Lossless Compression of Adaptive Multiresolution Meshes

Felix Kälberer, Konrad Polthier, Christoph von Tycowicz
Freie Universität Berlin, Germany
{*Felix.Kaelberer, Konrad.Polthier, Christoph.vonTycowicz*}*@fu-berlin.de*

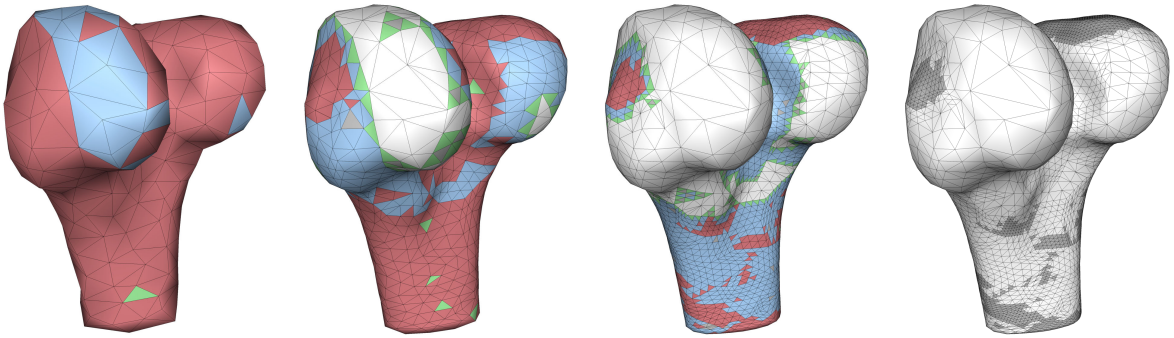


Figure 1. Adaptive refinement of a bone model. Elements are colored according to our coding scheme. We store a bit for every blue and red triangle, specifying whether it is further refined or not. These bits are sufficient to reconstruct the connectivity of the model. All other triangles can be reconstructed using rules of the refinement scheme as explained in the text.

***Abstract*—We present a novel coder for lossless compression of adaptive multiresolution meshes that exploits their special hierarchical structure. The heart of our method is a new progressive connectivity coder that can be combined with leading geometry encoding techniques. The compressor uses the parent/child relationships inherent to the hierarchical mesh. We use the rules that accord to the refinement scheme and store bits only where it leaves freedom of choice, leading to compact codes that are free of redundancy. To illustrate our scheme we chose the widespread red-green refinement, but the underlying concepts can be directly transferred to other adaptive refinement schemes as well. The compression ratio of our method exceeds that of state-of-the-art coders by a factor of 2 to 3 on most of our benchmark models.**

***Keywords*-multiresolution; subdivision surfaces; level-of-detail; connectivity coding; lossless; progressive mesh coding; 3D mesh compression**

I. Introduction

Multiresolution meshes, *i. e.*, meshes obtained through successive subdivision of a coarse base complex, are commonplace in a variety of areas such as movie industry, computer aided design (CAD) and numerical simulations. The computing power of todays computer systems and the availability of advanced modeling software make it easy to generate grids with up to several million vertices. Storing those meshes in a raw data format is notoriously expensive due to the sheer amount of data; this is where compression comes into play.

Adaptive refinement, *i. e.*, the introduction of detail only where it is needed, is an essential strategy to master the processing, rendering, transmission and storage of such meshes. For uniform refinement, the connectivity of the mesh can be represented by just the base complex and the number of subdivision levels. In contrast, Adaptively refined meshes exhibit a non-trivial hierarchical structure. To the best of our knowledge, the lossless compression of adaptive triangle hierarchies has not been researched into before.

Generally compression comes in two stages: First a lossy stage, where essential information of the input is extracted and negligible data is dropped. Second, the data decimation is generally followed by lossless encoding, in which the remaining data is transcoded into a compact byte stream, typically using entropy coding like Huffman or arithmetic coding.

In view of mesh coding, the mesh data consists of connectivity, geometry, and possibly attribute data such as colors and texture coordinates. 3D mesh coders are often referred to as *lossless* if they preserve the original connectivity of the mesh, even if floating point data of coordinates and attributes are truncated to a fixed precision. This tolerance within the ”lossless” category may be due to the fact that geometry data will never be free from errors, and errors introduced by truncating the least significant bits of a `float` value are often negligible compared to noise, discretization, and quantization errors during mesh acquisition.

Lossy mesh coders consider the connectivity of the mesh as auxiliary information that does not contribute to the shape of the model. In the same manner, tangential positions of vertices within the surface are regarded as negligible. The intention of those coders is usually the best reproduction of the shape with respect to some distance norm within a given byte limit. The mesh is often remeshed to a semi-

regular mesh that allows wavelet analysis to compress the geometry data.

