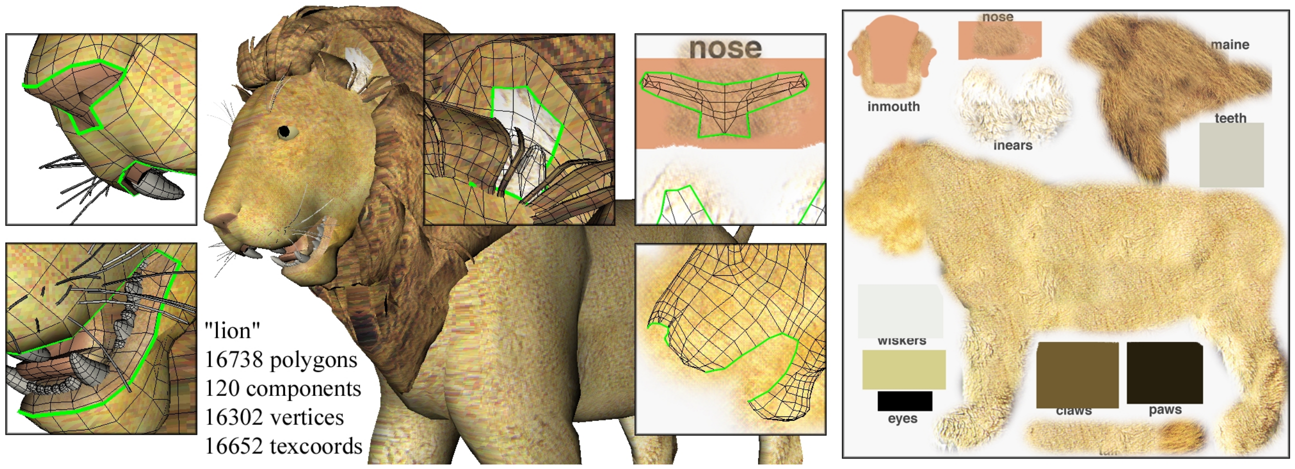
Compressing Texture Coordinates with Selective Linear Predictions

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



Abstract

In this paper we describe a strategy for efficient predictive compression of texture coordinates. Previous works in mesh compression often claim that this mesh property can simply be compressed with the same predictor that is already used for vertex positions. However, in the presence of discontinuities in the texture mapping such an approach results in unreasonable predictions. Our method avoids such predictions altogether. Rather than performing an unreasonable prediction, we switch to a less promising, but at least reasonable predictor. The resulting correctors are then compressed with different arithmetic contexts.

1. Introduction

Polygon meshes are the most widely used primitive for representing three-dimensional geometric models. They consist of mesh *geometry* and mesh *connectivity*, the first describing the positions of the vertices in 3D space and the latter describing how they are connected together into polygons. Often there are also mesh *properties* such as texture coordinates, colors, or normals that specify the visual appearance of the mesh surface.

The standard representation of a polygon mesh uses an array of floats to specify the vertex positions and an array of integers containing indices into the vertex array to specify the polygons. A similar scheme is used to specify the various properties and how they are mapped to the mesh. For large and detailed models this representation results in files of substantial size—hampering storage and transmission.

The need for more compact representations has motivated research on mesh compression and a number of efficient compression schemes [4, 22, 23, 7, 18, 13, 11, 8, 14, 15, 9] have been proposed. The majority of these techniques focus on only two aspects of mesh compression: coding the connectivity and coding the geometry. While these two are undoubtedly the basic ingredients of a polygon mesh, many polygonal datasets also contain properties. Typically their size is a significant portion of the mesh representation. The array of texture coordinates and the array of texture coordinate indices of the “lion” model shown above, for example, contribute about 45 % to the total file size.

The array of texture coordinate indices can be efficiently compressed with one of the proposed schemes for compressing property mappings [7, 21, 12]. The compression of texture coordinates, however, has received very little attention. Some papers on geometry compression [21, 2] suggest that texture coordinates could be compressed with the same predictive scheme that they propose for vertex positions, but give no further details and report no experimental results.

