8.7	64	14.1	53	19.9	45	25.8	39	31.7	34
cupie	6.7	72	11.7	61	17.5	51	23.5	44	29.6	39
tommygun	5.9	75	8.9	70	12.5	65	16.7	60	20.7	57
cow	8.9	63	14.6	51	20.6	43	26.6	37	32.7	32
cow_poly	7.4	69	12.7	58	18.7	48	24.6	41	30.7	36
teapot	6.6	73	11.1	63	16.1	56	21.2	50	26.2	46
average	7.2	70	11.5	62	16.6	54	21.8	48	27.0	44

Table 2: This table reports geometry compression rates in bits per vertex (bpv) at different quantization levels and the corresponding gain compared to the uncompressed geometry. The latter is simply three times the number of precision bits per vertex.

5 SUMMARY AND DISCUSSION

We have presented a simple technique that allows to exploit polygonal information for improved predictive geometry compression with the parallelogram rule. Our scheme is a natural generalization of the geometry coder by Touma and Gotsman [34] to polygon meshes and gives compression improvements of up to 40 percent. A proof-of-concept implementation of the proposed geometry coder is available in the form of an interactive Java applet on our Web page.

Can we further improve the polygonal geometry compression rates using only a simple linear predictor? Assume a parallelogram prediction is performed within a regular polygon (e.g. a planar and convex polygon with unit-edge lengths) of degree d. The prediction within regular quadrilaterals ($d = 4$) is perfect, but for higher-degree polygons ($d > 4$) the prediction error grows with the degree. Measurements on our test meshes show a similar behavior: predictions within quadrilaterals have the smallest average prediction error and the error becomes larger as the degree increases.

This observation suggests two ways of improvement: On one hand one could traverse the mesh such that a maximal number of vertices are predicted within a quadrilateral or a low-degree polygon. On the other hand one could change the linear prediction rule depending on the degree of the polygon a vertex is predicted within.

The parallelogram rule can be written as the linear combination $P = \alpha * A + \beta * B + \gamma * C$ where $\alpha = \gamma = 1$ and $\beta = -1$. It has the advantage that it can be implemented with pure integer arithmetic. If we allowed α, β, and γ to be floating point numbers, we could formulate a *pentagon rule* or a *hexagon rule*. Such rules would not be limited to base their predictions on only three vertices. The prediction of the last unknown vertex within a hexagon, for example, could use a linear combination of all five vertices.

The challenge is then to find *generic* coefficients that improve compression on all typical meshes. We tried to compute such coefficients for various polygonal degrees by minimizing the Euclidean error over all possible predictions in our set of test meshes. During compression we then switched the coefficients α, β, and γ based on the degree of the polygon we predicted within. This approach slightly improved the compression rates on all meshes; even on those that were not part of the set used to compute the coefficients.

Also predictions across polygons can be improved by switching between different floating point coefficients based on the degrees of the two polygons involved. Initial experiments show that such degree-adapted prediction rules result in small but consistent improvements in compression. However, these gains are bound to be moderate because on average more than 70 percent of the vertices are predicted within a quadrilateral. We found that the best linear predictor for these vertices is the standard parallelogram rule.

6 ACKNOWLEDGEMENTS

Many thanks go to Costa Touma for providing us an executable binary of his compression algorithm. The first author thanks Jack Snoeyink, Olivier Devillers, Agnès Clément Bessière, and Jean-Daniel Boissonnat for arranging his stay in France. This work was done while both authors were at the INRIA, Sophia-Antipolis and was supported in part by the ARC TéléGéo grant from INRIA.

References

[1] P. Alliez and M. Desbrun. Progressive encoding for lossless transmission of 3D meshes. In *SIGGRAPH'01 Conference Proceedings*, pages 198–205, 2001.

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

[3] C. Bajaj, V. Pascucci, and G. Zhuang. Progressive compression and transmission of arbitrary triangular meshes. In *Visualization'99*, pages 307–316, 1999.

