Compressed Adaptive Multiresolution Encoding

Markus Grabner

Institute for Computer Graphics and Vision
Graz University of Technology, Inffeldgasse 16/II
8010 Graz
Austria
e-mail: grabner@icg.tu-graz.ac.at

ABSTRACT

A new data structure *compressed adaptive multiresolution encoding* (CAME) for efficient storage of adaptive multiresolution non-manifold triangle meshes is presented. Unlike previous methods, CAME offers both data compression and view-dependent simplification. Triangle vertices are referenced by their relative path through the vertex hierarchy, therefore no neighborhood relations need to be stored to perform local mesh transformations (vertex split and edge collapse). Mesh dependencies are shown to be highly redundant and can be encoded with little overhead. CAME is built upon the *meta-node tree* recently introduced by El-Sana and Chiang (*External Memory View-Dependent Simplification*, in *Proceedings Eurographics 2000*), but compresses this data structure by a factor of 17.
The new method also offers external memory scene navigation, progressive transmission, and asynchronous access to the scene database.

Keywords: Multiresolution, adaptive LOD selection, compression

1 Introduction

Distribution of large three-dimensional datasets over the internet is becoming increasingly important. Applications include virtual habitat (large urban environments), virtual archaeology (ancient sites reconstructed at high detail), and virtual engineering (distributed CAD). The main tasks to be performed by such systems are data storage (which usually involves some kind of compression), data transmission (which usually involves decompression), and visualization.

3D data acquisition systems always seemed to be at least one step ahead of graphics hardware. This is also true today, current high-resolution models containing hundreds of millions polygons (e.g., the Digital Michelangelo Project [Levoy99]) by far exceed rendering capabilities even of high-end graphics workstations.

It is common to all of these and other similar applications that (due to the amount of data) the user can investigate only a small fraction of the scene at any time. Therefore it is possible (and required for performance reasons) to reduce the amount of data to be transmitted and displayed without degrading image quality perceived by the user. This is referred to as *view-dependent simplification.*

Furthermore it is useful to transmit a coarse approximation first to give the user an early impression of the requested model, which is then refined incrementally (*progressive transmission*). One major concern of mesh simplification algorithms is the ability to deal with arbitrary non-manifold triangle meshes as input (such as [Garla97, Luebk97, ElllS00]).

Compression techniques help reducing storage requirements and transmission times, but do not offer adaptive level of detail required for efficient visualization. On the other hand, multiresolution techniques don't care too much about memory consumption (see Figure 1 and Table 1). In this paper, a data structure is proposed that is equally well suited for storage, transmission and

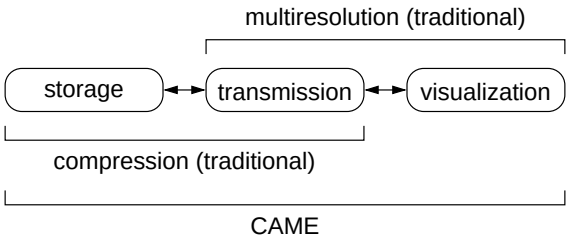


Figure 1: The scope of traditional compression and multiresolution techniques compared with our approach

visualization of large models. In particular, it offers both *compression* at a factor of 1:7 over conventional VRML files and a *hierarchical* scene description including all information required for view-dependent simplification. Since neighborhood relations between triangles are not explicitly represented in CAME, correct processing of non-manifold meshes comes at no extra cost.

2 Related work

2.1 View-dependent simplification

In the 90's, *continuous level-of-detail* algorithms were developed to better account for changing viewing parameters (e.g., viewer's position and orientation, illumination). Most notably, the *progressive meshes* representation introduced by Hoppe [Hoppe96] allows to store any two-manifold triangle mesh as a coarse *base mesh* together with a sequence of *detail records*.

By allowing to perform simplification and refinement operations in arbitrary order, the mesh can be adjusted selectively according to the current viewing parameters. Hoppe gives several refinement criteria (view-frustum, surface orientation, and screen-space geometric error), which are evaluated to decide if a vertex needs to be split. Xia and Varshney build a *merge tree* [Xia96] in a bottom-up fashion. They include illumination into their set of refinement criteria.

