Volume xx (200y), Number z, pp. 1–14

Context-Based Coding of Adaptive Multiresolution Meshes

Christoph von Tycowicz, Felix Kälberer, and Konrad Polthier

Freie Universität Berlin, Germany

Abstract

Multiresolution meshes provide an efficient and structured representation of geometric objects. To increase the mesh resolution only at vital parts of the object, adaptive refinement is widely used. We propose a lossless compression scheme for these adaptive structures that exploits the parent-child relationships inherent to the mesh hierarchy. We use the rules that correspond to the adaptive refinement scheme and store bits only where some freedom of choice is left, leading to compact codes that are free of redundancy. Moreover, we extend the coder to sequences of meshes with varying refinement. The connectivity compression ratio of our method exceeds that of state-of-the-art coders by a factor of 2 to 7.

For efficient compression of vertex positions we adapt popular wavelet-based coding schemes to the adaptive triangular and quadrangular cases to demonstrate the compatibility with our method. Akin to state-of-the-art coders, we use a zerotree to encode the resulting coefficients. Using improved context modeling we enhanced the zerotree compression, cutting the overall geometry data rate by 7% below those of the successful Progressive Geometry Compression. *More importantly, by exploiting the existing refinement structure we achieve compression factors that are 4 times greater than those of coders which can handle irregular meshes.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

1. Introduction

Multiresolution meshes, *i.e.*, meshes obtained through successive subdivision of a coarse base complex, are commonplace in a variety of areas such as the movie industry, computer aided design and in 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 a coarse 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 hierarchies has not been researched into before.

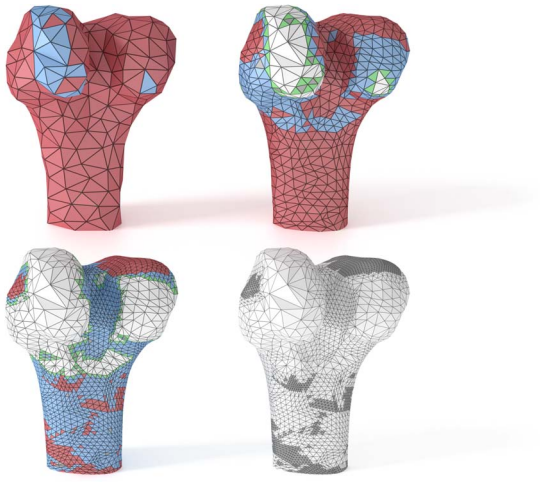


Figure 1: *Adaptive refinement of a bone model. Elements are colored according to our coding scheme. We store a bit for each blue and red triangle, specifying whether it is refined or not. These bits are sufficient to reconstruct the connectivity of the model. All other triangles can be reconstructed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

Generally, compression comes in two stages: The first is a lossy stage, where essential information of the input is extracted and negligible data is dropped. The data decimation is then 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 numerical analysis depend on an 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 some 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 optimal control problems with time-dependent PDEs 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 situation.

Related Work Numerous compression schemes for surface 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 [Ros99] and the method of Touma and Gotsman [TG98] are the most prominent coders for triangle meshes 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 [IS06] and the optimized Edgebreaker encoding of Szymczak [Szy02]. These coders profit from mesh regularity and are able to push the bit rate well below the Tutte limit [Tut62] of roughly 3.24 bits per vertex. Many triangle mesh coders have been generalized to polygonal meshes, such as Martin Isenburg's method [Ise02] which extends the Touma-Gotsman coder.

The FreeLence [KPRW05] and Angle Analyzer [LAD02] algorithm exploit correlation between connectivity and geometry by accessing already encoded geometry data when encoding connectivity and vice versa, allowing it to push the bit rates below that of [IS06] and [Szy02]. While FreeLence is especially performant in the triangular case, Angle Analyzer outperforms it for quadrangular meshes.

For progressive transmission, models are often simplified or remeshed [LSS*, GVSS00] to generate a simple base mesh from arbitrary connectivity meshes. In this context, wavelet-based coding has proven itself as *the* approach for efficient compression. Wavelet transforms recursively construct lower resolution approximations of a given input model, decorrelating high- and low-frequency geometry data. The difference of the details to predictions based on the coarse data are stored as wavelet coefficients. These typically feature a smaller entropy than the original data yielding superb compression rates.

Wavelet-based coding schemes exist for both irregular and semi-regular meshes. The first group is based on mesh simplification methods that progressively remove vertices causing the smallest distortion. Bits are written to identify a vertex within the mesh as well as its geometric position in order to be able to reconstruct the original mesh. Prominent coders of this group are [JS99, AD01, VP04].

The best results for geometry compression however have been achieved for semi-regular meshes. Based on stencils from subdivision schemes, efficient wavelet transforms have been derived. The best known wavelet-based coder in this category is the progressive geometry compression (PGC) codec by Khodakovsky *et al.* [KSS00] adapting the renowned zerotree coding scheme [Sha93] from image compression. A wealth of descendants have been proposed extending PGC to different types of meshes [KG03], resolution scalability [AMG05] and efficient embedded quantization [PA05]. However, these coders only aim at the compression of the geometry and do not allow lossless reconstruction of the connectivity even for meshes generated through adaptive refinement. (Although the normal mesh compression by Khodakovsky *et al.* [KG03] is able to generate such meshes, the adaptivity is controlled by the coder, thus neglecting any original criteria.)

Adaptively refined hierarchies also play a major role in point cloud compression. In [BWK02] and many follow-ups, a bounding box is adaptively refined up to a certain degree, and the leaf cells at the finest level then represent the point positions. Although these methods also compress adaptive trees, we propose optimization strategies specialized for the

situations on surface meshes, such as non-trivial root domains, red-green conformity, and balanced refinement. On the contrary, point cloud compression schemes utilize characteristics of surface geometry. For instance, close-by points are expected to be nearly coplanar, as done in [SK06].

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. 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 single-rate and progressive coders, outperforming state-of-the-art by factors of 2 to 7. Additionally, we present extensions to our compression that exploit correlations of the refinement structure in sequences of time-dependent grids.

To show that our connectivity compression for adaptive hierarchies works seamlessly with leading geometry compression schemes, we have extended the well-proven, wavelet-based progressive geometry coding (PGC) of Khodakovsky *et al.* [KSS00] to adaptive triangle and quad based hierarchies. As in connectivity coding we applied context-based modeling for geometry compression. For uniform hierarchies we could achieve gains of 7% on average over PGC for the coding of wavelet coefficients. Furthermore, we surpass non-specialized coders (Wavemesh [VP04], and PMC [Ise02]) by average factors of 2.8 for adaptive triangular and 4.5 for quadrilateral meshes.

