Progressive Compression for Lossless Transmission of Triangle Meshes

Pierre Alliez * Mathieu Desbrun †

University of Southern California

Abstract

Lossless transmission of 3D meshes is a very challenging and timely problem for many applications, ranging from collaborative design to engineering. Additionally, frequent delays in transmissions call for progressive transmission in order for the end user to receive useful successive refinements of the final mesh. In this paper, we present a novel, fully progressive encoding approach for lossless transmission of triangle meshes with a very fine granularity. A new valence-driven decimating conquest, combined with patch tiling and an original strategic retriangulation is used to maintain the regularity of valence. We demonstrate that this technique leads to good mesh quality, near-optimal connectivity encoding, and therefore a good rate-distortion ratio throughout the transmission. We also improve upon previous lossless geometry encoding by decorrelating the normal and tangential components of the surface. For typical meshes, our method compresses connectivity down to less than 3.7 bits per vertex, 40% better in average than the best methods previously reported [5, 18]; we further reduce the usual geometry bit rates by 20% in average by exploiting the smoothness of meshes. Concretely, our technique can reduce an ascii VRML 3D model down to 1.7% of its size for a 10-bit quantization (2.3% for a 12-bit quantization) while providing a very progressive reconstruction.

Keywords: Triangle Mesh Compression, Progressive Transmission, Connectivity Encoding, Geometry Encoding, Levels of Details, Mesh Decimation.

1 Introduction

With the growth of e-commerce and entertainment over the internet, the rapid transmission of 3D computer models becomes essential. Both virtual shopping malls and virtual worlds require massive transmissions of triangulated 3D geometric data over the network. In this paradigm, geometry is bound to become as commonplace as text, sound, pictures, or videos. Turning a geometric object into a bit stream is therefore a very timely and relevant problem. However, signals such as sounds, pictures, or movies can rely on Fourier analysis to determine their theoretical entropy, indicating what compression ratio can be achieved. Unfortunately, in the case of a 2-manifold in 3D, we do not have any theoretical results to measure how close to the theoretical compression limit we are: surfaces are almost impossible to analyze with the current mathematical tools due to irregular valences, non-uniform sampling, and the added notion of topology. Designing a new compression algorithm is therefore all the more fundamental as it gives a better understanding of what is the real *information content* of a surface.

*alliez@usc.edu

†desbrun@usc.edu

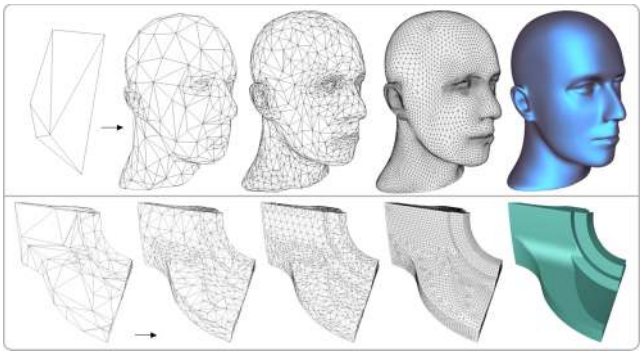

Figure 1: *Our compression technique progressively transmits an arbitrary triangle mesh vertex by vertex. With (*bottom*) or without (*top*) the use of a metrics to drive the vertex ordering, we maintain good mesh quality and good rate-distortion ratio all along the transmission.*

Since data transmission is a transaction between a client and a server, we must take the user's needs into account to judge the optimality of a compression algorithm, and not only the pure *information theory* side of it. An end user may not be very concerned about technical aspects like the bit rate, instead she is likely interested in getting a perceptually good geometric quality in the best time possible. If the server provides data using a single-rate coding, i.e., sending serially vertex after vertex, the user cannot judge the pertinence and quality of the information sent until the full transmission is achieved. This leads to a loss of time (and patience) on the user side, and a loss of network bandwidth for the server. The key idea to address this issue is to use *progressive coding*, where data are sent in a coarse-to-fine way. Optimizing the quality now equates to an optimization of the rate/distortion ratio.

Concretely, progressive compression requires the transmission of a very coarse approximation first, followed by subsequent bits that allow the progressive addition of more and more details. This process will allow the user to get an early grasp of the geometry. An ideal progressive *lossless* coder should reach the same rate as a single-rate one when the transmission is over, with a minimal granularity so that each new bit received by the decoder may be used to refine the current decoded mesh. Thus, progressive coding can be seen (again, ideally) as a simple reshuffling of the data. As recovering original connectivity and vertex position is very important for engineers, scientists, and for interactive collaborative design among other industrial applications, we focus on the design of a novel approach to *progressive, lossless encoding* of arbitrary meshes to try to narrow the current significant gap in bit ratio between single-rate and progressive encoding methods.

A mesh compression algorithm must compress two kinds of information: the *connectivity* (adjacency graph of triangles) and the *geometry* (positions of the vertices). These two types of data are not totally independent, since the Gauss-Bonnet theorem for instance states a necessary condition between geometry and topology of a surface. However, since this condition is global, it is safe and convenient to consider them separately. In order to present how we can optimize the compression ratios of both geometry and connectivity, we first review the previous work in this domain.

1.1 Previous Work

We first briefly mention some of the single-rate coding techniques currently known, since they offer insight into compression

of 3D objects. The reader can find a more detailed overview in [27].

Most of the initial mesh compression techniques use triangle strips as their encoding strategy [6, 2, 26], and/or vertex buffers [10]. The *EdgeBreaker* algorithm [20], however, uses a different strategy: it turns a mesh into a sequence of five-symbol strings using an edge conquest. Using this method a guaranteed 3.67 bit/vertex rate for connectivity is presented in [14]. A very efficient decompression of an *EdgeBreaker* code is introduced in [21], while a better rate for regular models is proposed in [24].

For the last three years, most papers refer (and use) the Touma and Gotsman algorithm [28] as the best single-rate encoder in terms of compression ratio, especially for regular meshes. This technique defines an edge-centered conquest, creating *one valence code per vertex*, and some additional, yet significantly less frequent codes: *dummy* for boundaries, *split* whose frequency is closely related to mesh irregularity, and *merge* for genus greater than 0. The ordered list of all valences and codes generated during the conquest are then processed by entropy encoding. The decoder simply reproduces the conquest according to the flow of valences sent. This technique results in amazingly low bit rates on very regular meshes (valence 6 almost everywhere), since the list is basically a string of sixes, containing almost zero entropy. Recently in [1], the same valence-driven approach is improved upon; more importantly, encoding only valences is proven to lead to the *optimal* bit rate per vertex for arbitrary connectivity, explaining the success of the valence-based techniques.