Although most position predictors are indeed well suited for compressing texture coordinates, the presence of discontinuities in the texture mapping can result in completely unreasonable predictions. The close-ups views of the “lion” model illustrate such discontinuities: Neighboring texture coordinates around the nose, the mouth, and the ear address distant locations in the texture image. Predictive schemes for compressing vertex positions assume that vertices that

[*]isenburg@cs.unc.edu http://www.cs.unc.edu/~isenburg/pmc
[†]snoeyink@cs.unc.edu

are *topologically close* are also *geometrically close*. This means, for example, that two vertices connected by an edge are assumed to have nearby positions in 3D space. The same assumption also holds for texture coordinates—unless there is a discontinuity. Instead of performing an unreasonable prediction near a discontinuity, our scheme switches to a less promising but at least reasonable predictor.

The most successful simple linear predictor for vertex positions is the *parallelogram rule* [23]. The position of a vertex is predicted to complete a parallelogram defined by the three vertices of a neighboring triangle. Intuitively it makes sense to apply the parallelogram rule also to texture coordinates. Usually the texture coordinates of two adjacent triangles also form two adjacent triangles in texture space. But if the edge connecting the two triangles coincides with a discontinuity in the texture mapping, the parallelogram rule gives a completely random prediction. In this case it is better to fall back to a simpler but meaningful prediction. It is crucial to the success of our approach to compress correctors resulting from the more promising parallelogram predictor with a different arithmetic context [24] than those resulting from less promising fall-back predictors.

2. Discontinuities in the Texture Mapping

Texture images are a simple way to increase the realism of polygonal meshes. The process of applying a texture image to a mesh is called *texture mapping*. It consists of putting every polygon of the 3D mesh into correspondence with a polygon in the 2D texture image. Although each polygon could be mapped independently, it is usually beneficial to map neighboring polygons in the mesh into neighboring polygons in texture space. The problem of finding a suitable mapping or *parameterization* for texturing a polygonal surface is a much studied problem [17, 20, 5, 16, 6, 19].

Vertices whose surrounding polygons are mapped into neighboring polygons in texture space appear at a single location in the texture image. They have a single texture coordinate that is used by all their surrounding polygons. In order to flatten a mesh without boundary or of non-trivial topology it is often cut open. Such cuts introduce discontinuities or *seams* in the texture mapping. Vertices along these seams appear at several locations in the texture image. Therefore they have multiple texture coordinates each of which is used by a subset of their surrounding polygons.

To minimize distortion in the texture-mapped image, the mapping between the polygons in 3D and the polygons in 2D is often sought to preserve angles and distances. Usually, it is impossible to flatten an entire mesh in one piece without creating overlapping or extremely distorted polygons. Distortion can be reduced by introducing additional cuts [6] or by cutting the mesh into several parts that are then parameterized separately [17, 20, 16, 19]. The latter results in the parameterization being broken up into several *charts*, which are then assembled into a single texture image called an *atlas* [17]. Both approaches create additional discontinuities in the texture mapping.

There are also other reasons to perform a piece-wise texture mapping: From an artist's perspective it is often convenient to cut a mesh into several parts in order to paint each of them separately, like it was done for the "cat" model from Figure 3. Also commercial software packages for automated generation of texture maps sometimes create a lot of seams. The "1510" model from Figure 3 was generated from a scan of an archaeological artifact. Its parameterization is broken into a many small squares that are tightly packed into a space-efficient atlas. This creates an immense number of discontinuities in the texture mapping.

In our experiments we use two sets of meshes with differently generated texture mappings. One set consists of hand-crafted polygon models of animals that were carefully textured by a skilled artist. Their texture mappings are relatively smooth with only a small number of seams that coincide with "natural" discontinuities. The other set consists of automatically generated triangle models of scanned archaeological artifacts that were textured using an automated method. Their texture mappings are not smooth at all.