We consider connectivity compression a vital issue since the outcome of many algorithms from geometry processing and numerics depend on exact reproduction of connectivity, think of animation or morphing. In our work we assume that the data reduction has already taken place. Our input models are hierarchical models that are adaptively refined. We assume that somebody has carefully selected important details and pruned the negligible ones by some criterion, be it by preservation of shape, normals, visual impact, or by numerical criteria. The use of a lossy black box encoder is prohibitive if no further detail should be lost. Such situations arise for example in numerical simulations where several frames with varying refinement structure have to be stored. In this case, the base mesh is stored just once, with different refinement stages for each time step. The storage of view-dependent refinements of objects in virtual environments creates a similar use case.

Related Work Numerous compression schemes for triangle meshes have been developed for single-rate coding (compressing the whole mesh in a region-growing fashion) as well as progressive coding (encoding the model from coarse to fine). On the single-rate side Edgebreaker [1] and the method of Touma-Gotsman [2] are the most prominent coders which have spawned a wealth of variants and improvements. Among the best-performing variants for connectivity compression is the early-split coder of Isenburg and Snoeyink [3] and the optimized Edgebreaker encoding of Szymczak [4]. These coders profit from mesh regularity and are able to push the bit rate well below the Tutte limit [5] of roughly 3.24 bits per vertex.

The FreeLence algorithm [6] exploits correlation between connectivity and geometry by accessing already encoded geometry data when encoding connectivity and vice versa, allowing it to push the bit rates well below that of [3] and [4].

For progressive transmission, models are often simplified or remeshed [7], [8] to generate such a simple base mesh from arbitrary connectivity meshes. In this context wavelet coding approaches seem to perform best [9], however, these coders only aim at the compression of the geometry and do not allow lossless reconstruction of the connectivity even for adaptive multiresolution meshes. (Although the normal mesh compression by Khodakovsky *et al.* [10] is able to generate such meshes, the adaptivity is controlled by the coder thus neglecting any original criteria.) There are lossless progressive coders as well [11], [12], but their compression rates usually stay below those of the best single-rate coders. The interested reader is referred to [13] for a broader survey of 3D compression schemes.

Contribution Our major contribution is a connectivity compression scheme that is adapted to the special characteristics of adaptive multiresolution meshes. We convert the tree-like hierarchical structure to a binary stream, and use the refinement scheme's rules, to prune redundant bits. This alone nearly halves the data size compared to a naïve coding of the structure. With context based arithmetic coding and an adapted traversal of the mesh we can furthermore take advantage of structural regularities that are typically present in real-world data. In combination, these measures push the data rates to significantly below those of general triangle mesh encoders, outperforming state-of-the-art by a factor of 2 to 3.

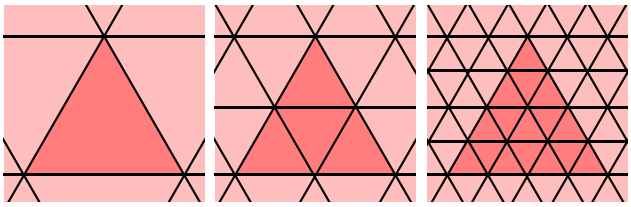


Figure 2. Triangle after one and two iterations of uniform 1-to-4 refinement.

We designed our coder with compatibility, extensibility, and scalability in mind. The compression scheme can be used in conjunction with any of the recent methods for geometry prediction [6], [14], data quantization [15], and bit allocation [16]. Our scheme is fast and can be implemented to run in linear time. The code is progressive, *i. e.*, after decoding the base mesh, the overall shape is reconstructed, and further details will be added from coarse to fine level.

Even though we illustrated the scheme for red-green refinement, the methods are not restricted to those meshes. The extension to adaptive quadrilateral meshes is straightforward. Even exploiting the similarities of adjacent frames in time-varying mesh sequences merely requires slightest changes to our methods.

II. Hierarchy coding

In this section we explain how we encode the hierarchical structure of adaptive multiresolution meshes and therefore its connectivity. First we will explain the concepts of red-green refinement, before we outline our encoding procedure in Section II-B. Sections II-C to II-E then elaborate the details.

A. Red-Green Refinement

Many refinement schemes, such as butterfly [17] or Loop [18] subdivision, are based on the dyadic split operation where each triangle is divided into four congruent subtriangles. First new vertices are introduced at each edge midpoint, dividing the edges into two. Connecting the three edge midpoints within each triangle then splits the triangle into four, see Fig. 2. Therefore, the scheme is often referred to as 1-to-4 split as well. Commonly, vertices inserted during the refinement step are denoted as odd vertices whereas vertices that have been present prior to refinement are termed even. We will refer to subtriangles formed entirely by odd

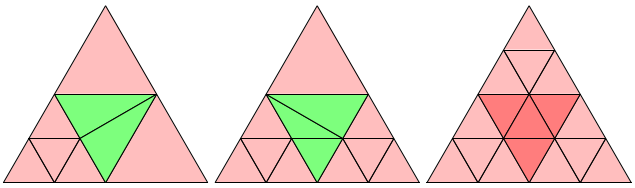


Figure 3. Conformization of one, two and three hanging nodes. In case of three hanging nodes the 1-to-4 split is applied. Up to symmetry, these are the only non-conforming situations that can occur in *balanced meshes*.

vertices as *center* triangles and those containing an even vertex as *outer* triangles.

