Compressed Progressive Meshes

Renato Pajarola and Jarek Rossignac
Graphics, Visualization & Usability Center
Georgia Institute of Technology

10 February 2000 11:02 AM

Abstract

Most systems that support the visual interaction with 3D models use shape representations based on triangle meshes. The size of these representations imposes limits on applications, for which complex 3D models must be accessed remotely. Techniques for simplifying and compressing 3D models reduce the transmission time. Multi-resolution formats provide quick access to a crude model and then refine it progressively. Unfortunately, compared to the best non-progressive compression methods, previously proposed progressive refinement techniques impose a significant overhead when the full resolution model must be downloaded. The CPM (Compressed Progressive Meshes) approach proposed here eliminates this overhead. It uses a new "Implant Sprays" technique, which refines the topology of the mesh in batches, which each increase the number of vertices by up to 50%. Less than an amortized total of 4 bits per triangle encode where and how the topological refinements should be applied. We estimate the position of new vertices from the positions of their topological neighbors in the less refined mesh using a new estimator that leads to representations of vertex coordinates that are 50% more compact than previously reported progressive geometry compression techniques.

1. Introduction

Although many representations have been proposed for 3D models [Ros94], polyhedra (or more precisely triangular meshes) are the de facto standard for exchanging and viewing 3D data sets. This trend is reinforced by the wide spread of 3D graphic libraries (OpenGL [NDW93], VRML [CBM97]), and of 3D graphics adapters for PCs that have been optimized for triangles. Therefore, triangle count is a suitable measure of a model's complexity. Common representations of triangulated meshes usually store the triangles as an indexed face list, where the coordinates of each vertex are 3 floating-point numbers and the incidence relation between triangles and vertices uses 3 integer vertex-references per triangle. Therefore, each triangle requires 12 bytes for the indices and every vertex 12 bytes for the coordinates which adds up to 18 bytes per triangle (there are roughly twice as many triangles as vertices in a typical model), not counting color and texture information. Even when using short-integer fixed point coordinates and short-integer incidence indices, an indexed face list for triangles still requires 9 bytes per triangle.

The complexity of 3D models in CAD, AEC, GIS, and medical applications has been rising steadily, fueled by the improvements in interactive 3D design tools, in data acquisition technologies, and in the storage, processing and graphics capabilities of personal workstations. Early designers and scientists were deliberately limiting the accuracy of their data sets to what could be stored and manipulated on their workstations. Today, the more complex models used by the automotive, aerospace, construction, petroleum, and architecture industries contain millions or even hundreds of million triangles. Their complexity will continue to increase rapidly in response to a need for higher accuracy during analysis, planning, and inspection.

The internet and the intranet provide a convenient medium for posting 3D databases on-line for general or restricted access. However, users who need to access these databases are often equipped with personal computers and standard telephone line connections. They do not have enough storage to locally cache all the models they wish to interact with and lack automatic consistency maintenance procedures for keeping such local copies updated. Consequently, most PC users must download the 3D models each time they wish to inspect or use them. A transmission cost of 18 bytes per triangle over a 56Kbps line implies a transmission rate of 400 triangles per second, or equivalently only 1.5M triangles per hour.

Because the exact representation of the visible geometry is not always required to produce an image of sufficient accuracy for navigation or inspection, geometric simplification and compression techniques may be invoked to reduce the transmission and rendering time. Geometric compression reduces the number of bits used to encode a geometric model. Simplification techniques reduce the number of triangles in an object's representation, and may be viewed as a form of lossy compression. Progressive transmission first sends a coarse model and then sends information used to refine the representation of the entire model or of specific features of the model.

This paper introduces several novel techniques, which improve upon previously reported compression and progressive transmission approaches [Hop96, TGHL98, LK98]. The comparison of progressive transmission solutions often involves subjective (visual) criteria and may not be safely reduced to the comparison of a single scalar measure. We suggest the use of error/time curves (Figure 1), which express how the accuracy of the model increases with time, or more generally with the total number of transmitted bits.

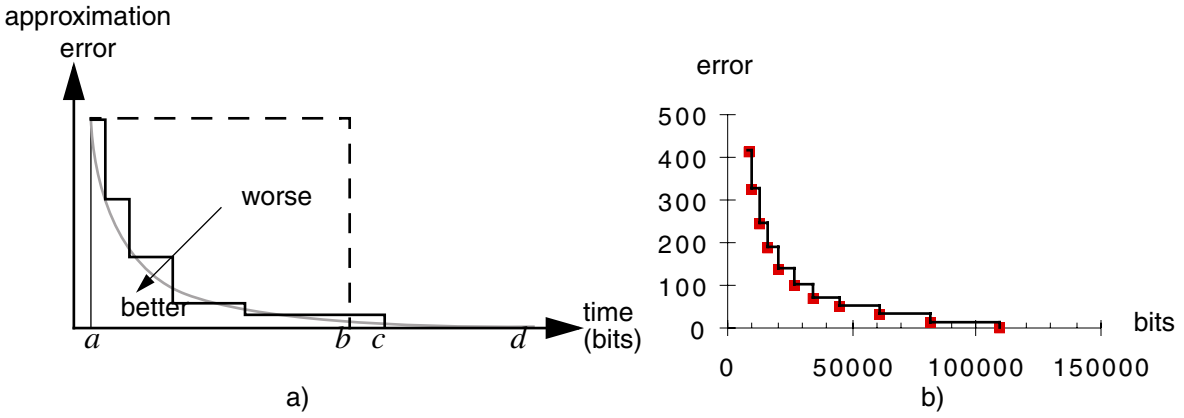


FIGURE 1. Progressive transmission solutions are compared in terms of the accuracy of the resulting model expressed as a function of the number of transmitted bits. a) For simplicity, we assume that it takes *a* bits to transmit the crude model for all cases. A simple approach of sending a precomputed crude model and subsequently the full resolution mesh is depicted by the dashed line, after receiving the crude model the user has to wait until the full resolution model is received at time *b*. An ideal fine-grain progressive technique (grey curve) immediately starts improving the crude model, but takes much longer (*d* bits) than a non-progressive technique to download the entire model. The black staircase curve illustrates batched updates which take slightly longer to download the full model, in time *c*, than a non-progressive approach, but much less time than the fine-grain methods. b) Presents actual CPM compression results for the small Bunny model.

Non-progressive approaches are often unacceptable, because they require the user to wait a long time before the entire model is decoded. A simple alternative would be to precompute a crude model, send it first, and then send the full resolution model independently, not taking advantage of the information received as part of the crude model. The user may start navigating the crude model immediately, but will have to wait more than the full transmission time of the complete model to see a more accurate resolution. For reference, this approach is shown by the dashed curve in Figure 1.

Fine-grain compression techniques, which refine the model by inserting one vertex at a time may offer increasingly better accuracy early on, but require significantly longer to reproduce the full resolution model, and thus tie up the communication channel for longer than needed. This penalty results from the overhead of encoding the refinement steps individually. The grey curve in Figure 1 shows an ideal fine-grain progressive transmission.

Techniques that group refinements into batches strike an optimal compromise. Although the accuracy remains constant while the next batch of upgrades is received and decoded, the overall waiting time is reduced, because batched upgrades may be compressed more efficiently than individual upgrades exploiting economy of scale. In Figure 1, the black staircase curve illustrates the approach presented in this paper, which groups the refinements in batches, and hence achieves a better compression. This technique may at times show higher error values than the fine-grain one, and take slightly longer to download the full model than a non-progressive approach.

The novel techniques introduced here encode each batch of refinements more efficiently than previously reported solutions [TGHL98, LK98]. As a result, they provide a whole series of accuracy refinements with little or no effect on the overall transmission time for the full resolution model, when compared to previous single-resolution compression techniques [Dee95, TR98, GS98]. The actual error curve produced by the CPM method for the *Bunny* model of Table 1 is shown in Figure 1.

The following section reviews the relevant prior art in geometric compression and progressive refinement. Then we provide a short overview of the CPM technique in Section 3. Next follow detailed descriptions of the CPM format (Section 4), of the compression algorithm (Section 5), and of the decompression process (Section 6). We analyze the storage cost in Section 7, and discuss our experimental results in Section 8. Finally, the paper is concluded with a summary in Section 9.

2. Prior art

We distinguish between lossless, lossy, and progressive compression. Previously reported lossless compression techniques include:

- A bit-efficient encoding of the connectivity graph, which captures triangle-vertex incidence from which other incidence and adjacency relations may be derived. See for example [Dee95, TR98, GS98, TG98]. A more comprehensive survey may be found in [Ros98] and in [TR98b]. In practice, these approaches produce a compressed format with less than 2 bits per triangle for the connectivity information alone.
- A predictor-based compression of the vertex locations: These solutions encode the corrections between the actual and the estimated location of each vertex. If the predictions are accurate, the corrective vectors are short and their integer coordinates may be efficiently encoded using entropy coding [Huf52, CNW87]. In [TR98] each vertex is predicted using a linear combination of its ancestors in a vertex spanning tree. In [Hop96], if a vertex v is split into an edge, v is used as a predictor for the other end of the new edge. The approach in [TG98] constructs a chamfered parallelogram to estimate the location of the free vertex of a new triangle that is adjacent to a known triangle.