Only recently, edge-collapse based multiresolution algorithms were extended to handle non-manifold meshes. Garland and Heckbert define *pair contractions* [Garla97] allowing to join unconnected portions of the mesh. Popović and Hoppe introduce *progressive simplicial complexes* [Popov97], which always end up with a single vertex as the coarsest representation of the input mesh. El-Sana and Varshney introduce the *view-dependence tree* [ElllS99] which enables both topology modification and view-dependent simplification.

2.2 Fold-over prevention

To avoid the mesh folding over itself, mesh dependencies must be investigated (see [Xia96] for details). Different strategies have been proposed to ensure consistent meshes. Hoppe requires some vertices [Hoppe96] or faces [Hoppe97] adjacent to the affected region to be present in the mesh before proceeding with simplification or refinement.

The face dependency conditions in [Hoppe97] can be expressed as a *directed acyclic graph* (DAG) of vertex-to-vertex dependencies as shown in [Grabn00]. A proof is given that triangle adjacencies are maintained correctly within this framework without being considered explicitly.

El-Sana and Varshney introduce *implicit dependencies* [ElllS99] allowing more compact storage of dependency information. Moreover, their dependency tests are localized and require only a small part of the mesh to be examined. This algorithm is also used in [ElllS00] to provide navigation through virtual worlds exceeding main memory size.

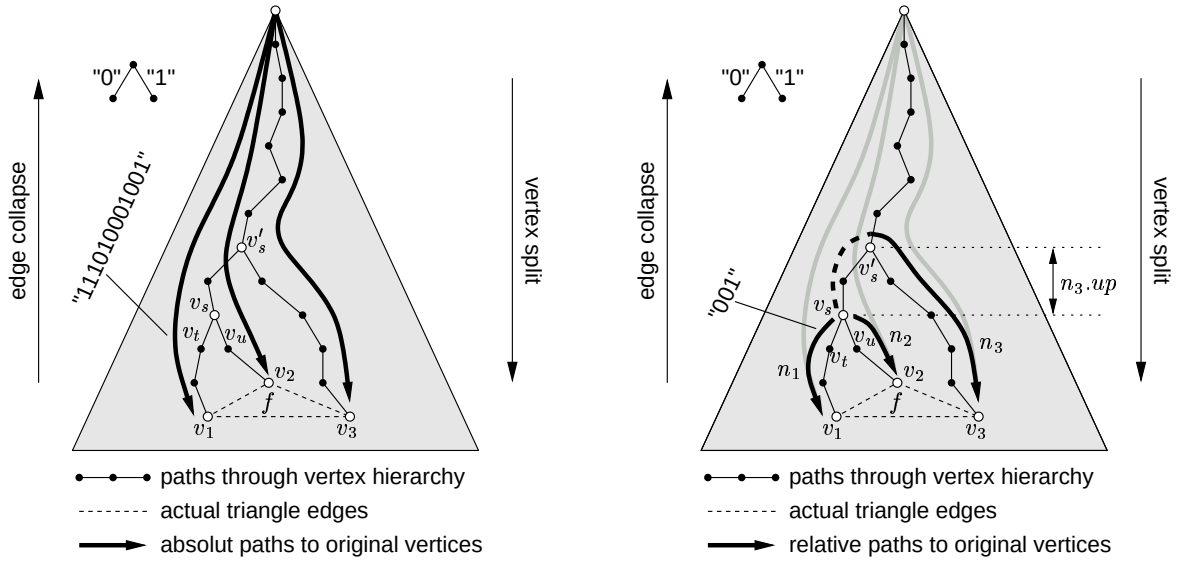
2.3 Mesh compression

Several techniques exist to reduce redundancy of triangle mesh data for compact storage. Hoppe encodes the sequence of vertex splits in an efficient way [Hoppe98]. Pajarola and Rossignac improve these results by grouping vertex splits into *batches* [Pajar00].

Efficient compression algorithms include the *edgebreaker* algorithm by Rossignac [Rossi99]. Starting at an arbitrary edge, the mesh is traversed in a spiral-like way. The topological relation of each triangle to previously processed parts of the mesh is encoding by less than two bits on average.

For geometry compression, Taubin and Rossignac use a *vertex predictor* that takes into account the K most recently encoded vertices [Taubi98]. Instead of storing absolute vertex coordinates, only the *correction vector* between the predicted and the actual position of a vertex is encoded. Predictor parameters are estimated to minimize the least square error of all correction vectors.