3. Previous Work

Previous work in mesh compression has mostly focused on compressing the connectivity and the geometry of triangular [4, 22, 23, 7, 18, 2] or polygonal [11, 8, 14, 15, 9] meshes. There are only a few papers on compressing the texture coordinate mapping [7, 21, 11, 12] and even fewer mention texture coordinate compression [21, 2].

The common approach for compressing vertex positions first quantizes the floating point numbers uniformly and then applies some form of predictive coding. Quantization with k bits of precision converts each floating point number into an integer value between 0 and $2^k - 1$, which could then be stored using k bits. Predictive coding reduces the variation and thereby the entropy of the sequence of k bit numbers. Rather than specifying each coordinate individually, previously decoded information is used to predict the next coordinate and only a correcting term is stored. The simplest prediction method simply predicts the next position as the last position. This is called delta-coding and was first suggested by Deering [4]. Better methods are the spanning tree predictor by Taubin and Rossignac [22] and the parallelogram predictor introduced by Touma and Gotsman [23].

Most authors suggest to use their position predictors to also compress other mesh properties. Deering [4] proposes delta-coding for compressing quantized RBG colors. Similarly, Taubin et al. [21] apply their spanning predictor to compress various quantized mesh properties. Although they capture discontinuities in the property mapping with *discontinuity bits*, they do not use this information to prevent unreasonable predictions near discontinuities. Bajaj et al. [2] limit their linear predictor to meshes with perfectly smooth property mappings and perform all computations in spherical coordinates. For RGB colors quantized to 4, 6, and 8 bits per component they report rather small compression gains of 12, 11, and 10 %.

A completely different approach for texture coordinate compression was proposed by Sorkine and Cohen-Or [19]. They completely re-texture a mesh by computing a new piece-wise parameterization that is implicitly defined by the mesh and by warping the texture image accordingly. This eliminates the need to explicitly store the texture coordinates as they can be computed from the mesh. However, the requirement to warp the texture image makes this method unsuitable as a general purpose compressor.

Another approach that avoids explicit texture coordinates re-samples the mesh onto a regular grid in texture space [6]. However, this method should be thought of as a new and compact representation for geometric shapes rather than a polygon mesh compression scheme.

4. Preliminaries

The driving component of a typical mesh compression engine is the connectivity coder. The coder grows a region on the mesh by traversing its vertices and polygons with a fixed [22, 23, 7, 18, 11] or an adaptive [1, 8, 14, 15] strategy and records a stream of symbols from which the connectivity can be reconstructed. Vertex positions and other mesh properties are compressed in the order that the vertex, the face, or the corner they are associated with is encountered during this traversal. Special care needs to be taken when mesh properties are attached *per-corner*.

A corner is the point where a polygon connects to a vertex. A triangle, for example, has three corners, each of which connects to a different vertex. A vertex has as many corners as there are polygons connected to it. Around every vertex of a manifold mesh there is a consistent cycle of corners and edges, which we refer to as a *vertex ring*. In this paper we use a counterclockwise order to talk about a next/following and a previous/preceding edge or corner. After encountering a vertex ring through one of its edges there is always an unique traversal of the corners. The traversal starts with the corner following the respective edge and ends with the corner preceding it.

Texture coordinates are usually attached per-corner as this allows to specify discontinuities in the texture mapping. The corners around a vertex located in a smooth part of the texture map use a single texture coordinate. The corners around a vertex along a seam use multiple texture coordinates each of which is shared by a set of adjacent corners.

We say a corner is a *smooth corner* if it uses the same texture coordinate as the previous corner, otherwise we call it a *crease corner*. The edge preceding a smooth corner is a *smooth edge*. Consequently the edge preceding a crease corner is a *crease edge*. A vertex is a *smooth vertex* if all of its corners are smooth; otherwise it is a *crease vertex*. The classification of vertices and corners into *smooth* and *crease* is sufficient to encode the mapping from mesh corners to texture coordinates [12] as illustrated in Figure 1.