Additional compression may be achieved through the following lossy approaches:

- Vertex locations may be quantized by expressing the vertex coordinates as k-bit integers in a normalized coordinate system derived from a minimum axis-aligned bounding box [Dee95, PH97, TR98]. The origin is placed at one vertex of the box and the units are selected so that the all coordinates span the range $[0..2^k]$. The choice of k is dictated by the absolute preci-

sion required by the application and by the size of the bounding box. Often k may be kept below 12 [Dee95, THLR98], which makes the entropy coding of the corrective vectors, discussed above, very effective, bringing the vertex location storage to between 12 and 18 bits per vertex for uniform tessellations of smooth surfaces [THLR98].

- The mesh may be simplified by coalescing vertices [RB93], by decimating them [SL96, SZL92], or by collapsing edges [HRD+93, RR96, GH97]. A more complete discussion may be found in [HG97]. Most of these techniques remove vertices one at a time in an order that attempts to maximize the accuracy of the approximating model at each stage. Saving intermediate results generates a series of approximations at several levels of detail. The differences between the various techniques lie principally in the error metric they use.

When the bandwidth precludes the transmission of the full resolution model, a crude model may be used initially, and then refined when necessary by downloading higher levels of detail [FS93] or by downloading upgrades which contain information sufficient to refine the current model. Refinements [Hop96, XEV97] which insert vertices one at a time, provide a fine-grain control of the accuracy and support view-dependent (non-uniform) refinements. However, this flexibility limits the compression ratio significantly and such progressive models require about 13 bits per vertex to encode the mesh connectivity. Grouping refinements into larger batches reduces the flexibility, but results in an economy of scale. For example, the *Progressive Forest Split* (PFS) technique [TGHL98], cuts the triangulated surface at a subset of its edges. The connected components of the cuts open up to become the boundaries of holes. The cuts and the internal triangulations of these holes may be encoded efficiently using the Topological Surgery compression [TR98]. The amortized connectivity encoding of PFS takes 10 bits per vertex. The geometry is encoded with about 30 bits per vertex. Concurrently to the presented method in this paper other progressive mesh compression techniques have been developed that use similar batch processing [CLR99] of updates, and that can handle meshes of arbitrary topology [BPZ99].

Because simplified models are crude approximations of the original model, their vertices may be quantized with fewer bits without significantly increasing the geometric error. Thus, upgrades should combine mesh refinements and the encoding of the higher precision bits for vertex coordinates [LK98, KR99].

3. Overview

The CPM approach introduced here is based on the notion of a global upgrade, also called *Implant Sprays* technique. As in *Progressive Meshes* (PM) [Hop96], a crude model is transmitted first and then refined progressively through a series of vertex splits, which are the inverse of the edge collapse operations introduced in [HRD+93], see Figure 2 for an example.

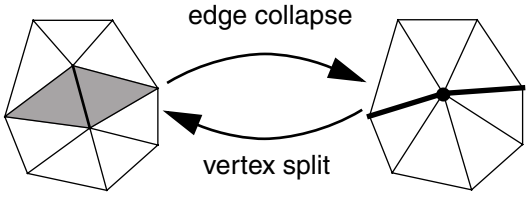


FIGURE 2. Edge collapse and vertex split for triangle mesh simplification and reconstruction. The cut-edges are marked with thick lines.

The cost of encoding each vertex split operation individually in the Progressive Meshes approach presented in [Hop96] has three components:

1. $\lceil \log_2(n) \rceil$ bits are needed to identify the vertex v to be split, where n is the number of previously recovered vertices.
2. $\lceil \log_2(d \cdot (d-1)) \rceil$ [1] bits are used to identify the two cut-edges amongst all the $d$edges incident upon vertex v.
3. 31 to 50 bits are used to encode the displacements of the new vertices with respect to v.

The CPM approach improves on all three components:

1. We group the vertex splits into batches, each splitting about 50% of the previously decoded vertices. We traverse the mesh and specify split-vertices by marking vertices with one bit instead of indexing them explicitly as done in [Hop96]. The amortized cost of this marking is less than 3 bits per vertex for the entire model – compared to more than 10 bits needed by the PM approach when compressing an average model. In [SvK97] a triangle-tree traversal is used to avoid expensive point location tests for incremental 2D Delaunay triangulations, whereas in CPM a vertex-tree traversal is used to reduce the storage space needed to specify progressive refinements in 3D meshes.
2. We encode the identifiers of the two cut-edges for each split-vertex as a choice of two out of d incident edges. Given the degree d, this can be done using exactly $\lceil \log_2((d \cdot (d-1))/2) \rceil$ [2] bits. Since d is known to the decompressor as well no bounds on the degree have to be assumed. Because split-vertices originate from collapsing two adjacent vertices into one, their degree d tends to be higher than the average (6) in a triangle mesh, in contrast to what was assumed in [Hop96]. Despite the higher average degree of split-vertices, our method requires less than 5 bits per split-vertex on average to encode the cut-edges.
3. We use a novel prediction for the displacement of the new vertices. It reduces the average length of the corrective vectors and compresses the quantized coordinates by 50%. Our vertex displacement prediction could be viewed as a reverse variant of the edge-split *Butterfly subdivision* scheme [DLG90, ZSS96]. It is based on the solution of two simultaneous vector equations, constructed using a generalization of the Butterfly scheme.

In order to reduce the total cost of marking the split vertices, we seek to maximize the ratio of split-vertices at each batch. On the other hand, we must ensure a clear separation of the different cut-edges, so that each pair – belonging to a vertex split – may be encoded without ambiguities with a minimum number of bits. Our solution, detailed in Section 5.1, is the most effective compromise amongst all the variations that we have considered.

As a result, the CPM format takes 50% less storage than the PM format. In fact, our experimental results show that, thanks to the combination of the three new techniques mentioned above, the CPM format is competitive with previously reported *non-progressive* compressed formats [TR98, TG98, GS98, Dee95]. Thus the benefits of progressive refinements come at little or no additional cost.

The CPM method compares favorably with other progressive transmission methods. For example, the PFS experiments in [TGHL98] need up to 50% more bits per triangle for the connectivity information and may require even up to 100% more storage cost for the vertex coordinates than our CPM format. Similarly, [LK98] requires $\lceil \log_2(n+6) \rceil$ bits per vertex for connectivity, which is 50% more than our CPM format for average meshes.

---

1. In [Hop96] the permutation $P(n,k) = \frac{n!}{(n-k)!}$ was proposed to encode the selection of two cut-edges.

2. We propose to use the combination $C(n,k) = \binom{n}{k} = \frac{n!}{k!(n-k)!}$ to encode the *unordered* selection of two cut-edges.

4. CPM format

The CPM compressed format is organized as follows. The crude model, M_0, is stored using a single resolution compressed format [Ros98]. The vertex geometry in M_0 is stored at a reduced resolution optimized for M_0 using ideas from [KR99]. M_0 usually contains 5% to 10% of the number of triangles of the entire model. The second part of the CPM format contains the missing least significant bits of the vertex coordinates of M_0. (For simplicity, we chose not to implement the full progressive coordinate encoding scheme by [LK98] in our experiments.)

The third part of the CPM format contains the sequence of the Implant Sprays refinement batches. These may be downloaded systematically, or only on request, and create the sequence of increasingly precise approximations M_1, M_2, …, $M_{\max}$. For instance, if the model's screen representation remains small, only M_0 may be needed. Each batch $M_i \rightarrow M_{i+1}$ includes the split-vertex marking bits in M_i, the cut-edges encoding for every split-vertex, and the entropy encoded prediction correction vectors of the split-vector.

5. Compression algorithm

5.1 Batched simplification

The full resolution mesh, $M_{\max}$, is simplified in batches, creating a series of meshes $M_{\max}$, $M_{\max\text{-}1}$, …, M_{i+1}, M_i, …, M_1, M_0 of decreasing accuracy. The simplification stops at a crude model M_0, when a given error threshold or mesh complexity is reached. This model M_0 is then used as the initial base mesh for reconstruction, and encoded using an efficient single-resolution mesh compression method [Ros98]. In each simplification batch $M_{i+1} \rightarrow M_i$, the number of triangles $|M_{i+1}|$ is decreased by $3 \cdot \tau_e \cdot |M_{i+1}|$. The ratio τ_e denotes the fraction of edges of M_{i+1} that can be collapsed in the same batch (typically 11%).

Figure 3 shows three out of the eight different levels of detail produced by the CPM method, where the triangles inserted by the previous refinement batch are indicated in red. The batches are created by the CPM compression process by selecting, at each batch about 11% of the model's edges, by collapsing them, and by encoding the information necessary to reverse these steps.[1]

FIGURE 3. Batches of edge collapses

To optimize coding, CPM attempts to maximize the selection ratio τ_e of collapsed edges for a simplification batch $M_{i+1} \rightarrow M_i$. However, to be able to uniquely identify the respective independent

1. Collapsing 11% of the edges reduces the number of vertices by 33% because there are about three times more edges than vertices. A 33% reduction of the number of vertices during simplification results in a 50% relative increase during refinement.

vertex splits in the simplified mesh M_i and to avoid singularities, the following three restrictions for collapsing edges in M_{i+1} must be fulfilled:

1. At most two vertices may be collapsed into one.
2. For each edge $e = (v_1, v_2)$ that will be collapsed and any vertex w that is connected by an edge to both v_1 and v_2, the triple (v_1, v_2, w) must define a valid triangle in M_{i+1}.
3. For each edge $e_1 = (v_1, v_2)$ that will be collapsed and any edge $e_2 = (w_1,w_2)$ forming a quadrilateral (v_1, v_2, w_1,w_2) with e_1 in M_{i+1}, e_1 and e_2 cannot be collapsed in the same batch.