A valence-drive approach to mesh compression has been presented in [Allie01b]. It has been extended to support progressive transmission in [Allie01a].



(a) absolut

(b) relative

Figure 2: Triangle f and the hierarchy paths to its vertices v_1, v_2, and v_3

3 The approach

The novelty of our approach is the way mesh vertices are referenced by triangles such that triangle adjacency relations are maintained implicitly. In fact, neither at the decompression stage nor at the rendering stage it is required to store information about a triangle's neighbors, allowing efficient storage and rendering. In this section, the data structure is explained, and it is shown how hierarchical and spatial coherence is used for mesh compression.

3.1 Data structure

In contrast to assigning more or less arbitrary indices to mesh vertices [Hoppe96] or having vertex indices reflect simplification order [ElllS99], in CAME each vertex is identified by the path to be taken in the simplification hierarchy from the root to the corresponding node. These *node identificators* are simply bit strings with "0" for the left branch (v_t) and "1" for the right branch (v_u) as indicated in Figure 2. We find the following bit strings in the example of Figure 2(a):

$$\begin{aligned} v_1 &= \text{“111010001001”} \\ v_2 &= \text{“11101000111”} \\ v_3 &= \text{“111010011111”} \end{aligned}$$

However, the bit strings identifying the paths to the vertices v_1, v_2, and v_3 are highly redundant since they all have a common prefix ("1110100" and two more common bits between v_1 and v_2). It is therefore sufficient to store only those portions of the strings *relative* to the node containing vertex v_s in Figure 2(b). This gives the *relative node identificators*

$$\begin{aligned} n_1 &= \text{“001”} \\ n_2 &= \text{“11”} \\ n_3 &= \text{“11111”}, \end{aligned}$$

where n_3 also has to include the number $n_3.up$ of levels to go up in the hierarchy (dashed line in Figure 2(b)) to reach the common junction v'_s of all vertex paths of triangle f. See Section 3.4 for implementation details of the bit string data structure.

3.1.1 Hierarchy traversal

Following the refinement criteria and mesh dependency enforcements, the *active vertex front* is moved up or down (see also [Hoppe97, Luebk97, Puppo98]), which corresponds to collapsing or expanding nodes of the hierarchy, respectively. Each triangle involved in an edge collapse or vertex split has to update its vertices to reflect the new level of detail. This requires to know all triangles adjacent to each vertex in [ElllS99] or to know each triangle's neighbors in [Hoppe98].

In our method, updating triangle vertices is completely separated from hierarchy traversal and doesn't require any additional information. We

store n_1, n_2, and n_3 together with the *vertex split record* [Hoppe96] of v_s. Therefore any information required to reconstruct triangle f is known as soon as v_s is split (see Figure 2). By evaluating the relative paths n_i, each triangle can autonomously update its vertices. Moving up and down one level in the hierarchy simply corresponds to truncating and appending one bit, respectively, at the end of the bit string identifying a triangle's vertex in the current level of detail.

3.1.2 Dependency representation

Defining mesh dependencies as relations between adjacent triangles has been shown by Hoppe to allow highly adaptive level of detail generation [Hoppe97]. These face-to-face relations are transformed to node-to-node relations as demonstrated in [Grabn00]. We can therefore use the same data structures (namely, the simplification hierarchy and relative paths between nodes) for identifying both vertices and mesh dependencies.

3.1.3 External memory management

To be able to visualize data sets much larger than main memory, we utilize the *meta-node* concept introduced by El-Sana and Chiang [ElllS00]. A constant number L of levels of the vertex hierarchy is grouped in a meta-node and kept on disk as a separate unit. In our implementation we chose $L = 8$.

This concept is also well suited for asynchronous access to the scene database. Each time the view-dependent refinement algorithm tries to access a meta-node not yet in main memory, the refinement procedure skips this branch. At the same time, a separate database thread is woken up to fetch and decode the requested data.