All our methods are described for triangular and quadrilateral surface meshes and we provide connectivity and geometry compression results for both cases. Although we illustrate our scheme for adaptive surface hierarchies, the methods are not restricted to these meshes, and an extension to adaptive tetrahedral or hexahedral meshes is straightforward.

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 topology and coarse shape is reconstructed. Further details will be added from coarse to fine levels as more code is processed.

2. Hierarchy Coding

In this section we explain how we encode the hierarchical structure of adaptive multiresolution meshes and thus their connectivity. First we will explain the concepts of adaptive triangular and quadrilateral refinement, before we outline our encoding procedure in Section 2.2. Sections 2.3 to 2.6 then elaborate on the details.

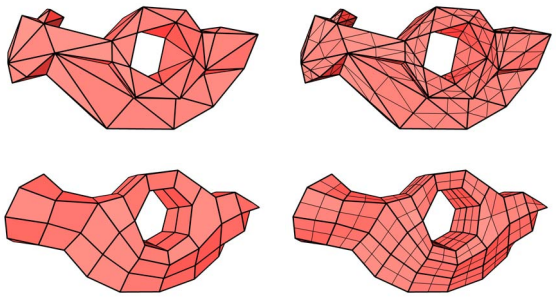


Figure 2: *Uniform refinement of coarse domains using the dyadic split (top) and face split (bottom).*

2.1. Adaptive Refinement Schemes

In the field of subdivision surfaces and FEM a variety of refinement schemes have been devised. Among them, the dyadic split (*cf.* butterfly [DLG90] or Loop [Loo87]) for triangle surfaces and the face split (*cf.* Catmull, Clark [CC78]) for polygonal surfaces are widely spread. The dyadic split operation divides each triangle 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. Consequently, this scheme is often referred to as 1-to-4 split. The face split, on the other hand, can be applied to elements of arbitrary degree. New vertices are inserted not only at edge midpoints but also at the center of the element. Then the vertices at the edge midpoints are connected to the center vertex, thus splitting an n-gon into n quadrilaterals. Figure 2 illustrates both the dyadic and the face split refinements.

Unlike *global mesh refinement* where all elements in the mesh are subdivided to obtain a finer mesh, *adaptive* or *local mesh refinement* performs the split operation only for selected elements. The transition between elements of different refinement levels requires extra care since a new vertex is inserted on an edge of the refined element, but the coarse neighboring face still contains the edge undivided. These irregular vertices are called *hanging nodes*. To resolve the non-conforming situations between elements of various refinement grades, the adjacent unrefined elements must also be refined. To maintain locality of the conformization, non-conformal elements must be split without introducing additional hanging nodes (see Figure 3, top). In the finite element community triangles introduced to resolve hanging nodes are called *green* triangles whereas elements that are generated by the dyadic split are called *red* triangles (hence the name *red-green refinement* [BSW83]). Several conformization schemes exist for quadrilateral hierarchies, some of them introducing non-quadrilateral elements. We implemented the scheme described in [SMFF04] (see Figure 3, bottom). Following the aforementioned terminology

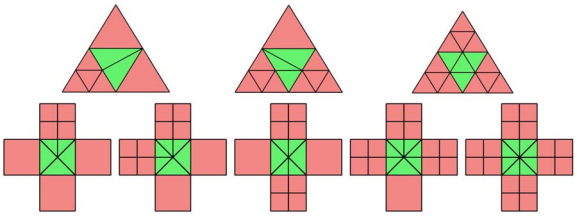


Figure 3: *Conformization of hanging nodes. Up to symmetry, these are the only configurations that can occur in* balanced meshes.

elements inserted by these schemes will also be referred to as *green elements*.

Since green elements 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 refinement of green elements and therefore ever thinner faces.

The refinement strategy yields a canonical hierarchical structure where each split element acts as a parent for the new sub-elements. Therefore each element of the coarse base mesh will have an associated tree that specifies its refinement. We refer to the entities of the trees as *nodes* to underline the parental relationship between elements at different resolutions of the mesh. For a split element we assign the sub-element incident to the n-th vertex as the n-th child of the related node. In the triangle setting, the remaining central sub-element will be the fourth child. Figure 4 shows three root elements with associated refinement trees as well as the represented adaptive grid.

2.2. Encoding

Akin to existing progressive coders we separately encode the base domain from the hierarchy. Typically the root grid is described by a small, carefully laid out mesh that can be compressed well using single-rate coders. Our prototype uses FreeLence [KPRW05] and PMC [Ise02] to losslessly encode the triangular and quadrilateral root grids, respectively. This compression will generally alter the order of the base domain elements 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 grid 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 the encoder and decoder agree on a common node traversal strategy, the

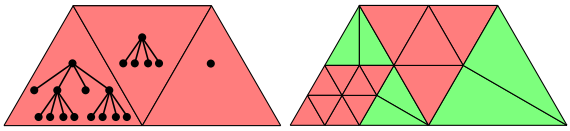


Figure 4: *Base mesh with refinement trees and its corresponding red-green refined hierarchical mesh.*

refinement hierarchy can be stored with one bit per node where each bit specifies whether the node has children (is refined) or not. Exceptions are made when the conformization schemes leaves freedom of choice, *e.g.* triangles with two hanging nodes, see Figure 3. The other possibility to resolve this non-conforming situation arises by flipping the diagonal edge. In practice, the concrete conformization is often determined exclusively by local indexing. Since we change the local indexing during the compression of the base mesh, the original indexing is lost. Therefore we need to store additional symbols to determine the conformizations. For the triangle hierarchies these will be one bit per triangle with two hanging nodes. Using a conformization scheme that introduces only quadrilaterals (see *e. g.* [Sch96]) at most one bit for each border between regions of different refinement must be coded for quadrilateral hierarchies. The conformization scheme of Settgast *et al.* [SMFF04] on the other hand has no such ambiguities. We entropy code these bits, but found that they where virtually incompressible without knowing the exact implementation of the grid manager. If, however, a geometric criterion is used to determine the conformizations, we can omit these bits altogether. The same is true if the application does not expect any green elements and the conformization is constructed on the fly, for example the *Progressive Geometry Compression* software from Caltech [KSS00].

