ISSN 1470-5559

Mesh Connectivity Compression for Progressive-to-Lossless Transmission

Pengwei Hao, Yakup Paker and Alan Pearmain

Queen Mary
University of London
Department of Computer Science

RR-05-05

June 2005

Mesh Connectivity Compression for Progressive-to-Lossless Transmission

Pengwei Hao[1] Yakup Paker[1] Alan Pearmain[2]

[1]*Department of Computer Science* [2]*Department of Electronic Engineering*
Queen Mary, University of London, UK
[1]*{phao, paker}@dcs.qmul.ac.uk* [2]*alan.pearmain@elec.qmul.ac.uk*

Abstract

A progressive mesh connectivity compression technique is proposed in this paper. Our method is based on the edge collapse and vertex splitting technique for progressive mesh compression-decompression. This is an efficient and reversible method that can be used for progressive-to-lossless transmission. We derive a theoretical upper bound for the lossless connectivity compression bit rate, when the isolated vertices are ignored. Our experiments show that all the meshes we have used can be compressed better than the derived bit rate upper bound. Furthermore, our method can ensure that less than 10% of the vertices generate accidental code. According to our analysis, the optimal bound for triangular mesh coding can be 3, smaller than some of the reported results in recent literature.

1. Introduction

1.1 Purpose of our work

3D mesh coding methods are important for efficient storage and fast transmission of large models. Such polygon models come from a number of sources including computer-aided design, range scanners, terrain mapping and iso-surface extraction from volume data. As model sizes continue to grow, finding efficient and improved compression methods will remain an important problem in computer graphics. In particular if complex scenes are built in 3D and sent over communications lines and need to be animated in real time the compression methods become crucial for the quality of animation to be achieved. A particular application where this is true is the computer games played interactively over the net and shared immersive worlds. In our case, a UK collaborative project called *Prometheus* [1, 11] undertook to develop an end-to-end 3D television chain. In this project virtual scenes are used as sets and avatars represent actors in a studio. Virtual worlds require transmission of massive amounts of triangulated 3D geometric data over the network. For both efficient transmission and real-time rendering of models to and within the end-user computers at various levels of detail, there was a need for an efficient technique of mesh simplification and progressive compression. The technique is based on three aspects of current research: mesh simplification, single-resolution mesh compression, and progressive mesh reconstruction. Our idea has been to find a mesh compression method for progressive-to-lossless transmission which produces good quality simplified meshes with low bit rates and with lossless results comparable to the single-resolution compression methods.

1.2 Summary of related work

The key idea of progressive coding is that data are sent in a coarse-to-fine way. Concretely, progressive compression transmits a coarse approximation first, and uses the following subsequent bits to refine the mesh progressively for more and more details.

As for the progressive encoders, Hoppe introduces in [7] an algorithm for progressive transmission, starting from a coarse mesh and inserting vertices one at a time to refine the mesh. A method called Progressive Forest Split compression is proposed by Taubin et al. in [13], using a base mesh and a forest of vertex splits. Gandoin and Devillers [5] uses the kd-tree encoding of the geometry to drive the connectivity coding process. Pajarola and Rossignac [10] group vertex-split operations into batches, then traverse the mesh and specify splits by marking each vertex using one bit. Khodakovsky et al. [8] proposed a progressive compression algorithm based on the wavelet transform. Cohen-Or et al. [4] presented the patch coloring algorithm for progressive mesh compression based on vertex decimation. In Alliez and Desbrun's valence-driven progressive method [2], they use the minimal granularity for connectivity, i.e., they remove (or insert) only one vertex at a time during the encoding (or decoding respectively) phase, and encode (or decode) the valences of the removed vertices afterwards (or beforehand respectively). A deterministic conquest avoids an explicit transmission of order over the vertices. Of course, progressive compression methods cannot compete with those single-resolution ones [12, 14, 3] since their techniques basically increase the dispersion of valence due to the re-triangulation.

So far, the performance of [2] looks the best, but in terms of the upper bound of bit rate, the method is not so satisfactory for all meshes.

