Proc. Vision, Modeling, and Visualization VMV'99
Erlangen, Germany, pp. 253-259, Nov. 1999

Fully Embedded Coding of Triangle Meshes

Marcus Magnor and Bernd Girod

University of Erlangen-Nuremberg, Telecommunications Laboratory
Cauerstrasse 7, D-91058 Erlangen, Germany
email: {magnor,girod}@nt.e-technik.uni-erlangen.de
Tel.: +49-9131-8528904 FAX: +49-9131-8528849

Abstract

An algorithm for triangle mesh compression is presented. Any mesh topology can be coded. Vertex coordinates and mesh connectivity are simultaneously coded with increasing level-of-detail. The bit stream can be sequentially decoded, gradually refining vertex accuracy and mesh resolution. The embedded code allows progressive transmission of triangle meshes and continuous decoding of 3-D geometry. Coding performance is validated on a number of standard geometry data sets. The measurements show that the presented algorithm compresses topology information to less than 7 bits per triangle, and vertex positions are accurately reconstructed from 60 bits per vertex. Images document the perceptive quality of rendering results from meshes of reduced resolution.

1 Introduction

Triangle meshes allow convenient modeling of arbitrary 2-D surfaces in 3-D space. Planar triangles serve as basic primitives to approximate smooth surfaces. Most frequently, triangle meshes are used in Computer Graphics to model surfaces of solid objects. The number of triangles determines model accuracy. Because smooth surface appearance and small details are lost if the number of triangles is small, geometry models typically consist of many triangles. For complex objects, the number of triangles and vertices can amount to several hundred megabytes of data if no compression scheme is used.

Large triangle meshes require increased rendering time, longer processing, more storage capacity and longer transmission times. Several algorithms have been proposed to compress triangle mesh information. Triangle strip coding [1] requires between 8 and 11 bits per triangle if an optimal stripper is available [2]. In [3], connectivity of uniform, 2-manifold triangle meshes is coded with less than 2 bits per triangle, on average. Both algorithms do not feature progressive mesh reconstruction, i.e., the entire coded information is required to reconstruct a mesh.

Progressive Meshes (PM) [4] provide a hierarchical representation of mesh geometry. By adding triangles to an existing mesh, mesh accuracy is gradually refined. Vertex coordinates are not embeddedly coded, but have to be specified with full precision. Also, a starting mesh of the object's topology must be provided.

In this paper, a new algorithm is presented that embeddedly codes triangle mesh topology and vertex coordinates. The bit stream features sequential decoding and continuous mesh refinement, up to accurate mesh reproduction. Vertex position and mesh connectivity are jointly coded. Any mesh topology as well as manifold and non-manifold meshes can be coded. Compression efficiency and reconstruction quality are evaluated using several standard geometry data sets. Measurements show that the proposed *Embedded Mesh Cod-*

ing (EMC) algorithm compresses geometry information to less than 7 bits per triangle, and vertex positions are correctly reconstructed from ≈ 60 bits per vertex.

2 Progressive Mesh Representation

A triangle mesh data set consists of F triangles, defined by $3F$ vertex indices, and coordinates for V vertices. For large triangle meshes describing 2-D surfaces in space, Euler's Formula yields $F \approx 2V$. If vertex coordinates are expressed by three 4-byte float values, and 4 bytes are needed to index a vertex,

$$3*4*V+3*4*F \approx 36*V \approx 18*F \quad \text{bytes} \quad (1)$$

are required to store mesh information. Objects with complex geometry can require triangle meshes of up to $F = 10^6$ triangles. Such triangle meshes cannot be rendered at interactive frame rates, they are time-consuming to transmit and require extensive storage capacities. Reducing the number of triangles F and representing vertex coordinates with less than 3*4 bytes yields smaller triangle meshes which allows faster transmission and rendering rates, at the cost of introducing some deviations from the original surface description. In addition, neighborhood relationships among triangles can be exploited to efficiently code mesh topology without losing geometry information, as is done, e.g., in the directed-edge mesh representation [5].