Unlike *global mesh refinement* where all elements in the mesh are refined to obtain a finer mesh, *adaptive* or *local mesh refinement* performs the split operation only for selected elements. Care must be taken at the transition between elements of different refinement levels, since a new vertex is inserted upon an edge of the refined element, but the coarse neighboring triangle still contains the edge undivided. These irregular vertices are called *hanging nodes*. To remove the non-conforming situations between elements of various refinement grades, the adjacent unrefined triangles must be refined too. To maintain locality of the conformization, non-conformal elements are bisected (see Fig. 3). In the finite element community these temporal elements are called *green* triangles whereas elements that are generated by the 1-to-4 split are called *red* triangles. Hence the name *red-green refinement*. Since green triangles are less shape regular than their parents the adaptive refinement is usually restricted to *balanced meshes* where the refinement level of neighboring elements must not differ by more than one level. This bounds the number of hanging nodes on each edge to one, preventing the bisection of green elements and therefore ever thinner triangles.

The refinement strategy yields a canonical hierarchical structure where each quadrisected element acts as a parent for the four new triangles. Therefore each triangle of the coarse base mesh will have an associated quad-tree that specifies its refinement. We will write *node* to refer to the entities of the quad-trees. This underlines parental relationships between triangles at different resolutions of the mesh. All refinement trees together form a forest whose leaf nodes either represent a red triangle or a pair or triplet of green triangles. We choose to associate the fourth child of a node with the center triangle introduced during refinement of its related triangle. The outer triangle incident to the parents first vertex will be the first child, and so forth. Fig. 4 shows three root triangles with associated refinement trees as well as the represented adaptive triangulation.

B. Encoding

Akin to existing progressive coders we separately encode the base domain from the hierarchy. Typically the root

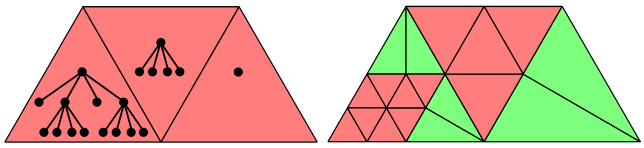


Figure 4. Red-green refined mesh and its corresponding hierarchy trees.

triangulation is described by a small, carefully laid out mesh that can be compressed exceedingly well using single-rate coders. Our prototype uses the state-of-the-art single-rate coder FreeLence [6] to losslessly encode its connectivity and geometry. This compression will in general alter the order of the base domain triangles as well the as their local indexing, *i. e.*, the ordering of references to vertices. To ensure the compatibility of the refinement hierarchy, the root triangulation and its associated refinement forest is matched to the reconstructed deterministic output of the FreeLence decoder so that refinement trees can be bijectively mapped to root elements without coding of further information.

Starting from the base domain, encoding the refinement hierarchy is sufficient to reconstruct the connectivity of the mesh at any level of detail. Provided that encoder and decoder agree on a common node traversal strategy, the refinement hierarchy can be stored with one bit per node where each bit specifies whether the node has children (is refined) or not. The only exception is caused by triangles with two hanging nodes, see Fig. 3, middle. The second possibility to resolve the non-conforming situation arises by flipping the diagonal edge. In this case we need to store one additional bit. In practice, the direction of the diagonal edge is often determined exclusively by local indexing. In this case, the extra bit can be omitted if the coder uses the same scheme that was used during mesh creation.

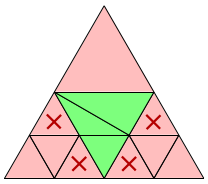
Due to the deterministic conversion of the hierarchical structure in to a bit stream we can estimate an upper bound for the code size. The number of leafs in the hierarchy is no greater than the number f of triangles of the finest resolution. Moreover, four nodes share a common parent node. This bounds the number of code symbols to $f+\frac{1}{4}f+\frac{1}{16}f\ldots<\frac{4}{3}f\approx\frac{8}{3}v$ where v the number of vertices. This also holds if we have conformizations of triangles with two hanging nodes. In this case we have to store one extra bit, but in fact we counted three green triangles in f that where represented by just one node in the hierarchy tree.

To maintain the progressivity of the generated bit code we traverse the hierarchy breadth-first, so that we successively visit the nodes at a certain depth in all trees, before switching to the next finer level. Finally the generated bit stream is entropy encoded. In the following sections the algorithm is explained in more detail.

C. Redundant Symbols

We converted the refinement structure into a compact bit stream. Nevertheless the knowledge of the structure can be used to further improve the hierarchy compression by culling out nodes from the bit stream whose state can be implicitly reconstructed. Because the hierarchy is directly related to the mesh, the mesh constraints implied by the refinement scheme instantaneously affect the hierarchy's structure. These dependencies are exploited by the following extensions:

1-REGULARITY As mentioned before, red-green refinement produces balanced meshes. There will be at most one hanging node per side of an element. Moreover, since the nodes of the hierarchy are conquered level-wise, we already know whether the neighbors of the node in question are green triangles that resolved a non-conforming situation in the parent level. As a consequence, nodes representing triangles adjacent to coarse green elements can not be refined. Hence, they can be skipped by the encoder.