As for the progressive encoders, Hoppe introduces in [11] an algorithm for progressive transmission, starting from a coarse mesh and inserting vertices one at a time. It uses the edge collapse topological operator in order to decimate and record a sequence of vertex split encodings. The granularity is optimal, but encoding each split requires $log_2(\#v)$ bits both to localize the vertex to split and several bits to still locate its two incident edges to cut. A method called Progressive Forest Split Compression is proposed by Taubin *et al.*in [25], using a base mesh and a forest of vertex splits. Pajarola and Rossignac [18] group vertex-split operations into batches, then traverse the mesh and specify splits by marking each vertex using one bit, leading to an amortized cost of less than three bits per vertex for the marking process. They encode the inverse edge collapse operation by indicating the two edges cut during the vertex split. The geometry is encoded using a butterfly-like prediction approach, but the faster version of that method [19] returns to a simpler predictor for geometry. The connectivity of typical meshes is compressed down to approximately 7.2 bits per vertex.

Leading to an even better bit rate, Cohen-Or *et al.* [5] propose to alternate between a 2- and 4-coloring technique to decimate the mesh. The choice of the coloring is driven by the distribution of valences in a given level of detail. The authors use vertex removal and a deterministic (or geometric) angle-driven retriangulation. Unfortunately, the inside Z-triangulation leads to degenerate meshes, i.e. with long thin triangles. The authors try to compensate for the degeneracy through an immediate 2-coloring pass. However, compared to [18], the geometric quality of the progressive mesh remains worse. On the other hand, they achieve up to 15% better compression rates. Here again, these results cannot compete with single rate methods [28, 20] since their techniques basically increase the dispersion of valence due to the retriangulation.

For encoding the geometry, most papers use prediction, quantization and arithmetic coding. Khodakovsky *et al.* [13] point out the great importance of normal versus tangent decomposition of the relative position for bit allocation in geometry. Devillers and Gandoin [8] totally suppress the order of the vertices, assuming that a geometry-centered triangulation [3] is later able to progressively rebuild the connectivity from the regularity of the point cloud transmitted. Snoyeink *et al.* [23] and Denny and Solher [7] stress that any data already transmitted have defined an implicit order which can be used to save significant entropy.

Since compression ratios and geometry quality are intricately related, King and Rossignac [15] and Khodakovsky *et al.* [13] really look at 3D compression as a rate/distortion problem, rather than from a pure rate viewpoint. Note that [13] obtains the best geometry compression ratios by far, but through a complete remeshing of the 3D surface, which is definitely the best thing to do if only the visual aspect of a surface needs to be transmitted. On the other hand, we propose in this paper a **lossless** encoder, that will transmit an arbitrary mesh in full, yet in a progressive way.

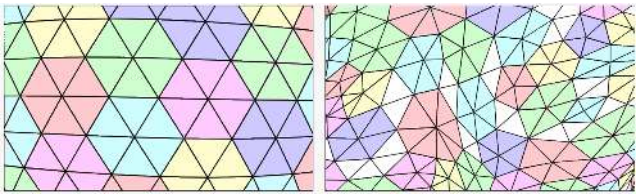

Figure 2: *Left: an optimal independent set of patches, tiling the mesh. Right: a non-optimal independent set of patches with white-colored null patches.*

1.2 Overview

From the exploration of previous encoders, we make the following simple observations. 1) A deterministic conquest avoids an explicit transmission of order over the vertices. It implicitly builds an order that the coder and the decoder will agree on. If we want the geometry to also be progressive, this conquest must depend uniquely on connectivity. 2) An optimal progressive connectivity encoder should generate one valence code per vertex of the original mesh, in any order. This will achieve "minimal" entropy as we know it from [1]. 3) Decimation quality and compression ratio do not seem to be mutually optimizable. Although a perfect decimation is not crucial, care needs to be taken in order to avoid any mesh degeneracies.

Our contributions are built upon these observations. We propose a novel method for the *progressive, lossless encoding* of meshes, aiming at getting as close as possible to the single-rate compression ratios. We use the minimal granularity for both connectivity and geometry, i.e., we decimate (resp. insert) only one vertex at a time during the encoding (resp. decoding) phase. In a nutshell, the encoding algorithm can be roughly described in two stages:

- A valence-driven decimating conquest constructs an independent set of patches (1-rings), alternating between two very simple strategies. Each patch center vertex is then removed, and the patch gets strategically re-triangulated to promote a balanced valence everywhere. We refer to this 2-step decimation as the 3-to-1 (or *inverse* $\sqrt{3}$) simplification for reasons that will be made clear in Section 3.5 (see Figure 11).
- The list of valences of the removed vertices (plus some "cosmetic" codes) is then compressed by a code simplification method that simulates the decoding process to suppress redundancies and a subsequent adaptive arithmetic encoder.

The remainder of this paper details this algorithm, and is articulated as follows: in Section 2, we give conventions and definitions of terms we will use during the description of our algorithm in Section 3, which details the decimation strategy, the patch conquest, and the patch retriangulation. Section 4 will explain how the end user can easily and efficiently decode the bit stream previously encoded. We detail geometry encoding in Section 5, and give multiple results in Section 6 to show both the quality of the hierarchy transmitted and the rate achieved, between 30% and 40% better than previous progressive encoders. We finally give conclusions in Section 7.

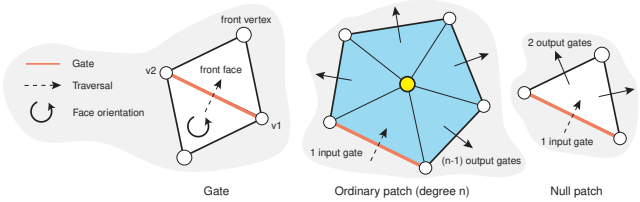

Figure 3: *Left: a gate is defined by its oriented edge. Each gate stores a reference to its front face and its front vertex. Right: the gate item allows the conquest to achieve the mesh traversal through ordinary or null patches.*

2 Definitions

In this section, we briefly describe some relevant definitions we will use throughout this paper. When necessary, we define a graphics convention for each of these definitions to enhance the clarity of the figures.

- Patch: a degree-d patch is a set of faces incident to valence-d vertex (Figure 4.A).

- Vertex removal: operation consisting in removing a vertex and its incident triangles, and subsequently remeshing the remaining hole (Figures 4.B and 4.C).

- Independent set: a set of patches on a mesh where each face belongs to at most one patch. An optimal independent set is achieved when each face of the mesh belongs to exactly one patch (Figure 2.A): the patches then tile the mesh completely.

- Null patch: a face that does not belong to any patch. This occurs when a mesh is not completely tiled by the patches. Null patches are colored in white on Figure 2.B.

- Gate: an oriented edge, storing a reference to its front face (see Figure 3). A gate allows us to go from one patch to an adjacent one during a mesh traversal.