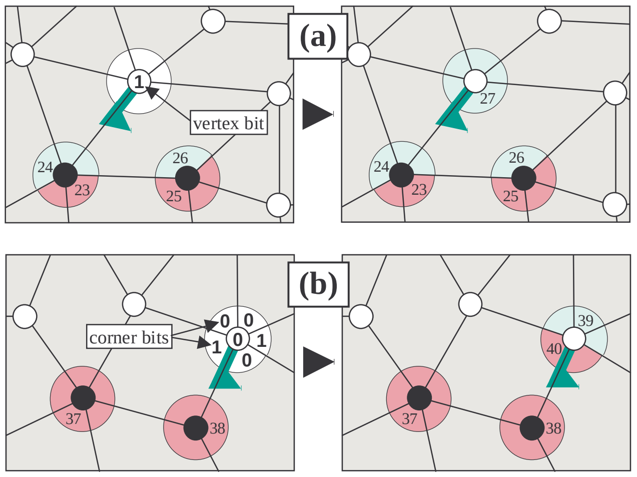


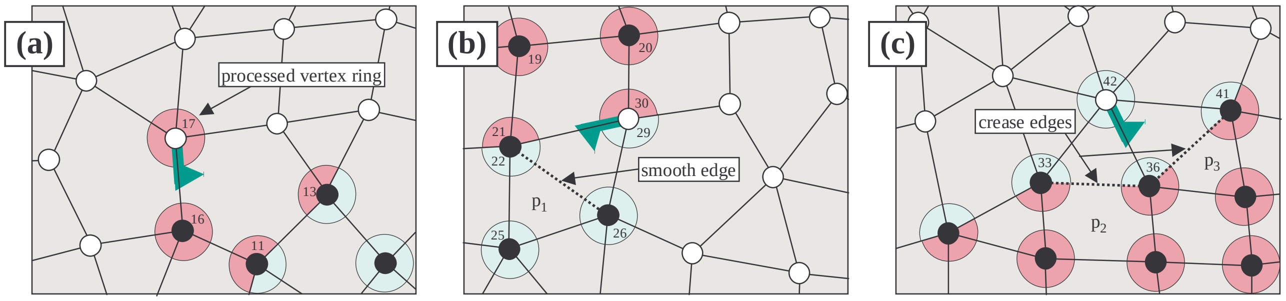
Figure 1. Coding with vertex and corner bits: **(a)** A vertex bit of 1 tells the decoder that it is a smooth vertex. It assigns the next texture coordinate to all its corners. **(b)** A vertex bit of 0 tells the decoder that it is a crease vertex. This means that the vertex uses two or more texture coordinates. Here the decoder has to read the corner bits. A corner bit of 1 tells the decoder to assign the next texture coordinate to this corner, otherwise it assigns the current texture coordinate.

5. Compressing Texture Coordinates

We compress both the texture coordinate mapping and the texture coordinates by processing the vertex rings in the order they are encountered by the connectivity coder. The texture coordinate mapping is coded with vertex and corners bits as illustrated in Figure 1, which are then compressed with the predictive scheme proposed in [12]. Subsequently we predict the texture coordinate(s) associated with the vertex ring using one of four prediction rules. The offset vectors that correct the predicted value to the actual value are then compressed with an arithmetic coder [24].

Whenever possible, we predict a texture coordinates using the parallelogram rule [23]. For polygonal meshes we try to perform the parallelogram prediction *within* a polygon instead of *across* two polygons. This was shown to improve the bit-rates when compressing vertex positions [9] and also works for texture coordinates. The intuition is that four texture coordinates from a single polygon are more likely to be in a parallelogram-shaped configuration in texture space than four texture coordinates from two adjacent polygons.

Parallelogram predictions across polygons are *only* used when the four texture coordinates belong to the same chart.