Marked triangles can not be further refined due to green triangles in the parent level.

HANGING NODES If all three neighbors of the current node are already encoded, the total number of hanging nodes within the element is known to the decoder. In case of three hanging nodes, the decoder can immediately perform a dyadic split. Consequently no symbol will be stored. Some implementations of red-green refinement also prohibit two hanging nodes so that those symbols could be culled out from the output as well. Anyhow, this case will be handled by our coder without overhead, *cf.* Section II-D.

UNIFORM REFINEMENT Uniformly refined meshes exhibit a featureless hierarchical structure—the whole forest can be described by one single scalar that specifies the overall height of the refinement trees. Because many meshes in practice are uniformly refined to a certain degree, we exploit this property to reduce the number of code symbols. We store a single byte encoding the degree of uniform refinement separately, allowing the coder to skip all nodes on coarser levels.

STREAM TRUNCATION An interesting observation is that decoding a 0 from the stream has no effect on the hierarchy while a 1 causes a refinement of the current node (or its associated triangle, respectively). As the refinement forest is conquered in a breadth-first manner, nodes at the finest level are visited last, thus concluding the output stream. These are all 0 entries and are only needed for *closing* the forest, *i. e.*, generating a valid hierarchical structure. This structure can be constructed by the decoder without using these entries. Therefore the encoder omits the finest nodes from the output and even truncates 0's written after the last 1 as these can be implied. The decoder thus simply skips nodes for which no bit was stored (*i. e.*, the code contains no further symbols that can be read).

Encoding the degree of uniform refinement in combination with the omission of trailing zeros guarantees that not a single symbol ends up in the output when a uniformly refined mesh is compressed. The results on the number of symbols that have to be coded for our benchmark models are shown in Table 1. Among the described optimizations *stream truncation* and *1-regularity* perform best and contribute most to the reduction of symbols. The effect of *hanging nodes* is rather small since the degree of uniform refinement is stored and the case where all three neighbors are know arises only rarely (see Section II-E for more details). The same holds for the *uniform refinement* as this optimization only affects the coarsest levels containing exponentially few nodes. After all, the number of symbols that have to be coded averages out at nearly half the number of nodes.

D. Context Groups

With the steps in the last section we used the rules of the refinement scheme to eliminate code symbols in cases where the refinement scheme leaves no room for choice. The steps above reduce the binary representation of the refinement tree to a compact, redundancy free representation, without even looking at the characteristics of the particular input model. Models that appear in practice, however, *do* show certain characteristics. Just like two adjacent pixels in a digital photograph are likely to be similar, the refinement grades in hierarchical meshes typically tend to be locally similar.

Luckily, our forest structure admits the definition of neighbors, which lets us easily determine the effects of locality. We call two nodes within one level of the hierarchy *adjacent*, if their triangle counterparts in the mesh of that level share an edge. Due to the locality of the refinement depth, the split information of two adjacent nodes is highly correlated, so the refinement state of the neighbor node is a valuable hint. For instance, 23 thousand of the 96 thousand nodes of the fandisk model have children, which gives each hierarchy triangle the probability of 24 % of being split. Given the knowledge that a particular neighbor is a leaf, the probability of being subdivided drops to 7 %. If all three direct neighbors are leafs, that probability is a mere 1.2 %.

Let X be the binary stream of split data. As shown by Shannon [19], the *entropy* $H(X) = \sum_{i=0}^{1} -p(i)\log(p(i))$, measured in bits per symbol, is the information content of X. It poses a lower bound for the code length of any encoding of the message X, where $p(0)$ and $p(1)$ are the probabilities of X being 0 or 1, respectively. Good entropy coders, such as arithmetic coding [20], approach this lower bound in the limit and are thus optimal in the limit sense.

If we do have additional information about X, for instance the state of the neighbor elements, the code length can be reduced. Let Y be an information source that is correlated

model	f	#nodes	drop 0s	hanging	1-regul.	uniform	#left	code size
bones	5622	5438	8%	0.0%	16%	0.0%	76% (4123)	1496 (1912)
bunny	90771	103002	28%	0.1%	22%	0.2%	50% (51369)	28800 (47992)
fandisk	86092	95532	34%	0.1%	23%	0.0%	42% (40518)	23416 (38520)
feline_gauss	199008	229924	43%	0.2%	20%	0.2%	37% (85486)	48112 (74232)
feline_mean	254044	303072	38%	0.1%	16%	0.8%	46% (137906)	62040 (128608)
femur	8944	10608	27%	0.2%	13%	0.0%	60% (6361)	2440 (4224)
heat_transfer	96412	115034	1%	0.1%	15%	0.6%	84% (96242)	28056 (74144)
horse	96368	113228	37%	0.1%	18%	0.2%	45% (51481)	25792 (47392)
rabbit	68506	78570	23%	0.1%	21%	1.3%	55% (43261)	21592 (40544)
venus	138672	154792	40%	0.2%	24%	0.0%	36% (55812)	36360 (47352)
average	104444	120920	28%	0.1%	19%	0.3%	53% (57256)	27810 (50492)