- State flags: each face and vertex can be tagged *free*, *conquered* or *to_be_removed*, depending on their current state.

- Retriangulation tags: each vertex can be assigned a minus $\ominus$ or a plus $\oplus$ according to whether one wants to strategically minimize their valence or to maximize it respectively during a retriangulation. These tags are displayed on our figures when it is required.

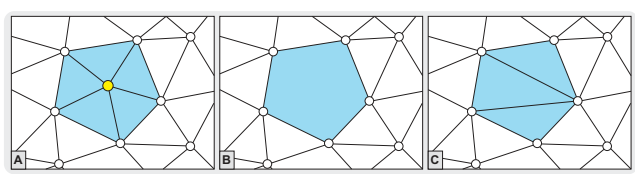

Figure 4: *A: a degree-5 patch. B: removal of the middle vertex. C: retriangulation of subsequent hole.*

3 Progressive Connectivity Encoding

In this section, we detail the core of our progressive encoding algorithm. We explain how a valence-driven conquest, similar in spirit to [1], allows us to decimate a mesh layer by layer using vertex removals, while both maintaining a good mesh quality throughout the simplification and guaranteeing near-optimal compression ratios.

3.1 Decimation/Coding Trade-off

Progressive transmission implies model simplification. We therefore have to define a decimation strategy that will drive our encoder. Decimation techniques usually need an atomic decimation operator, an error metric, and a set of topological constraints. In our case, the most appropriate decimation operator is vertex removal, since it corresponds to the finest granularity of the mesh, therefore inducing the most progressive decimation. Unfortunately, the use of error metrics in reliable decimation algorithms (for instance [9, 12, 17]) leads to almost random vertex removals on the mesh. Coding the random access of such a decimation would be extremely costly compared to a single-resolution conquest [28, 1], since it requires the coding of a localization in a large set of vertices. Moreover, we would prefer not to rely heavily on geometry-centered decisions during the decoding process since we seek independence between connectivity and geometry encoding. It thus seems that one cannot have an optimal decimation and an optimal connectivity encoding at the same time. Our goal to obtain the best rate/distortion ratio at any time during the transmission presents us with a delicate tradeoff. This naturally led us to investigate whether a valence-driven decimation would be more appropriate.

3.2 Valence-driven Decimation

Importance of Low Valence Vertices

We first make the following simple observation: removing a vertex with a valence greater than six (resp., lower than 6) from a triangle mesh and remeshing the subsequent hole leads to an *increase* (resp., a *decrease*) in the sum of the valences of the remaining vertices. This is a direct consequence of the Euler formula. If we write V' the new sum of valences after one valence-v vertex removal and a local remeshing, and V the original sum of all valences excluding this vertex, we have: $V' = V + (v-6)$. Therefore, a vertex removal leads to a systematic change in valence distribution, as shown in Figure 5. Using the connectivity entropy analysis described in Section 1.1, we thus claim that removing a vertex of valence more than six increases entropy: the data excursion of the list of valences globally increases, resulting eventually in a lower compression rate. It is therefore a bad strategy in a compression algorithm seeking the lowest bit cost.

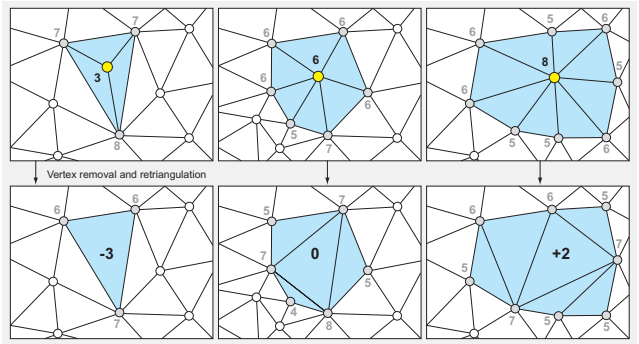

Figure 5: *Influence of a vertex removal on the sum of the valences of remaining vertices: only removals of vertices with valence less than six decrease the sum of remaining valences.*

Additionally, our experiments have shown that removing a high valence vertex often leads to two major inconveniences, *independent of the error metrics chosen*. First, it creates badly shaped triangles (even using a Z-triangulation as in [5]) if no local geometry adjustment is done after the removal, while low valence vertex removals are much safer. Second, a large valence vertex removal is more likely to violate the manifold property or to change the topology of the surface. We thus strongly advocate, as already done in [7], for the removal of the vertices of valence ≤ 6 (≤ 4 on boundaries) since it maintains a low statistical valence dispersion around the average value 6. Such a simple decimation strategy provides an appropriate trade-off between mesh quality and valence excursion. However, we need to deal with a few topology and geometry constraints to ensure good decimation.

Safety Conditions for Vertex Removal

A vertex removal can safely be performed only if it does not violate the manifold property of the surface. However, to make our method more flexible, we let the user select other types of conditions that

may be desirable. Among the vertices of valence ≤ 6 encountered during the conquest and potentially removable, we forbid:

- vertices whose removal leads to violation of the manifold property of the mesh, i.e. when the corresponding remeshing process would create already existing edges.
- vertices whose removal leads to a normal flipping locally; we let the user enable or disable this option according to the desired tradeoff between quality and compression rates, since the normal flipping is not an issue for the encoder, but may be one for the user.
- vertices violating any metrics-related decision designed by the user; simply put, one can decide at any point if a vertex can be removed or not in order to tune the quality of the progressivity. In essence, the decimation algorithm is open to any metrics that would make a better rate/distortion tradeoff for a given application. As mentioned before, this flexibility in quality will obviously result in an additional bit cost.

This valence-driven decimation is very flexible, since any or no error metrics can be used. To prove that using only the valence is safe even without error metrics, all the examples in this paper (except for the fandisk on Figure 1, bottom) do not use any other error metric other than a decimation purely driven by valence. Figure 1 (top) illustrates such a decimation down to an eight-vertex polyhedron.

For objects with specific geometric characteristics like sharp edges, a geometry-driven decimation can substantially improve the perceptual rate-distortion dramatically. As we mentioned, one can skip some important vertices by just sending null patch codes if these particular vertices may better remain present at this stage. We designed a very simple metrics-based decimation mode where each vertex is first checked for validity. We experimented with two metrics-based tests: one based on a vertex-to-patch distance, and one based on the volume embedded between a patch and its retriangulated version (and the area change for boundaries) as depicted in Figure 6. The latter is similar in spirit to [17], except that we use a binary decision for each vertex removal. We normalize the volume error as following: $error = \sqrt[3]{(volume)}/(perimeter/degree)$ so that the same threshold can be used for all the scales. Figure 1 (bottom) shows an example of the fandisk mesh decimation using the volume-based metric and a threshold parameter set to 0.25.