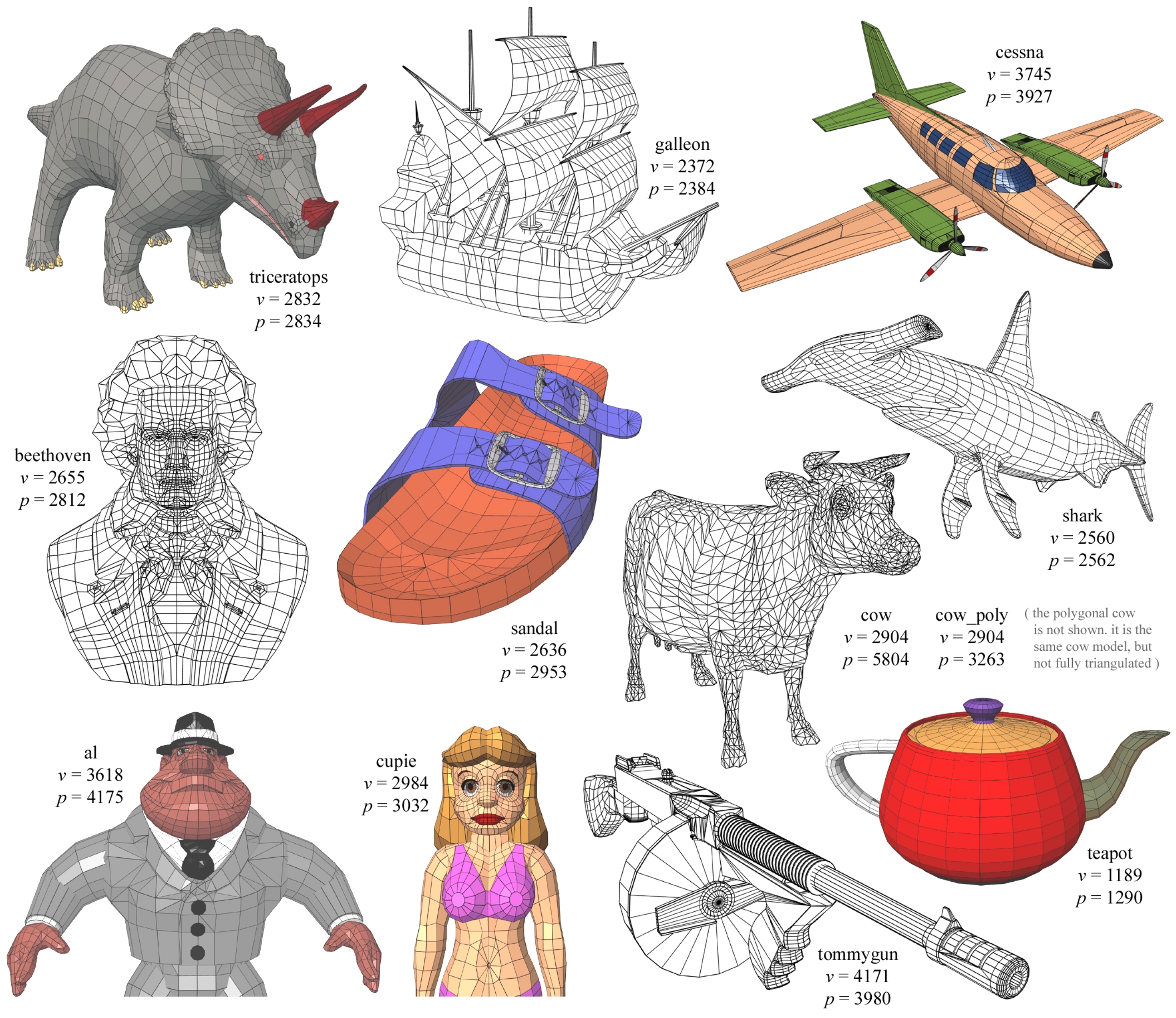
[4] C. Bajaj, V. Pascucci, and G. Zhuang. Single resolution compression of arbitrary triangular meshes with properties. In *DDC'99*, pages 247–256, 1999.

[5] D. Cohen-Or, R. Cohen and R. Irony. Multi-way geometry encoding. TR-2002.

[6] D. Cohen-Or, D. Levin, and O. Remez. Progressive compression of arbitrary triangular meshes. In *Visualization'99 Conf. Proc.*, pages 67–72, 1999.