Algorithms have been devised to reduce the number of triangles, while minimizing the resulting geometry distortion [4, 5, 6, 7]. At the core of the *Progressive Meshes* (PM) algorithm [4], the mesh is searched for that edge whose elimination results in the least geometry distortion. Collapsing an edge results in removing two triangles because in a 2-manifold mesh an edge always belongs to two triangles (Fig.1). To simplify the mesh further, the PM algorithm can be applied multiple times. By reversing the PM algorithm, a coarse mesh can be augmented by additional triangles to progressively refine mesh accuracy. The edge-collapse operation is inverted

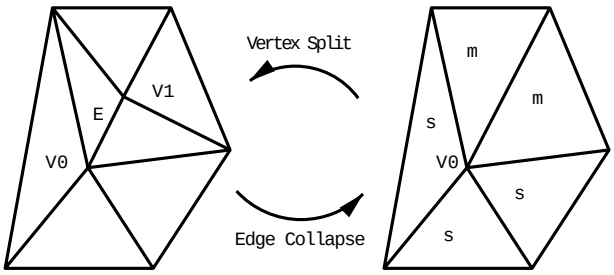


Figure 1: Progressive Meshes.
Edge Collapse: E is eliminated; *V1* falls onto *V0*; 2 triangles disappear.
Vertex Split: *V0* is specified; a new vertex *V1* is created; *V0*'s neighboring triangles are rearranged (s: stay at current vertex, m: move to new vertex); 2 new triangles are created along the connecting edge.

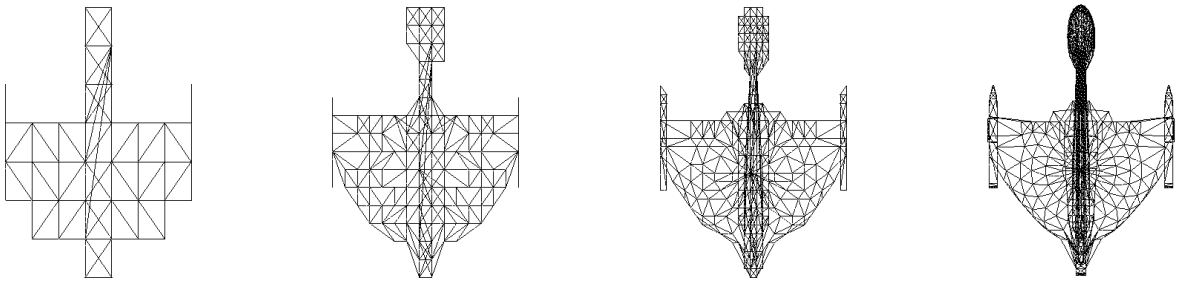
by splitting a vertex: a new vertex is created, connected to an existing vertex, the local triangles are rearranged, and two new triangles are created along the connecting edge. By indexing an existing vertex, the mesh is refined always at the point of greatest deviation from the original geometry.

For lossless mesh reconstruction, a vertex split requires the new vertex's position (3*4 bytes), the index of the splitting vertex (4 bytes), and information on how to redistribute neighboring triangles between new and old vertex ($\approx$ 1 byte). Because vertex splitting preserves mesh topology, the PM algorithm also needs a basic mesh of the object's topology to start from. Neglecting the starting mesh, V vertex splits are necessary to generate a final mesh of $F = 2V$ triangles, amounting to

$$\approx 17 * V \quad \text{bytes} \quad (2)$$

of data.

Because the PM algorithm exploits a priori knowledge about the 2-manifold mesh, its topology and previously coded local triangle neighborhood, it requires less than half as much memory as Eq. 1 suggests. While the PM algorithm reduces the number of triangles in a mesh, vertex coordinates always have to be coded with full accuracy, regardless of the amount of approximation. A scheme to efficiently compress triangle meshes using the PM representation can be found in [8].



Resolution level 3 Resolution level 4 Resolution level 5 Original (level 24)

Figure 2: The EMC algorithm progressively refines vertex positions and mesh connectivity.