Due to the deterministic conversion of the hierarchical structure to a bit stream we can estimate an upper bound for the code size. Let us begin with hierarchies generated by face splits. Except for the root level, each parent is split into four children. Therefore, the number of bits in the stream will sum to one bit per leaf plus one $\frac{1}{4}$ bit for their parents, $\frac{1}{4^2}$ bit for their grandparents and so on until reaching $\frac{1}{4^{d-1}}$ bit for the first level, where d is the depth of the leaf in the tree. Additionally, we have to add $\frac{1}{4^{d-1}}\frac{1}{n}$ bit for the n-gon contained in the root level. We can write this sum as

$$\sum_{i=0}^{d-1}\frac{1}{4^i}+\frac{1}{4^{d-1}}\frac{1}{n}=\frac{4}{3}+\frac{1}{4^{d-1}}\left(-\frac{1}{3}+\frac{1}{n}\right)\leq\frac{4}{3}$$

This bound also holds for the triangle hierarchies since they are composed entirely of quad trees so the n in the last term will be 4. Since the number of leaves in the hierarchy is no greater than the number f of elements of the finest resolution, our stream will contain no more than $\frac{4}{3}f$ bits. So far we did not account for symbols needed to resolve multiple possibilities of conformization. In these cases we have

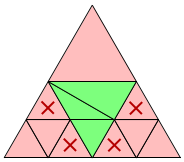
to store additional bits, but in fact, we already counted multiple green elements in f that where represented by just one node in the hierarchy.

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.

2.3. 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 are mirrored the hierarchy's structure. These dependencies are exploited by the following extensions:

1-REGULARITY As mentioned before, adaptive 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 elements that resolved a non-conforming situation in the parent level. As a consequence, nodes representing faces adjacent to coarse green elements cannot be refined and can thus be skipped by the encoder.



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

HANGING NODES Some implementations of adaptive refinement prohibit more than a certain number of hanging nodes per element. In these setups, the hanging nodes are conformized by applying the usual split operation, introducing hanging nodes in adjacent coarse elements. During compression, if all neighbors of the current node are already processed, the number of hanging nodes within the element will be known to the coder. Hence, elements that exceed the maximum number of allowed hanging nodes can be split immediately and no symbol has to be coded. Anyhow, we do not need to treat these cases explicitly since they will be handled by our coder without overhead, *cf.* Section 2.4.

UNIFORM REFINEMENT Uniformly refined meshes exhibit a featureless hierarchical structure—the whole forest can be described by a 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 Note 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 element, 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, *cf.* [Sai04]. 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 of 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. Because *uniform refinement* only affects the coarsest levels only a few symbols are omitted for our adaptive test models. Overall, the number of symbols that have to be coded averages out at nearly half the number of nodes.

2.4. 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 element 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, 23k of the 96k 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 leaves, that probability is a mere 1.2 %.

Let X be the binary stream of split data. As shown by

model	f	#nodes	drop 0s	1-regul.	uniform	#left	code size
bones	5622	5438	8%	16%	0.0%	76% (4133)	1520 (1904)
fandisk3	86092	95532	34%	23%	0.0%	43% (40770)	23208 (40520)
feline	254044	303072	37%	16%	0.8%	46%(138523)	60096 (138512)
femur	8944	10608	26%	13%	0.0%	60% (6417)	2392 (4680)
heat_transfer	96412	115034	5%	15%	0.6%	79% (91111)	28632 (82840)
horse	96368	113228	37%	18%	0.2%	46% (51545)	25368 (51440)
rabbit	68506	78570	23%	21%	1.3%	55% (43368)	21496 (42928)
venus	138672	154792	40%	24%	0.0%	36% (56112)	35560 (50320)
blade	51157	35905	16%	36%	0.0%	49% (17425)	9960 (17488)
fandisk4	24595	30662	24%	3%	0.0%	73% (22521)	1424 (17888)
fertility	192635	129420	14%	37%	0.0%	49% (63513)	42848 (63592)
shoulder	108361	77219	16%	34%	0.4%	49% (38120)	23448 (38192)
rockerarm	30747	27160	30%	21%	6.8%	42% (11482)	5392 (11360)
torso	54918	38583	12%	36%	0.7%	52% (19929)	11968 (19968)
average	72111	86802	23%	22%	0.8%	54% (43212)	20951 (41545)

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 6 list percentage of bits that can be omitted by dropping trailing zeros, exploiting 1-regularity, storing the number of levels of uniform refinement. Column 7 list the percentage and actual number of bits that have to be stored, and the last row shows the size of the compressed code in bits, with (and without) the use of context groups.*

Shannon [Sha48], 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 [WNC87], approach this lower bound in the limit and are thus optimal in that 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 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 the definition of contexts: a neighbor can either be *refined* (△), *not refined* (△), or it has not been processed before (**?**).

The status of the neighbor elements 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 is obtained 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 exactly how we define the contexts. The contexts arise from the number of △, △, and **?** neighbors of a hierarchy node. We write (x,y,z) to denote the context with x △ situations, y times △, and z times **?**. For the triangle hierarchy setting the all possible cases are listed in Table 2.

Group △,△,?	Naïve traversal #symb	%1's	bits/symb	Improved traversal #symb	%1's	bits/symb
(0, 3, 0)	13606	2%	0.123	681	10%	0.517
(1, 2, 0)	5603	37%	0.954	30	50%	1.400
(2, 1, 0)	5913	82%	0.693	91	78%	0.945
(3, 0, 0)	12146	100%	0.003	685	100%	0.041
(0, 2, 1)	14314	6%	0.340	29789	5%	0.289
(1, 1, 1)	6373	43%	0.992	12079	72%	0.860
(2, 0, 1)	14130	99%	0.049	33227	98%	0.129
(0, 1, 2)	17256	10%	0.487	40996	23%	0.772
(1, 0, 2)	18520	94%	0.342	20944	96%	0.222
(0, 0, 3)	30855	55%	0.994	1	100%	5.000
culled	164356		0	164549		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 *model. For each strategy we provided the number of symbols in each context group, the percentage of 1's among those symbols, and the efficiency in terms of bits per symbol.*

2.5. Traversal Order

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 level-wise 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 affects the distribution of the symbols 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. Here *naïve traversal* refers to a strategy where children are visited in order.

In our implementation elements incident to the parent's vertices are visited first. Additionally local indexing of parent elements determines the order of its children. Hence context group $(0,0,n)$, where none of the neighbors are known, contains the most entries. This group, though, is virtually incompressible as no advantage can be taken of mutual information. The same holds for context groups where the extra information is rather ambiguous *e. g.* $(1,1,1)$, $(1,2,0)$, and $(2,1,0)$ in the triangle setting. Contrarily, the other context groups perform very well but are less populated.