3.2 Topology compression

3.2.1 Node identificators

Common substrings of each triangle's vertex paths can be omitted as explained in Section 3.1. Moreover, since v_1 is referenced by the left path (via v_t), the first bit of n_1 is always "0", and the first bit of n_2 is "1". We also know the first bit of n_3 because the paths to v_1/v_2 and v_3 enter different subtrees below v'_s. Omitting each path's first bit saves additional three bits per triangle. The Huffman-encoded path lengths are inserted into the bit stream before each path.

3.2.2 Mesh dependencies

Fold-over prevention is performed based on Hoppe's method [Hoppe97], with dependencies defined between nodes of the simplification hierarchy [Grabn00] as relative node identificators. In a two-manifold mesh[1], each node n depends on a maximum of four other nodes n_i, $i = 0 \ldots 3$ (corresponding to the faces $f_{n0} \ldots f_{n3}$ adjacent to the affected region, see [Hoppe97] for details). However, there are two cases when a reference to one (or more) of these four nodes can be omitted:

- A node occurs more than once in the set of node dependencies (i.e., $n_i = n_j$, $i \neq j$). This is true if some of the triangles $f_{n0} \ldots f_{n3}$ share an edge which is collapsed during the simplification process after node n has been created.
- A node n_i in the set of node dependencies lies on the path between the hierarchy root and n (i.e., n_i is a prefix of n). Since the vertex hierarchy implicitly defines dependencies between any (inner) node and its two children, storing n_i doesn't provide additional information in this case.

The number of non-redundant dependencies (i.e., not falling into one of the two categories above) of each node was found to be between 1.5 and 2 on average. By omitting redundancies, mesh dependencies can be encoded even more compact than El-Sana and Varshney's implicit dependencies (which requires two integer vertex-ids per node [ElllS99]).

3.3 Geometry compression

3.3.1 Vertex locations

Since each meta-node represents a localized part of the mesh, it is possible to exploit spatial coherence for data compression. Similar to other compression schemes [Taubi98, Touma98], we encode vertex coordinates relative to previously encoded locations. For each vertex split operation creating vertices v_t and v_u from v_s, the *difference vectors* $\mathbf{d}_t = \mathbf{v}_t - \mathbf{v}_s$ and $\mathbf{d}_u = \mathbf{v}_u - \mathbf{v}_s$ are stored. Note the difference between v (identifying one vertex in the mesh) and $\mathbf{v}$ (coordinate vector of v in $\mathbb{R}^3$). Instead of using a global coordinate system, principal component analysis is performed for all $\mathbf{d}_i$

[1] The following holds for meshes of arbitrary topology, we use the two-manifold case for demonstration purposes only.

within one meta node. A difference vector $\mathbf{d}$ can then be expressed as

$$\mathbf{d} = c_1\mathbf{e}_1 + c_2\mathbf{e}_2 + c_3\mathbf{e}_3,\ ||\mathbf{e}_i|| = 1,\ i = 1\ldots 3$$

where $\mathbf{e}_i$ are the eigenvectors (sorted by decreasing corresponding eigenvalue λ_i) of the matrix

$$\mathbf{M} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{d}_i\mathbf{d}_i^T.$$

Two difference vectors per node contribute to $\mathbf{M}$, hence $N = 2(2^L - 1)$. In a sufficiently smooth region of the mesh, c_3 is significantly smaller than c_1 and c_2 and therefore requires less bits of storage. The user can select the maximum number of bits per component (see also Figure 4).

The difference vectors are largest near each meta-node's root because the approximation error grows as the hierarchy is built bottom-up. We therefore use *normalized difference vectors*

$$\widetilde{\mathbf{d}} = c2^{l/2}\mathbf{d}$$

instead. A constant c is assigned to each meta node to bound the lengths of the normalized difference vectors ($||\widetilde{\mathbf{d}}|| \leq 1$). l is the length of the path from the meta-node's root to the node $\mathbf{d}$ belongs to. The factor $2^{l/2}$ is due to the following consideration. The number of nodes (and therefore the number of triangles) differs by a factor of two between two successive levels in the hierarchy. However, in a smooth region of the mesh, the surface area is only slightly modified by collapsing edges in that region. Therefore twice the number of triangles share the same area when going down one level, giving a factor of roughly $1/\sqrt{2}$ for edge lengths.

