Improved Compression of Topology for View-Dependent Rendering

Christopher Zach[*] VRVis Research Center

Markus Grabner[†] TU Graz

Konrad Karner[‡] VRVis Research Center

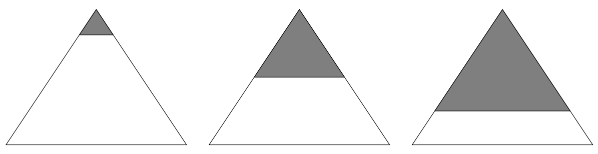
Abstract

We present a simple and efficient representation to store and transmit connectivity data of view dependent meshes for out-of-core rendering of large datasets. Resident mesh data available for rendering is organized as vertex tree to support real-time visualization of huge models. Our approach is not restricted to manifold meshes and can be used in the presence of non topology-preserving refinement operations as well. In contrast to progressive mesh compression our representation allows selective access to relevant fractions of the refinement hierarchy. After new mesh refinements are received, the update of the resident hierarchy available for display does not rely on known topology in the neighborhood of the patches to be refined. Therefore our approach does not require dependencies. We compare our connectivity coding with an existing framework for efficient transmission of view-dependent meshes and obtained substantially better compression results.

1 Introduction

Interactive visualization of huge 3D models, that fit only partially into main memory, requires enhanced rendering techniques. Additionally, there is an increasing need for network based visualization systems with a client/server architecture, e.g. in the context of geoinformation and virtual tourism. In our work we address the challenging task of displaying huge polygonal models in real-time using *view-dependent rendering* and *multiresolution* techniques. Since our system handles very large datasets in a client/server environment, special purpose compression methods for faster transmission of scene data over a network are required.

Single rate compression methods for geometry encode 3D geometry very efficiently, but the model can only be displayed on screen if all data was received. *Progressive compression and transmission* of geometric data display the 3D model initially at low resolution and gradually refine the displayed mesh when receiving new data. Therefore these methods avoid the latency until the 3D model is displayed on the client computer at some level of detail. Pure progressive compression increases the available resolution of the model uniformly, which yields to low visual quality, if the full model is substantially larger than available memory on the client computer (Figure 1(a)). Selective transmission of geometry data depending on the current and expected viewing parameters allow high quality visualization even if only a fraction of the model data fits into main memory (Figure 1(b)). Additionally, it is reasonable to utilize a view-dependent rendering (or level of detail) approach for already resident geometry data, since even the available model will be typically too large to be rendered in real-time at full resolution. Figure 2 illustrates the selective retrieval of geoemtry data for a terrain data set.


(a) View-independent progressive transmission

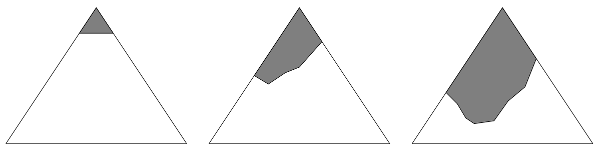
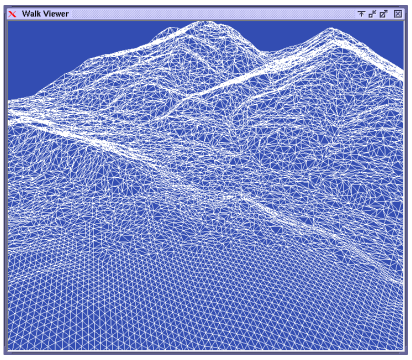

(b) View-dependent progressive transmission

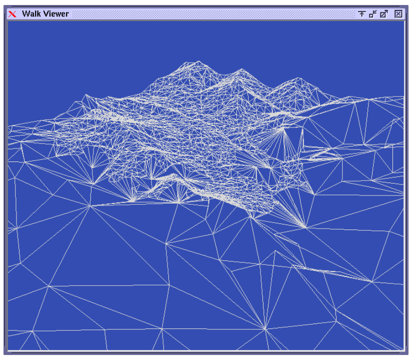
Figure 1: Progressive transmission of multiresolution models and view-dependent transmission. Shaded regions indicate resident portions of the vertex tree.

The main contribution of this work is an improved encoding for connectivity of view-dependent meshes over earlier work. We allow non-manifold meshes to be represented and non-topology-preserving mesh refinement operations are possible as well.

[*]zach@vrvis.at
[†]grabner@icg.tu-graz.ac.at
[‡]karner@vrvis.at



(a) High resolution data is available for the initial viewing position.



(b) A larger view on the model after disabling further retrieval of geometry. The in-memory model has a non-uniform resolution.

Figure 2: Selective paging of a terrain dataset.