In [6], Gotsman presented some theoretical upper bounds for valence-based mesh coding, 3.24 bpv for triangular meshes. However, according to our analysis, the upper bound for a triangular mesh can be even smaller, 3 bpv.

1.3 Overview of our work

Our work has mainly concentrated on the efficient and reversible mesh simplification methods. In this paper, we mainly summarize the connectivity compression for progressive decompression. Because the edge collapse method is better than a vertex removal method with respect to the simplified mesh quality, our technique is based on the edge collapse and vertex splitting method. We use gates to locate patches, patches to cover the whole mesh, and two indices counted from the gate to locate vertex splitting. We also give a theoretical upper bound estimation of the bit rate for recording the indices. Our experiments show the efficiency of our method.

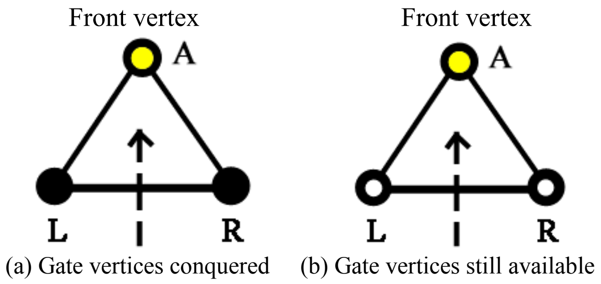
2. Progressive Simplification and Reconstruction

In order to simplify a mesh, we need to select an edge of a vertex pair to collapse. Different strategies of vertex pair selection result in different simplified meshes, which also give different encoding efficiencies. We use a strategy that conquers patches of collapsed edge and sustaining vertices to cover the whole mesh.

2.1 Definitions

A *patch* consists of all adjacent triangles of one vertex. The vertex is, therefore, the *central vertex* of the patch. A *gate* is an edge along the boundary of a patch used to traverse or conquer the patch. Those traversed patches and the vertices that belong to the patch are considered as *conquered*. A vertex or a patch is called *available* if it has not been conquered. A central vertex of a patch is called *isolated* if there is no adjacent edge that can be selected to collapse in the encoding phase or there is no vertex splitting needed in the decoding phase. Therefore, any patch of a simplified mesh has a central vertex that is either isolated or collapsed and is waiting for splitting when decoding. Isolated vertices generate additional accidental codes in the encoded bit stream, for which the additional operations are taken as *splits* in [6] and some other publications.

The vertex at the front triangle of a gate is named the *front vertex*, similar to that in [2]. As illustrated in Figure 1, the two gate vertices are the left vertex L and the right vertex R, where the front vertex is A. The two gate vertices can be conquered (a, filled black vertices) or not (b, hollow circles).


Front vertex
A
L
R
(a) Gate vertices conquered
Front vertex
A
L
R
(b) Gate vertices still available
Figure 1. Definition of a gate and its front vertex

A gate to find the edge to collapse is also a gate to the new patch. Gates are used to locate patches in the encoding phase and relocate patches in the decoding phase.

We take *levels-of-detail* (LoDs) as a level of the simplified mesh at which we cover a mesh with adjacent but un-overlapped patches.

2.2 Encoding: Edge Collapse and Patch Conquest

The encoding phase is to simplify a mesh and to record the necessary information for mesh reconstruction in the decoding phase. Starting from the first gate in the original mesh, we collapse an edge adjacent to the front vertex and take the newly re-triangulated patch as conquered, and then push all the patch boundary vertices into a queue. Then we pop up these vertices in pairs as gates to locate the next adjacent patches to conquer. We continue with this process to conquer the adjacent patches, if available, until the whole mesh is covered. Figure 2 gives a demonstration of a general case of edge collapse in the encoding phase.