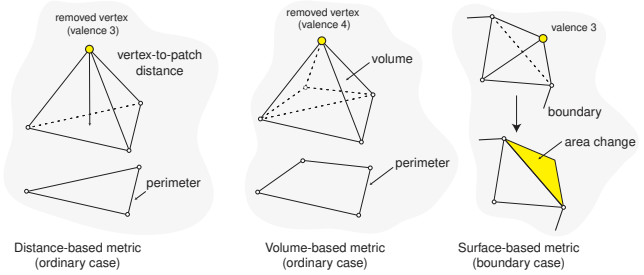

Figure 6: *Left: the vertex-to-patch distance, normalized by the patch perimeter, is a possible error metric. Middle: Better results can be achieved by an error metric defined by the volume included between the original patch and the retriangulated one, still normalized by the patch perimeter. Right: the area change error metric is computed for a boundary vertex.*

3.3 Overview of the Algorithm

Now that our notion of decimation is selected, we can describe the outline of the encoding algorithm. The key idea is that in an orientable manifold, faces incident to a vertex can be ordered. Considering an arbitrary set of patches, any vertex removal followed by a local retriangulation of the patch leaves the patch borders unchanged (see Figure 4). Therefore, these borders can be known by both the encoder (that has the original patch) and the decoder (that has the patch now retriangulated, but with the same borders). As a logical consequence of what we have presented, we can create a *decimating conquest* that decimates vertices of valence less than six encountered along the way, while isolating them in patches to leave the borders intact. For reasons that will become obvious in the two following sections, we alternate such a decimating conquest with a *cleaning conquest*, targeting valence-3 vertices only. This will guarantee an overall nice 3-to-1 decimation.

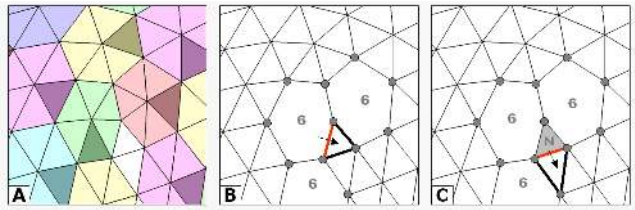

Figure 8: *Null patch during conquest - A: the conquest cannot reach an optimal independent set (null patches are colored in white). B: this null patch has been created when the red gate is popped out from the queue. Its front vertex is already conquered, a code null_patch(N) is therefore output, its front face becomes conquered, and subsequently its two adjacent gates are pushed onto the queue. C: same behavior when the (newly popped) red gate becomes active.*

3.4 Decimating Conquest

Our decimating conquest uses the notion of a gate described in Section 2 as the base tool in order to traverse the mesh and collect adjacent patches. We start with an initial seed gate arbitrarily chosen among the edges of the original mesh. This gate (g_1 in Figure 7), is pushed onto a first-in-first-out (fifo) queue after we flag its two vertices as *conquered*. We then pop the gate out of the queue, and consider the three following cases:

1. *if its front face is tagged conquered or to_be_removed*: There is nothing to do, since the patch we enter has already been or cannot be conquered. We discard the current gate, and proceed to the next gate available on the fifo queue.

2. *else, if its front vertex is free and has a valence* ≤ 6: The corresponding patch will be decimated and retriangulated. The front vertex is flagged *to_be_removed*, its neighboring vertices are flagged *conquered* and its incident faces are flagged *to_be_removed*. The symbol v corresponding to the valence of the removed vertex (or equivalently, the degree of the patch's boundary polygon) is output, and the $v-1$ output gates are generated and pushed to the fifo queue. We discard the current gate, and proceed to the next gate available on the fifo queue. Figure 10(a) and Figure 7 illustrate this general behavior of the conquest.

3. *else, (if its front vertex is free and has a valence* > 6*)*
   or (if its front vertex is tagged conquered): The front face must be a null patch; we declare it conquered, a code *null_patch* is generated and the two other output gates of the triangle are pushed onto the fifo queue (see Figure 8). We discard the current gate, and proceed to the next gate available on the fifo queue.

3.5 Patch Retriangulation

We must now provide a completely deterministic remeshing strategy to fill up the patches after each vertex removal, since the decoder must be able to still find the original borders of the patches. To guarantee a nice remeshing, we designed an adaptive patch retriangulation process driven by a vertex-tagging procedure that allows us to maintain, at no extra cost, a good mesh quality during the decimation while keeping the deterministic behavior. The underlying

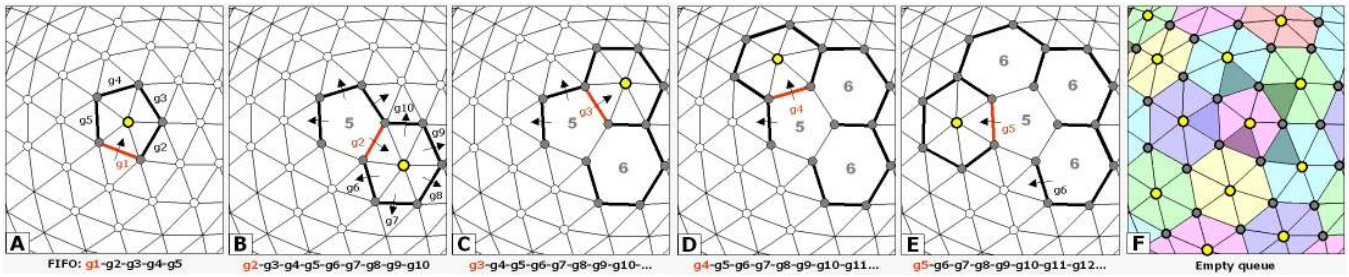

Figure 7: *Standard conquest - A: g_1 denotes the first seed gate pushed into the fifo queue. This gate can conquer the patch of its free front vertex. This valence-5 vertex and its adjacent faces are flagged to_be_removed. Every patch border vertex becomes conquered, four gates are pushed in counterclockwise order into the gate queue and a code 5 is output. B: g_2 becomes active, its valence-6 front vertex becomes to_be_removed, the patch's vertices are flagged conquered, five gates are pushed into the queue and a code 6 is output. C: code 6 patch; D,E: Same behavior, two successive codes 6 being output. F: Eventually, all vertices become either to_be_removed (center of patches) or conquered (borders of patches), colored respectively yellow and grey. The queue is now empty.*

idea is to keep the valence distribution as compact as possible: as we will show, our patch retriangulation leads to a perfect 3-to-1 (or "$\sqrt{3}^{-1}$") simplification if the mesh is regular, and to a good triangulation otherwise.