Table 1. Removal of redundant symbols. Column 2 contains the number of faces and column 3 the number of tree nodes, *i. e.*, the number of binary decisions the decoder has to make. Columns 4 to 7 list percentage of bits that can be omitted by dropping trailing zeros, forbidding 3 hanging nodes, exploiting 1-regularity, storing the number of levels of uniform refinement. Column 8 list the percentage and actual number of bits that have to be stored, and the last column shows the size of the compressed code in bits, with (and without) the use of context groups.

to X (in our case $y \in Y$ describes a particular configuration of refinement states of the neighbor elements). The amount of information that actually has to be stored is measured by the conditional entropy

$$H(X|Y) = \sum_{y \in Y} p(Y{=}y) \sum_{i=0}^{1} -p(i|Y{=}y) \log p(i|Y{=}y),$$

that is, the amount of *new* information in X, given that we already know Y. If X and Y are correlated, $H(X|Y)$ is strictly less than $H(X)$.

In our implementation we use *context groups* as a simple measure to profit from the correlation of hierarchy elements. Recall that we specify with one bit whether the current node is refined as we traverse the nodes in the trees level by level. Whenever a binary code symbol is produced, we check the status of the neighbors. If the neighbor has already been visited during traversal, its refinement status will be known to the decoder. Thus we can use the refinement status of already processed neighbors as for the definition of contexts: a neighbor can either be *refined* ($\bigtriangleup$), *not refined* ($\bigtriangledown$), or it has not been processed before (**?**).

The statuses of the neighbor triangles define the context group in which the symbol is encoded. We write symbols of different context groups in separate arrays, which are entropy coded independently. With arithmetic coding, each context group y will compress to

$$H(X|Y{=}y) = \sum_{i=0}^{1} -p(i|Y{=}y) \log(p(i|Y{=}y))$$

in the limit. The total code size per symbol results by averaging the entropies individual contexts, weighted by their respective probabilities,

$$\sum_{y \in Y} p(Y{=}y) H(X|Y{=}y) = H(X|Y),$$

which proves that contexts are an appropriate tool to capture *all* the mutual information that is inherent in the correlation of neighboring elements.

So far we have not specified, how exactly we define the contexts. The contexts arise from the number of $\bigtriangleup$, $\bigtriangledown$, and **?** neighbors of a hierarchy node. *Counting* the respective neighbor states is enough to capture substantially different situations, since neighbor configurations with the same number the respective states differ at most by cyclic permutation or reversal. We write (x, y, z) to denote the context with x $\bigtriangleup$ situations, y times $\bigtriangledown$, and z times **?**. Of the ten possible contexts we can eliminate the context $(3, 0, 0)$ as we never have to store a symbol in this context: if all three neighbors are split, then the enclosed element *must* be split too, as no three hanging nodes in a triangle are allowed. The nine remaining cases are listed in Table 2.

E. Traversal Order

Grouping of the refinement information in separate contexts can immensely improve the efficiency of the entropy coder, see Section II-D. In this section we review the compression rates within the single context groups and discuss the impact of the hierarchy traversal strategy on them.

The breadth-first traversal of the hierarchy is an important factor in our coder. Yet it leaves freedom to choose any iteration of nodes within each level. This choice directly effects the distribution over the context groups, as the context of a node solely depends on the refinement state of its neighbors, and therefore on the fact whether these have been already visited or not.

A customary traversal scheme would visit the children of each node in a fixed ordering. Table 2 shows the symbols distribution in each context group for one of our test models (which is close to the average of our test set). Here *naïve traversal* refers to a strategy where children are visited in order. Since in our implementation center triangles are specified as the fourth child of its parent, outer triangles are visited first. Hence context group (0,0,3), where none of the three neighbors are known, contains most entries. This group, though, is virtually incompressible as no advantage can be taken of mutual information. The same holds for

Group △,△,?	Naïve traversal #symb.	%1's	bits/symb.	Improved traversal #symb.	%1's	bits/symb.
(2, 1, 0)	5913	81%	0.69	161	69%	1.04
(2, 0, 1)	14130	99%	0.05	29098	96%	0.21
(1, 2, 0)	5603	36%	0.96	128	26%	1.00
(1, 1, 1)	6141	41%	0.99	13466	56%	0.99
(1, 0, 2)	18752	93%	0.35	34718	89%	0.47
(0, 3, 0)	13606	1%	0.12	681	10%	0.52
(0, 2, 1)	12657	6%	0.34	32371	4%	0.29
(0, 1, 2)	16355	8%	0.43	27282	14%	0.60
(0, 0, 3)	33413	52%	1.00	1	100%	8.00
culled	176502		0	165166		0
total	303072		0.23	303072		0.20

Table 2. Population of the context groups for the naïve and improved traversal strategy on the *feline_mean* model. For each traversal strategy we provided the number of symbols in each context group, the percentage of 1's among those symbols, and the compression efficiency in terms of bits per symbol.

(1,1,1), (1,2,0), and (2,1,0) where the extra information is rather ambiguous. Contrary, the other context groups perform very well but are less populated.