The specific algorithm is as follows:

1. At each LoD, add the first candidate gate into the gate candidate queue as the program initialisation, and repeat, following the patch conquest steps, until there is no vertex collapsed in the iteration.
2. Get a candidate gate from the queue. If the queue is empty, go to Step 1 to simplify the mesh for the next LoD.
3. Check if both the gate vertices L and R are available. If neither, check the next candidates in the queue.
4. Cross the gate of vertex L and vertex R and find the front vertex A. (As in the direction of the dashed arrow in Fig. 2a)
5. Starting from vertex L, go clockwise to vertex R to search all the adjacent vertices of A for an edge AB to collapse. (For demonstration, we search along the solid arrows in Fig. 2a and we find the edge AB in Fig. 2b as our selected edge to collapse.) If there is no such edge to collapse, we tag the central vertex as isolated, record this information in the bit stream, and go to Step 2.
6. Collapse the edge AB, insert a new vertex A’ as the central vertex of a new patch, and rebuild the connectivity of the new patch. (As from Fig 2b to Fig. 2c)
7. Count clockwise sequentially from the gate vertex L to locate the first vertex adjacent to the collapsed edge and get one index. (We get vertex 2 in Fig. 2c)
8. Count counter-clockwise from the gate vertex R to locate the other vertex adjacent to the collapsed edge, and get the other index. (We get 1 in Fig. 2c)
9. Encode the two indices of the last two steps by an entropy coder to record them in the bit stream in order to re-locate the vertex splitting connectivity for perfect reconstruction in the decoding phase.
10. Starting from the entrance gate vertex L, go clockwise to vertex R to add all the adjacent vertices into the queue as the candidate gate vertices to adjacent patches. (As in Fig. 2d)
11. Tag vertex A’ and all its adjacent vertices of the patch as conquered (As in Fig. 2d), and go to Step 2.
12. Encode the final coarsest mesh.

Various criterion for the edge selection can be used. In our experiments, we use the maximum valence to determine which edge is selected to collapse. As mentioned in [2], this can reduce the data dispersion and then the final bit rate.

If there is no edge available to collapse, we just take the central vertex as an isolated vertex, but we need to record this information in the bit stream, which obviously reduces encoding efficiency.

The bit streams at different LoDs are reverse recorded in the whole bit stream of mesh coding. The coarsest mesh is encoded and put at the head of the whole code bit stream so that the decoder can reconstruct the mesh.

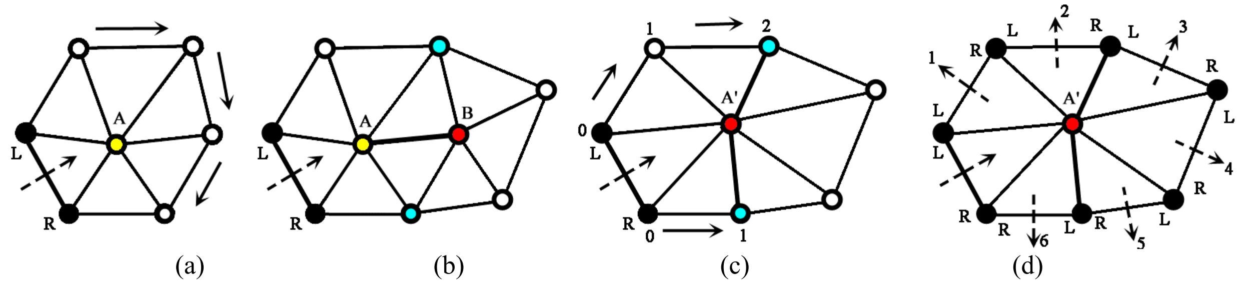


Figure 2 Edge Collapse

2.3 Decoding: Patch Conquest and Vertex Splitting

The decoding phase is just the reverse of the encoding phase. Starting from the first patch in the coarsest mesh, we tag the patch as a conquered patch and push all the patch boundary vertices into a queue, and split the central vertex to produce a finer mesh, and then pop up these vertices by pairs as gates to the next adjacent patches. Afterwards, we conquer these adjacent patches if they are still available.

The specific algorithm is as follows:

1. Decode the coarsest mesh.
2. At each LoD, add the first candidate gate into the gate candidate queue as the program initialisation, and repeat following the patch conquest steps until there is no data in the bit stream.
3. Get a candidate gate from the queue. If the queue is empty, go to Step 2 to refine the mesh for the next LoD.