2 Related Work

2.1 View-Dependent Simplification

The key prerequisite for view–dependent mesh visualization is the generation of a multiresolution data structure containing a set of smooth levels of detail of the original model. One popular multiresolution concept is the *progressive mesh* [Hoppe 1996], which is essentially a sequence of meshes with successively lower geometric accuracy and complexity. Starting with the original mesh the next sequence element is obtained by applying one *edge collapse* operation (Figure 3) to remove one edge and several (usually two) triangles from the previous mesh. The sequence of edge collapses is chosen such that the overall shape of the model is preserved. Edge collapses are performed until the mesh distortion (according to some quality metric) is larger than some threshold. A successful metric to guide the simplification procedure is the quadric error metric proposed by Garland and Heckbert [Garland and Heckbert 1997].

Several authors observed that the linear sequence of edge collapses can be generalized to a partial ordering represented by the *vertex tree* [Floriani et al. 1997; Hoppe 1997; Pajarola 2001; Xia and Varshney 1996]. Selective refinement at runtime determines the currently displayed mesh. Our framework is based on `VDSlib` [Luebke 2002], which is described by Luebke and Erikson [Luebke and Erikson 1997]. Figure 4 gives an overview of the most important data structure, the *vertex tree*. At runtime the nodes in the vertex tree fall into three categories: *active nodes*, which contain the currently rendered triangles (called *subtriangles* or *subtris* for short), *boundary nodes*, which correspond to vertices in the displayed mesh, and *inactive nodes*. For each frame to render, the vertex tree is traversed in top down order and active nodes are determined according to some screen error metric.

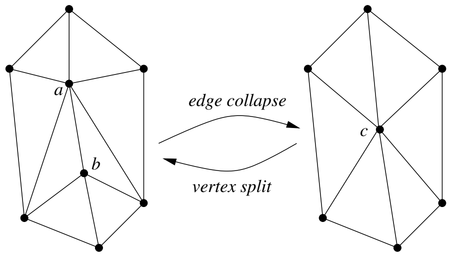


Figure 3: Edge collapse operation. The vertices a and b are collapsed into vertex c, therefore removing one edge and 2 triangles. The inverse operation, *vertex unfolding* (or *vertex split*), is applied at runtime for mesh refinement.

2.2 Geometry Compression and Streaming

The necessity to transmit large geometric models over slow network connections has initiated much research on geometry compression. Early work addressed single-rate, non-progressive compression of static 3D models [Deering 1995; Rossignac 1999; Taubin and Rossignac 1998; Touma and Gotsman 1998]. Although these compression techniques are very effective, the non-progressive nature introduces a significant latency until the model can be displayed on the target computer. In order to reduce this lag progressive transmission methods for geometry were developed. These methods allow display of a coarse model very quickly after the transmission

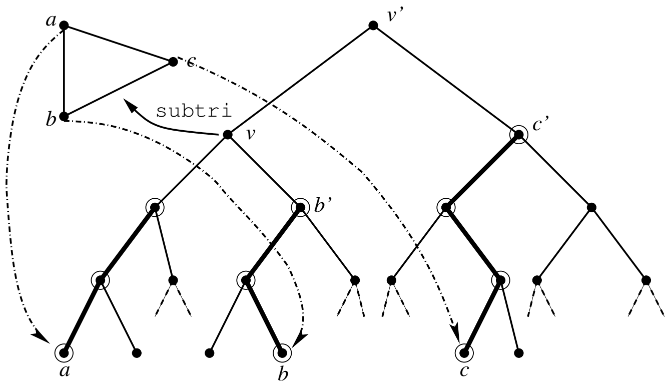


Figure 4: The vertex tree as found in `VDSlib`. Every node stores geometric data (point coordinate, color, a bounding sphere etc.), its children and a list of associated triangles, the so called *subtris*. The triangle (abc) is a subtri of v, because it exists only if v is unfolded at runtime. The nodes emphasized with a circle are possible corners of (abc) chosen for rendering. v', b' and c' are explained in Section 4.

has been initiated and refine the displayed model as soon as more data is received. Progressive Meshes allow a somewhat compact external representation [Hoppe 1996; Hoppe 1998]. Cohen-Or et al. [Cohen-Or et al. 1999] describe a progressive encoding of geometry based on successive vertex insertion and graph coloring to identify affected triangles. Giving up some flexibility of the original progressive mesh approach, grouping several mesh updates into batches yields to even more compact representation [Pajarola and Rossignac 2000; Taubin et al. 1998]. Alliez and Desbrun [Alliez and Desbrun 2001] achieve very low bitrates using a valence-driven vertex decimation method. Devillers and Gandoin [Devillers and Gandoin 2000; Gandoin and Devillers 2002] tightly couple connectivity compression with progressive coding of vertices using spatial subdivision methods.