3.3.2 Error estimation

It is vital to any view-dependent visualization system to be able to compute (or at least estimate) the screen-space geometric approximation error. This is often done by defining an *error volume* associated with the approximation error introduced by each simplification step [Xia96, Luebk97, Hoppe97]. Since we are looking for a compact representation, we decided to use a simple error volume, which is a sphere centered at the simplified vertex v_s bounding all vertices in the subtree below v_s.

Hierarchical coherence can again be exploited by relative encoding of the spheres' radii. Let r_i and r_j be the error volume radii associated with nodes n_i and n_j, respectively, and n_j be an child node of n_i. Then $q_{ij} = r_j/r_i$ is stored instead of r_j, where the q_{ij} can be encoded with as few as two bits per node. To avoid underestimating the error, q_{ij} is rounded up to the next larger value that can be represented by the selected number of bits.

3.4 Memory management

Each meta-node is not only stored and transmitted separately, but is also assigned a single chunk of memory for visualization. This is possible because the number of nodes and triangles within a meta-node is known as soon as the data has been transmitted. Therefore dynamic memory management overheads can be reduced significantly.

Similar considerations apply to the relative paths n_1, n_2, and n_3 stored for each triangle. Although a variable length data structure in principle, it is far more efficient to allocate a fixed number of bits in main memory large enough to represent any mesh of practical interest[2]. The current implementation packs n into a 64 bit integer, using six bits for $n.up$ (see Figure 2(b)), six bits for the length of the bit string, and the remaining 52 bits for the bit string itself. Assuming a well-balanced hierarchy, this is sufficient to encode meshes with up to $2^{51} \approx 2{\cdot}10^{15}$ vertices. To enforce generation of a well-balanced hierarchy, we define a balancing factor b to obtain a modified error function

$$E(i) = 2^{bd(i)}E_q(i),$$

where $E_q(i)$ is the quadric error metric [Garla97] associated with vertex i, and $d(i)$ is the depth of the subtree below i created so far. For any $b > 0$, the modified error functions penalizes the creation of an unbalanced hierarchy.

Note that only the error metric guiding the simplification process is modified. The screen-space geometric error for view-dependent simplification is evaluated based on explicitly calculated error volumes (see Section 3.3.2).

4 Results

Since adaptive level of detail and mesh compression have not been addressed together before, we first compare the feature set of our approach with some recent related techniques in Table 1. While the most efficient compression methods don't support adaptive LOD, the view-dependent algorithm by El-Sana and Chiang doesn't deal with compression.

[2] Dynamic memory allocation causes an overhead of 24 bytes on the GNU/Linux system, while 16 bytes are sufficient to identify each molecule on the earth's surface.

	Rossignac [Rossi99]	El-Sana & Chiang [ElllS00]	Alliez & Desbrun [Allie01a]	CAME
compression	++	– –	++	+
progressive encoding	–	+	+	+
adaptive level of detail	–	+	–	+
non-manifold meshes	–	+	–	+

Table 1: Feature comparison chart

model name	number of triangles	processing time	storage size (in bytes) VRML	gzip	E&C	CAME	compr. ratio
bones	4204	0:01	157k	49.1k	-	25.7k	0.164
dino	10777	0:03	370k	94.7k	-	56.5k	0.153
terrain	39300	0:14	1.81M	311k	-	231k	0.127
bunny	69451	0:29	2.81M	836k	9.7M	557k	0.198
dragon	202520	1:12	9.55M	2.34M	29.6M	1.74M	0.182
city	601666	7:55	36.5M	5.43M	-	4.46M	0.122
buddha	1080946	8:34	54.6M	10.8M	-	6.26M	0.115

Table 2: Compression achieved with CAME

Table 2 demonstrates the results achieved by our approach for seven different models. The sizes of the models as plain and gzip-compressed VRML files are given since these are common techniques to transmit 3D data over the internet. CAME stores twice the number of vertices (i.e., the simplified geometry) and additional information for adaptive LOD (e.g., mesh dependencies, error estimation). However, it still outperforms the standard gzip method and achieves an improvement of factor 17 over the uncompressed method by El-Sana and Chiang [ElllS00] for the "bunny" and "dragon" model (column "E&C").