4. Check if both the gate vertices L and R are available. If neither is available, check the next candidates in the queue.
5. Cross the gate of vertex L and vertex R and find the patch and its central vertex A'. (As in Fig. 2c & 2d)
6. Starting from the entrance gate vertex L, go clockwise to vertex R to add all the adjacent vertices into the queue as the candidate gate vertices to adjacent patches. (As in Fig. 2d)
7. Tag vertex A' and all its adjacent vertices of the patch as conquered (As in Fig. 2d).
8. If the central vertex of the patch is isolated, go to Step 3.
9. Get the indices for vertex splitting from the bit stream.
10. Starting from the gate vertex L, go clockwise and starting from the gate vertex R go counter-clockwise, using the two indices to re-locate the two adjacent vertices of the collapsed edge. (As in Fig. 2c)
11. Split the central vertex into two, and re-triangulate the patch. . (As from Fig 2c to Fig. 2b)
12. Go to Step 3.

From the gate derived from the queue, for each LoD, we locate all the patches in the same order as in the encoder.

If the decoding time is limited such as for periodic rendering of an animated scene, the decoding procedure can be stopped at any time. In this case, the decoded mesh is not losslessly decompressed.

2.4 The Exceptional First Gate

In order to exactly re-locate all the patches in the decoding phase, the first gate must be fixed so that the decoder can find it at any LoD. Therefore, the mesh encoder needs to fix the positions of the first two gate vertices and these are kept the same for all LoDs. Actually, if the first gate is fixed and both of the gate vertices are forbidden to collapse, they stay unchanged through the entire simplification at all LoDs, even to the final coarsest mesh. Obviously, this results in worse mesh quality.

If we take the first gate vertices as the un-conquered vertices, and allow the front vertex to choose either of them to collapse, the simplified mesh should have better quality. Thus, we have three cases of edge collapse for the first gate, as shown in Figure 3. Figure 3a shows a gate to the first patch. We search all the adjacent vertices for an edge to collapse along the arrows. For the first patch, all the adjacent vertices are available for edge collapse. If the vertex chosen is not either of the gate vertices, it is the normal edge collapse case (3b), just the same as the following patch conquests as in Figure 2. If the collapse is with the left vertex of the gate (3c), the re-location index is just from the fixed right vertex. If the collapse happens with the right vertex of the gate (3d), the re-location index should be from the fixed left vertex. To let the decoder distinguish the three cases of the first gate, we only need one number to identify them, say 0, 1 and 2.

3. Upper Bound Estimation of Bit-Rate

The proof of upper bound estimation of bit rate when a mesh is encoded with a valence-driven method is given in [3]. We use a similar idea to give a general formula to estimate the upper bound of similar methods and give an estimate for our method.

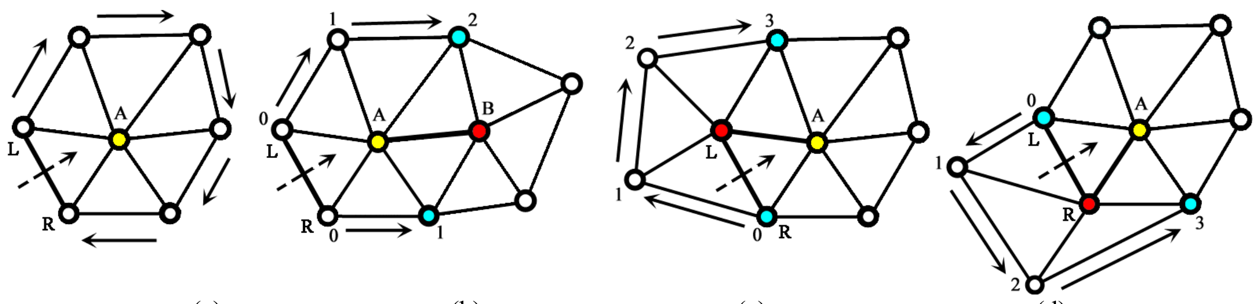


Figure 3 Three cases of the first gate for vertex pair searching

Suppose the average value of an integer sequence is m, and the minimum is k. Let H stand for the entropy of the integers, for $n = m - k$, the upper bound of H is:

$$\max H = \log_2 \frac{(n+1)^{n+1}}{n^n}$$

For the mathematical derivation, please refer to the Appendix of this paper.