The positive effects of an adaptive traversal order have been observed in Angle-Analyzer [LAD02] for surface meshes. Also, Schnabel *et al.* [SK06] optimize the octree traversal in each level. In this spirit, 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 a 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 entropy of its current context. 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 2.3 feature a zero entropy and will hence be traversed before all other nodes. A promising candidate for prioritization of the remaining nodes is to use the actual entropies of the individual contexts. The greedy strategy traverses nodes in the context with lowest entropy first in order to minimize the overall compression rate. Experiments showed that this strategy already achieves significant improvements in compression. Learning the entropies of the contexts, however, is expensive in terms of compression performance as well as computational cost. Once the learning phase is settled, the algorithm sticks with a fixed prioritization of contexts. To remove all effects of the learning process of the contexts' entropies from the traversal, we additionally investigated fixed priorities for the nodes, *i.e.*, a fixed order of contexts during the search for the next node to conquer. We looked at different orderings:

- increasing by the learned (settled) entropies of contexts,
- increasing by the number of unknown neighbors, and
- decreasing by the difference of known coarse and known refined neighbors.

In the case of ties we also tried all possible permutations.

The tests revealed that we can substantially improve the performance of the traversal by skipping the learning process. Although we could not identify a single strategy that performed best on the entire set, ordering the contexts as in Table 2 (increasing by number of unknown neighbors and breaking ties increasing by the number of known refined neighbors) constantly provided good results. Therefore we chose this ordering as the default for our coder and thus for all the presented tests.

As a result of our choice, the context group $(0,0,n)$ contains almost no entries—in fact, it will always be comprised of one symbol if the mesh represents a connected surfaces. The nodes are thus conquered in a region-growing manner, so nodes whose neighbors are all known become extremely rare (*cf.* group $(3,0,0)$, $(2,1,0)$, and $(1,2,0)$ in Table 2). Furthermore, the traversal directly affects the nodes' order which causes the change in the number of culled out symbols. Reviewing the final compression rates on our test models shows an average improvement of 7 %.

2.6. Time-varying Sequences

As we observed, we can profit well from the correlations between adjacent elements. We can profit in the same way

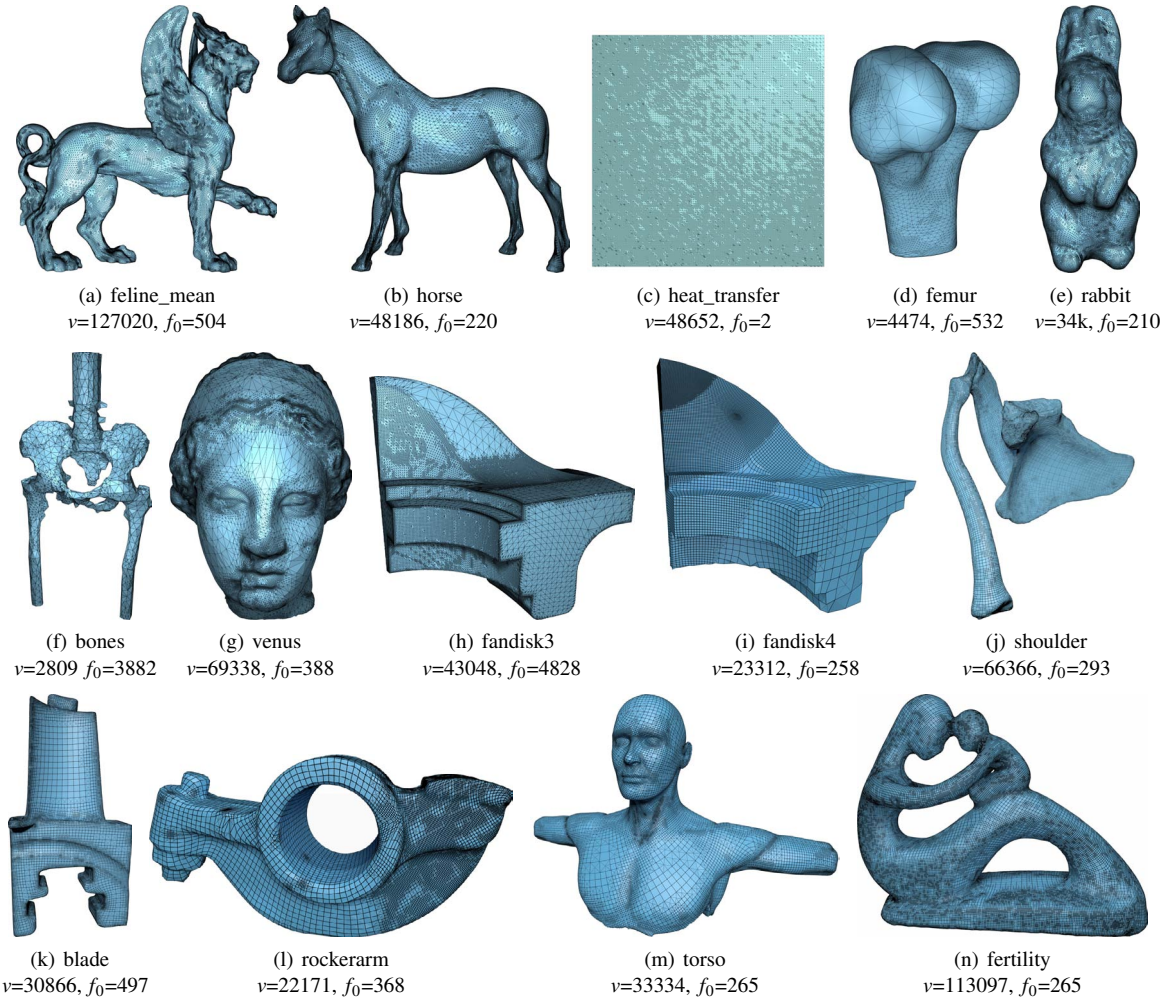


Figure 5: *The test set we used, the number v of vertices at the finest resolution, and the number f_0 of elements of the base mesh.*

when we have a time-varying series of refinements of a common base mesh. Here, we assume that the degree of refinement varies only a little from one frame to another, just as one assumes smooth transitions for animated geometries or videos.