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

[8] O. Devillers and P.-M. Gandoin. Geometric compression for interactive transmission. In *Proc. of IEEE Visualization 2000*, pages 319–326, 2000.



mesh	number of		holes, handles			mesh	connectivity			geometry (8 bits)			geometry (10 bits)			geometry (12 bits)		
name	vertices	& polygons	& components			name	TG	IA	gain	TG	IA	gain	TG	IA	gain	TG	IA	gain
triceratops	2832	2834	–	–	1	triceratops	767	421	45	2990	2362	21	4936	3798	23	7095	5226	26
galleon	2372	2384	–	–	12	galleon	619	621	0	3666	2555	30	5436	3920	28	7396	5470	26
cessna	3745	3927	–	–	11	cessna	1165	1191	-2	3776	2269	40	6075	3824	37	8943	5856	35
beethoven	2655	2812	10	–	8	beethoven	793	698	12	3589	3247	10	5607	5119	9	8072	7106	12
sandal	2636	2953	14	12	9	sandal	699	697	0	2996	2364	21	4648	3692	21	6709	5253	22
shark	2560	2562	–	–	1	shark	484	242	50	2345	1515	35	3859	2346	39	5703	3384	41
al	3618	4175	–	–	21	al	952	1099	-1	4732	3957	16	7413	6369	14	10344	8997	13
cupie	2984	3032	–	–	6	cupie	784	612	22	3003	2505	17	5017	4361	13	7315	6535	11
tommygun	4171	3980	–	6	39	tommygun	1077	1178	-9	4415	3076	30	7040	4653	34	10197	6517	36
cow	2904	5804	–	–	1	cow	682	647	5	3096	3244	-5	5153	5316	-3	7397	7487	-1
cow_poly	2904	3263	–	–	1	cow_poly	667	554	17	3114	2673	14	5178	4628	11	7417	6776	9
teapot	1189	1290	–	1	1	teapot	158	168	-6	1392	981	30	2209	1650	25	3127	2387	24
						average			10			24			24			23

Table 3: The table reports the number of vertices, polygons, holes, handles, and components for each mesh. The resulting compression rates in bytes for connectivity and geometry for the Touma and Gotsman coder (TG) and our coder (IA) are listed side by side. The coding gains of our coder over the TG coder are reported in percent. The results for geometry compression are given at three different precision levels of 8, 10, and 12 bits. Note that for purely triangular models (e.g. the cow) we get roughly the same bit-rates as the Touma and Gotsman coder. This validates that our improvement really comes from using the polygonal information.