We designed a new traversal scheme (*Improved traversal* in Table 2) to redistribute the symbols and maximize the overall compression. Instead of ordering the children in a fixed manner we first iterate over every tree and collect all nodes at that certain depth. This allows a global optimization of the level-wise node traversal.

The principle of our algorithm is to maximize the mutual information that can be exploited for encoding each node. For that purpose we prioritize each node by the loss of entropy we can expect from knowing more of its neighbors. Therefore nodes that already profit from encoded neighbors will be conquered first, which in turn provides more information to its unprocessed neighbors. Clearly all nodes that are skipped by the coder due to optimizations from Section II-C feature a zero entropy and will hence be equipped with the highest priority (0). To lower the computational complexity we use a heuristic to prioritize the other nodes. On initialization these nodes are set up with a priority of 4 and each time a node's neighbor is processed its priority decreases by one. As a result the context group (0, 0, 3) contains almost no entries—in fact it will always comprise one symbol if the mesh represents a connected surfaces. The nodes are thus conquered in a region-growing manner, so nodes with all three neighbors known become extremely rare (*cf.* group (2, 1, 0) and (1, 2, 0) in Table 2). This traversal directly affects the nodes' order which causes the change in the number of culled out symbols. Reviewing the compression rates on our test models shows an average improvement of more than 7 %.

III. Interplay with Geometry Coders

So far we described the strength of our coder in terms of connectivity compression. We also designed the coder with compatibility in mind. Therefore it can be used in conjunction with recent techniques for geometry compression from the multi- as well as the single-resolution setting.

Geometry Prediction Schemes Several schemes for point position prediction have been proposed over the last decade. The parallelogram rule [2] has enjoyed great popularity in the single-rate compression community for both its simplicity and efficiency. Such schemes can be applied progressively for each individual resolution or run as a post-processing step for the finest level. Here, our improved mesh traversal strategy would even enhance the method since multiple parallelograms can be used for prediction, *i. e.*, a bigger stencil can be employed (*cf.* FreeLence [6] for results on prediction gains).

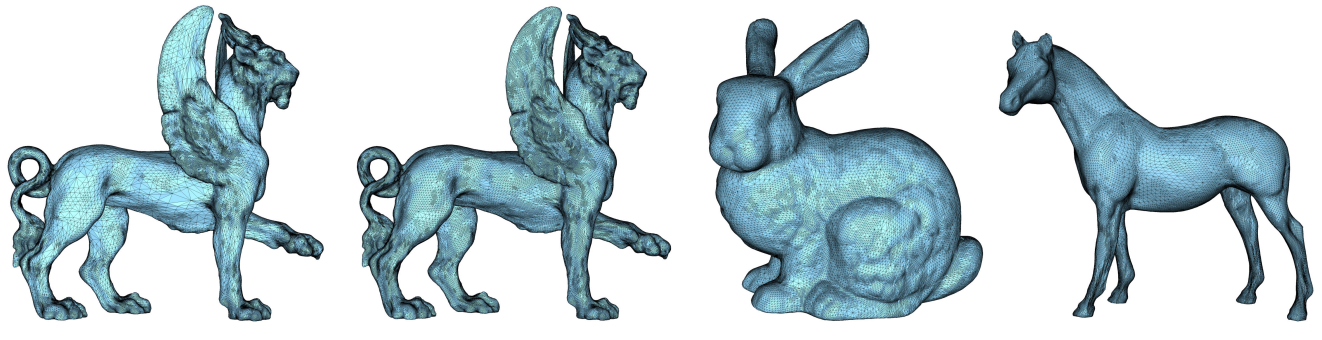
In the field of multiresolution coders, the wavelet coding approaches stand out not only for the compression rates, but also because they have spawned numerous descendants. In fact, wavelet decompositions on surfaces usually apply Loop [21] or butterfly [14], [10] stencils as prediction schemes and subsequently define the local frame details to be the coefficients of the wavelet transform. These steps can be performed independently of the connectivity encoding and require no adjustments.

Data Quantization Classical global and local quantization techniques are fully independent of our coder as these have no effect on the connectivity. We extended our prototype to apply either no quantization or logarithmic quantization by implementing Isenburg's [15] fast floating-point compression. Most of the wavelet based progressive geometry compression schemes are based on zerotree coders [22], [21], which efficiently encode the location of coefficients below threshold in subtrees. These coders could profit from a combination with our coder as it restores the complete refinement hierarchy. This information can then be used to limit the vertex-based hierarchies of the zerotree coder. Whenever the hierarchy is coarse, wavelet coefficients do not have to be stored, so the zerotree coder needs less code. Just setting the wavelet coefficients to zero does not have effect, as the corresponding regions will be any quadric surface patch. Bit allocation techniques like [16] have been proposed to optimize the trade-off between the bit rate and the quality of the reconstructed mesh. These only affect the quantization of the coefficients and works jointly with the zerotree coder, thus being compatible with our coder.