When processing a hierarchy node in a series, we query the state of the corresponding node in the previous time frame, which can give us one of three states:

- it was a node with children,
- it is a leaf node, or
- it didn't exist

in the previous frame. Thus, the number of contexts triples, if we also include the status of the previous frame in the contexts.

If the refinement trees don't vary much between the time steps, then contexts corresponding to the first case will be mainly filled with ones, while the latter two will primarily contain zeros. Thus, grids which equal their preceding frame can be stored at no cost, aside from a small overhead due to the use of more contexts. At the contrary, if the grids are not correlated, the entropy of the individual contexts can never be worse than in the static setting, since symbols from one context are simply spread to three, maintaining or improving symbol possibilities. Table 3 shows the results of the time series adaption applied to three time-varying hierarchies.

3. Geometry Compression

Thus far, we have described our coder in terms of connectivity compression. This section covers the compression of the geometry of the adaptive hierarchies by extending progressive coding schemes from the uniform setting. The following text will cover wavelets and zerotrees only very briefly as extensive literature on these topics exist. We refer the reader

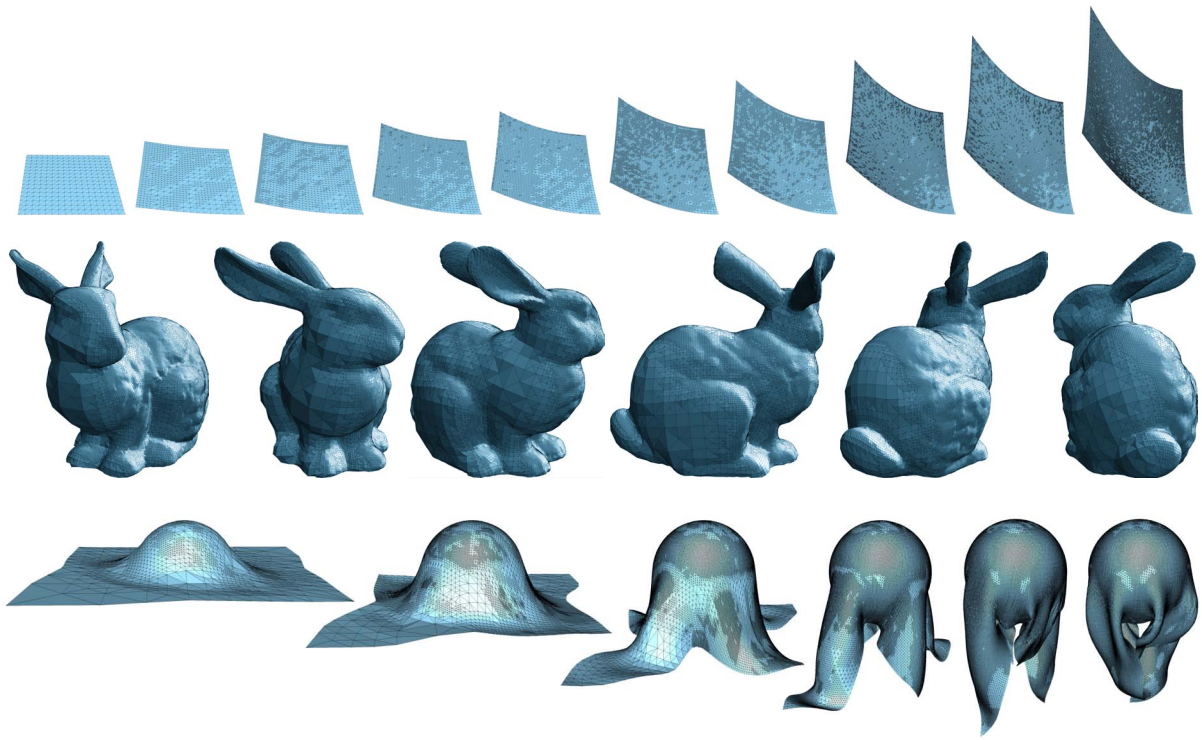


Figure 6: *The three test sequences used. Top: planar domain refined driven by heat transfer computations (showing temperature as z-component). Middle: View-dependent refinement taking curvature, visibility and silhouette into account (as proposed in [SMFF04]). Bottom: cloth simulation with curvature-dependent refinement.*

model	#frames	average	total	diff.
bunny	21	585 (1002)	12280 (21046)	-42%
heat	11	420 (1125)	4621 (12376)	-63%
cloth	15	459 (472)	6888 (7085)	-3%

Table 3: *Compression results for time-varying sequences. Left to right: Model, number of frames in the sequence, average code size per frame in bytes for dynamic (static) coder, total code size in bytes for dynamic (static) coder, and the difference in code size between dynamic and static coder.*

to [KSS00] for a discussion on wavelet transforms of semi-regular meshes and to [SPM96] for a detailed description of zerotrees.

3.1. Wavelet transform

Wavelet transforms convert a mesh hierarchy into a coarse base mesh and nested sets of wavelet coefficients representing the mesh details. The coefficients are computed as differences to predictions derived from the parent level. Thus, any resolution of the mesh can be reconstructed recursively from the base mesh and the coefficients from all intermediate resolutions. Various subdivision schemes can be used to predict new points from the parent level.

In our implementation we compute the wavelet transform based on interpolatory subdivision schemes. Although lifted wavelet transforms have been reported to provide better compression, we used unlifted schemes for simplicity. We applied butterfly subdivision [DLG90] for triangle hierarchies and Leif Kobbelt's scheme [Kob96] for quadrilateral hierarchies. An exception is the uniformly refined triangular case, where we used the PGC software for the lifted butterfly wavelet transform. This way, we can ensure the exact same quantization errors, which simplifies comparison.

We represent the vector-valued wavelet coefficients in a local frame induced by the tangent plane of the surface. Since normal errors have more impact on the distortion, we increase the precision of the normal component. We chose a factor of 4, which has been reported as being reasonable in the literature. Each component is encoded individually using zerotree coding.

Once the coefficients in level j are computed, we adjust their scaling by a factor of 2^{-j}. Such scaling arises from L_2-normalizing the subdivision basis functions. It is commonly applied in compression to compensate for their shrinking support on finer levels. Note that the scaling provides higher accuracy on coarser levels and causes stronger quantization for coefficients on finer levels.