Examples of invalid edge collapses that do not fulfill the above requirements are depicted in Figure 4. As a result of these constraints, a split-vertex in the simplified mesh M_i will yield a single edge in the refined mesh M_{i+1}, and no cut-edge is used for more than one vertex split.

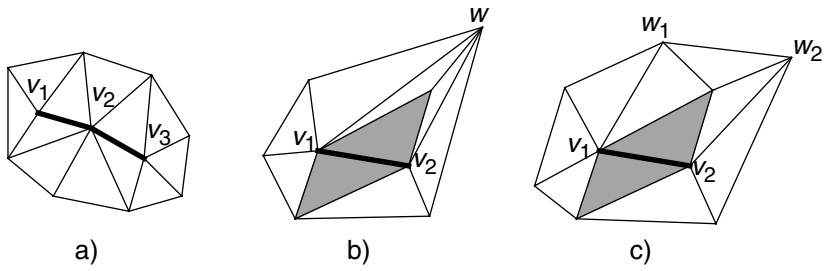


FIGURE 4. Invalid edge collapses during CPM simplification. a) The three vertices v_1, v_2 and v_3 may not be collapsed into one. b) The triple (v_1, v_2, w) is not a valid triangle of the mesh. c) Edges v_1,v_2 and w_1,w_2 may not be collapsed in the same simplification batch.

To achieve a good approximation at each stage, the CPM method uses an error metric to evaluate the error introduced by every single edge collapse. However, the CPM approach is independent of the error metric, which may be selected so as to satisfy the application requirements. The error metric used in our current implementation is an estimation of the *Hausdorff* distance, see Section 5.4 for details. Based on that error metric, the CPM method selects a subset of the less expensive edges that do not violate the constraints defined above. These will be collapsed in the next batch. Different selection strategies might be applied to achieve an optimum with respect to the approximation error introduced per batch. In fact, the batch-wise processing of simplifications in CPM cannot anymore guarantee the same optimal order as proposed in [Hop96] or [GH97].

The current CPM implementation greedily selects collapsible edges in order of increasing approximation error, as long as they do not conflict with the topological restrictions mentioned above. Updating the ordered list of collapsible edges and maintaining a dynamic heap during this selection process is not necessary because all edges that change their error are incident to a selected edge collapse and cannot be collapsed within the same batch because of the restrictions mentioned above. Thus it is both necessary and sufficient to compute the approximation errors and sort the edges accordingly once for each batch. One could also avoid the sorting by selecting edges iteratively with increasing threshold until no more can be selected due to topological restrictions. Choosing a good initial and incremental threshold will result in few iterations.

To reduce the number of coordinates that define a vertex split, all selected edges are always collapsed to their midpoint.

5.2 Connectivity coding

The encoding of the connectivity information needed to restore M_{i+1} from M_i can be summarized as follows:

1. We construct and traverse a vertex spanning tree of M_i and mark split-vertices (i.e. the results of an edge collapse in M_{i+1}). For every marked split-vertex v, we encode its cut-edges as follows:
   2. We compute the indices of the two cut-edges in the sorted list of the incident edges on v, clockwise, starting with the edge from v to its parent in the vertex spanning tree (Figure 5).
   3. Given the degree d of the split-vertex in mesh M_i, the two edge numbers are identified as one possible choice out of $\binom{d}{2}$ for selecting the cut-edges, we encode this choice using exactly $\left\lceil \log_2 \binom{d}{2} \right\rceil$ bits.

Since the degree d of a split-vertex in M_i is known by the encoder and the decoder, in Step 3 we can use a table look-up mechanism for the conversion between the two cut-edge numbers and the index out of $\binom{d}{2}$. Figure 5 illustrates a vertex spanning tree and the corresponding vertex enumeration order. Four vertices are marked as split-vertices (7, 10, 13 and 15). The corresponding cut-edges are indicated. The two cut-edges of a split are identified by a pair of numbers as explained above. For example {0,2} means that edges 0, edge to parent in vertex tree, and 2, second in clockwise ordering, are the two cut-edges of the current split-vertex.

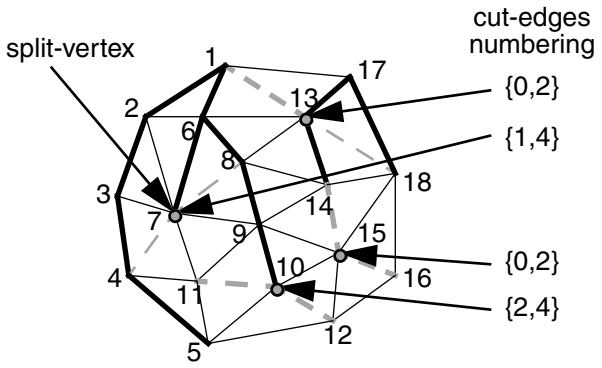


FIGURE 5. Vertex spanning tree traversal, split-vertex marking and cut-edges encoding. The vertex spanning tree is shown using thick lines (solid and dashed), vertices 7, 10, 13 and 15 are marked as split-vertices. The cut-edges are drawn as dashed grey lines and numbered clockwise with youngest first (edge to parent vertex).

5.3 Geometry prediction and coding

To complete the compression we encode the geometry coordinates of the original vertices of the collapsed edges. In CPM we apply the prediction error coding model used for image compression [Kou95] to 3D geometry coordinates. The basic idea is to predict an unknown vector from known vertices and to encode the prediction error. The decompression process uses the same prediction and reconstructs the correct vector by adding the decoded correction.

1. We estimate the originally collapsed vertex locations by A' and B', based on the split-vertex V and cut-edges e and f in mesh M_i.
2. We calculate the prediction correction vector $E = \overrightarrow{BA} - \overrightarrow{B'A'}$ between the estimated and actual vertex locations, see Figure 6.
3. We encode the prediction error E using an entropy coding scheme.

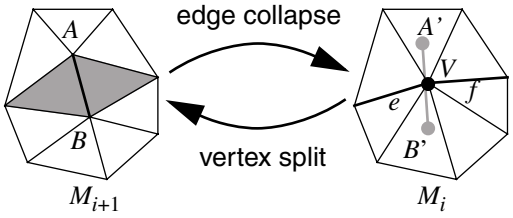


FIGURE 6. Geometry prediction.

CPM uses a prediction method inspired by the *Butterfly subdivision* [DLG90, ZSS96] to estimate the original non-collapsed vertex locations. The Butterfly interpolation inserts a new vertex by splitting an edge as shown in Figure 7, and the location of the new vertex is a weighted sum of the surrounding vertices using the weights shown in Figure 7.

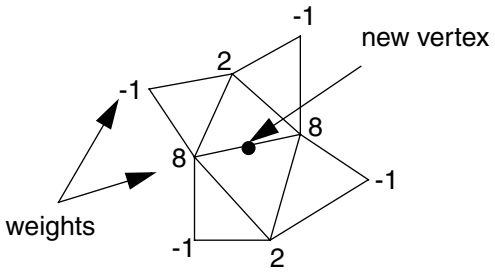


FIGURE 7. Butterfly interpolation.

We extend this idea as follows. A vertex A can be approximated by a linear combination of its immediate neighbors a_i, with topological distance 1 on the triangulation graph, and the vertices c_i at topological distance 2 as shown in Figure 8. The approximation A' of A is:

$$A' = \alpha \cdot \frac{\sum_{i=1}^{k} a_i}{k} + (1 - \alpha) \cdot \frac{\sum_{i=1}^{k} c_i}{k} \qquad \textbf{(EQ 1)}$$

Specifying a value of less than 1 for the parameter α denotes a weighted averaging between the two crowns a_i and c_i, and a value of more than 1 expresses more an extrapolation based on the difference between the two crowns for estimating A. The value of α can be adjusted for each model if needed, in our experiments we constantly used 1.15 which produced consistently good estimates for all models.

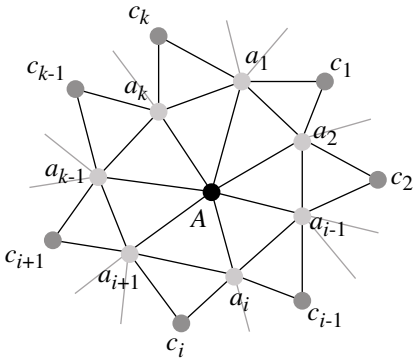


FIGURE 8. Vertex prediction as weighted sum of surrounding vertices.