Our proposed EMC algorithm improves the PM approach by employing slightly different elimination and reconstruction operations, which allow compression of manifold or non-manifold meshes of arbitrary topology. Further, vertex coordinates and connectivity are jointly and progressively encoded in the bit stream. Because vertex positions and mesh topology are simultaneously and progressively coded, geometry models can be continuously decoded up to arbitrary levels-of-detail, and decoding can stop at any point.

3 Embedded Mesh Coding

The coding scheme presented in this paper is based on expressing vertex coordinates by an octree description similar to vertex clustering [7], introducing a hierarchy of resolution levels (Fig. 2). In contrast to PM, vertex coordinates and mesh connectivity are simultaneously coded at continuously increasing resolution levels.

The EMC algorithm starts by finding the bounding box around the mesh. The bounding box is hierarchically subdivided using octree decomposition. The octree is realized by combining three binary trees for the X-, Y- and Z-direction. Each vertex is assigned its octree representation, according to its relative coordinates within the bounding box. If vertex coordinates are assumed to be given by 4-byte float values, which exhibit a 23 bit-long mantissa plus sign, relative position of vertices can be accurately expressed by 24 bits. Decomposing the bounding box down into 24 resolution levels is sufficient to uniquely represent vertex coordinates, because if 2 vertices exhibit the same 24-level octree representation, they can be considered identical. Each vertex is also assigned a list stating its affiliated connecting edges to other vertices, and the vertices are ordered according to their octree representation.

Next, a list is created which states the succession of operations necessary to simplify the mesh: Starting at the second-highest (23.) resolution level and in direction of the shortest bounding box extension, all vertices are examined to determine whether two vertices fall into the same subvolume (Fig. 3). For solitary vertices, a *Refinement* (RF) operation is memorized in the list, stating to which half of the subvolume the vertex belongs. If two vertices fall into the same subvolume, a *Vertex Merge* (VM) operation is written to the list, and the vertices are merged: One of the coinciding vertices is removed (by definition the vertex in the right half of the subvolume), its edges are redirected to the remaining vertex, and, if a connecting edge between both vertices existed, it is deleted from the remaining vertex's edge list. Because the VM operation does not require a connecting edge between the vertices, the VM operation is more versatile than PM's edge-collapse operation and allows to change mesh topology.

After all vertices have been passed once, the remaining vertices are examined again along the other two bounding box directions. After three passes, the vertices in each subvolume have been merged. Then, the next-coarser oc-

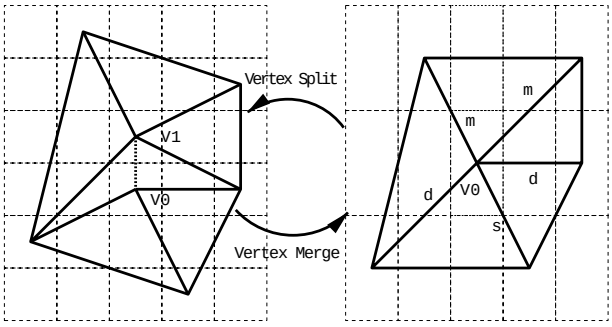


Figure 3: EMC Operations.

Vertex Merge: At some coarse resolution level, 2 vertices fall into the same octree subvolume; one vertex moves to the center of the subvolume; the other vertex is eliminated and its edges connected to the remaining vertex. Solitary vertices move to the center of the subvolume.
Vertex Split: A new vertex is created and assigned to one half of the subdivided volume; the current vertex moves to the other half; its affiliated edges are rearranged (s: stay, m: move, d: duplicate), and if needed a connecting edge between current and new vertex is established.

tree resolution level is considered. With each pass, the 3-D grid of vertex coordinates becomes coarser, and vertex positions move to remaining locations.