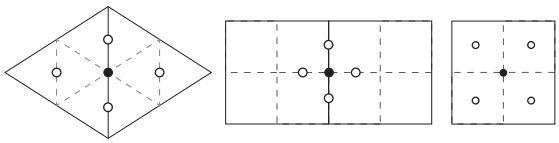


Figure 7: *Parent-child relationships in triangular and quadrilateral hierarchies. Solid circles indicate parents; empty circles indicate children.*

3.2. Zerotrees

For decreasing thresholds $T_k = T_0/2^k$, the zerotree coder successively transmits the location and sign of coefficients that become significant, *i.e.*, when $|c_i| > T_k$. In order to transmit this information efficiently, coefficients are grouped in a tree structure implementing the parent-child relationship depicted in Figure 7.

For triangle hierarchies, coefficients have a one-to-one association with edges of the coarser level. For quadrilateral hierarchies there are two types of associations: vertices inserted at face midpoints are associated with these faces, whereas vertices on edge midpoints are associated to these edges. Based on this structure we use the SPIHT encoding [SPM96] to transmit significance information either for a single coefficient or for sets of coefficients, requiring only a single bit if all coefficients in a set are insignificant. Thus, the zerotree coder exploits the fact that the distribution of the wavelet coefficients is centered around zero and that their magnitudes tend to decrease at finer subbands depending on the smoothness of the input surface.

In addition to the significance and sign data, refinement bits have to be sent in each pass for coefficients that became significant in earlier passes. The initial threshold T_0 and the number of zerotree passes determine the quantization of the coefficients. In our experiments we choose T_0 to be the greatest magnitude of all coefficients' components. The number of bit planes was chosen to achieve L_2 errors that are comparable to those caused by uniform 12-bit quantization.

For the progressive transmission of multiresolution meshes it is important to interleave 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.

3.3. Entropy Coding

Although very efficient, the compression performance of the zerotree coder can be further improved by applying entropy coding to significance bits. Refinement and sign bits, on the other hand, are much harder to compress.

As Denis *et al.* recently observed [DSM*10], there is a strong correlation between neighboring coefficients within each level (intraband correlation) and a correlation between parents and their child coefficients (interband correlation), with the former being stronger than the latter. The SPIHT encoder (and therefore PGC) exploits the interband correlation by transmitting the significance information for children of a parent as a j-bit symbol, where $j \in \{1,2,3,4\}$ is the number of insignificant children.

However, as seen in Figure 8, there are coefficients in the same level (squares) that are closer than the siblings defined by the zerotree (light gray). Since there is a higher correlation between direct neighbors within a subband, we decided to use their significance information for entropy coding.

Following the ideas of our connectivity coding, we define contexts based on the number of significant, insignificant, and unprocessed neighbor coefficients. Since the wavelet coefficients of different subbands are of different magnitudes, we consider only neighbors of the same subdivision level to guarantee similarity of the data. We devise four contexts representing the neighbor status:

- all processed neighbors are significant
- all processed neighbors are insignificant
- all neighbors are unprocessed.
- there are significant and insignificant neighbors

This choice of contexts can be used for various mesh types and is robust if the number of neighbors varies due to boundaries, adaptivity, or irregular face degrees.

The significance bits in the SPIHT algorithm occur not only for single coefficients but also for sets of coefficients, both contributing roughly the same amount to the code size. Following the context construction for coefficients we also apply context-based coding for *type A* and *type B* set significance bits separately, tripling the number of contexts to twelve (see [SPM96] for a definition of set types). For the set contexts, neighbors are counted as significant if their respective set significance test is positive.

To also exploit interband correlation we double the number of contexts for coefficients and type *A* sets, increasing the total number to 20. For a coefficient, the context decision is based on the significance value of its parent. For a type *A* set,

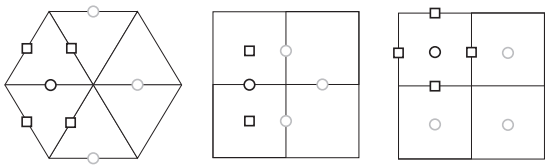


Figure 8: *The neighbors (squares) that are used for context-based coding of significance bits. The black circle marks the current coefficient and gray circles represent sibling coefficients that are grouped in the SPIHT coder.*

			PMC			Wavemesh			PGC			our		
	triangle mesh	*v*	**geom**	**conn**	L_2	**geom**	**conn**	L_2	**geom**	**conn**	L_2	**geom**	**conn**	L_2
adaptive	bones	2809	20.63	2.34	47	21.60	4.44	47	–	–	–	18.98	1.84	46
	fandisk3	43048	11.15	1.53	38	11.94	2.87	37	–	–	–	6.06	0.58	32
	feline	127020	10.18	1.23	37	10.33	2.40	37	–	–	–	2.39	0.48	35
	femur	4474	17.12	1.32	38	18.53	2.75	38	–	–	–	13.19	0.71	31
	heat_trans	48652	2.22	1.21	0	3.57	2.52	0	–	–	–	0.10	0.64	4
	horse	48186	11.30	1.32	40	11.39	2.37	40	–	–	–	3.10	0.53	40
	rabbit	34255	12.94	1.47	43	13.28	2.81	43	–	–	–	5.12	0.64	32
	venus	69338	12.60	1.50	33	12.39	2.60	33	–	–	–	4.30	0.52	31
uniform	fandisk3	154498	4.47	0.03	40	4.40	0.04	28	1.99	0.01	26	1.67	0.01	26
	feline	258046	5.34	0.01	37	2.64	0.01	31	1.14	0.00	34	1.08	0.00	34
	horse	112642	5.51	0.01	40	3.45	0.01	32	1.35	0.00	40	1.30	0.00	40
	rabbit	107522	5.65	0.01	43	3.90	0.01	43	1.93	0.00	32	1.89	0.00	32
	venus	198658	5.67	0.01	35	3.63	0.01	40	2.04	0.00	30	1.96	0.00	30
	avg. (adaptive)	47222	12.27	1.49	35	12.88	2.85	34	–	–	–	6.66	0.74	31
	avg. (uniform)	166273	5.33	0.01	39	3.54	0.02	35	1.69	0.00	32	1.58	0.00	32

Table 4: *Connectivity and geometry rates (columns **conn** and **geom**) for triangle hierarchies in bits per vertex. Column v lists the number of vertices, and L_2 the root mean square errors reported by* `metro` *in units of* 10^{-6} *w.r.t. the bounding box diameter.*

we check the parent of the coarsest coefficients in the set and switch the context if it is significant. For type *B* sets, we do not need to store a bit if the parents are insignificant. In this case, the set will always be significant. Hence, no doubling of contexts is needed for type *B* sets.