Each vertex is tagged either $\ominus$ or $\oplus$ during the conquest, depending on whether it is desirable to locally minimize or maximize its valence during the remeshing of its adjacent patches. At the beginning of the conquest, the right and the left vertex of the seed gate are flagged $\oplus$ and $\ominus$ respectively. Then, using the tag table shown in Figure 9, we *retriangulate* the patch and tag its vertices accordingly. We will see during the description of the decoder (see Section 4) that it will allow us to find the borders of a retriangulated patch automatically as soon as the valence of the decimated middle vertex is known. Now, for the subsequent gates, their two vertices will already be tagged, and just a quick look at the table *according to the tags on the gate* allows us to tag the rest of the new patch entered.

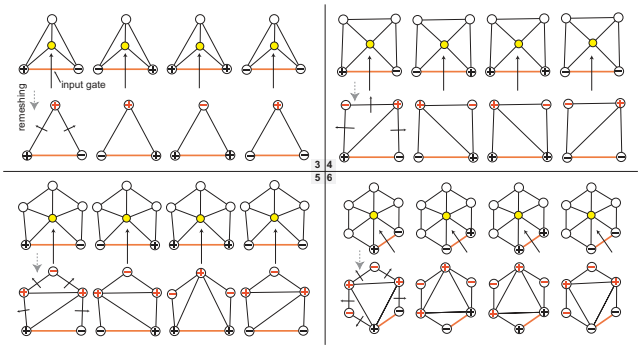

Figure 9: *Adaptive retriangulation process from degrees 3 to 6. The input gates are colored in red, the remeshing decision being only taken from the tags of their two vertices.*

There may be clashes between tags as we conquer more and more patches: if most of the vertices of a patch have already been tagged, we may not be able to have a configuration similar to our table. In this case, we keep the tagged vertices with their original tag, but we triangulate the patch according to the table anyway, and assign the vertices still untagged with the tags from the table. Since the decoder is able to do the same operations (see Section 4 and Figure 9), we keep the deterministic behavior of our retriangulation and suppress any further ambiguities.

The previous remeshing strategy has encouraged on average one out of two vertices on a patch border to be of low valence. As a consequence, our mesh will be now be littered with valence-3 vertices. Our remeshing strategy may therefore seem to be counter-intuitive since we sought to reduce the statistical valence dispersion around the value 6. However, it turns out that a cleaning conquest right after a decimating conquest will in fact improve the mesh considerably, and result in the promised valence balance.

3.6 Cleaning Conquest

The cleaning conquest is almost exactly the same as the decimating conquest, previously defined. The main difference lies in the gates we put in the queue. As it can be noticed in Figure 11.B, the valence-3 patches are now separated by a triangle in the regular case. Therefore, we modify the conquest to put some gates not directly on the border of the current patch, but on the two edges of every face adjacent to the border, and we flag those faces as conquered (see Figure 10(b)).

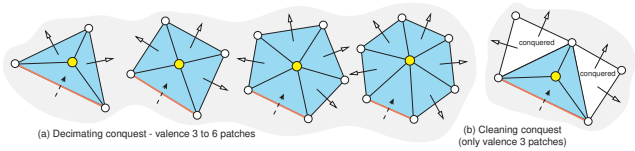

Figure 10: *(a) Mode used during the decimating conquest. Each edge from the patch boundary (but the input gate) is pushed as a gate to the fifo queue. (b) Mode used during the cleaning conquest (only valence-3 patches). Each exterior face adjacent to an edge of the patch boundary is flagged conquered and two gates per conquered face are pushed to the fifo.*

The *only* other difference with the previous decimating conquest is that we restrain our conquest to valence-3 vertices. Figure 11 demonstrates that applied on a locally regular mesh, our coupled conquests will perform a 3-to-1 simplification, which corresponds to an inverse "$\sqrt{3}$ subdivision" [16]. Therefore, a decimation conquest followed by a cleaning conquest will suppress two faces out of three : we obtain near-optimal valence encoding *and* good mesh decimation (see also Figure 1).

The conquest is terminated when the gate queue is empty. We are then done with the first *layer* of decimation; we now may begin the next layer of conquest, starting with a decimating one again. We will stop when we reach the final number of vertices requested by the user, or, very rarely, when we cannot suppress any vertices in a layer without violating topology or metric constraints.

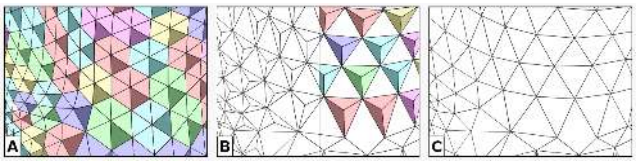

Figure 11: *A: a regular area generates optimal patch tiling. B: remeshing resulting from the vertex tagging process. Formation of extremal valences are encouraged, i.e. valence 3 and 9 in the regular case; the cleaning conquest then targets the valence-3 patches. C: after the cleaning decimation, the simplified mesh is still regular.*

3.7 Arithmetic Coding

The code sequence generated by a decimating conquest is composed of valence codes between 3 and 6, plus some *null_patch* codes, while the sequence generated by a cleaning conquest is only composed of codes 3 and *null_patch*. The decoder, knowing that

we alternate between decimating and cleaning conquests, can systematically replace a 3 code by a 6 code for cleaning odd layers. Indeed, during the decimating conquest, we try to minimize the valence of every other vertex (tagged $\ominus$) on a patch in order to get a significant number of valence-3 vertices, which are easy to remove safely during the cleaning conquest. However, these valence-3 vertices are created only temporarily to keep our retriangulation deterministic, but they were vertices of valence 6, hence the substitution we perform. Since an arithmetic encoder is very sensitive to the occurrence of codes, it allows us to keep the peak of occurrence at valence 6 to optimize the compression. Note also that our technique of cleaning decimation is related to the 2/4-coloring alternation in [5], since we alternate between two different conquests that, conceptually, always go by pairs too.

Since we must also reorder these codes for the decoder, we proceed as follows: Let A_1 be the first sequence generated by a decimating conquest, followed by a sequence B_1 resulting from cleaning, then A_2 is the second decimating conquest followed by B_2, ..., and A_n and B_n being the two final sequences. We first substitute all the 3s by 6s in $B_1,\ldots,B_n$, then we feed the sequence $B_n \cdot A_n \cdot B_{n-1} \cdot A_{n-1} \ldots A_2 \cdot B_1 \cdot A_1$ to an order-0 adaptive arithmetic encoder [29, 22]. We point out that valence codes have *not* been reordered within a layer, only the layers themselves has been reshuffled. We will see in Section 4 that the decoder will then be able to decode this sequence layer by layer *in the same order of conquest than the coder did*, guaranteeing synchronization and correct decoding. Notice that for a very irregular mesh, numerous *null_patch* codes may impede the compression ratio. We thus naturally tried to remove every unnecessary code *null_patch*, and found that *simulating* a decoding stage removes on average one tenth of these accident codes. Since this tasks is achieved by the decoder, we describe it further in Section 4.2.