The bit stream is constructed by first writing a header consisting of 6 float values (4 bytes each), describing bounding box position and dimensions. A vertex list is established and initialized by one single vertex in the center of the bounding box, with no affiliated edges. Now, the operations list is passed from end to front. Starting at resolution level 1 along the longest bounding box dimension, the last entry of the operations list (VM operation) is reversed by a *Vertex Split* (VS) operation:

- The existing vertex's position is refined to the left subvolume's center position.
- A new vertex is appended to the vertex list with coordinates at the center of the right subvolume.
- If both vertices are connected, an edge is created and appended to both vertices' (empty) edge lists, and a codeword is written to the bit stream indicating whether or not a connecting edge exists.
- The operations list is searched for both vertices' next VM=VS^{-1} operations, and the resolution levels and directions of the next occuring VS operations are stored.
- The differences to the current resolution level and direction are written to the bit stream using a fix Huffman code table. A special codeword indicates that a vertex will undergo no further VS operations.

For the second bounding box direction, the vertex list, consisting now of 2 vertices, is passed again.The information on each vertex's next VS operation is compared to the current resolution level and direction; if the next VS operation is not yet due, the vertex position is refined (RF operation) and a single bit is written to the bit stream, indicating whether the vertex belongs into the left or the right half of the divided subvolume. If current resolution level and direction suggest that the vertex undergoes a VS operation, the VS-operation steps are performed and each edge in the vertex's edge list is classified:

- STAY: the edge remains in the old vertex's edge list; the STAY codeword is written to the bit stream;
- MOVE: the edge is removed from the old vertex's edge list and appended to the new vertex's edge list; the MOVE codeword is written to the bit stream.
- DUPLICATE: a new edge is created and appended to the new vertex's edge list; the DUPLICATE codeword is written to the bit stream.

The third bounding box direction is considered by passing again through the list of vertices and performing the necessary RF or VS steps. After all spatial directions have been refined, the resolution level is incremented and the vertex list is passed again three times for all directions. Coding ends after the 24th resolution level has been completed. The resulting bit stream contains all information necessary to fully reconstruct original vertex positions and connectivity.

Data Set	Vertex#	△#	PM bytes	EMC bytes	Conn. bits	V.Pos. bits	bits/△	bits/Vertex
Beethoven	2515	5036	42755	23216	34511	151206	6.85	60.1
Bunny	3318	6600	56406	30416	42727	200593	6.47	60.5
VW	5217	9860	88689	46875	65389	309599	6.63	59.3

Table 1: Geometry models used for EMC evaluation; the coding rates for PM are calculated from Eq. 2; EMC bit stream size is measured at resolution level 24; Separated EMC bit-rate for connectivity and vertex position coding; bits per triangle and bits per vertex position.

4 EMC Decoding

Decoding follows the same scheme described for writing the bit stream: the vertex list is repeatedly passed and appended, edges are created, moved and duplicated. Triangles are reconstructed during bit stream decoding by checking local vertex connectivity if edges have been created or moved. If the original mesh contains edges that can be connected to triangles that are not part of the surface description, a few additional vertices need to be introduced to the mesh prior to coding. Decoding can be interrupted at any point and the triangle mesh reconstructed with according resolution.

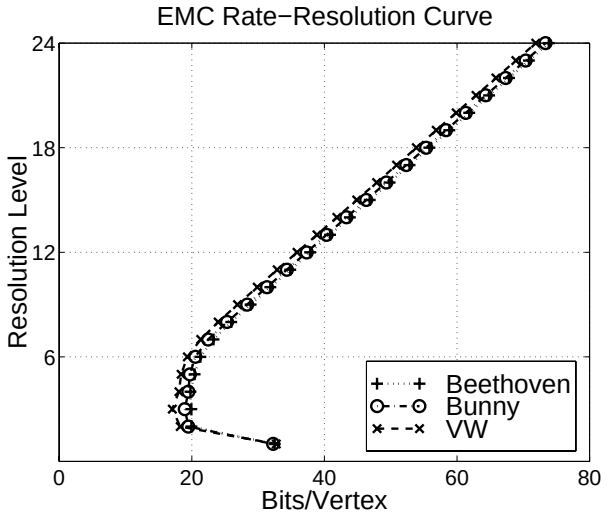


Figure 4: Resolution level vs. bit-rate of the EMC algorithm; at low resolution levels, the additional header information causes higher apparent bit-rates per vertex.