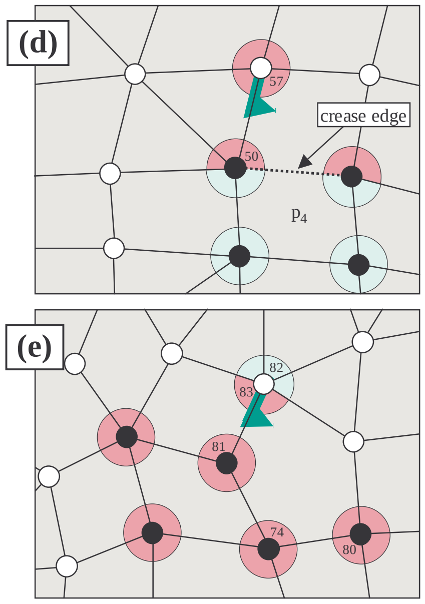


Figure 2. These five example scenarios illustrate the four different predictions that are used for compressing texture coordinates. The processed vertex ring and the edge through which it was encountered are marked with a green flag. Corners from the same vertex that are shaded with the same color use the same texture coordinate. Corners from different vertices that are shaded with the same color use a texture coordinate from the same chart. Corners that are shaded with different colors use texture coordinates from different charts. **(a)** a smooth vertex. Its texture coordinate T_{17} is used by four corners. It is *within*-predicted as $T_{16} - T_{11} + T_{13}$. **(b)** a crease vertex. Its texture coordinates T_{29} and T_{30} are each used by two corners. T_{29} is *across*-predicted as $T_{22} - T_{25} + T_{26}$ by involving polygon p_1, which connects to the vertex ring via a smooth edge. T_{30} is *within*-predicted as $T_{20} - T_{19} + T_{21}$. **(c)** another smooth vertex. Its texture coordinate T_{42} is used by six corners. It is *across*-predicted inside the vertex ring as $T_{33} - T_{36} + T_{41}$. The two *across*-predictions involving the polygons marked p_2 and p_3 are forbidden as they connect to the vertex ring via a crease edge. **(d)** again a smooth vertex. Its texture coordinate T_{57} is used by four corners. It is *nearby* predicted as T_{50}. An *across*-prediction involving the neighboring polygon marked p_4 is forbidden because of the crease edge. **(e)** a crease vertex. Its texture coordinates T_{82} and T_{83} are used by two and three corners respectively. T_{82} is *center*-predicted as the center of the texture image. There is no reasonable prediction for T_{82} because it could address any location in the image. T_{83} is *within*-predicted as $T_{81} - T_{74} + T_{80}$.

This is the case when the edge connecting the two polygons is smooth. Applying the parallelogram rule to texture coordinates from different charts (e.g. across a crease edge) would result in a completely random prediction. Instead of performing an unreasonable prediction we fall back to a less successful but at least reasonable predictor. We simply use a *nearby* texture coordinate from the same chart that is connected by an edge as the prediction. If there is no such texture coordinate, then no reasonable prediction is possible. In this case we predict it to lie in the *center* of the texture image. In Figure 2 the possible scenarios are illustrated.

It is crucial to the success of our method to compress the correctors with different arithmetic contexts [24] depending on which prediction was performed. Using a single context for all correctors would spoil the potentially low entropy of correctors that are result of more promising predictions with the anticipated poor outcome of the fall-back predictions.

In Table 1 we list the percentages with which the different prediction rules were used. Note that *within*-predictions can only occur in polygonal meshes. Furthermore the tables report separate bit-rates for the different prediction rules, which—as expected— confirm their varying success. In Table 2 we reports the performance of our compression scheme at different quantization levels. Notice that predictive compression does not scale with increasing precision. The achievable compression ratio is strongly dependent on the number of precision bits. Such a technique mainly predict away the high-order bits, so the relative compression ratios decrease if more precision (= low bits) is added.

6. Summary and Current Work

Taking into account discontinuities in the texture mapping, we have introduced simple prediction rules for efficient compression of texture coordinates. The average compression gain achieved by our method is 71 % for our polygonal and 50 % for our triangular test set for texture coordinates quantized at 10 bits of precision. To our knowledge this is the first work that rigorously investigates texture coordinate compression in a quantitative, in-depth manner.