Meshes with regular topology can be converted into a multiresolution representation based on wavelets [Eck et al. 1995]. Wavelets provide a general tool for progressive and compressed transmission of arbitary data. Khodakovsky et al. [Khodakovsky et al. 2000] present a progressive mesh encoding scheme based on wavelet transformation.

None of the above mentioned compression techniques allow selective transmission of refinement updates based on the current or predicted future viewing parameters, since the resident mesh is refined uniformely. For very huge models like large terrains and other outdoor environments, it is reasonable to refine the mesh near to the current viewing position sooner. Additionally we explicitly allow unbalanced refinement hierarchies resident in memory to store only relevant parts of the hierarchy in the clients memory.

Wang and Li [Wang and Li 2000] propose an octree-based clustering method to obtain a multiresolution model similar to the VDS representation. In order to transmit fractions of the model depending on current viewing parameters, they employed an absolut path coding of vertices. Yang et al. [Yang et al. 2001] allow partially view-dependent transmission of progressive meshes by splitting the base mesh in partitions, which can be transmitted independently.

Our work is a significantly refined and optimized version of the CAME topology coding approach [Grabner 2002]. The CAME framework stores triangle connectivity similar to the VDS framework, but uses paths through the simplification hierarchy to access mesh vertices instead of node IDs. Hence, mesh vertices are identified by bit strings indicating the path to be taken in the simplification hierarchy to reach the leaf node corresponding to the vertex. Each bit in the string identifies the branch to be taken after a vertex split. Topology compression is achieved by omitting redundant prefixes of bit strings. A slightly improved variant of connectivity coding overcoming some limitations found in the CAME framework is described in Section 4 in more detail. In this paper we address solely the compression of connectivity information. A improved method to encode vertex positions based on the CAME framework can be found in [Grabner and Zach 2003].

3 Dependency-Free Transmission

With sufficiently known topology in the neighborhood of the considered refinement, compact encoding of updates (vertex splits) is possible by indicating cut edges. Determined topology can be enforced by dependencies between refinements to narrow the possible sequences of refinements. If we assume an average valence of 6 for every mesh vertex, approximately 5 bits are sufficient to encode the connectivity update for a vertex split [Hoppe 1996]. This compact representation requires dependency data to be encoded additionally. View-dependent progressive meshes [Hoppe 1997] and merge trees [Xia and Varshney 1996] rely on these dependencies at run-time, therefore this data must be transmitted anyway.

The VDS framework [Luebke and Erikson 1997] does not require dependencies between vertex nodes in general, but dependencies can be beneficial for image quality, since the probability of run-time mesh foldovers is reduced. Typically only thin or otherwise fragile triangles are worth to add dependencies to avoid run-time artefacts. Therefore it is reasonable to insert vertex dependencies only for selected nodes. Without dependencies throughout the refinement hierarchy the neighborhood of a vertex ready to split is only known partially, therefore compact cut edge coding is inhibited.

Single refinements are usually grouped into *meta nodes* [El-Sana and Chiang 2000] to be more efficiently transmitted over a network. Within a framework relying on dependencies a predefined clustering of nodes based on the vertex tree can result in some waste of transmission time, since refinements that depend on non-resident vertices cannot be decompressed. It is unclear whether this is a severe penalty. Nevertheless, the virtual memory management required for networked out-of-core rendering frameworks is substantially simplified in the absence of dependencies. Hence, we assume in this work that no or only few dependencies are transmitted and we focus purely on encoding the connectivity of triangles.

4 Relative Encoding of Triangle Topology

In this section we describe the basic method to encode triangle vertices efficiently for an external memory representation. This basic method is loosely based on the encoding used in the CAME framework [Grabner 2002].

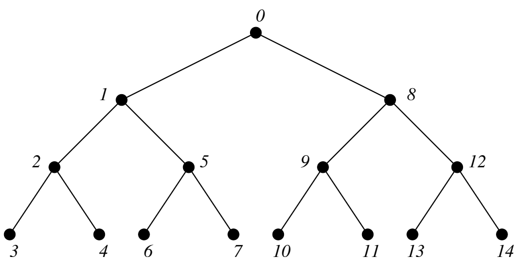
Unlike view-independent progressive compression schemes view-dependent meshes require index-based specification of triangle vertices. The methods to encode vertex splits compactly as described by Hoppe [Hoppe 1998] and Pajarola and Rossignac [Pajarola and Rossignac 2000] are not applicable for view-dependent representations, since the topology in the neighborhood of the vertex to split is only partially known. The knowledge of surrounding connectivity depends on the parts of the vertex hierarchy already transmitted.