5 Measurements

The EMC algorithm[1] was tested on several standard geometry models (*Beethoven, Bunny* and *VW*). Table 1 depicts the number of vertices and faces of the triangle meshes. If the PM algorithm's calculated bit stream size is compared to the results measured with EMC at 24 resolution levels, the EMC algorithm achieves more than 45% better data compression. When separating the total bit-rate into vertex positional and connectivity information, the EMC algorithm requires less than 7 bits per triangle to code mesh connectivity. About 60 bits per vertex are needed to code vertex coordinates.

Fig. 4 depicts obtained resolution level in dependence on the number of bits spent per vertex. At low resolution levels, only few vertices are reconstructed, and the header information's additional bits cause apparently higher bit-rates per vertex. The rate-resolution curves nearly coincide for all three test models, even though model characteristics differ significantly: the Beethoven model exhibits fine details and sharp edges, the Bunny shows curving surfaces of varying radii, while the VW model has smooth surfaces, as can be seen from the upper row in Fig. 5. The images are rendered with flat shading. The lower rows depict images rendered from increasingly compressed triangle meshes. Models reconstructed up to the $l = 10$th resolution level cannot be visually distinguished from the original full-resolution models. Good geometry reconstructions are obtained after $l = 8$

[1] available at:
http://www.lnt.de/~magnor/Geometry.html

resolution refinements, and reconstruction results are still satisfactory when decoding only up to the $l = 6$th resolution level.

Vertex coordinate error relative to bounding box extension depends on resolution level l by $2^{-(l+1)}$. The positional error is also an upper limit on the Hausdorff distance measure [6] expressing mesh accuracy. The Hausdorff distance is defined by the maximum distance between an original vertex and the approximative mesh's surface.

6 Conclusions

A new coding algorithm for triangle meshes was presented, and its compression performance was validated on several geometry data sets. The EMC algorithm features progressive bit stream decoding for simultaneous refinement of vertex position and mesh connectivity. Less than 7 bits per triangle are required to code connectivity, and 60 bits per vertex suffice to accurately reconstruct vertex positions. For lossy compression, much lower bitrates are obtained.

The current implementation of the EMC algorithm has not been optimized for fast compression. Future work will also include supplementing the algorithm by an efficient coding scheme to progressively code texture. Several texture maps are intended to be jointly coded to enable view-dependent rendering from variable texture maps [9, 10]. The presented algorithm will be applied to light field coding and rendering [11, 12], after approximate geometry has been reconstructed from the light field images using, for example, the algorithm presented in [13].

References

[1] F. Evans, S. Skiena, and A. Varshney. Optimizing triangle strips for fast rendering. In *IEEE Visualization '96*, pages 319–326, October 1996.

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

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

[4] H. Hoppe. Progressive meshes. In *SIGGRAPH 96 Conference Proceedings*, pages 99–108, August 1996.

[5] S. Campagna, L. Kobbelt, and H.-P. Seidel. Directed edges – a scalable representation for triangle meshes. *Journal of Graphics Tools*, 1999. accepted for publication.

[6] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. Interactive multiresolution modeling on arbitrary meshes. In *SIGGRAPH 98 Conference Proceedings*, pages 105–114, July 1998.

[7] J. Rossignac and P. Borrel. Multiresolution 3D approximations for rendering complex scenes. In *Modeling in Computer Graphics*, pages 455–465, 1993.

[8] R. Pajarola and J. Rossignac. Compressed progressive meshes. Technical Report GIT-GVU-99-05, Georgia Institute of Technology, January 1999.

[9] P. Debevec, Y. Yu, and G. Boshokov. Efficient view-dependent image-based rendering with projective texture-mapping. Technical Report CSD-98-1003, University of California, Berkeley, May 1998.

[10] P. Debevec, C. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *SIGGRAPH 96 Conference Proceedings*, pages 11–20, August 1996.

[11] M. Levoy and P. Hanrahan. Light field rendering. In *SIGGRAPH 96*, pages 31–42, August 1996.

[12] S. Gortler, R. Grzeszczuk, R. Szeliski, and M. Cohen. The lumigraph. In *SIGGRAPH 96 Conference Proceedings*, pages 43–54, August 1996.