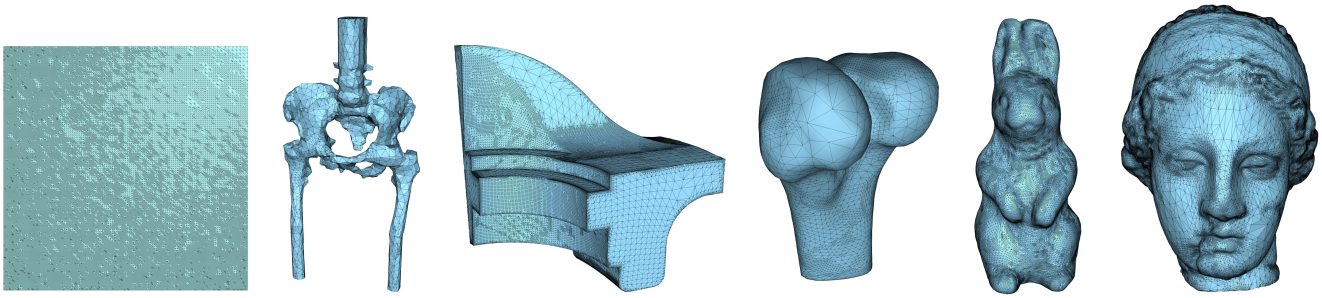
Embedded Bit Streams For the progressive transmission of multiresolution meshes it is important to interlace the connectivity and geometry data. This enables the decoder to reconstruct an intermediate mesh from any prefix of the stream. Therefore a common mesh traversal strategy has to be chosen for the geometry and hierarchy compression. Nevertheless, the traversal used by our coder can be changed to any progressive conquering of the mesh in order to facilitate a special geometry coder.

IV. Results and Conclusion

We measured the performance of our coder using the ten models in Fig. 5. The feline, bunny, horse, rabbit, and



(a) feline_gauss v=99502, f_0=504 (b) feline_mean v=127020, f_0=504 (c) bunny v=45547, f_0=230 (d) horse v=48186, f_0=220



(e) heat_transfer v=48652, f_0=2 (f) bones v=2809 f_0=3882 (g) fandisk v=43048, f_0=4828 (h) femur v=4474, f_0=532 (i) rabbit v=34k, f_0=210 (j) venus v=69338, f_0=388

Figure 5. The test models we used, the number v of vertices at the finest resolution, and the number f_0 of triangles of the base mesh.

model	f	**TG**	**optEB**	**FL**	**Ours**
bones	5622	885	853	**632**	638
bunny	90771	9132	8740	5219	**3656**
fandisk	86092	8217	8041	4521	**3164**
feline_gauss	199008	17200	16499	10909	**6111**
feline_mean	254044	21092	19347	12079	**7853**
femur	8944	791	784	530	**400**
heat_transfer	96412	7979	7612	**3559**	3823
horse	96368	n/a	7811	5251	**3266**
rabbit	68506	n/a	6325	3469	**2735**
venus	138672	10986	10505	7312	**4601**

Table 3. Code sizes in bytes compared with the Touma-Gotsman algorithm (TG), Szymczak's optimized Edgebreaker encoding (optEB), and FreeLence (FL). Note that the rates of FreeLence is not directly comparable to our method, as FreeLence uses the model's geometry to enhance connectivity prediction, whereas the other methods compress the connectivity independently.

venus models are constructed by adaptive coarsening of the remeshes by Khodakovsky *et al.* [21]. For the feline_gauss and venus model we used Gauss curvature for coarsening, the others have been generated using mean curvature as criterion. Bones and heat_transfer are courtesy of Zuse Institute Berlin (ZIB), result from heat transfer simulations. The femur model, courtesy of Oliver Sander, was refined according to criteria of a two-body contact problem in a human knee joint. The fandisk model was produced from a QuadCover [23] parameterization, and was also coarsened using the mean curvature criterion. Note that we included the entropy coded orientation bits for the conformization as well when necessary. In particular this concerns the bones, heat_transfer, and femur models, as these were not generated with our own hierarchy manager.

Table 1 shows the efficiency of our strategies for redundancy removal from the binary representation of the refinement hierarchy. Here especially two approaches contribute to the overall result: **stream truncation** and **1-regularity**. 1-regularity heavily profits from the fact that the refinement scheme only allows the subclass of balanced meshes. Stream truncation exploits the rather simple observation that trailing 0 symbols can be omitted as these cause no change in the reconstructed hierarchy. The effect of stream truncation cannot be achieved by the context based entropy coder alone. Keeping the zeros expanded our codes by 15%. The strategies of Section II-C nearly halve the number of symbols that have to be coded.

Arithmetic coding is another vital part of our compression scheme. Without context groups, the compact binary representation of the hierarchy is almost incompressible due to a rather uniform distribution of the symbols, as confirmed by values in parentheses in Table 1. Again, knowledge about the mesh structure is used to apply context based arithmetic encoding. The introduced context groups again reduced the code length by approximately 50 %.