The processing times refer to an Intel Pentium III/1000MHz. Since the city model consists of many small objects, it requires an additional Delaunay tetrahedralization step (as proposed in [ElllS00]) to be able to connect disjoint parts of the scene, thus the relatively large execution time. See also Figure 3 for the effectiveness of topology simplification.

More detailed information is given in Table 3. The vertex hierarchy depths were found with the balancing factor $b = 1$ (see Section 3.4). The number of bits to represent topology (connectivity, mesh dependencies) and geometry (vertex coordinates, multiresolution data including error estimation) are given.

Figure 3 shows screenshots of an interactive walkthrough of the "city" data set. The figure also contains triangle counts and frame rates of the models simplified by the adaptive LOD algorithm.

Geometry compression is demonstrated in Figure 4. Three bits per component of the normalized difference vectors $\tilde{\mathbf{d}}$ (see Section 3.3.1) are already sufficient to resemble the original shape (five bits per component were used in Tables 2 and 3).

5 Conclusions and future work

The CAME framework offers a solution of two problems of interactive visualization. First, it is a compact representation reducing storage requirements and transmission times of large 3D models. Second, it is organized hierarchically, allowing efficient view-dependent simplification and progressive transmission.

While the experiments with the city data set showed the effectiveness of topology simplification, they also made the need for a more sophisticated error measure obvious. We expect improved image quality when taking into account object semantics in the error measure.

6 Acknowledgements

Many thanks to Andrej Ferko for his great help to improve this paper and to the reviewers for their comments. This work has in part been funded by the European Union under contract no. IST-1999-20273.

model name	depth of hierarchy	detailed size (in bits) connect.	depend.	coord.	MR data	bits total	bits per triangle
bones	13	68.8k	31.2k	97.0k	8.61k	206k	48.9
dino	15	163k	90.4k	177k	21.9k	452k	42.0
terrain	16	587k	298k	879k	80.1k	1.84M	46.9
bunny	17	1.11M	596k	2.62M	139k	4.48M	64.2
dragon	19	3.37M	1.59M	8.56M	402k	13.9M	68.7
city	21	10.7M	5.35M	18.2M	1.36M	35.7M	59.3
buddha	22	17.3M	8.45M	22.2M	2.17M	50.1M	46.3

Table 3: Detailed split up of memory consumption

REFERENCES

[Allie01a] Pierre Alliez and Mathieu Desbrun. Progressive compression for lossless transmission of triangle meshes. In *SIGGRAPH 2001 Conference Proceedings*, Annual Conference Series, August 2001.

[Allie01b] Pierre Alliez and Mathieu Desbrun. Valence-driven connectivity encoding for 3D meshes. In *Proceedings Eurographics*, volume 20 of *Computer Graphics Forum*, September 2001.

[ElllS99] Jihad El-Sana and Amitabh Varshney. Generalized view-dependent simplification. In *Proceedings Eurographics*, volume 18 of *Computer Graphics Forum*, pages 83–94, September 1999.

[ElllS00] Jihad El-Sana and Yi-Jen Chiang. External memory view-dependent simplification. In *Proceedings Eurographics*, volume 19 of *Computer Graphics Forum*, pages 139–150, August 2000.

[Garla97] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In Whitted [Whitt97], pages 209–216.

[Grabn00] Markus Grabner. Consistency of the VDPM framework. In *Proceedings of SCCG 2000*, pages 147–155, May 2000.

[Hoppe96] Hugues Hoppe. Progressive meshes. In *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108, August 1996.

[Hoppe97] Hugues Hoppe. View-dependent refinement of progressive meshes. In Whitted [Whitt97], pages 189–198.

[Hoppe98] Hugues Hoppe. Efficient implementation of progressive meshes. *Computers & Graphics*, 22(1):27–36, February 1998. ISSN 0097-8493.

[Levoy99] Marc Levoy. The digital Michelangelo project. In *Proceedings of the Second International Conference on 3D Digital Imaging and Modeling*, 1999.

[Luebk97] David Luebke and Carl Erikson. View-dependent simplification of arbitrary polygonal environments. In Whitted [Whitt97], pages 199–208.

[Pajar00] Renato Pajarola and Jarek Rossignac. Compressed progressive meshes. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):79–93, January/ March 2000.

[Popov97] Jovan Popović and Hugues Hoppe. Progressive simplicial complexes. In Whitted [Whitt97], pages 217–224.