Using the vertex prediction model of Equation 1, we can estimate the new vertex positions A and B after the vertex split as a linear combination of each other. Figure 9 depicts the vertex prediction

situation for a vertex split, the area of the vertices which are used for the prediction of one of the new vertices is shown in different greys for the two estimates *A*' and *B*'. Using the prediction function of Equation 1 and the notations of Figure 9, we can get a prediction formula for the two original vertices *A* and *B* of an edge collapse and estimate them as:

$$A' = \alpha \cdot \frac{\sum_{i=1}^{k_a} a_i + v_1 + v_2 + B'}{k_a + 3} + (1-\alpha) \cdot \frac{\sum_{i=1}^{k_a+1} c_i + b_1 + b_{k_b}}{k_a + 3} \quad \textbf{(EQ 2)}$$

$$B' = \alpha \cdot \frac{\sum_{i=1}^{k_b} b_i + v_1 + v_2 + A'}{k_b + 3} + (1-\alpha) \cdot \frac{\sum_{i=1}^{k_b+1} d_i + a_1 + a_{k_a}}{k_b + 3} \quad \textbf{(EQ 3)}$$

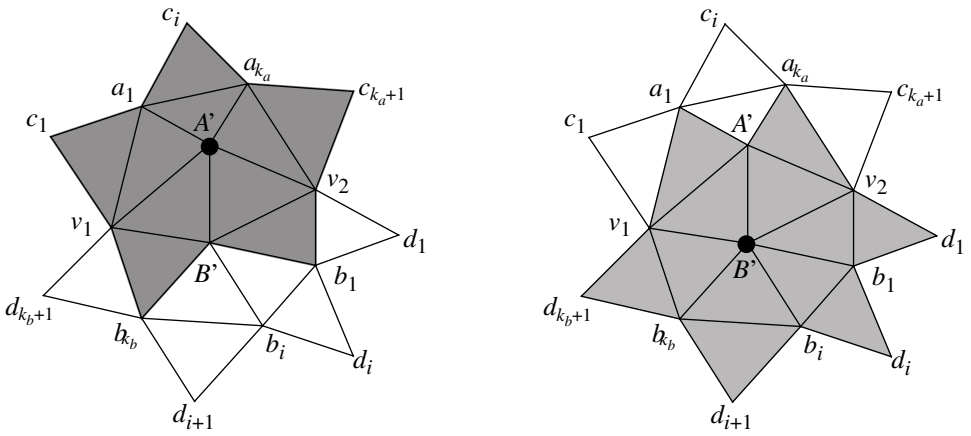


FIGURE 9. CPM Butterfly vertex prediction. a) Shows the vertices used to predict *A*, and b) depicts the situation for estimating *B*.

Based on the collapsed vertex $V = (A+B)/2$ and the split-vector $D = B - A$, we have $A = V - 0.5D$ and $B = V + 0.5D$. Therefore, we can express both Equations 2 and 3 in terms of an estimated split-vector *D*'. To simplify the expressions, we introduce $S_A = (\sum_{i=1}^{k_a} a_i + v_1 + v_2)/(k_a + 3)$ and $C_A = (\sum_{i=1}^{k_a+1} c_i + b_1 + b_{k_b})/(k_a + 3)$ (for S_B and C_B respectively). Thus using *D*', Equations 2 and 3 can be rewritten as:

$$V - 0.5D' = \alpha \cdot \left(S_A + \frac{V + 0.5D'}{k_a + 3} \right) + (1-\alpha) \cdot C_A \quad \textbf{(EQ 4)}$$

$$V + 0.5D' = \alpha \cdot \left(S_B + \frac{V - 0.5D'}{k_b + 3} \right) + (1-\alpha) \cdot C_B \quad \textbf{(EQ 5)}$$

Now we are left with two equations and only one unknown *D*'. Therefore, we have two estimates for *D*' (Equations 6 and 7), D'_A based on the prediction of *A*' and D'_B based on the prediction of *B*'.

$$-0.5 \cdot D'_A = \frac{(k_a+3)((1-\alpha)C_A + \alpha S_A) + (\alpha - k_a - 3)V}{k_a + 3 + \alpha} \quad \textbf{(EQ 6)}$$

$$0.5 \cdot D'_B = \frac{(k_b+3)((1-\alpha)C_B + \alpha S_B) + (\alpha - k_b - 3)V}{k_b + 3 + \alpha} \quad \textbf{(EQ 7)}$$

At decompression time everything is known but the split-vector *D*, for which we want to have a short encoding. Using Equations 6 and 7 we can predict $A' = V - 0.5D'_A$ and $B' = V + 0.5D'_B$, and with $D' = B' - A'$ we get $D' = 0.5(D'_A + D'_B)$. Since *D*' is known at compression and decompression time, we

can encode the prediction error $E = D - D'$ only. At decompression, D is reconstructed by adding the decoded correction vector E to the estimate D'.

One can observe that prediction errors for good estimators have a probability distribution that decreases exponentially with the absolute value of the prediction error. Instead of storing frequency or coding tables for the actual prediction errors of a refinement batch, we approximate the prediction error histogram by a *Laplace* probability distribution:

$$L(x) = \frac{1}{\sqrt{2\upsilon}} e^{-\sqrt{\frac{2}{\upsilon}}|x-\mu|}$$

For symmetric error distributions, the mean μ is 0, and the variance υ uniquely defines the shape of the Laplace distribution. For each batch of vertex splits the variance of the prediction errors, $\upsilon = \sum_i^{E_i \in \text{batch}} (E_i - \mu)^2 / |\text{batch}|$, is computed and encoded with the compressed refinement batch.

Given this probability distribution, entropy coding methods can efficiently compress the quantized coordinate prediction errors. Based on the variance υ and the probability distribution $L(x)$, we compute a different Huffman code [Huf52] for each batch to compress the corresponding prediction errors. In contrast to other geometry compression approaches, CPM does not require to store an entire Huffman coding table for each batch, but only the variance value υ. This is sufficient for the decompression algorithm to reconstruct the required Huffman coding table.

Even though Huffman and arithmetic coding [CNW87] both provide minimal redundancy entropy codes, arithmetic coding yields slightly better compression ratios in the limit. However, the slowness in processing arithmetic codes, and its difficulty streaming different encoding methods within the same bit sequence make it inappropriate fot the intended use.

The CPM experiments in Section 8 demonstrate that this prediction error coding model indeed produces short codings for the geometry coordinates of the tested models.

5.4 Error metric

Most attempts at estimating the error that results from using approximations to nominal shapes for graphics are either limited to view-independent geometric deviations [RB93, HRD+93, KT96, GH97] or to heuristic measures focused on preserving image characteristics, such as the location of silhouettes or highlights [Hop97, LE97]. In the current CPM implementation we chose to use the Hausdorff distance to measure the approximation accuracy. However, instead of using the exact Hausdorff distance the CPM simplification process uses a variation of the *Quadric Error Metrics* [GH97] to estimate the error of a collapsible edge. In CPM we enhanced this error measure by a normalization factor for every error quadric – the number of planes. Note that CPM does not depend on a specific error measure, many other object-space error metrics could be used to estimate the error introduced by an edge collapse.

6. Decompression algorithm

The decompression algorithm performs the inverse operations of the compression process to reconstruct the sequence of meshes $M_0, M_1, \ldots, M_{\max-1}, M_{\max}$. Geomorphs [Hop96] may be used to eliminate the popping effect of each update. At the beginning, prior to the actual CPM method, the base mesh M_0 is decompressed. Thereafter, the individual refinement batches $M_i \rightarrow M_{i+1}$ are decompressed from the CPM file as needed. In each batch, the number of triangles is increased by $\tau_v \cdot |M_i|$

on average. The decompression builds a vertex spanning tree of M_i and uses the same vertex traversal order as the compression algorithm to read and process the CPM vertex markings. Within each batch, the following steps are performed for every visited vertex in the vertex tree traversal of M_i:

1. Read bit of CPM data. If 0, the vertex is not marked to split, continue reading the marking bit for the next vertex. If 1, the vertex has to be split, and we extract the additional vertex split information in steps 2 and 3.
2. Read $\left\lceil \log_2 \binom{d}{2} \right\rceil$ bits of CPM data, where d is the degree of the current marked vertex in M_i and use these bits to identify the corresponding two cut-edges in M_i.
3. Decompress the prediction error vector E from the CPM data, and add it to the estimated split vector D' in M_i to reconstruct the correct split vector as $D = D' + E$ for the current split-vertex.

Note that in Step 1 all cut-edges are numbered with respect to mesh M_i. Therefore, the actual mesh refinements of one batch have to be performed only after all cut-edges of that batch have been identified in M_i.

At the beginning of each batch, the variance υ is read from the CPM data, and the corresponding Huffman table is constructed in the same way as in the compression algorithm. Using that Huffman code, the prediction errors E can exactly be decompressed in Step 3.

7. Amortized storage cost analysis for connectivity

We want to express the number of bits used to encode the connectivity of a triangle mesh as a function of the size of the final mesh. For this, we first consider only one batch of refinement steps that increase the number of triangles from $|M_i|$ in the coarser model to $|M_{i+1}|$ in the finer model. Assume that, in each batch, we can select $\tau_v \cdot |M_i|$ vertices to be split, thus we have $|M_{i+1}| = (1+\tau_v)|M_i|$. Furthermore, to encode such a refinement we need $B \cdot |M_i|$ bits (B bits per triangle in M_i), which means $B/(1+\tau_v)$ bits per triangle in M_{i+1}.