The techniques proposed in this paper have been integrated into the benchmark compressor that we have recently made available [10]. We encourage the reader to visit the two web-pages that contain an interactive version of this coder together with all of the animal models[1] and all of the archaeological artifact models[2] used in this paper.

[1]http://www.cs.unc.edu/~isenburg/pmc/animals.html
[2]http://www.cs.unc.edu/~isenburg/pmc/artifacts.html

mesh	predicted [%]				bit-rate [bpt]			
name	within	across	nearby	center	within	across	nearby	center
lion	82	14	3	1	5.6	8.5	9.7	20.2
wolf	84	14	2	0.6	5.9	9.1	11.1	20.2
raptor	76	18	5	1	5.5	8.4	8.9	19.5
fish	90	10	0.3	0.1	6.6	10.6	14.6	20.1
snake	91	8	1	0.2	3.5	7.9	9.5	20.4
horse	86	11	3	0.5	4.3	7.4	10.1	20.4
cat	80	15	4	0.7	4.3	7.1	8.4	19.6
dog	59	37	4	0.7	6.2	7.2	10.0	20.6
average	**81**	**16**	**3**	**0.6**	**5.2**	**8.3**	**10.3**	**20.1**

mesh	predicted [%]				bit-rate [bpt]			
name	within	across	nearby	center	within	across	nearby	center
“1398”	–	55	35	10	–	7.5	9.8	18.9
“1412”	–	49	39	12	–	8.3	10.3	19.0
“1510”	–	53	36	11	–	7.3	10.7	18.3
“1568”	–	51	38	11	–	8.4	10.5	19.0
“17”	–	58	33	9	–	8.2	10.4	17.7
“1814”	–	52	38	10	–	7.5	10.7	15.9
“1823”	–	52	37	11	–	7.6	10.4	19.1
“2441”	–	49	38	13	–	8.3	10.2	19.2
average	**–**	**52**	**37**	**11**	**–**	**7.9**	**10.4**	**18.4**

Table 1. These tables report which percentage of texture coordinates is predicted *within* a polygon, *across* polygons, as a *nearby* texture coordinate, and as the *center* of the bounding box. The corresponding bit-rates at a precision of 10 bits confirm the different success of these predictions.

mesh characteristics				8 bit		10 bit		12 bit	
name	c	v	t	bpt	gain	bpt	gain	bpt	gain
lion	120	16302	16652	3.8	77	6.3	69	9.7	60
wolf	35	7068	7234	3.8	76	6.6	67	10.3	57
raptor	79	7454	6984	3.7	77	6.3	68	10.0	58
fish	7	4685	4685	4.2	73	7.0	65	10.7	55
snake	6	11137	11610	2.3	85	3.9	80	6.5	73
horse	5	9199	9988	2.9	82	4.9	76	8.2	66
cat	39	9627	10350	3.0	81	5.0	75	8.2	66
dog	19	6650	6522	3.9	76	6.8	66	10.6	56
average				**3.5**	**78**	**5.9**	**71**	**9.3**	**61**

mesh characteristics				8 bit		10 bit		12 bit	
name	c	v	t	bpt	gain	bpt	gain	bpt	gain
“1398”	1	1487	3133	6.3	61	9.5	53	13.5	44
“1412”	1	1180	2712	7.0	56	10.4	48	14.4	40
“1510”	1	644	1422	6.7	58	9.9	51	13.8	43
“1568”	1	1394	2999	7.2	55	10.5	48	14.4	40
“17”	1	1178	2354	6.6	59	9.9	51	13.8	43
“1814”	1	1145	2475	6.4	60	9.6	52	13.4	44
“1823”	1	1202	2700	6.8	57	10.0	50	14.0	42
“2441”	1	1204	2727	7.0	56	10.5	48	14.5	40
average				**6.8**	**58**	**10.0**	**50**	**14.0**	**42**

Table 2. These tables list for each model the number of connected components c, of vertices v, and of texture coordinates t. The achieved compression rates in bits per texture coordinate (bpt) are given at the three common quantization levels of 8, 10, and 12 bits and the compression gain (%) in comparison to uncompressed texture coordinates is reported. The bit-rate for uncompressed texture coordinates is simply the number of quantization bits times two.