[Puppo98] Enrico Puppo. Variable resolution triangulations. *Computational Geometry*, 11(3–4):219–238, December 1998.

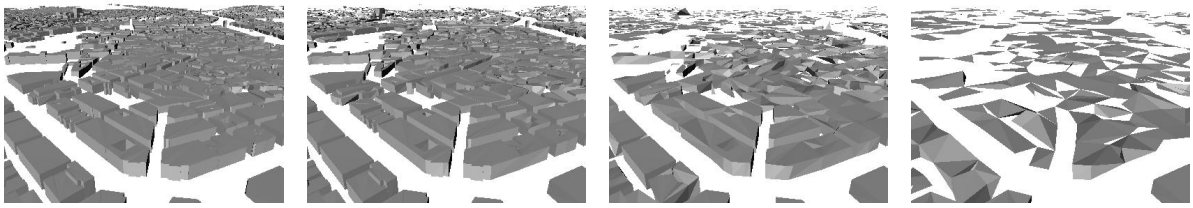
[Rossi99] Jarek Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1), 1999.

[Taubi98] Gabriel Taubin and Jarek Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17(2):84–115, April 1998.

[Touma98] Costa Touma and Craig Gotsman. Triangle mesh compression. In *Proceedings of the 24th Conference on Graphics Interface (GI-98)*, pages 26–34, June 18–20 1998.

[Whitt97] ACM SIGGRAPH, ACM SIGGRAPH. *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

[Xia96] Julie C. Xia and Amitabh Varshney. Dynamic view-dependent simplification for polygonal models. In *IEEE Visualization '96*, October 1996.

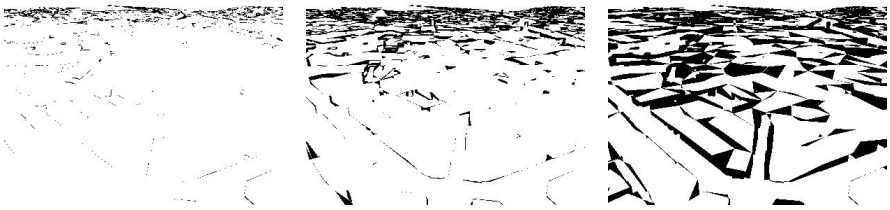


(a) original model (601666 faces, 2 frames/second)

(b) 1% error (23478 faces, 10 frames/second)

(c) 3% error (7168 faces, 30 frames/second)

(d) 10% error (1862 faces, 84 frames/second)

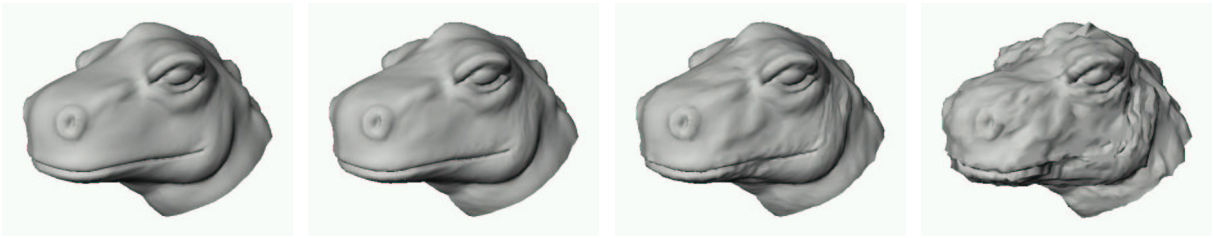


(e) 1% error (error screen)

(f) 3% error (error screen)

(g) 10% error (error screen)

Figure 3: Topology simplification of the "city" data set (screen space geometric error threshold is given in % of screen size), error screens are computed by subtracting the foreground masks of original and simplified models to remove the influence of shading, frame were times measured on an Intel Pentium III/600MHz with GeForce256 DDR graphics hardware



(a) original model

(b) 5 bits per component

(c) 4 bits per component

(d) 3 bits per component

(e) 5 bits per component (error screen)

(f) 4 bits per component (error screen)

(g) 3 bits per component (error screen)

Figure 4: Geometry compression artefacts of the "dino" data set at different numbers of bits per component, error screens are pixel-wise subtractions and include the influence of shading