4.1 Relative Vertex ID Coding

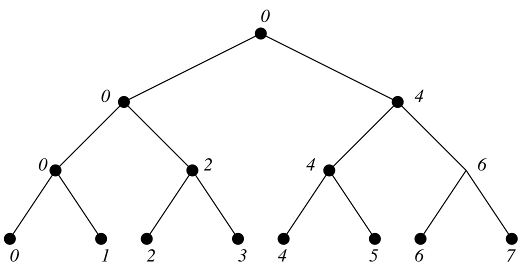
Referring to Figure 4, the subtriangle (abc) of a node v is represented by its three vertices (node IDs) pointing to the appropriate leaves in the vertex tree. Node IDs are assigned using a depth-first traversal (see Figure 5(a)). At least two of these corners (a and b) are leaves in the subtree of the node v owning the subtriangle. We call these corners *internal nodes* (wrt. the owning node). Therefore it is reasonable to encode the relative path from v to the leaf vertices [Grabner 2002]. Instead of directly encoding the relative paths, we employ relative coding of node IDs. Internal corners are stored as the difference $n.id - v.id$ (here n is either a or b). These differences are finally entropy coded. By the numbering scheme for node IDs, this difference is always positive.

The third corner c of the triangle is not in the subtree of v. We call this node *external*, and relative coding is slightly different. The leaf node c belongs to the subtree of the common ancestor v' of v and c (recall Figure 4). Therefore the external node c is represented as the pair $(h, c.id - v'.id)$, where h is the height difference between v' and v. These values are again entropy coded in a final pass.

During the vertex tree construction the assigned node IDs are based on a preorder traveral (see Figure 5(a)). At runtime this unique numbering can be relaxed and we can improve incremental coding by the modified numbering scheme illustrated in Figure 5(b). For binary vertex trees the achieved benefit is about one bit per vertex, since the range of ID values is effectively halved.



(a) VDS node numbering



(b) Compact node numbering

Figure 5: Node numbering in the VDS framework and the ID assignment scheme used for improved relative coding.

We achieve even higher compression if the following considerations are taken into account: if corner a is already encoded, b must be in the right subtree of v. Therefore the ID of b is at least the ID of the leftmost leaf b' in the right subtree of v (Figure 4) and we only need to encode the difference $b'.id - b.id$. In practice this additional improvement saves 1.5 to 2 bits per triangle. Note that there is only a benefit if the first encoded vertex is in the left subtree. The improvement can be applied to the external node c.

So far this encoding method does only exploit the coherence of triangle corners within the vertex tree, but it does not exploit mesh connectivity found in the original 3D model. Nevertheless this incremental coding of node IDs alone yields to substantial reduction of connectivity data compared with an indexed face set representation (14.9 bits per vertex on average instead of $3 \times 17 = 51$ bits for the crater model).

4.2 Entropy Coding of Difference Values

Since the range of differences between node IDs is rather large, but concentrated near small values, we utilize a variation of Huffman coding in order to avoid very large dictionaries: we store the 511 most frequent difference values in the Huffman table and the 512th code represents the prefix for all other differences, which are encoded directly with the appropriate number of bits. This encoding is only slightlier inferior to true Huffman coding, but has a bounded size of the dictionary to transmit.

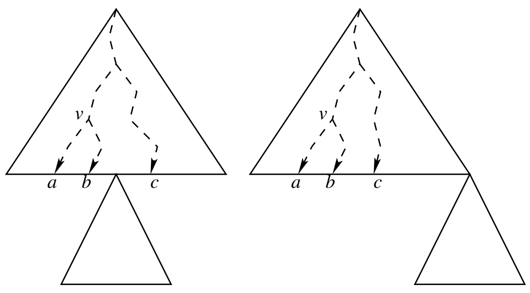
Additionally we tested separate Huffman tables for encoding differences of internal and external vertices. In these cases the allowed table size is halved to obtain a fair comparison. Separate tables yield to a reduction of 0.5 bits per triangle.

4.3 Restructuring the Vertex Tree

Without changing the semantics of the vertex tree the successors of a node may be permuted arbitrarily. The distribution of difference values depends on the shape of the vertex tree as seen in Figure 6. Rearranging the tree such that deeper parts move to the right end proved beneficial in all our experiments. Although the gain is rather marginal in most cases (0.3–0.5 bits/triangle), it is a simple enhancement with no additional memory costs.

4.4 Path Coding versus ID Difference Coding

Unlike the CAME framework [Grabner 2002] we explicitly allow unbalanced vertex trees in our approach. Binary (relative) paths form another strategy of node identification, but the range of assigned values is sparse for unbalanced trees. Highly unbalanced trees can appear in practice, e.g. if the scene database has large variations in scale. A mostly uniform height field representing a large area might be enhanced with highly detailed local 3D models. In such cases the restriction of a maximum path length of 52 nodes (as found in the current CAME implementation) may be too restrictive. For these reasons we decided to assign node IDs from a compact range of values, which proved somewhat superior to relative path coding.