An evaluation of compression rates within each context group was given in Table 2 and revealed huge gaps between

the performance of individual groups. These gaps can be attributed to the mutual information inherent to each context group. Therefore we devised a hierarchy traversal scheme that shifts the distribution of symbols over the contexts and thus equilibrates the mutual information available for the coding of each node. Despite the fact that this raised the number of encoded symbols, an average gain of over 7 % for the overall compression rates could be achieved.

Table 3 summarizes our results in terms of compression rates. Our coder outperforms the optimized Edgebreaker coder (and the Touma-Gotsman coder, resp.) by factors of up 2.7 (2.9), with factor of 2.2 (2.3) on average. The FreeLence algorithm achieves an average factor of 1.3 and even surpasses our coder for two of the test models (bones and heat_transfer). Regarding the bones model these results are not surprising as the outcome for the root mesh dominates the overall code size (440 bytes). For the heat_transfer model it is not quite as obvious. In this case FreeLence profits from the regularity and planarity of the grid. Only discrete angles with a small set of free valences emerge, leading to exceptional compression rates. FreeLence however achieves this compression rates by employing knowledge of the geometry. Contrary, our method is independent of the geometry and can be combined with any geometry compression scheme which could in return employ knowledge of the connectivity, *i. e.*, the hierarchical structure. Thus, the other algorithms are more appropriate for comparison to our coder.

ACKNOWLEDGMENT

This work was supported by the DFG Research Center Matheon "Mathematics for key technologies" and mental images GmbH. We thank Martin Weiser and Sebastian Götschel of the Zuse Institute Berlin as well as Oliver Sander for providing their multiresolution data. Models are courtesy of Stanford University, Caltech, and the Visible Human Project. Finally we thank Andrzej Szymczak for providing the Optimized Edgebreaker compression rates.

REFERENCES

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

[2] C. Touma and C. Gotsman, "Triangle mesh compression," in *Graphics Interface Conference Proceedings*, 1998, pp. 26–34.

[3] M. Isenburg and J. Snoeyink, "Early-split coding of triangle mesh connectivity," in *Graphics Interface 2006 Proceedings*.

[4] A. Szymczak, "Optimized edgebreaker encoding for large and regular triangle meshes," in *DCC '02 Proceedings*. Washington, DC, USA: IEEE Computer Society, 2002, p. 472.

[5] W. Tutte, "A census of planar triangulations," *Canadian Journal of Mathematics*, vol. 14, pp. 21–38, 1962.

[6] F. Kälberer, K. Polthier, U. Reitebuch, and M. Wardetzky, "Freelence - coding with free valences," *Computer Graphics Forum*, vol. 24, no. 3, pp. 469–478, 2005.

[7] A. W. F. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin, "Maps: multiresolution adaptive parameterization of surfaces," in *SIGGRAPH '98 Proceedings*, 1998.

[8] I. Guskov, K. Vidimče, W. Sweldens, and P. Schröder, "Normal meshes," in *SIGGRAPH '00 Proceedings*, 2000.

[9] J. Peng, C.-S. Kim, and C.-C. J. Kuo, "Technologies for 3d mesh compression: A survey," *Journal of Visual Communication and Image Representation*, vol. 16, 2005.

[10] A. Khodakovsky and I. Guskov, "Compression of normal meshes," in *In Geometric Modeling for Scientific Visualization*. Springer-Verlag, 2003, pp. 189–206.

[11] M. I. Jack and J. Snoeyink, "Mesh collapse compression," in *In Proceedings of SIBGRAPI99*, 1999, pp. 27–28.

[12] P. Alliez and M. Desbrun, "Progressive compression for lossless transmission of triangle meshes," 2001.

[13] P. Alliez and C. Gotsman, "Recent advances in compression of 3D meshes," in *Advances in Multiresolution for Geometric Modelling*, Dodgson, Floater, and Sabin, Eds. Springer, 2005.

[14] F. Morán and N. N. García, "Hierarchical coding of 3d models with subdivision surfaces," 2000.

[15] P. Lindstrom and M. Isenburg, "Fast and efficient compression of floating-point data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1245–1250, 2006.

[16] F. Payan and M. Antonini, "An efficient bit allocation for compressing normal meshes with an error-driven quantization," *CAGD*, vol. 22, no. 5, pp. 466–486, 2005.

[17] N. Dyn, D. Levine, and J. A. Gregory, "A butterfly subdivision scheme for surface interpolation with tension control," *ACM Trans. Graph.*, vol. 9, no. 2, pp. 160–169, 1990.

[18] C. Loop, "Smooth subdivision surfaces based on triangles." Utah University, USA, 1987.

[19] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical J.*, vol. 27, 1948.

[20] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[21] A. Khodakovsky, P. Schröder, and W. Sweldens, "Progressive geometry compression," in *SIGGRAPH '00 Proceedings*, 2000, pp. 271–278.

[22] A. Said, W. A. Pearlman, and S. Member, "A new fast and efficient image codec based on set partitioning in hierarchical trees," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, pp. 243–250, 1996.

[23] F. Kälberer, M. Nieser, and K. Polthier, "Quadcover - surface parameterization using branched coverings," *Computer Graphics Forum*, vol. 26, no. 3, pp. 375–384, 2007.