A similar technique can also be used to compress RGB colors. However, on our test set of colored meshes the parallelogram predictor continuously “over-shot” the variation in color and was outperformed by the simpler nearby predictor. We achieved the best rates of 5.7 (11.4) bits per color at quantization levels of 4 (6) bit per color component by using an average of all possible nearby predictions—this equals a compression gain of 52 (37) % respectively.

Currently we are investigating an approach for predicting texture coordinates from vertex positions. Many techniques for (semi-)automatic texture map generation take mesh geometry into account to compute texture coordinates that minimize some distortion metric. The correlation between vertex positions and texture coordinates is high when shape preserving metrics are used that minimize geometric stretch [20], angle distortion [16], or other intrinsic measures [5]. Sorkine and Cohen-Or [19] establish complete correlation between the two, because they define the texture coordinate mapping through the mesh geometry.

Instead of using mesh geometry to define the texture coordinates, we can use it to predict them. This predictor works best if the shape of a polygon in 3D space is similar to its shape in texture space. Although initial results are promising we have to evaluate if the achievable gains are always worth the additional computational effort. For example *space-optimized* texture maps as proposed by Balmelli et al. [3] can have a fairly low correlation between texture coordinates and vertex positions.

Acknowledgments: We thank Curious Labs for the animal models and the Archaeology Technologies Laboratory[*] at North Dakota State University for the models of archaeological artifacts.

[*]http://atl.ndsu.edu/archive/

References

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

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

[3] L. Balmelli, G. Taubin, and F. Bernardini. Space-optimized texture maps. In *Eurographics’02 Conference Proceedings*, pages 411–420, 2002.

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

[5] M. Desbrun, M. Meyer, and P. Alliez. Intrinsic parameterisations of surface meshes. In *Eurographics’02 Conference Proceedings*, pages 209–218, 2002.

[6] X. Gu, S. Gortler, and H. Hoppe. Geometry images. In *SIGGRAPH’02 Conference Proceedings*, pages 355–361, 2002.