(a) Large difference value to encode c (b) Small difference value to encode c

Figure 6: The effect of restructuring the vertex tree to move deep subtrees to the right tail. The triangle (a, b, c) is a subtri of v. The difference $c.id - v.id$ is much larger in tree (a) than in (b).

5 Improved Coding of Internal Vertices

Relative coding of triangle topology as described in the previous section still requires about 15 bit per triangle, which is substantially more than required for pure progressive coding. In this section we argue that encoding and transmitting only the external corner of each subtriangle is sufficient and the two internal corners can be mostly omitted. Unfortunately these improvement does not reduce the encoded size to one third, since referencing external nodes is much more expensive than referring to internal corners.

When a vertex split $u \rightarrow (u_1, u_2)$ and external corners v_l and v_r are received by the client, typically two new triangles $\Delta(u_1, u_2, v_l)$ and $\Delta(u_2, u_1, v_r)$ are available for rendering (see Figure 7).

Triangles adjacent to u must be updated to reflect the refined corners. Adjacent triangles fall into two categories:

1. Triangles, that reference u as external corner.
The update of affected corners is done implicitly by the run-time mesh selection procedure, since external corners specify the full path to the corresponding leaf vertex. Notice that nodes owning such triangles

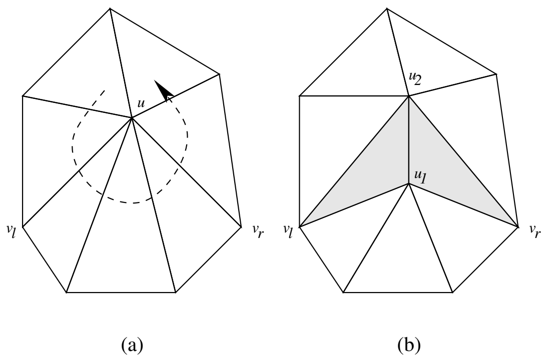


Figure 7: Triangles adjacent to u are affected by the vertex split $u \rightarrow (u_1,\, u_2)$.

are not ancestors of u.

2. Subtriangles owned by ancestor nodes of u.
   These subtris refer to the appropriate leaf vertex below u only implicitly and the correct substitution $u \rightarrow u_1$ or $u \rightarrow u_2$ for the corner must be determined.

If we presume an orientable manifold, all triangles adjacent to u (either by referencing a leaf node in the subtree below u or only implicitly referencing u) can be arranged in an oriented triangle fan (see Figure 7(a)). If u is on the boundary, the fan does not constitute a closed loop. The correct update information can be distributed across this triangle fan. In order to avoid the time consuming search for triangles currently adjacent to u, faces need to store pointers to neighboring triangles (similar to the `Face` structure used in the efficient implementation of progressive meshes [Hoppe 1998]). These pointers must be updated if new refinements are received by the client or unused parts of the vertex tree are removed from main memory.

This approach requires strictly orientable 2-manifolds as source mesh and only topology preserving edge collapses are permitted. Since we allow general input meshes and do not restrict edge collapses (and explicitly permit general vertex pair contractions), we encode the required control information to correctly update adjacent subtris of parent nodes directly. The affected subtriangles of ancestor nodes can be serialized into a linear sequence, therefore the mapping of a control bit to the target subtri is unique. Additionally, the number of control bits needs not to be encoded. In our experiments with mostly manifold meshes about 4 control bits are necessary for every refinement, which implies that two bits per triangle are additionally required. Figure 8 and Figure 9 give a small example, how adjacent triangles of parents are affected and encoded by vertex splits.

Finally, we give details on our implementation for the

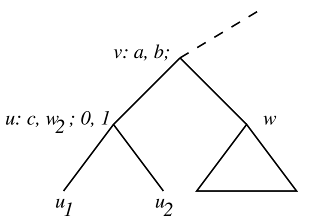


Figure 8: The control bits associated with the nodes. The notation $u\,:\,c, w_2; 01$ inserts the two triangles $(u_1 c u_2)$ and $(u_1 u_2 w_2)$ and updates the subtris of its parent v appropriately.

data encoded for the refinement $u \rightarrow (u_1,\, u_2)$: Usually the two inserted subtriangles are encoded as a pair of external vertices (see Section 4). We extend the domain of the height difference h with two new symbols to represent non-existent subtris and non-manifold updates introducing more than 2 triangles.