For a mesh without boundaries, if the number of faces, edges and vertices are *F*, *E*, and *V* respectively, we have $3F = 2E$; $F + V - E = 2$. Then, we have $E = 3V - 6$ and the total valence of all vertices is $2E = 6V - 12$. So the average valence for arbitrary mesh is approximately 6. If we take the average valence $m = \sum_{i=3}^{\infty} i \cdot p_i = 6$, where the minimum valence is $k = 3$. Then, we have $n = m - k = 3$. Therefore, the corresponding entropy upper bound of valences is the same as given in [3]:

$$\max H = \log_2 \left(\frac{4^4}{3^3} \right) = 3.245$$

For our method, the minimum indices k_1 and k_2 are 0, and the average value of the sum of two indices (m_1, m_2) and 2 should be less than the average valence of a simple mesh, 6. Therefore, we have $k_1 = k_2 = 0$, and we take $m_1 + m_2 + 2 = 6 - 1$ or $m_1 + m_2 = 3$.

Suppose $n_1 = m_1 - k_1$, $n_2 = m_2 - k_2$, we have $n_1 + n_2 = 3$ and $0 \le n_1, n_2 \le 3$.

In the case of the uniform distribution, we suppose $n_1 = n_2 = 3/2$. Then, we can get the corresponding upper bound of bit rate, which is the addition of the bit rate upper bounds of two indices:

$$\max H = \log_2 \frac{(n_1 + 1)^{n_1 + 1}}{n_1^{n_1}} + \log_2 \frac{(n_2 + 1)^{n_2 + 1}}{n_2^{n_2}} = 2 \times 2.427 = 4.85$$

If we use a triangle-remove method for progressive mesh compression, with the upper bound of an integer sequence with the average constraint as shown in the Appendix of this paper and based on our analysis in Section 3, we can easily find the bit rate upper bound of the three indices. To use a new vertex to replace three vertices of a triangle, there are totally two vertices removed after each operation and three indices are needed for decoding. In such a case, we take $k_1 = k_2 = k_3 = 0$, and $m_1 + m_2 + m_3 + 2 = 6 - 1$ or $m_1 + m_2 + m_3 = 3$. Thus, we have $n_1 + n_2 + n_3 = 3$. The worst case is $n_1 = n_2 = n_3 = n = 1$, and we obtain the upper bound:

$$\max H = 3 \log_2 \frac{(n+1)^{n+1}}{n^n} \Big/ 2 = 3 \times 2 / 2 = 3 \quad \text{(bpv)}$$

However, in above analysis, the additional operations for isolated vertices (as *splits* are needed in [6]) are not considered yet.

4. Experiments

We used nine models studied in the literature to make the experiments and test their performance. The encoding bit rates of the complete mesh (for lossless reconstruction) are listed in Table 1. They are as good as we expected. The numbers of the isolated vertices are all less than 10%, and the bit rates of connectivity are all less than our estimated upper bound, 4.85. The time for decoding is very close to that for encoding the same model.

In comparison with the best results by [2], Alliez & Desbrun, 2001, we list their results in the last column.

From the common models used, our results are not always better than theirs, but it is interesting to mention that in the case of their worst examples ours perform better. For models fandisk and horse, their bit rates are 4.99 and 4.61 bpv, respectively, and above than their upper bound. In comparison, our results are 4.06 and 4.47 bpv, respectively, and the bit rates for all our tested meshes are below our upper bound.

The upper bound of their vertex removal method is 3.245 bpv, but their worst case is 4.99 bpv, 1.75 more than that. We believe that their method has less influence on the number of the isolated vertices than ours, which results in cases that exceed the upper bound. In our experience, therefore, our method resulting less than 10% isolated vertices performs better. Therefore, our method gives more flexibility by selecting an edge to collapse and then locating the patch than a technique where a patch is located and then the central vertex is removed.

5. Conclusions

In contrast to the vertex removal method [2], the efficiency results from a lower number of isolated vertices, and this makes our estimated upper bound tight and reasonable. Edge collapse strategy has the flexibility to choose one of the adjacent vertices to collapse, which is obviously produces a lower number of isolated vertices (less than 10%) and also better simplifies mesh quality.