3.8 Discussion

With our valence-driven decimating conquest, we generate one valence code per vertex optimally, as in [28, 1]. Indeed, if the mesh was very regular to start with, our global strategy will only generate codes six (one per vertex) and the mesh after conquest will remain regular: we will obtain extremely high compression ratios, just like [1] since we encoded *exactly the same zero-entropy sequence, just reordered to create progressivity*. We therefore achieved the optimality sought for very regular meshes. Although a perfectly regular mesh is not at all representative of typical meshes, any local regularity of a mesh will generate a 3-to-1 simplification, while the other regions will have more *null_patch* codes. Roughly, we found our coder to be always 25% to 40% better in compression ratio than other progressive connectivity coders on very irregular meshes, and easily up to 200% better on more regular meshes. We postponed the discussion of the different results with measured rates to Section 6.

4 Progressive Connectivity Decoding

The decoder receives sequences of valence or *null_patch* codes. For a given layer, it *refines* the mesh in the same order as the conquest decimated the vertices during the encoding process. However, we receive the layers in reverse order, so we will start with a "un"-cleaning conquest (we call *3-patch discovery*), followed, by a "un"-decimating conquest (called *patch discovery*), and we repeat. The discovery and the vertex insertions are synchronized through the state of the fifo queue; thus, the decoder will also know which layer needs to be processed, and will be able to toggle between the two-gates/one-gate modes for the patch/3-patch discovery (as was described in Section 3.5). We now detail how the patch discovery and the vertex insertions are done from the transmitted valences or *null_patch* codes, and explain the principle of code packing used in our encoder.

4.1 Patch Discovery and Vertex Insertion

The decoder uses exactly the same strategy defined in the coder (see Section 3), except for the significant difference that *we now have to find the border of a patch each time we cross a gate*. Aside from this particular problem, the implementation is perfectly similar, with the same flags, tags, and the same fifo queue. We therefore refer the reader to the section on coding (Section 3), and we only address the patch border discovery.

When we cross a gate at any time during the discovery, its front face is the seed face of a retriangulated patch that we seek. Now, from the valence we receive through the order-0 adaptive arithmetic decoder [22], we must find a deterministic way to find the borders of this patch. It turns out that the decoder can *deduce* the way to incrementally *walk* on the faces using the current vertex tagging and the same tag tables. Indeed, since we perform the same propagation of tags and flags in the same order, and since the coder had retriangulated the patch according the current tags using the tag table, we know exactly how these retriangulation faces are connected. A few cases are possible, depending on the tags read on the two vertices of the gate, as illustrated in Figure 12. For a valence v code, the discovery must be achieved successfully by *walking on* $(v-2)$ *free faces* (remember this remark for the code packing section). The vertices are then flagged conquered and tagged to minimize or to maximize according to the vertex tagging table: we are then ready to add a vertex of the correct valence inside the patch, and we can proceed to the rest of the discovery.

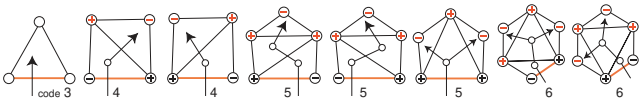

Figure 12: *The polygon discovery is achieved from an active input gate with tagged vertices and one valence code. The tags assigned to the vertices determine how to incrementally walk onto the faces from the current front face of the gate. A valence $v \geq 3$ transmitted code leads to the discovery of $(v-2)$ free faces.*

4.2 Code Packing for Redundancy Elimination

We described in the previous section the basic decoding algorithm that makes the assumption that a valence v code *must lead* to a "discovery" walk onto $(v-2)$ free faces. We can further reduce the bit rate significantly by eliminating some *null_patch* codes due to the fact that this walk is *sometimes not possible* because of neighboring already-conquered faces. Let's suppose the decoder gets a code of valence v. If it is not possible to walk on the $(v-2)$ adjacent faces as defined by the tag table, it must be that the gate being currently treated is actually a null patch, and the vertex valence we received is for later. The decoder can, in that case, resume its discovery after tagging this face as null patch. At the next gate where the $(v-2)$-walk is possible, we know that the valence code was meant for this very gate. This naturally defines a very simple code packing idea for the coder, as we mentioned in Section 3: at the end of the encoding process, we give the current sequence of valence codes to the decoder, which typically removes one tenth of the *null_patch* codes, then we feed the arithmetic encoder with the packed code sequence. To our knowledge, using a simulated decoder to further eliminate redundancies in a code sequence is a novel idea. Due to its extreme simplicity, we believe it could be useful for conquest-centered existing coding techniques.

5 Geometry encoding

Now that we have defined a connectivity encoding method, we must also encode the geometry of the mesh, i.e., the positions of the vertices. Like the vast majority of previous work, we first apply a global quantization step to the mesh vertices, typically using 8 to 12 bits. Then, we can send the geometry information right after each vertex's valence code, to use the implicit order defined by our

conquests. Local prediction, using the probable smoothness and regularity of the mesh, are used in order to further compress these generated geometrical values. Inspired by [13], we also separate normal and tangential components to further reduce the bit rate.

Barycentric Prediction

Each vertex transmitted in our algorithm is added in the middle of a patch, where all the neighbors are perfectly known both by the coder and the decoder (up to the desired quantization). The coder and the decoder can therefore use the barycenter of all the vertices of the patch as a first approximation for the position of the newly created vertex. More sophisticated methods such as butterfly or Loop stencil for prediction turn out to behave badly for irregular meshes; in the absence of smoothness assumption on the mesh, the barycentric prediction is as good as another in practice, as already noticed for instance in [19].

Approximate Frenet Coordinate Frame

The coder or the decoder can also approximate the normal of the mesh locally by a area-weighted sum of the normals of each triangle present in the patch processed. The normal $\mathbf{n}$ and the barycenter $\mathbf{b}$ now define the approximate tangent plane of the surface. We can proceed to compute an approximate Frenet frame, in order to reduce the excursion of the offset vector for smooth surfaces.

To compute the first tangent vector $\mathbf{t}_1$, we simply project the gate onto the normal plane, and normalize the result. The second tangent vector $\mathbf{t}_2$ is obtained through a cross product of $\mathbf{n}$ and $\mathbf{t}_1$: this completes the coordinate frame. Figure 13 illustrates the construction for a set of vertices $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ defining a patch and its faces.