3.4. Adaptive Zerotree Coding

So far, we have only described geometry encoding of uniformly refined hierarchies. Using adaptive meshes, we can profit from the fact that we don't have to store any bits for coefficients that are not present in the adaptive mesh. In our context model for the significance bits, we consider nonexistent coefficients as unprocessed.

An alternative is to fill up the zerotree with zero coefficients producing a uniformly refined tree structure, as done in Khodakovsky's and Guskov's normal mesh compression [KG03]. Due to the efficiency of the zerotree coder, the impact on the code size is small. Nevertheless, to reconstruct the original adaptivity, the refinement information must be transmitted, which is more costly than our proposed method, especially if the mesh refinement is not geometry related.

4. Results and Conclusion

We measured the performance of our coder using the 14 models in Figure 5 as well as the three time-varying sequences in Figure 6. The uniform feline, horse, rabbit, and venus models are from the PGC data set, courtesy of Khodakovsky *et al*. We constructed the respective adaptive models by local coarsening according to various curvature measures. Bones and heat_transfer—courtesy of Zuse Institute Berlin—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 quadrilateral hierarchies are remeshes based on QuadCover [KNP07] parameterizations. The fandisk4 and the rockerarm refinements have been constructed based on heat diffusion and thin shell simulation, respectively. The other quad models were coarsened based on curvature. Note that we included the entropy coded orientation bits for the conformization of triangle hierarchies as well when necessary. In particular, this concerns bones, heat_transfer, and femur, as these models 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 17%. The strategies of Section 2.3 nearly halve the number of symbols that have to be coded. If the mesh is not balanced, the 1-regularity optimization can not be applied. In this case, refinement bits can be entropy coded using the same context models as above, counting the green neighbors as unrefined.

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

			PMC			our		
	quad mesh	v	**geom**	**conn**	L_2	**geom**	**conn**	L_2
adaptive	blade	36358	9.31	2.14	55	3.63	0.28	52
	fandisk4	23657	5.50	0.33	58	2.70	0.07	45
	fertility	134351	8.11	2.22	55	1.61	0.32	53
	rockerarm	24460	8.52	1.46	61	3.49	0.23	49
	shoulder	77573	8.80	2.08	56	1.75	0.31	61
	torso	39086	9.23	2.10	58	2.72	0.31	54
uniform	blade	127234	4.48	0.00	55	1.02	0.00	58
	fandisk4	263682	3.61	0.00	52	0.72	0.00	40
	fertility	274426	4.15	0.00	55	0.68	0.00	58
	rockerarm	300546	5.06	0.00	61	1.20	0.00	70
	shoulder	94208	4.10	0.00	56	0.42	0.00	62
	torso	269826	4.03	0.00	58	0.35	0.00	46
	avg. (adaptive)		8.11	1.72	57	2.65	0.25	52
	avg. (uniform)		4.24	0.00	56	0.73	0.00	56

Table 5: *Connectivity and geometry rates for quadrilateral hierarchies in bits per vertex.*

the mesh structure is used to apply context based arithmetic encoding. The introduced context groups reduced the code length by approximately 50 %.

An evaluation of compression rates within each context group is 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 balances the mutual information available for the coding of each node. As a result, an average gain of 7 % for the overall compression rates could be achieved.

In Tables 4 and 5 we provide compression results for the connectivity as well as the geometry of our test models. Although the connectivity is trivial in the uniform case, we provide results for all models for which the finest level geometry information was also available.

Table 4 summarizes our results for the triangle multiresolution meshes. We are comparing our results to the single-rate Polygonal Mesh Compression [Ise02] (PMC) and to the progressive wavelet based coders Wavemesh and PGC. In the connectivity case, our scheme significantly outperforms PMC by an average factor of 2.0 and Wavemesh by an average factor of 3.8 for the adaptively refined meshes.

For geometry compression, we used uniform 12-bit quantization for PMC for all models. This is also true for the coarse base meshes needed by PGC and our coder. For Wavemesh, we used the current version 2.1 and chose for each model the options that provided the best results: uniform 12-bit quantization for the adaptive models, and zerotree encoding with lifting radius 1 for the uniform ones. Wavemesh was able to detect the subdivision connectivity of the uniform models and was thus able to handle the uniform models much better than the adaptive ones.

Overall, we exceed PGC codec by 7% on average for uniform multiresolution models, due to the use of improved context models for significance bits and we surpass PMC and Wavemesh by average factors between 2.0 and 3.0 for the uniform as well as the adaptive models.

Similarly, Table 5 presents compression results for quadrilateral hierarchies for our coder and PMC. One reason for the drastic drop in the connectivity rate is due to the fact that we encode face-based symbols and quadrilateral meshes have half as many faces per vertex. Evaluating the connectivity compression results, we exceed PMC by an average factor of 6.9 for the connectivity and 2.8/3.4 for the adaptive/uniform geometry, which demonstrates the efficiency of our scheme for quadrilateral multiresolution meshes.

Table 3 shows compression results for time-varying sequences. Overall, using the proposed extensions to the our static compression scheme, can improve compression rates significantly. Even though the extension is only on par with the static version if the refinement varies a lot (see cloth), we recommend using the extension due to its simple implementation.

Even though we have a non-optimized prototype implementation, the execution times in our tests were dominated by the times for loading and parsing the uncompressed models from the hard disk. After processing the small base mesh, connectivity encoding and decoding of a model is performed by a single traversal of the hierarchy in which each node is touched only once. Connectivity processing of a node merely involves finding its neighbors and children, arithmetic coding of one bit, and finding a node with highest priority. Each of those operations is done in constant time. This applies even to finding a highest priority node if the nodes of each priority are collected in separate doubly linked lists.

When coding the wavelet coefficients, every coefficient is visited once for each bit plane. Thus, geometry coding time is linear in the number of coefficients and the number of bit planes.

4.1. Further Extensions and Future Work

We took an in-depth look at connectivity compression, but only briefly covered geometry compression. Progressive geometry compression has been an active field of research and many tweaks have been reported to further increase geometry compression performance, but consideration of all of them is beyond the scope of this paper. Many of those tweaks can be transferred from the uniform to the adaptive case. For instance, second generation wavelets derived via the lifting scheme [SS95] have often been applied and we expect improved rate-distortion curves when using them in our scheme. However, a thorough treatment of lifting schemes on adaptive mesh hierarchies is still lacking. Empirical results of [DSM*10] further show that there are correlations between sign bits that can be exploited. Furthermore, we observed that the components of vector-valued wavelet coeffi-

cients are also correlated, making them suitable for further context modeling.