Now we can examine a sequence of refinement batches, each of which increases the number of triangles by a factor of $1+\tau_v$. Expressing the overall cost based on the refined mesh M_i, we derive the following recursive cost function:

$$C(|M_i|) = B \cdot |M_{i-1}| + C(|M_{i-1}|) = B\frac{|M_i|}{1+\tau_v} + C\left(\frac{|M_i|}{1+\tau_v}\right).$$

This recursive cost function can be rewritten as a geometric sum with $\delta = 1/(1+\tau_v)$, as:

$$C(|M_{\max}|) = B\delta|M_{\max}| + B\delta^2|M_{\max}| + \ldots = B\delta|M_{\max}|(1+\delta+\delta^2+\ldots) = B \cdot \delta|M_{\max}|\frac{\delta^k - 1}{\delta - 1}.$$

When the number k of refinement batches is large and because $\delta < 1$, this cost can be bounded as follows:

$$C(|M_{\max}|) \le B \cdot \frac{\delta}{1-\delta}|M_{\max}| = B \cdot \frac{1}{\tau_v}|M_{\max}| \qquad \textbf{(EQ 8)}$$

Therefore, the overall cost to transmit a series of refinement batches can be expressed as B/τ_v bits per triangle. Thus the coding scheme depends strongly on the fraction τ_v of split-vertices that are selected in every batch, and on the encoding of one single batch expressed as B bits per triangle of the batch's input mesh.

The relationship of split-vertices in the coarse mesh M_i to the corresponding edge-collapses in the refined mesh M_{i+1} can be expressed as $\tau_v = 3\tau_e/(1-3\tau_e)$ and $\tau_e = \tau_v/(3-3\tau_v)$. For example:

$$\tau_e = 1/15 \Leftrightarrow \tau_v = 1/4\,,$$

$$\tau_e = 1/12 \Leftrightarrow \tau_v = 1/3\,,$$

$$\tau_e = 1/9 \Leftrightarrow \tau_v = 1/2\,.$$

One can observe that any independent set of vertices in M_i could be used as split-vertices defining an update batch, since the cut-edges of two independent vertices in M_i cannot possibly interfere with any of the restrictions mentioned in Section 5.1. Thus a set of edge collapses that forms an independent set of split-vertices provides a lower bound for τ_e in M_{i+1}, respectively split-vertex selection ratio τ_v in M_i. However, we can select more independent edge collapses in M_{i+1} than only those which map to an independent set of split-vertices in M_i. Therefore, the 4-coloring theorem of planar graphs provides a lower bound for the ratio $\tau_v \geq 1/4$ of simultaneous vertex splits in M_i, and thus also for the ratio $\tau_e \geq 1/15$ of independent edge collapses in M_{i+1}.

Given the vertex split selection ratio τ_v the overall cost for transmitting the connectivity information of one batch is one bit per vertex, and $\left\lceil \log_2\binom{d}{2} \right\rceil$ bits for every vertex split, thus $B = 1/2 \cdot (1 + \tau_v \cdot \left\lceil \log_2\binom{d}{2} \right\rceil)$ bits per triangle in M_i. Using Equation 8 to calculate the amortized cost, and expressing the cost as bits per triangle in the full resolution mesh $M_{\max}$, CPM achieves the following connectivity encoding cost per triangle:

$$\frac{1}{2} \cdot \left(\frac{1}{\tau_v} + \left\lceil \log_2\binom{d}{2} \right\rceil \right)$$

For practical situations our experiments have shown that $\left\lceil \log_2\binom{d}{2} \right\rceil$ is less than 5 bits on average, and that split ratios of $\tau_v \geq 1/3$ are achievable. Overall, the experiments show that CPM can encode the connectivity of the complete mesh $M_{\max}$, including all intermediate incremental meshes M_0, …, M_i, …, $M_{\max}$ in about 3.5 bits per triangle of the original mesh $M_{\max}$.

8. Experimental results

In all the experiments presented in this section the base mesh M_0 is encoded using the Edgebreaker coding method [Ros98]. It uses 2 bits per triangle to encode the connectivity. The vertex coordinates are not compressed and each vertex uses 3 times the number of quantization bits. In the tables, we use the notation C / Δ and G / Δ to denote the number of bits per triangle used to encode the connectivity (C / Δ) and the geometry (G / Δ). The different meshes that result from incrementally applying the CPM refinement batches, starting with the base mesh M_0, are called levels of detail (LODs).

First we briefly discuss the efficiency of our prediction method. Second, we present compression results for small and hard to compress models that are not overtesselated in terms of size of triangles versus vertex quantization. We decided to include tests with small models to compare our method more accurately against other compression methods which report compression results on such small models. Third, we also tested the CPM method with several large and highly tessellated models.

8.1 Geometry prediction

The efficiency of our geometry prediction method can be measured not only in terms of the raw compression ratio as presented below, but can also be expressed in relation to the average length of collapsed edges. In progressive meshes (PM) [Hop96] and their efficient implementation [Hop98] the coordinates of the vector representing a collapsed edge is encoded without prediction using entropy coding. We compare our prediction error (i.e. the length of the corrective vector) to the length of the collapsed edge. For example if the prediction error is a quarter of the collapsed edge, we can expect the prediction error encoding to use 2 bits less than an encoding of the collapsed edge itself.

Figure 10 shows the prediction errors expressed as percentage of the length of the collapsed edges. The prediction errors are 3 to 4 times smaller than the respective edges. Together with the optimized frequency distribution (exponential, Laplace like error distribution) it allows a short encoding of the coordinates. The last few batches encode the locations of most of the vertices and therefore their compression ratio keeps the average cost low, despite the more expensive encoding of the early batches (see also Table 1).

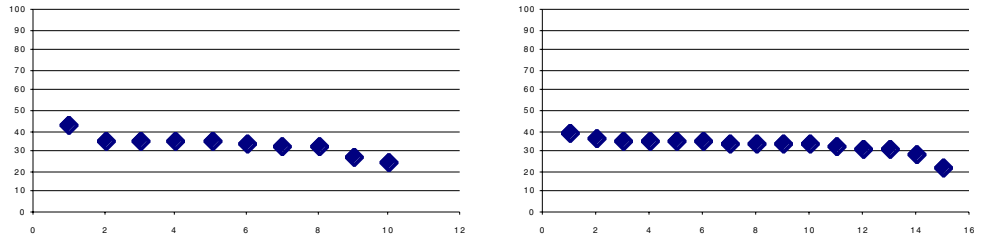


FIGURE 10. Geometry prediction performance. a) Prediction error of the different refinement batches (LODs) expressed as percentage of edge length for the small 10-bit quantized bunny model (Figure 14 and Table 1), and b) for the larger 12-bit quantized bunny (Figure 13 and Table 5).

8.2 Comparative study

Table 1 shows the detailed coding and compression results of the CPM method, applied to a simplified, non-overtesselated bunny model with 4833 vertices, quantized to 10 bits per coordinate. The CPM simplification was stopped at a base mesh M_0 containing approximately 5% of the number of input vertices. With selection ratio of $\tau_v > 1/3$ on average, the CPM method generated a sequence of 10 refinement batches. The rows $M_i \rightarrow M_{i+1}$ show the number of new vertices per batch, the connectivity and geometry bits per batch, and also per triangle. Row M_{10} presents the cumulative costs of the CPM method (including all batches and M_0). The CPM representation requires 3.6 bits per triangle for encoding the connectivity of the bunny model. The vertex coordinates are encoded using about 5 bits each which is equivalent to 7.7 bits per triangle. The approximation errors, i.e. the *Hausdorff* distance, between the original model and the different LODs produced by the CPM method were estimated using the *Metro* tool [CMRS98]. The error graph is plotted in Figure 1 on page 2.

Although the current CPM implementation is aimed more at compression efficiency than speed performance, it processes about 2500 vertex splits per second on a 175MHz R10000 SGI O2. This time includes reading and decoding marking bits, cut-edge codes and geometry coordinates, as well as performing the actual vertex splits and creating the new triangles. Reading and decoding the variable length codes is extremely fast, it uses less than 10% of the processing time. Most of the time is consumed at equal shares for vertex prediction and reconstruction, and for refining the triangular mesh. For example when transmitting the bunny model (of Table 1) over a 56Kbps communication line in a total of 1.9 seconds for all batches (109336 bits), the decoder can process the incoming data on the fly (1.8 seconds for 4590 vertex splits).