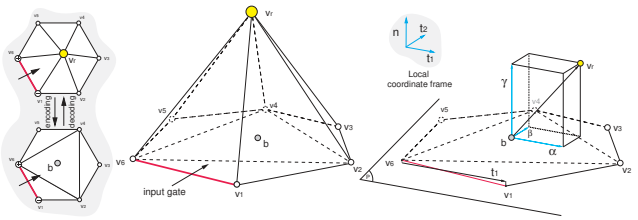

Figure 13: *Prediction method for geometry encoding. The current input gate is colored in red. Residuals are expressed in terms of both tangential and normal components deduced from the current patch's frontier, known for both the coder and the decoder.*

Quantization of Frenet Frame Coordinates

We denote $\mathbf{v_r}$ the vertex position we now want to encode/decode. With the Frenet frame we built, the coder finds the new Frenet coordinate of this point by projection on our basis vectors (see Figure 13): $\mathbf{v}_r = \mathbf{b} + \alpha \cdot \mathbf{t}_1 + \beta \cdot \mathbf{t}_2 + \gamma \cdot \mathbf{n}$. Furthermore, we can round each Frenet coordinate (α, β, γ) to a signed integer value so that the decoder is able to restore the position of $\mathbf{v}_r$ to the same quantized value using a post-quantization. We process all the decimated vertices of a layer as indicated, and find both the range and the offset of each set of tangential and normal coordinates. We communicate this information to the decoder before sending the $\mathbf{t}$ and $\mathbf{n}$ values of all the vertices of the coming layer. With this two-pass technique, we help the arithmetic coder to adapt the number of bits to the range of the geometry. The decoder will just have to add the Frenet coordinates to the barycenter to find the final position of the vertex inserted.

Our experiments showed that this local Frenet quantization allows the adaptive arithmetic coder to achieve a better adaptation to normal and tangential distributions of each layer: as noted in [13], most fine meshes contain much more information in the normal direction that in the tangential directions. Our encoder can therefore adapt to the smoothness (preponderance of normal) and/or the uniformity (very little tangential components) of the mesh encoded. As demonstrated in the next section, this way to deal with geometry is always 15% better than previous approaches, and can potentially be much more for very uniform, smooth meshes.

6 Results

We tested our method on many different meshes, more or less regular, more or less uniform, to fully evaluate our coder performances. On a typical mesh like the horse (see Figure 14), we can reduce the original ascii VRML file to only 2.33% of its initial size for a 12-bit quantization (and only 1.67% for a 10-bit quantization). Note that this is respectively only 1.11 and 1.08 times more than the same mesh encoded by an efficient single-rate encoder [28], but with added benefit of a *fully progressive transmission*. Figure 14 and Table 1 sum up our compression results obtained from some typical meshes; the rate/distortion curve, obtained from Metro [4], proves the scalable behavior of the algorithm. Our current implementation encodes and decodes 5,000 faces/s on a regular PIII PC, handling arbitrary genus and arbitrary number of holes.

The average rate for arbitrary meshes of our progressive connectivity encoding is measured at **3.69 bits per vertex**, not including the optimally regular torus (only shown here to confirm the excellent asymptotic behavior of the algorithm) or the fandisk using an error metric. We obtain a gain of 30% to 45% for the connectivity encoding ratios compared with the best progressive coders published so far [5, 18]. Indeed, Cohen-Or *et al.* [5] and Pajarola *et al.* [18] obtain in average 5.98 b/v and 7.0 b/v respectively on their similar examples. We need to emphasize that our technique allows for a fully progressive transmission, since we use our decimating technique *down to less than 1 thousandth of the vertices* of the original mesh (see Table 1), and often down to the minimum triangulated mesh of equivalent genus. Some previous work remains unclear on their decimation limits, while some others stop at about 20%, significantly loosing a part of the purpose of progressivity encoding.

The horse mesh mentioned above is encoded using 5.72 and 20 b/v by Cohen-Or *et al.* [5] for the connectivity and the geometry respectively. In comparison, we achieve 4.61 and 16.24 b/v for the same geometry quantization. The fandisk is encoded using 6.8 and 15 b/v by Pajarola *et al.* [18] respectively. We achieve 4.99 and 12.34 b/v for the same quantization (see Table 1) when using a volume-based metric (see Figure 1, bottom). These examples are characteristic of our gains for arbitrary meshes. Highly regular and/or highly uniform meshes can be coded much more compactly, since our method exploits both regularity in valence and uniformity in geometry. Therefore, all our results point to the fact that, even if progressive lossless encoding seems to have an intrinsic additional entropy, our technique minimizes the extra cost and renders progressive coding almost as good as single-rate coding.

Models	**Vertices**	**Quant.** #bits	**Connect.** bit/v	**Geometry** bit/v	**Total** bit/v
fandisk	$6475 \rightarrow 4$	10	4.99	12.34	17.39
horse	$19851 \rightarrow 4$	12	4.61	16.24	20.87
nefertiti	$3407 \rightarrow 12$	10	3.95	11.88	16.01
mannequin	$11703 \rightarrow 4$	10	3.58	9.98	13.59
venus	$11217 \rightarrow 16$	10	3.59	10.15	13.82
tiger	$2738 \rightarrow 4$	10	2.67	12.67	15.46
torus	$36450 \rightarrow 24$	10	0.39	3.58	4.02

Table 1: *Compression rates for typical meshes. Notice that all these results decimate the meshes down to about 1 thousandth of their original vertex count to be fully progressive.*

7 Conclusion and future work

We have presented a novel method for progressive, lossless compression of arbitrary triangle meshes. Our contribution is two-fold: we improved significantly upon existing encoders for both connectivity and geometry. We showed that our encoding strategies for

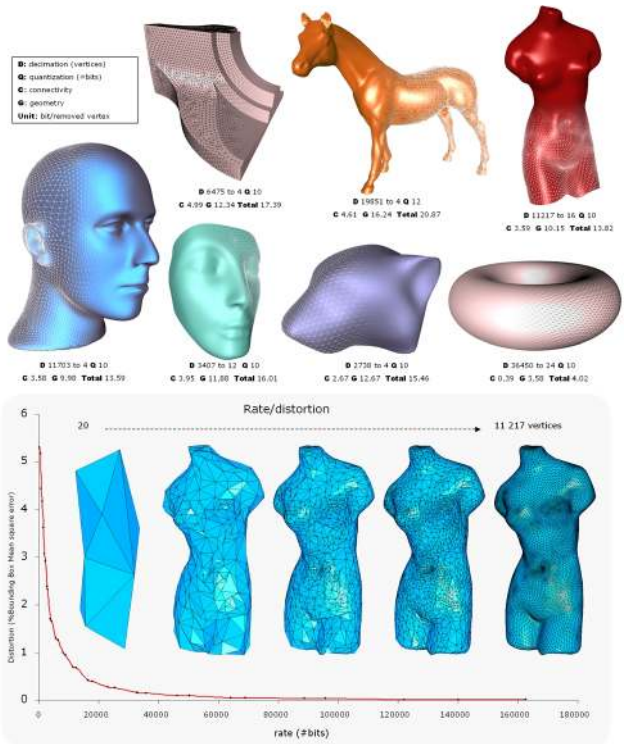

Figure 14: *Top: progressive compression ratios in bits per vertex for models with increasing regularity. Bottom: the venus mesh is progressively transmitted from 20 to 11362 vertices, while its distortion [4] is rapidly decreasing. Note that the sample points on the curve go by adjacent pairs, as our coupled decimating/cleaning conquests do (see Section 3).*