Extending the context model based on the significance of a coefficient in a prior time frame also enables us to take advantage of correlation in animated sequences. We expect similar gains for the geometry code as for our connectivity part.

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, Aim@Shape, Polygon Technologies and the Visible Human Project.

References

[AD01] ALLIEZ P., DESBRUN M.: Progressive compression for lossless transmission of triangle meshes, 2001.

[AMG05] AVILÉS M., MORÁN F., GARCÍA N.: Progressive lower trees of wavelet coefficients: efficient spatial and SNR scalable coding of 3D models. *Advances in Mulitmedia Information Processing-PCM 2005* (2005), 61–72.

[BSW83] BANK R., SHERMAN A., WEISER A.: Some refinement algorithms and data structures for regular local mesh refinement. *Scientific Computing, Applications of Mathematics and Computing to the Physical Sciences* (1983).

[BWK02] BOTSCH M., WIRATANAYA A., KOBBELT L.: Efficient high quality rendering of point sampled geometry. In *Proc. of the Eurographics workshop on Rendering* (2002), pp. 53–64.

[CC78] CATMULL E., CLARK J.: Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-Aided Design 10*, 6 (1978), 350 – 355.

[DLG90] DYN N., LEVINE D., GRAPHORY J. A.: A butterfly subdivision scheme for surface interpolation with tension control. *ACM Trans. Graph. 9*, 2 (1990), 160–169.

[DSM*10] DENIS L., SATTI S., MUNTEANU A., CORNELIS J., SCHELKENS P.: Scalable Intraband and Composite Wavelet-Based Coding of Semiregular Meshes. *IEEE Transactions on Multimedia 12*, 8 (2010), 773–789.

[GVSS00] GUSKOV I., VIDIMČE K., SWELDENS W., SCHRÖDER P.: Normal meshes. In *SIGGRAPH '00 Proceedings* (2000), pp. 95–102.

[IS06] ISENBURG M., SNOEYINK J.: Early-split coding of triangle mesh connectivity. In *Graphics Interface Proceedings* (2006).

[Ise02] ISENBURG M.: Compressing polygon mesh connectivity with degree duality prediction. In *Graphics Interface Conference Proceedings* (2002), pp. 161–170.

[JS99] JACK M. I., SNOEYINK J.: Mesh collapse compression. In *In Proceedings of SIBGRAPI'99* (1999), pp. 27–28.

[KG03] KHODAKOVSKY A., GUSKOV I.: Compression of normal meshes. In *In Geometric Modeling for Scientific Visualization* (2003), Springer-Verlag, pp. 189–206.

[KNP07] KÄLBERER F., NIESER M., POLTHIER K.: Quadcover - surface parameterization using branched coverings. *Computer Graphics Forum 26*, 3 (2007), 375–384.

[Kob96] KOBBELT L.: Interpolatory subdivision on open quadrilateral nets with arbitrary topology. In *Computer Graphics Forum* (1996), vol. 15, Wiley Online Library, pp. 409–420.

[KPRW05] KÄLBERER F., POLTHIER K., REITEBUCH U., WARDETZKY M.: Freelence - coding with free valences. *Computer Graphics Forum 24*, 3 (2005), 469–478.

[KSS00] KHODAKOVSKY A., SCHRÖDER P., SWELDENS W.: Progressive geometry compression. In *SIGGRAPH '00 Proceedings* (2000), pp. 271–278.

[LAD02] LEE H., ALLIEZ P., DESBRUN M.: Angle-analyzer: A triangle-quad mesh codec. vol. 21, pp. 383–392.

[Loo87] LOOP C.: Smooth subdivision surfaces based on triangles. Utah University, USA.

[LSS*] LEE A. W. F., SWELDENS W., SCHRÖDER P., COWSAR L., DOBKIN D.: Maps: multiresolution adaptive parameterization of surfaces. In *SIGGRAPH '98 Proceedings*.

[PA05] PAYAN F., ANTONINI M.: An efficient bit allocation for compressing normal meshes with an error-driven quantization. *CAGD 22*, 5 (2005), 466–486.

[Ros99] ROSSIGNAC J.: Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics* (1999), 47–61.

[Sai04] SAID A.: Introduction to arithmetic coding-theory and practice. *Hewlett Packard Laboratories Report* (2004).

[Sch96] SCHNEIDERS R.: Refining quadrilateral and hexahedral element meshes. In *5th International Conference on Grid Generation in Computational Field Simulations* (1996), CRC Press.

[Sha48] SHANNON C. E.: A mathematical theory of communication. *The Bell System Technical J. 27* (1948).

[Sha93] SHAPIRO J. M.: Embedded image coding using zerotrees of wavelet coefficients. In *IEEE Transactions of Signal Processing* (1993), vol. 41, pp. 3445–3462.

[SK06] SCHNABEL R., KLEIN R.: Octree-based point-cloud compression. In *Symposium on Point-Based Graphics 2006* (July 2006), Botsch M., Chen B., (Eds.), Eurographics.

[SMFF04] SETTGAST V., MÜLLER K., FÜNFZIG C., FELLNER D.: Adaptive tesselation of subdivision surfaces. *Computers & Graphics 28*, 1 (2004), 73–78.

[SPM96] SAID A., PEARLMAN W. A., MEMBER S.: A new fast and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology 6* (1996), 243–250.

[SS95] SCHRÖDER P., SWELDENS W.: Spherical wavelets: Efficiently representing functions on the sphere. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (1995), ACM, pp. 161–172.

[Szy02] SZYMCZAK A.: Optimized edgebreaker encoding for large and regular triangle meshes. In *DCC '02 Proceedings* (Washington, DC, USA, 2002), IEEE Computer Society, p. 472.

[TG98] TOUMA C., GOTSMAN C.: Triangle mesh compression. In *Graphics Interface Conference Proceedings* (1998).

[Tut62] TUTTE W.: A census of planar triangulations. *Canadian Journal of Mathematics 14* (1962), 21–38.

[VP04] VALETTE S., PROST R.: Wavelet-based progressive compression scheme for triangle meshes: Wavemesh. *IEEE Transactions on Visualization and Computer Graphics 10*, 2 (2004).

[WNC87] WITTEN I. H., NEAL R. M., CLEARY J. G.: Arithmetic coding for data compression. *Communications of the ACM 30*, 6 (1987), 520–540.