6 Summary of the Method

In this section we summarize the proposed encoding method. The vertex tree obtained by the mesh simplification procedure is initially reordered as described in Section 4.3. Subsequently, node IDs are assigned to each leaf. A manifold vertex split $v \rightarrow (v_1, v_2)$ creating two new triangles (v_1, v_2, a) and (v_2, v_1, b) is encoded as the bit string $h_a, a.id - v_a.id; h_b, b.id - v_b.id; control\_bits$. The height differences h_n and the ID differences $n.id - v_n.id$ of the external vertices $n \in \{a, b\}$ are entropy coded (with separate dictionaries). v_n denotes the common ancestor of v and n (recall Section 4). The control bits are determined as explained in Section 5. It is not necessary to store the number of control bits explicitly, since this number can be reconstructed from the already decoded vertex tree. The dictionary of height differences is extended by special symbols indicating non-manifold vertex splits. This initial symbol is followed by the number of triangles to decode. The triangles in the non-manifold case are encoded explicitly with two internal and one external corner. Since non-manifold splits are enountered only infrequently (e.g. 227 out of 34800 nodes for the bunny model), a more efficient encoding is not required.

7 Results

We tested our connectivity coding on several geometric datasets with various complexities. The first columns in table 1 gives an overview on the complexity of the evaluated models. The Bunny, Hand and Dragon models were obtained from the Large Geometric Models Archive of the Georgia Institute of Technology. The Crater dataset

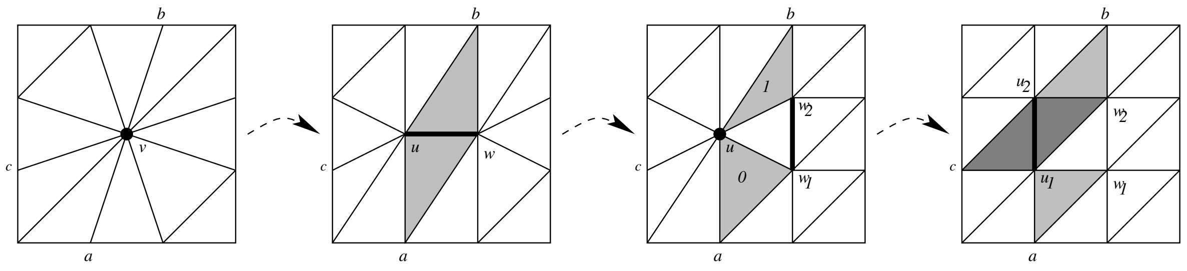


Figure 9: A small sequence of vertex refinements. Dark shaded triangles indicate subtris of u (splitted into u_1 and u_2). Lighter triangles depict subtriangles of the parent v, which are adjacent to u and hence control bits are required for correct updates. The numbers in the light triangles determine, how they need to be adjusted. '0' represents the replacement of u by u_1, whereas '1' indicates the substitution $u \rightarrow u_2$. With this information the triangles (uaw) and (uwb) can be updated correctly. Note, that the triangle adjacent to u and inserted with the second vertex split references u as external corner.

is provided with the QSlim mesh simplification software by M. Garland. The inner city and large city datasets consist of an urban 3D model with detailed roofs and flat facades. The DEM model showing an urban area was obtained by an airborne laser scanner.

The vertex trees for these models were generated with the quadric error metric, if not noted otherwise. The obtained compression results for various encoding techniques for view-dependent meshes are summarized in Table 1. We evaluated relative path compression found in CAME, relative ID coding (Section 4), and our proposed coding (Section 5). Additionally we specify the maximum depth of a node in the corresponding vertex tree for each model to show the influence of balancing on compression results.

The presented numbers of bits per triangle refer to encoding only connectivity information without the structural data required to recover the tree structure. The benefit of encoding ID differences (7th column) instead of relative paths (6th column) varies with the dataset, but the gain of external corner coding (last column) is usually 3 bits per triangle (with the exception of the city datasets).

8 Analysis and Discussion

8.1 Analysis for Internal Vertices

In this section we argue, that the observed 2 bits per triangle to encode internal vertices are no coincidence. The proof is based on a simple counting argument: If we consider a balanced vertex hierarchy with height d, the number of subtris (for 2-manifold meshes) is $2 \times (2^d - 1)$. At level k in the hierarchy ($k < d$), 2×2^k triangles refer to $2 \times 2 \times 2^k$ internal vertices. Furthermore, for triangles at level k, $d - k - 1$ control bits are required at successor nodes. Summation over all levels yields, that

$$4 \sum_{k=0}^{d-1} 2^k (d-k-1) = 4\,(2^d - d - 1)$$

control bits are required in total. Consequently,

$$\frac{4 \times (2^d - d - 1)}{2 \times (2^d - 1)} \rightarrow 2$$

bits are required to encode the internal vertices of each subtriangle. In summary, encoding internal vertices for (mostly) 2-manifold meshes and somewhat balanced hierarchies requires about 1 bit per internal vertex.