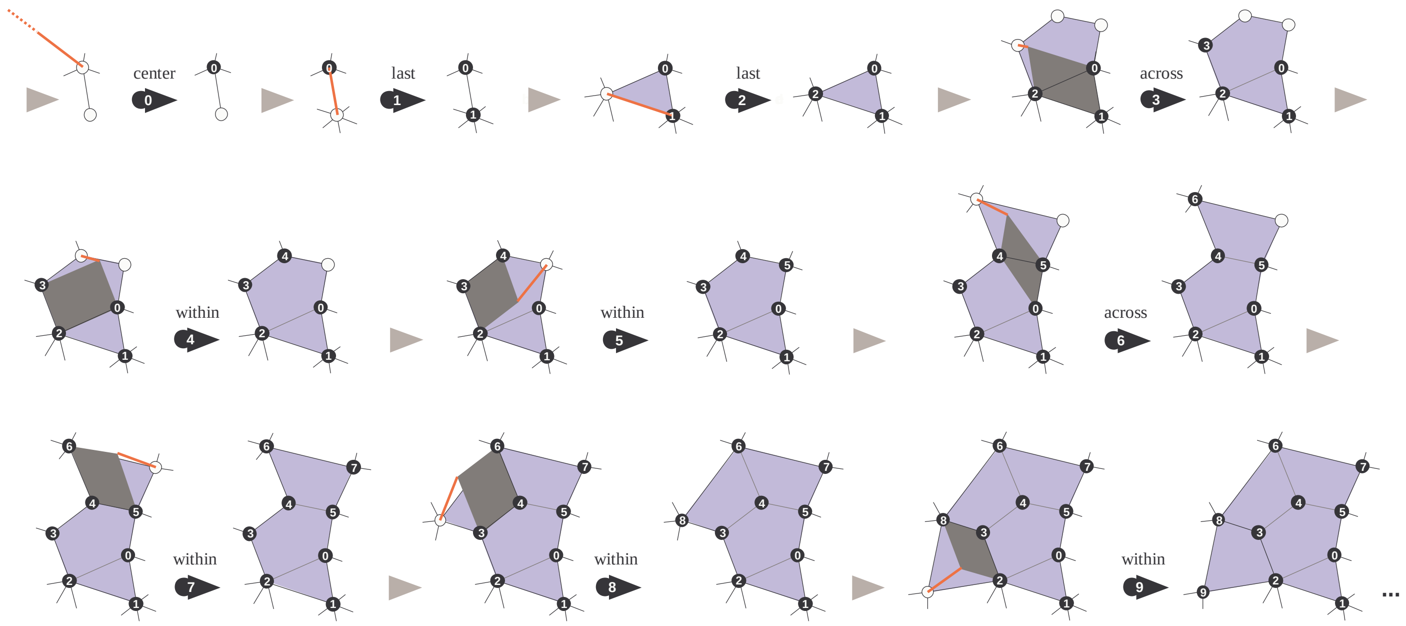


Figure 3: The decoding process is an exact replay of the encoding process. Here a small decoding example that demonstrates in which order the connectivity decoder traverses the vertices and which parallelogram predictions the geometry decoder uses to decode their position. The parallelogram used for prediction is shown in dark grey and the corrective vectors are shown in red. The light-grey arrows stand for one or more steps of the connectivity decoder. They are not important here (but are described in [12] using the same example). The black arrows with numbers denote a step of the geometry coder. They are described here: **(0)** The decoder predicts vertex 0 as the center of the bounding box. **(1)** The decoder predicts vertex 1 as vertex 0. **(2)** The decoder predicts vertex 2 as vertex 1. **(3)** The decoder *across*-predicts vertex 3 as completing the parallelogram spanned by vertices 0, 1, and 2. **(4)** The decoder *within*-predicts vertex 4 as completing the parallelogram spanned by vertices 0, 2, and 3. **(5)** The decoder *within*-predicts vertex 5 as completing the parallelogram spanned by vertices 2, 3, and 4. **(6)** The decoder *across*-predicts vertex 6 as completing the parallelogram spanned by vertices 5, 0, and 4. **(7)** The decoder *within*-predicts vertex 7 as completing the parallelogram spanned by vertices 5, 4, and 6. **(8)** The decoder *within*-predicts vertex 8 as completing the parallelogram spanned by vertices 6, 4, and 3. **(9)** The decoder *within*-predicts vertex 9 as completing the parallelogram spanned by vertices 8, 3, and 2. And so on ...

[9] O. Devillers and P.-M. Gandoin. Progressive and lossless compression of arbitrary simplicial complexes. In *SIGGRAPH'02*, pages 372–379, 2002.

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

[11] H. Hoppe. Efficient implementation of progressive meshes. *Computers & Graphics*, 22(1):27–36, 1998.

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

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

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

[15] Java3D. http://java.sun.com/products/java-media/3D/.

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

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

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

[19] A. Khodakovsky and I. Guskov. Normal mesh compression. preprint, 2001.

[20] A. Khodakovsky, P. Schroeder, and W. Sweldens. Progressive geometry compression. In *SIGGRAPH'00 Conference Proceedings*, pages 271–278, 2000.

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

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

[23] B. Kronrod and C. Gotsman. Optimized compression of triangle mesh geometry using prediction trees. In *Proceedings of 1st International Symposium on 3D Data Processing, Visualization and Transmission*, pages 602–608, 2002.

[24] E. Lee and H. Ko Vertex data compression for triangle meshes. In *Pacific Graphics'00 Conference Proceedings*, pages 225–234, 2000.

[25] H. Lee, P. Alliez, and M. Desbrun. Angle-analyzer: A triangle-quad mesh codec. to appear in *Eurographics'02 Conference Proceedings*, 2002.

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

[27] MPEG4. http://mpeg.telecomitalialab.com/.

[28] R. Parajola and J. Rossignac. Compressed progressive meshes. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):79–93, 2000.

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

[30] D. Shikhare, S. Bhakar, and S.P. Mudur. Compression of 3D engineering models using discovery of repeating geometric features. In *Proceedings of Workshop on Vision, Modeling, and Visualization*, 2001.

[31] 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.

[32] G. Taubin, A. Guéziec, W.P. Horn, and F. Lazarus. Progressive forest split compression. In *SIGGRAPH'98 Conference Proceedings*, pages 123–132, 1998.

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

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

[35] Viewpoint. *Premier Catalog (2000 Edition) www.viewpoint.com*.

[36] Virtue3D. http://www.virtue3d.com/.

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