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

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

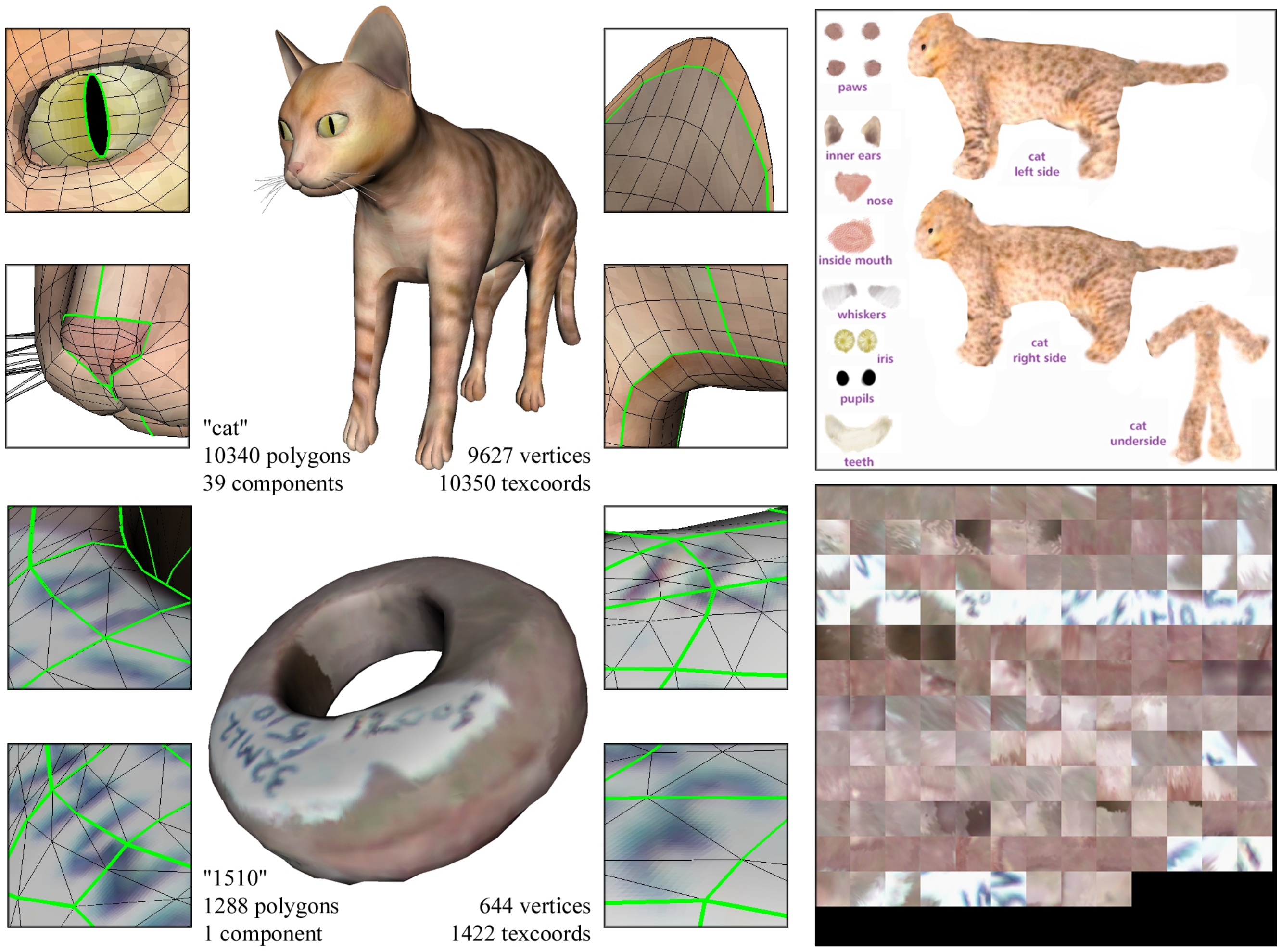


Figure 3. The closeup views show the texture mapping discontinuities of the "cat" model from the animal set and the "1510" model from the archaeological artifact set. These can also be inferred from their corresponding texture atlas.

[9] M. Isenburg and P. Alliez. Compressing polygon mesh geometry with parallelogram prediction. In *Visualization'02 Conference Proceedings*, pages 141–146, 2002.

[10] A benchmark coder for polygon mesh compression. *http://www.cs.unc.edu/˜isenburg/pmc/*

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

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

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

[14] A. Khodakovsky, P. Alliez, M. Desbrun, and P. Schroeder. Near-optimal connectivity encoding of 2-manifold polygon meshes. In *Graphic Models*, 64(3-4):147–168, 2002.

[15] H. Lee, P. Alliez, and M. Desbrun. Angle-analyzer: A triangle-quad mesh codec. In *Eurographics'02 Conference Proceedings*, pages 383–392, 2002.

[16] B. Levy, S. Petitjean, N. Ray, and J. Maillot. Least squares conformal maps for automatic texture atlas generation. In *SIGGRAPH'02 Conf. Proceedings*, pages 362–371, 2002.

[17] J. Maillot, H. Yahia, and A. Verroust. Interactive texture mapping. In *SIGGRAPH'93 Proc.*, pages 27–34, 1993.

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

[19] O. Sorkine, D. Cohen-Or, R. Goldenthal, and D. Lischinski. Bounded-distortion piecewise mesh parameterization. In *Visualization'02 Conf. Proceedings*, pages 355–362, 2002.

[20] P. Sander, J. Snyder, S. Gortler, and H. Hoppe. Texture mapping progressive meshes. In *SIGGRAPH'01 Conference Proceedings*, pages 409–416, 2001.

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

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

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

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