8.2 Worst-Case Results for External Vertices

The figures given in Table 1 suggest that the number of bits to encode one triangle are relatively constant for all datasets regardless of the model complexity. In this section we disprove the conjecture, that in fact a constant number bits is required to encode connectivity of view-dependent meshes.

In Figure 10 an example for an input mesh and corresponding vertex trees are given. Note, that the mesh bears some resemblance to a bipartite graph. The vertex trees presented in Figure 10(b) and (c) can actually be generated by a reasonable error metric (e.g. using the edge length as quality metric), if the vertex positions are adjusted accordingly. In this case all vertices on the left and on the right constitute separated subtrees below the root node and references to external corners are inefficient.

The vertex trees for this (somewhat artificial) example allow two conclusions: At first, even balanced vertex trees do not guarantee a constant number of bits per triangle to encode topology, and the number of bits per triangle required with relative path coding can be linear in the

Dataset name	#vertices	$\lceil \log_2 \#\text{vertices} \rceil$	#triangles	Depth	CAME	Relative ID	Proposed
Bunny	34834	16	69451	21	18.48	15.08	12.12
Bunny^a				27	18.80	14.01	11.06
Bunny^b				18	16.00	14.21	11.21
Crater	100000	17	199114	26	18.04	14.93	11.95
Inner city	72186	17	162146	132	18.99	12.03	12.32
Large city	144243	18	248755	128	19.06	11.45	10.70
Skeleton hand	327323	19	654666	118	18.41	15.32	12.23
DEM	361176	19	720000	29	19.99	15.49	12.39
Dragon	437645	19	871414	30	18.96	15.25	12.21
Dragon^a				48	19.51	13.70	10.91
Dragon^b				24	16.71	14.37	11.33

Table 1: Compression results for the evaluated datasets. The figures represent the required number of bits per triangle. The vertex hierarchies were created using the quadric error metric. Additionally the bunny and dragon model were simplified using the edge length metric (indicated by ^{a}) and a refined edge length metric including a balancing factor to penalize unbalanced hierarchies (denoted with ^{b}).

number of mesh vertices n (instead of $O(\log n)$ for an indexed representation without relative coding). Notice that the second observation applies only to path coding, since in the given example the relative paths comprise a sparse subset of bit strings. Node ID coding requires always $O(\log n)$ bits in the worst case.

9 Conclusions and Future Work

We have presented an improved method to encode view-dependent meshes for selective and efficient transmission over slow connections. Received refinements can arrive in arbitrary top-down order, since decompression is only dependent on ancestor nodes. Dependencies between refinement operations are only required on demand for improved visual quality to avoid run-time mesh foldovers.

In this work we have only addressed efficient encoding of connectivity data and we did not discuss compression of vertex positions and other attributes. Appropriate compression of vertex positions depends on the actual mesh simplification procedure, e.g. half-edge collapses [Pajarola 2001] benefit possibly from a different position coding method than optimal placement strategies (e.g. found in the quadric error metric approach [Garland and Heckbert 1997]). We suppose that geometry coding is largely orthogonal to connectivity compression in our approach. Further, we would like to emphasize, that our method works on given view-dependent meshes and encoding is not tightly coupled with the generation of the multiresolution representation (as opposed to many progressive methods).

There is still a significant gap between the performance of progressive mesh coders and our method. Gandoin and Devillers [Gandoin and Devillers 2002] report less than 2 bits per triangle for manifold surfaces in contrast to estimated 10–12 bits per triangle required by our approach (for mostly manifold meshes). Progressive methods cannot be directly compared with our approach, since we aim on efficient transmission of meshes suitable for view-dependent rendering at the client computer. Nevertheless, we expect improved compression methods for view-dependent meshes in the future.

10 Acknowledgements

We would like to thank Ivana Kolingerova and Sebastian Krivograd for helpful discussions.

This work has been done in the VRVis research center, Graz and Vienna/Austria (http://www.vrvis.at), which is partly funded by the Austrian government research program Kplus.

References

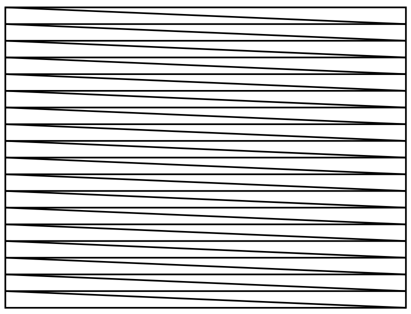
ALLIEZ, P., AND DESBRUN, M. 2001. Progressive compression for lossless transmission of triangle meshes. In *Proceedings of SIGGRAPH 2001*, 195–202.