mesh connectivity and mesh geometry are both more efficient than any comparable previous work on all our tests. We even compete with single-rate connectivity coding techniques for very regular meshes, and take advantage of uniformity for geometry encoding as lossy encoders do [13]. Our algorithm defines a new *decimating conquest* that allows us to be very close to the rate of one valence per vertex, the floor for connectivity entropy [1]. We also described an original inverse $\sqrt{3}$ simplification process that combines valence-driven decimation and adaptive retriangulation to maintain the regularity of valence along the progressive encoding. The method presented is also easy to implement while near optimal in the regular case. Finally, it is very flexible and open to any decimation error metrics if needed. In the irregular case, the *null_patch* codes turn out to be the only impeding factor to better the compression ratios. Even if we removed between one tenth and one half of them using an innovative simulation of the decoding process, we believe more work in this area would still increase the benefits of our method. Compression ratio improvements, handling of non-manifold meshes or even polygon soups, and genus reduction/encoding are obvious future work.

Acknowledgements

The authors want to thank: Michael Schindler for help with his range encoder, Peter Schröder and Wim Sweldens for initial discussions, Zoë Wood and Mark Meyer for advice and proof-reading, and finally Magali Mazière for her very precious help. This work has been partially supported by the Integrated Media Systems Center, a NSF Engineering Research Center, cooperative agreement number EEC-9529152.

References

[1] P. Alliez and M. Desbrun. Valence-Driven Connectivity Encoding of 3D Meshes. In *Eurographics Conference Proceedings*, 2001.

[2] R. Bar-Yehuda and C. Gotsman. Time/space Tradeoffs for Polygon Mesh Rendering. *ACM Transactions on Graphics*, 15(2):141–152, 1996.

[3] J-D. Boissonnat and F. Cazals. Smooth Surface Reconstruction via Natural Neighbour Interpolation of Distance Functions. In *ACM Symposium on Computational Geometry*, 2000.

[4] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: Measuring Error on Simplified Surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998.

[5] D. Cohen-Or, D. Levin, and O. Remez. Progressive Compression of Arbitrary Triangular Meshes. In *IEEE Visualization 99 Conference Proceedings*, pages 67–72, 1999.

[6] M. Deering. Geometry Compression. In *ACM SIGGRAPH 95 Conference Proceedings*, pages 13–20, 1995.

[7] M. Denny and C. Sohler. Encoding a Triangulation as a Permutation of its Point Set. *9th Canadian Conference on Computational Geometry*, pages 39–43, 1997.

[8] O. Devillers and P-M. Gandoin. Geometric Compression for Interactive Transmission. In *IEEE Visualization 00 Conference Proceedings*, pages 319–326, 2000.

[9] M. Garland and P. Heckbert. Simplifying Surfaces with Color and Texture using Quadric Error Metrics. In *IEEE Visualization 98 Conference Proceedings*, pages 263–269, 1998.

[10] S. Gumhold and W. Strasser. Real Time Compression of Triangle Mesh Connectivity. In *ACM SIGGRAPH 98 Conference Proceedings*, pages 133–140, 1998.

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

[12] H. Hoppe. New Quadric Metric for Simpliying Meshes with Apperance Attributes. In *IEEE Visualization 99 Conference Proceedings*, pages 59–66, 1999.

[13] A. Khodakovsky, P. Schröder, and W. Sweldens. Progressive Geometry Compression. In *ACM SIGGRAPH 00 Conference Proceedings*, pages 271–278, 2000.

[14] D. King and J. Rossignac. Guaranteed 3.67v bit Encoding of Planar Triangle Graphs. In *Proceedings of the 11th Canadian Conference on Computational Geometry*, pages 146–149, 1999.

[15] D. King and J. Rossignac. Optimal Bit Allocation in 3D Compression. *Journal of Computational Geometry, Theory and Applications*, 14:91–118, 1999.

[16] L. Kobbelt. $\sqrt{3}$-Subdivision. In *ACM SIGGRAPH 00 Conference Proceedings*, pages 103–112, 2000.

[17] P. Lindstrom and G. Turk. Fast and Memory Efficient Polygonal Simplification. In *IEEE Visualization 98 Conference Proceedings*, pages 279–286, 1998.

[18] R. Pajarola and J. Rossignac. Compressed Progressive Meshes. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):79–93, 2000.

[19] R. Pajarola and J. Rossignac. Squeeze: Fast and Progressive Decompression of Triangle Meshes. In *Proceedings of the Computer Graphics International Conference*, 2000.

[20] J. Rossignac. EdgeBreaker : Connectivity Compression for Triangle Meshes. *IEEE Transactions on Visualization and Computer Graphics*, pages 47–61, 1999.

[21] J. Rossignac and A. Szymczak. WrapZip Decompression of the Connectivity of Triangle Meshes Compressed with Edgebreaker. *Journal of Computational Geometry, Theory and Applications*, 14:119–135, november 1999.

[22] M. Schindler. A Fast Renormalization for Arithmetic Coding. In *Proceedings of IEEE Data Compression Conference, Snowbird, UT*, page 572, 1998. http://www.compressconsult.com/rangecoder/.

[23] J. Snoeyink and M. van Kerveld. Good Orders for Incremental (Re)construction. *13th annual ACM Symposium on Comp. Geometry*, pages 400–402, 1997.

[24] A. Szymczak, D. King, and J. Rossignac. An Edgebreaker-based Efficient Compression Scheme for Regular Meshes, 2000. To appear in a special issue of Journal of Computational Geometry: Theory and Applications.

[25] G. Taubin, A. Guéziec, W. Horn, and F. Lazarus. Progressive Forest Split Compression. In *ACM SIGGRAPH 98 Conference Proceedings*, pages 123–132, 1998.

[26] G. Taubin, W. Horn, J. Rossignac, and F. Lazarus. Geometry Coding and VRML. In *Proceedings of the IEEE, Special issue on Multimedia Signal Processing*, volume 86(6), pages 1228–1243, june 1998.

[27] G. Taubin and J. Rossignac. 3D Geometry Compression, 1999-2000. ACM SIGGRAPH Conference course notes.

[28] C. Touma and C. Gotsman. Triangle Mesh Compression. In *Graphics Interface 98 Conference Proceedings*, pages 26–34, 1998.

[29] I.H. Witten, R.M. Neal, and J.G. Cleary. Arithmetic Coding for Data Compression. *Communications of the ACM*, 30(6), june 1987.