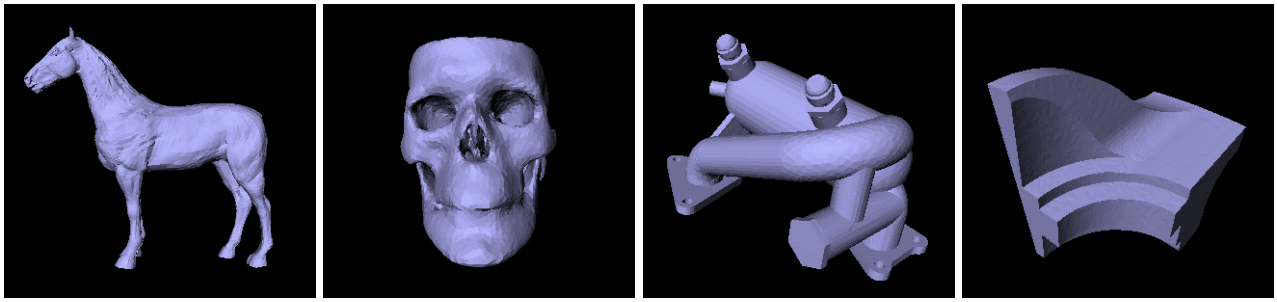
meshes	error	vertices	C bits	C/Δ	G bits	G/Δ	C+G/Δ
M_0	413	243	972	2	7290	15	17
$M_0 \rightarrow M_1$	324	56	516	4.6	1344	12	16.6
$M_1 \rightarrow M_2$	245	88	712	4.1	1888	10.7	14.8
$M_2 \rightarrow M_3$	185	115	943	4.1	2320	10.1	14.2
$M_3 \rightarrow M_4$	138	163	1279	3.9	3120	9.6	13.5
$M_4 \rightarrow M_5$	100	234	1752	3.8	4248	9.1	12.9
$M_5 \rightarrow M_6$	70	331	2467	3.7	5584	8.4	12.1
$M_6 \rightarrow M_7$	48	469	3416	3.6	7392	7.9	11.5
$M_7 \rightarrow M_8$	29	692	4976	3.6	10472	7.6	11.2
$M_8 \rightarrow M_9$	15	998	7081	3.6	13504	6.8	10.4
$M_9 \rightarrow M_{10}$	0	1444	10316	3.6	17744	6.1	9.7
M_{10}		4833	34430	3.6	74906	7.7	11.3

TABLE 1. Progressive compression results for a non-overtesselated 10bit quantized Bunny model. Connectivity information amounts overall to 3.6 bits per triangle, coordinate data requires 7.7 bits per triangle or 15.4 per vertex. See also Figures 3 and 14 for images of the bunny model.

We can compare the CPM compression results to the PFS method [TGHL98] using the bunny model from Table 1, and the horse and skull models from Table 2. The PFS method reports 4.4, 3.9 and 5.0 connectivity bits per triangle for these three models – summation over the 5 LODs of the connectivity per triangle times the number of inserted triangles, divided by the number of triangles in the final mesh – which means that the CPM method improves on the mesh connectivity by 18%, 10% and 32% respectively. The PFS experiments use only 6 bits for coordinate quantization, and apply a smoothing operation to avoid visual artifacts. This makes it difficult to directly compare the geometry compression results. Nevertheless, even using a much finer quantization of 10 instead of only 6 bits per coordinate, the CPM method outperforms the PFS method in terms of geometry compression. Whereas the PFS experiments in [TGHL98] report 18.9, 14.9 and 19.2 geometry bits for the same three models, CPM achieves 7.7, 7.1 and 7.5 bits only, which is an improvement of 59%, 52% and 61% respectively on the geometry compression. Overall, the CPM method improves by roughly 50% on the PFS compression results (bunny 23.3 : 11.3, horse 18.8 : 10.6, skull 24.2 : 10.9).

meshes	vertices	C bits	C / Δ	G bits	G / Δ	C+G / Δ	LODs
horse	10811	75492	3.5	152476	7.1	10.6	9
skull	10952	73381	3.4	163868	7.5	10.9	7
fohe	4005	25545	3.5	73332	10.1	13.7	7
fandisk	6475	48289	3.7	99626	7.7	11.4	9

TABLE 2. Progressive compression results for several 10bit quantized models. See also Figures 11 and 14.



horse skull fohe fandisk

FIGURE 11. Experimental 10-bit quantized models.

A comparison to state-of-the-art single-resolution mesh compression methods is given in Table 3 for two 8-bit quantized models. The column C+G denotes the overall data size in bytes, and the column LODs shows how many different meshes, LODs, were created with the CPM method. The numbers of the two comparing compression methods are replicated from the tables in [TG98]. One can observe that the CPM method mainly suffers in terms of connectivity encoding compared to the single-resolution methods. In some cases, the geometry compression is even better than the proposed single-resolution method [TR98], on the other hand [TG98] requires about half the size for the reported 8-bit models. In contrast to the single-resolution methods, the CPM algorithm provides different LODs, continuous progressive refinements, and produces good approximations of the final shape of the model at early stages during decompression.

		Topological Surgery [TR98]			Touma & Gotsman [TG98]			CPM			
model	vertices	C/Δ	G/Δ	C+G	C/Δ	G/Δ	C+G	C/Δ	G/Δ	C+G	LOD
triceratops	2832	2.2	5.2	5196	1.1	4.1	3701	3.5	5.8	6527	9
shape	2562	1.1	7.1	5291	0.1	4.7	3038	3.5	6.9	6637	10

TABLE 3. Comparison of compression results to single-resolution methods for two 8bit quantized models. See also Figures 12 and 14.

In Figure 12, a complete sequence of meshes demonstrates the progressive mesh refinements achieved by the CPM method. A vertex split ratio of $\tau_v \approx 0.43$ could be achieved, resulting in 9 different LODs M_0, …, M_8. The bits indicated for M_i denote the cumulative bits for transmitting all batches up to M_i, including all intermediate meshes $M_{j<i}$.

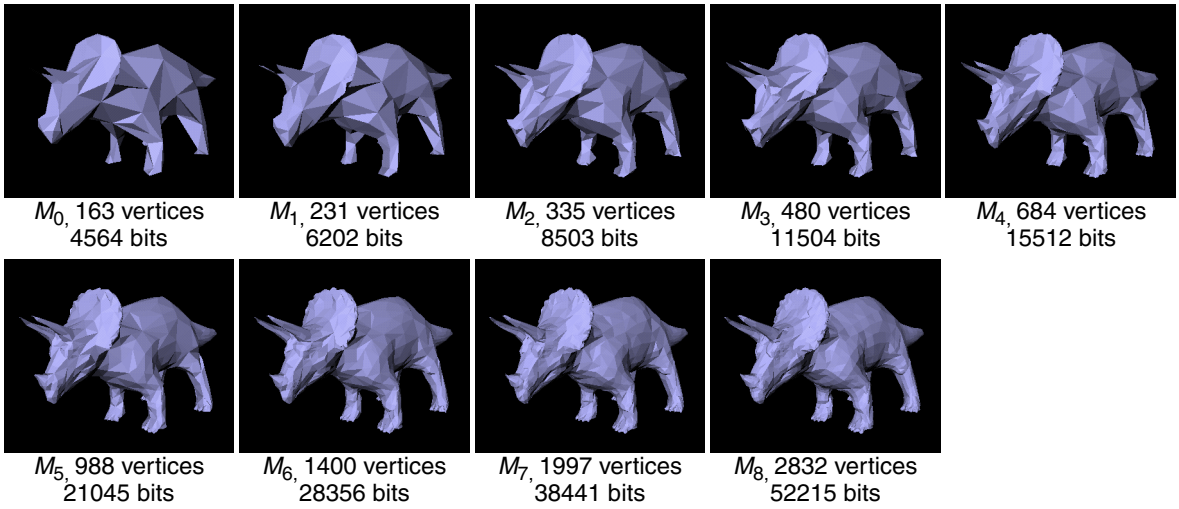


FIGURE 12. CPM sequence of meshes for the triceratops model, showing the number of vertices of each LOD and the cumulative bits representing all refinements up to the current LOD.

Figure 14 provides an overview of CPM compression results using a set of non-overtesselated 3D models. The rightmost column shows the original quantized models, the number of vertices, the number of bits needed to store it using a standard binary encoding (three vertex indices per face and three times the number of quantization bits per vertex), and the estimated transmission time needed for that representation using a 56Kbit per second communication line. The other three columns show the base mesh M_0, an intermediate mesh, and the full resolution mesh using the CPM method. The ratio of the number of vertices to the original model is given with each image. The number of bits shown is the accumulated cost needed by the CPM method, as is the transmission time indicated, thus the bits (transmission time) for mesh M_i include all the meshes $M_{j<i}$ too. The compression ratio of CPM compared to a simple binary representation, and thus also the gain in transmission time, is between 1:4 and 1:5. In the full-resolution models of the CPM method (column 3 of Figure 14), the edge collapses and the corresponding two triangles selected by the first batch of the simplification process are highlighted. Note that the CPM method not only saves time and space to transmit or store the complete model compared to a standard binary representation but also provides very good approximations already at small fractions of time, or number of bits, used for the full-resolution model.

An intuitive suggestion would be to apply a general purpose compression method like *gzip* to the binary indexed face list prior to evaluate CPM compression ratio. However, as Table 4 shows dramatically this is hardly useful since the binary indexed face representation cannot be compressed easily using a general purpose compression method. The compression gain using gzip is virtually zero, in some cases it even increases the file size. This might be due to the fact that the allocation of bits for vertex coordinates and vertex indices is already minimized for this representation, and it seems that the resulting bit stream does not incorporate much redundancy or repetition of bit patterns anymore.

meshes	binary	gzip	ratio
bunny	64827	64836	0.99
fohe	51190	50084	1.02
fandisk	87394	86969	1.01
triceratops	33967	33299	1.02
shape	30727	27662	1.11

TABLE 4. General purpose compression applied to binary indexed face list representations. The second column shows the number of bytes used for a binary indexed face list, the third column shows the number of bytes used after gzip compression. The last column provides the compression ratio of uncompressed binary indexed face list compared to gzip compressed representation.