COHEN-OR, D., LEVIN, D., AND REMEZ, O. 1999. Progressive compression of arbitrary triangular meshes. In *Proceedings of IEEE Visualization '99*, 67–72.

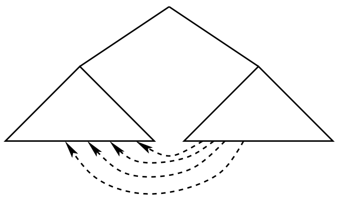
DEERING, M. 1995. Geometry compression. In *Proceedings of SIGGRAPH '95*, 13–20.

DEVILLERS, O., AND GANDOIN, P.-M. 2000. Geometric compression for interactive transmission. In *Proceedings of IEEE Visualization 2000*, 319–326.

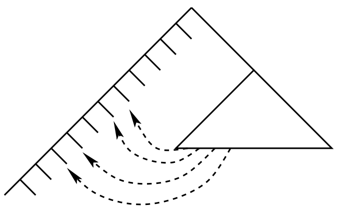
ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. 1995. Multiresolution analysis of arbitrary meshes. In *Proceedings of SIGGRAPH '95*, 173–182.



(a)



(b) Logarithmic tree depth



(c) Linear tree depth

Figure 10: The structure of a worst case mesh and possible vertex trees. The dashed arrows indicate references to external corners.

EL-SANA, J., AND CHIANG, Y.-J. 2000. External memory view-dependent simplification. In *Proceedings of Eurographics 2000*, 139–150.

FLORIANI, L. D., MAGILLO, P., AND PUPPO, E. 1997. Building and traversing a surface at variable resolution. In *IEEE Visualization'97*, 103–110.

GANDOIN, P.-M., AND DEVILLERS, O. 2002. Progressive lossless compression of arbitrary simplicial complexes. *ACM Transactions on Graphics (TOG) 21*, 3, 372–379.

GARLAND, M., AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH '97*, 209–216.

GRABNER, M., AND ZACH, C. 2003. Adaptive quantization with error bounds for compressed view-dependent multiresolution meshes. In *EUROGRAPHICS 2003, Short Presentations*, 77–82.

GRABNER, M. 2002. Compressed adaptive multiresolution encoding. *Journal of WSCG 10*, 1, 195–202.

HOPPE, H. 1996. Progressive meshes. In *Proceedings of SIGGRAPH '96*, 99–108.

HOPPE, H. 1997. View-dependent refinement of progressive meshes. In *Proceedings of SIGGRAPH '97*, 189–198.

HOPPE, H. 1998. Efficient implementation of progressive meshes. *Computers and Graphics 22*, 1, 27–36.

KHODAKOVSKY, A., SCHRÖDER, P., AND SWELDENS, W. 2000. Progressive geometry compression. In *Proceedings of SIGGRAPH 2000*, 271–278.

LUEBKE, D., AND ERIKSON, C. 1997. View–dependent simplification of arbitrary polygonal environments. In *Proceedings of SIGGRAPH '97*, 199–208.

LUEBKE, D., 2002. View-dependent simplification library. http://vdslib.virginia.edu.

PAJAROLA, R., AND ROSSIGNAC, J. 2000. Compressed progressive meshes. *IEEE Transactions on Visualization and Computer Graphics 6*, 1, 79–93.

PAJAROLA, R. 2001. Fastmesh: Efficient view-dependent meshing. In *Proceedings Pacific Graphics 2001*, 22–30.

ROSSIGNAC, J. 1999. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics 5*, 1, 47–61.

TAUBIN, G., AND ROSSIGNAC, J. 1998. Geometric compression through topological surgery. *ACM Transactions on Graphics 17*, 2, 84–115.

TAUBIN, G., GUEZIEC, A., HORN, W., AND LAZARUS, F. 1998. Progressive forest split compression. In *Proceedings of SIGGRAPH '98*, 132–132.

TOUMA, C., AND GOTSMAN, C. 1998. Triangle mesh compression. In *Proceedings of Graphics Interface*, 26–34.

WANG, H., AND LI, J. 2000. OctMesh - interactive mesh browsing over the internet. In *Int'l Conference on Information Technology: Coding and Computing (ITCC'00)*, 104–108.

XIA, J. C., AND VARSHNEY, A. 1996. Dynamic view-dependent simplification for polygonal models. In *IEEE Visualization '96*, 335–344.

YANG, S., KIM, C.-S., AND KUO, C.-C. K. 2001. View-dependent progressive mesh coding for graphic streaming. In *Proc. SPIE ITCOM*.