[13] P. Eisert, E. Steinbach, and B. Girod. Multi-hypothesis volumetric reconstruction of 3-D objects from multiple calibrated camera views. In *ICASSP*, March 1999.

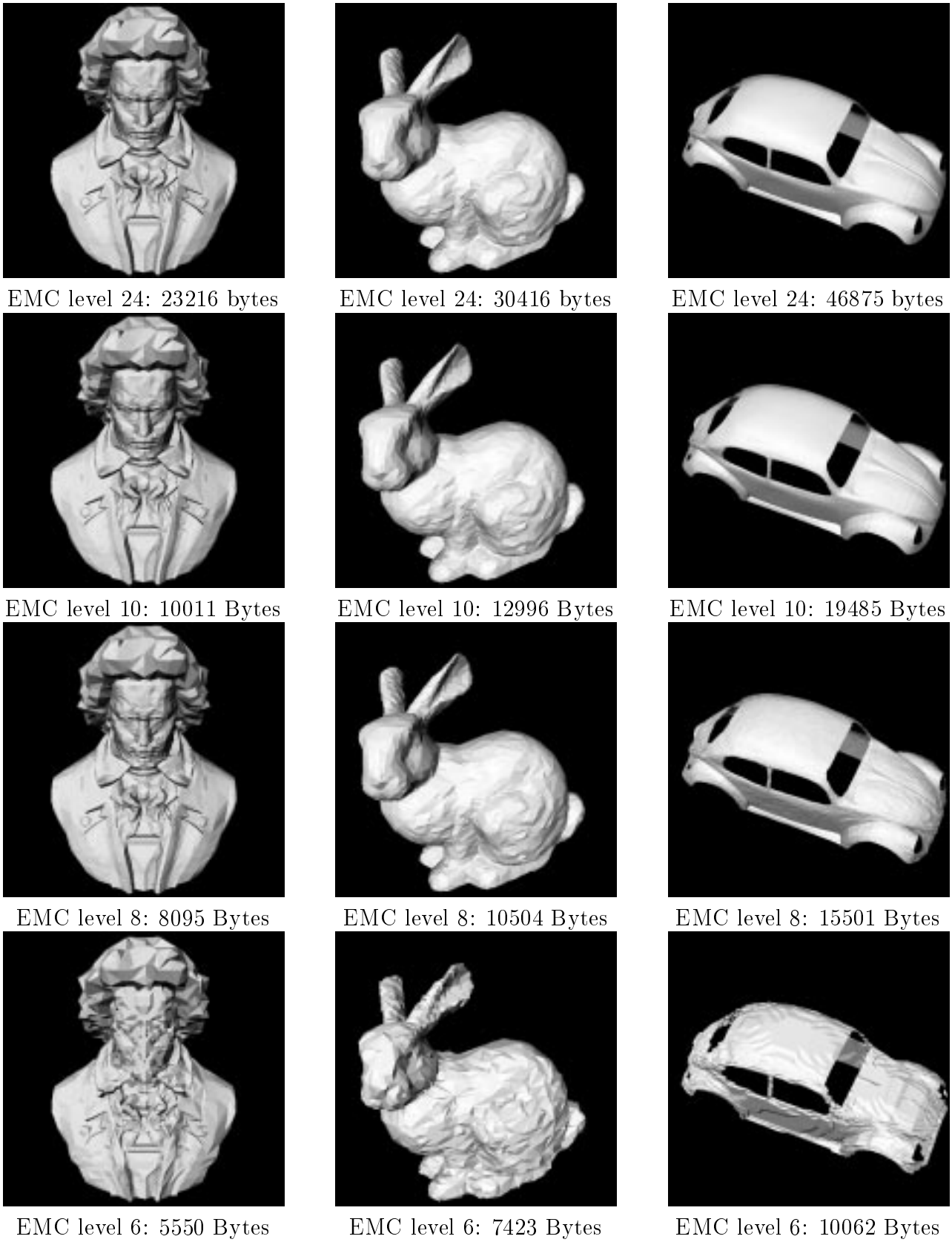


Figure 5: Images rendered from the test meshes at different resolution levels.