8.3 Large model experiments

In this section we present some experiments with much larger triangulated models than used in the previous section. The bunny model used here has about 70000 triangles, the phone has 150000 and the (happy) statue uses about 100000 triangles. Table 5 provides the CPM compression results for these models which are shown in Figure 13. As expected, the connectivity encoding cost of 3.6 bits per triangle is completely independent of the model size. On the other hand, the geometry compression performs even better on large and finely tessellated models as can be seen from Table 5. Although we used a higher quantization resolution (12 bits) to represent these large models accurately, compared to the smaller models (10 bits), the geometry compression cost is not higher. Therefore, comparing the geometry compression to the quantization resolution yields an ever better performance for the larger models.

meshes	vertices	C bits	C / Δ	G bits	G / Δ	C+G / Δ	LODs	compressed transmission time	uncompressed model size and transmission time
bunny	34834	248403	3.6	570340	8.2	11.8	15	14 sec	4598088 bits, 80 sec
phone	83044	585689	3.6	1132696	6.9	10.5	13	30 sec	11460072 bits, 200 sec
happy	49794	351460	3.6	858128	8.8	12.4	12	21 sec	6572808 bits, 115 sec

TABLE 5. Progressive compression results for large 12bit quantized models, see also Figure 13. The transmission time is estimated with 56Kbit per second. The rightmost column denotes a binary encoding using 36 bits for vertex coordinates and $\log_2|V|$ bits for the vertex indices of an indexed triangle list.

We also compare our CPM approach to an improved implementation of progressive meshes [Hop98]. Even without having the vertex splits ordered in decreasing approximation error, an efficient PM encoding [Hop98] requires more than 5 bits per triangle for connectivity. If approximation quality is taken into account for ordering the vertex splits, the cost increases to more than 7 bits per triangle, which is more than double the number of bits used for connectivity in the CPM method. For example the large bunny and happy models each only need 3.6 bits per triangle for connectivity with our CPM method, whereas [Hop98] reports 8.4 (bunny) and 10.6 bits (happy buddha) per triangle when using an accuracy dependent vertex split ordering. Our geometry compression method performs best on highly tessellated models with coarse quantization, such as the 12 bit quantized models reported in Table 5. Increasing the quantization resolution for the same model results in a lower geometry compression ratio. Nevertheless, also for the 16 bit quantized large bunny and happy models CPM achieves a geometry compression performance of 14 to 15 bits per triangle which is comparable to the coordinate compression ratio reported in [Hop98] when using vertex ordering.

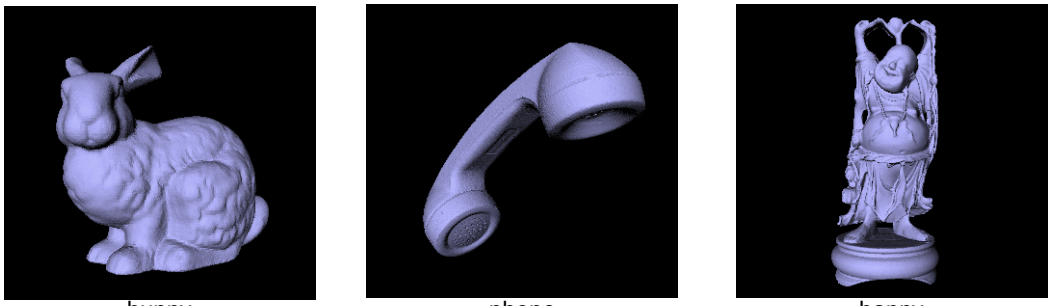


FIGURE 13. Experimental large 12-bit quantized models.

In addition to the timing presented in the previous section, we also timed the decompression speed using the large bunny model. The CPM decoder was able to process more than 3000 vertex splits per second in that case. Again, the vertex prediction and mesh refinement dominated the time cost over reading and decoding the variable length codes.

9. Conclusion

The CPM method introduced here transmits triangulated 3D models through a series of progressive mesh refinements. The progressive mesh representation constructed by CPM preserves topology of the given high-resolution input mesh, and handles manifold triangle meshes of arbitrary genus with boundaries. Although refinements of boundaries cannot be encoded using CPM exactly as described here, simple enhancements of the CPM connectivity coding will allow to do so without significant overhead in encoding cost. Handling of non-manifold meshes can be achieved by segmentation of the model into manifold mesh components [GBTS99] and applying CPM to each manifold part.

The compression efficiency of CPM and total transfer time is comparable to – and sometimes better than – the time required to transfer the original model when using the best non-progressive 3D compression techniques reported so far. However, instead of waiting until the entire transmission is over, the viewer may use a crude but often sufficient approximation of the model after the initial 5 to 10% of this period. Furthermore, that crude approximation is refined incrementally during the transfer through a series of 7 to 15 steps, and its accuracy drastically increased by the first few refinements, often reducing the need to ever transfer the final batches of refinements.

The significant improvements in compression ratios offered by CPM over previously reported compression and progressive transmission techniques result from a new approach which combines three new ideas that were introduced in this paper:

1. Split vertices are identified using a single bit per vertex, rather than $\log_2 n$ bits, as needed by the original PM solution [Hop96].
2. The two cut-edges amongst the d edges incident upon a given split-vertex are identified using an optimal code and a look-up table. The table is defined by the value of d, which is known to both the compression and decompression algorithms, for each split-vertex.
3. The location of the pair of vertices produced by each vertex split is predicted with unprecedented accuracy by our new *Butterfly estimator*, which takes into account the vertices in the topological vicinity of the split-vertex.

We have derived simple validity conditions, which govern the simplification steps of the CPM compression algorithm. These conditions guarantee that our 1-bit-per-vertex marking method is unambiguous and still offer sufficient flexibility for our greedy and simple algorithm to achieve a

25% or more vertex reduction for each batch. Furthermore, because at each batch CPM first marks all the edges that must be collapsed and then collapses them all at once, there is no need, nor benefit, from maintaining a priority queue of the edge candidates. The compression algorithm is thus significantly simpler and more efficient. In contrast, the optimized order of vertex splits in PM [Hop96] provides some approximation quality advantages at the cost of encoding efficiency.

The CPM algorithm and format are independent of the chosen complexity and of the particular technique used to compress the initial crude approximation and of the particular error metric used to select the best edge candidates for simplification at each batch. These may be selected based on the particular application needs.

Our experimental results conducted on a variety of models exhibit a 50% improvement over progressive compression ratios reported elsewhere [Hop96, TGHL98]. The average cost per triangle for transmitting the entire mesh using our progressive method is 3.6 bits for the connectivity and 7.7 for the vertex location (for the bunny model). For the same model, the PM approach would require 8 bits for the connectivity (improved to 5 to 7 bits in [Hop98]) and between 15 and 25 for the geometry. According to the experiments reported in [TGHL98] the PFS method would require 4.4 bits for connectivity and 18.9 bits for geometry.

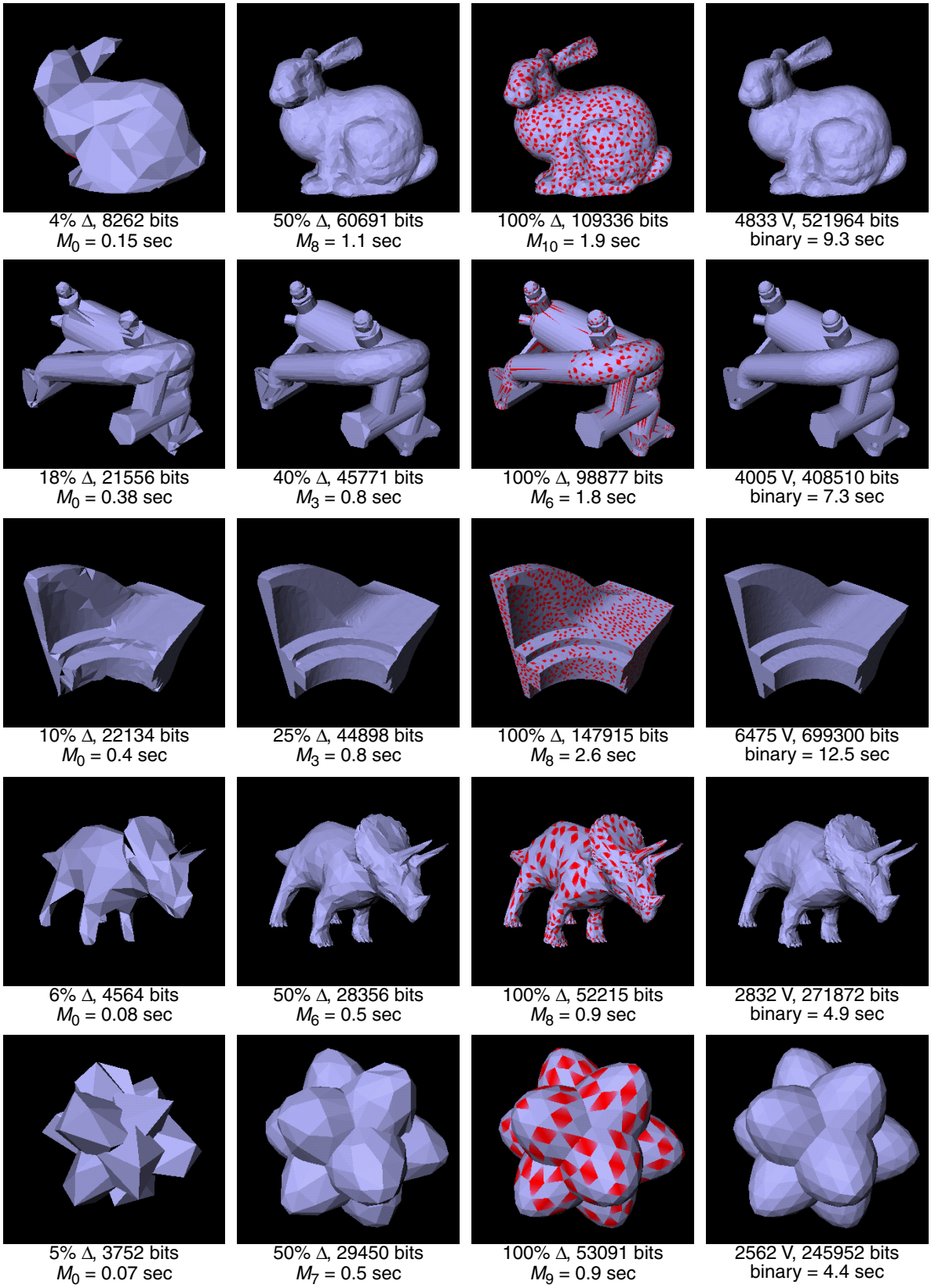


FIGURE 14. The first three columns show three LODs of the CPM method, and the cumulative bits and estimated transmission times using a 56Kbps connection. The forth column references a binary encoding using 30 bits (respectively 24 for the bottom two shapes) for vertex coordinates and $\log_2|V|$ bits for the vertex indices in the indexed face list, and presents the estimated transmission time as well.

Acknowledgments

This work was supported by the Swiss NF grant Nr. 81EZ-54524 and US NSF grant Nr. 9721358. We would like to thank Andrzej Szymczak for helping with coding schemes, Peter Lindstrom for discussions on geometric predictors and subdivision and for providing geometric models, and Davis King for input on accuracy matching vertex quantization. We also thank the anonymous reviewers for their comments, and suggestions to improve the original manuscript.

References

[BPZ99] Chandrajit L. Bajaj, Valerio Pascucci and Guozhong Zhuang. Progressive compression and transmission of arbitrary triangular meshes. In *Proceedings Visualization 99*, pages 307–316. IEEE, Computer Society Press, Los Alamitos, California, 1999.

[CBM97] R. Carey, G. Bell, and C. Martin. The Virtual Reality Modeling Language ISO/IEC DIS 14772-1. http://www.vrml.org/Specifications.VRML97/DIS, 1997.

[CMRS98] P. Cignoni, C. Montani, D. Rocchini and R. Scopigno. Metro: Measuring error on simplified surfaces. *IEEE Computer Graphics Forum*, 17(2):167–174, 1998.

[CNW87] John G. Cleary, Radford M. Neal, and Ian H. Witten. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, June 1987.

[CLR99] Daniel Cohen-Or, David Levin and Offir Remez. Progressive compression of arbitrary triangular meshes. In *Proceedings Visualization 99*, pages 67–72. IEEE, Computer Society Press, Los Alamitos, California, 1999.

[Dee95] Michael Deering. Geometry compression. In *Proceedings SIGGRAPH 95*, pages 13–20. ACM SIGGRAPH, 1995.

[DLG90] N. Dyn, D. Levin and J. A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Transactions on Graphics*, 9(2):160–169, 1990.

[FS93] T. Funkhouser and C. Sequin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Proceedings SIGGRAPH 93*, pages 247–254. ACM SIGGRAPH, 1993.

[GH97] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings SIGGRAPH 97*, pages 209–216. ACM SIGGRAPH, 1997.

[GBTS99] André Guéziec, Frank Bossen, Gabriel Taubin and Claudio Silva. Efficient compression of non-manifold polygonal meshes. In *Proceedings Visualization 99*, pages 73–80. IEEE, Computer Society Press, Los Alamitos, California, 1999.

[GS98] Stefan Gumhold and Wolfgang Strasser. Real time compression of triangle mesh connectivity. In *Proceedings SIGGRAPH 98*, pages 133–140. ACM SIGGRAPH, 1998.

[HRD+93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *Proceedings SIGGRAPH 93*, pages 19–26. ACM SIGGRAPH, 1993.

[Hop98] Hugues Hoppe. Efficient implementation of progressive meshes. Technical Report MSR-TR-98-02, Microsoft Research, 1998. (also in SIGGRAPH 98 Course Notes 21)

[Hop97] Hugues Hoppe. View-dependent refinement of progressive meshes. In *Proceedings SIGGRAPH 97*, pages 189–198. ACM SIGGRAPH, 1997.

[Hop96] Hugues Hoppe. Progressive meshes. In *Proceedings SIGGRAPH 96*, pages 99–108. ACM SIGGRAPH, 1996.

[Huf52] D. A. Huffman. A method for the construction of minimum redundancy codes. In *Proc. Inst. Electr. Radio Eng.*, pages 1098–1101, 1952.

[KT96] A.D. Kalvin and R.H. Taylor. Superfaces: Polyhedral approximation with bounded error. *IEEE Computer Graphics & Applications*, 16(3):64–77, May 1996.

[KR99] Davis King and Jarek Rossignac. Optimal bit allocation in compressed 3D models. Technical Report GIT-GVU-99-07, GVU Center, Georgia Institute of Technology, 1999.

[Kou95] Weidong Kou. *Digital Image Compression: Algorithms and Standards*. Kluwer Academic Publishers, Norwell, Massachusetts, 1995.

[LE97] David Luebke and Carl Erikson. View-dependent simplification of arbitrary polygonal environments. In *Proceedings SIGGRAPH 97*, pages 199–208. ACM SIGGRAPH, 1997.

[LK98] J. Li and C.C. Kuo. Progressive coding of 3D graphics models. *Proceedings of the IEEE*, 96(6):1052–1063, 1998.

[NDW93] J. Neider, T. Davis, and M. Woo. *OpenGL Programming Guide*. Addison Wesley, Reading, Massachusetts, 1993.

[PH97] Jovan Popovic and Hugues Hoppe. Progressive simplicial complexes. In *Proceedings SIGGRAPH 97*, pages 217–224. ACM SIGGRAPH, 1997.

[RR96] R. Ronfard and J. Rossignac. Full-range approximation of triangulated polyhedra. *IEEE Computer Graphics Forum*, 15(3):C67–C76, August 1996.

[Ros98] Jarek Rossignac. Edgebreaker: Compressing the incidence graph of triangle meshes. Technical Report GIT-GVU-98-17, http://www.cc.gatech.edu/gvu/reports/1998, GVU Center, Georgia Institute of Technology, Atlanta, GA, 1998. (to appear in IEEE Transactions on Visualization and Computer Graphics)

[Ros94] Jarek Rossignac. Through the cracks of the solid modeling milestone. In S. Coquillart, W. Strasser and P. Stucki, editors, *From Object Modelling to Advanced Visualization*, pages 1–75. Springer-Verlag, 1994.

[RB93] Jarek Rossignac and Paul Borrel. Multi-resolution 3d approximations for rendering complex scenes. In Bianca Falcidieno and Tosiyasu L. Kunii, editors, *Modeling in Computer Graphics*, pages 455–465. Springer-Verlag, Berlin, 1993.

[SvK97] Jack Snoeyink and Marc van Kreveld. Good orders for incremental (re)construction. In *13th Symposium on Computational Geometry*, pages 400–402. ACM, 1997.

[SL96] M. Soucy and D. Laurendeau. Multiresolution surface modeling based on hierarchical triangulation. *Comput. Vision Image Understanding*, 63():1–14, January 1996.

[SZL92] William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen. Decimation of triangle meshes. In *Proceedings SIGGRAPH 92*, pages 65–70. ACM SIGGRAPH, 1992.

[TG98] Costa Touma and Craig Gotsman. Triangle Mesh Compression. In *Proceedings Graphics Interface 98*, pages 26–34, 1998.

[TGHL98] Gabriel Taubin, André Guéziec, William Horn and Francis Lazarus. Progressive forest split compression. In *Proceedings SIGGRAPH 98*, pages 123–132. ACM SIGGRAPH, 1998.

[THLR98] Gabriel Taubin, William Horn, Francis Lazarus and Jarek Rossignac. Geometric coding and VRML. *Proceedings of the IEEE*, 86(6):1228–1243, 1998.

[HG97] Paul S. Heckbert and Michael Garland. Survey of polygonal surface simplification algorithms. In *SIGGRAPH 97 Course Notes 25*. ACM SIGGRAPH, 1997.

[TR98] Gabriel Taubin and Jarek Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17(2):84–115, 1998.

[TR98b] Gabriel Taubin and Jarek Rossignac. 3D geometric compression. In *SIGGRAPH 98 Course Notes 21*. ACM SIGGRAPH, 1998.

[XEV97] Julie C. Xia, Jihad El-Sana and Amitabh Varshney. Adaptive real-time level-of-detail-based rendering for polygonal models. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):171–183, April-June 1997.

[ZSS96] Denis Zorin, Peter Schröder, and Wim Sweldens. Interpolating subdivision for meshes with arbitrary topology. In *Proceedings SIGGRAPH 96*, pages 189–192. ACM SIGGRAPH, 1996.