For our future work, we need to implement the corresponding geometric data compression. The literature [9] provides us some clues for efficient encoding which should be useful for our further research.

For our theoretical analysis of our new bit rate upper bound with face removal method, the upper bound should be verified by experiments in future.

Table 1 Experiments with patch conquest strategy of vertex pair searching

Models	Vertices	LoDs	Average Indices		Isolated vertices (%)	connectivity (ours, bits/v)	Connectivity ([2], bits/v)
			Index1	Index2			
fandisk	6475 -> 4	38	1.50	1.24	5.7	4.06	4.99
horse	19851 -> 4	45	1.43	1.28	6.6	4.47	4.61
mannequin	11703 -> 4	41	1.49	1.25	5.7	3.83	3.58
torus	36450 ->12	42	1.36	1.48	3.5	3.04	0.39
dinosaur	14070 -> 4	42	1.47	1.26	7.8	4.59	N/A
eight	766 ->18	23	1.67	1.32	8.3	4.45	N/A
feline	49864 ->13	49	1.44	1.26	7.3	4.48	N/A
sampleavatar	1290 -> 4	32	1.52	1.39	8.0	4.53	N/A

Appendix: Upper bound of an integer sequence with constraints

This proof follows that given in [3], after the derivation of the Lagrange multipliers is corrected.

Consider a series of integer numbers, *i*, distributed from *k* to $+\infty$ (the minimum number is *k*), the distribution probability of number *i* is p_i , its corresponding entropy of the number sequence is $H = -\sum_{i=k}^{\infty} p_i \log_2 p_i$. And, we also have $\sum_{i=k}^{\infty} p_i = 1$ as our one constraint.

Suppose the average value of the sequence is *m*, we have the other constraint, $\sum_{i=k}^{\infty} i \cdot p_i = m$.

Using Lagrange multipliers to find the upper bound of *H* for the integer sequence:

$$f(p_i, \lambda, \mu) = -\sum_{i=k}^{\infty} p_i \log_2 p_i + \lambda(\sum_{i=k}^{\infty} p_i - 1) + \mu(\sum_{i=k}^{\infty} i \cdot p_i - m)$$

Derivatives of *f* with respect to each of p_i must be 0:

$$\log_2 p_i = \lambda - \log_2 e + \mu \cdot i$$

Let $p_i = 2^{\lambda - \log_2 e + \mu \cdot i} = 2^{\lambda - \log_2 e} \cdot \left(2^{\mu}\right)^i = \alpha \cdot \beta^i$, we have

$$\sum_{i=k}^{\infty} p_i = \alpha \cdot \sum_{i=k}^{\infty} \beta^i = \alpha \cdot \frac{\beta^k}{1-\beta} = 1$$

and

$$\sum_{i=k}^{\infty} i \cdot p_i = \alpha \cdot \sum_{i=k}^{\infty} i \cdot \beta^i = \alpha \cdot \frac{\beta^k}{1-\beta} \cdot \frac{k-(k-1)\beta}{1-\beta} = \frac{k-(k-1)\beta}{1-\beta} = m$$

Therefore, the entropy can reach the upper bound when

$$\alpha = \frac{(m-k+1)^{k-1}}{(m-k)^k}, \ \beta = \frac{m-k}{m-k+1}$$

The entropy upper bound then is

$$\begin{aligned}\max H &= -\sum_{i=k}^{\infty} \alpha \cdot \beta^i \cdot \log_2\left(\alpha \cdot \beta^i\right) \\ &= -\alpha \cdot \log_2 \alpha \cdot \sum_{i=k}^{\infty} \beta^i - \alpha \cdot \log_2 \beta \cdot \sum_{i=k}^{\infty} i \cdot \beta^i \\ &= -\log_2 \alpha - m \cdot \log_2 \beta \\ &= (m-k+1) \cdot \log_2 (m-k+1) - (m-k) \cdot \log_2 (m-k) \\ &= \log_2 \frac{(m-k+1)^{m-k+1}}{(m-k)^{m-k}}\end{aligned}$$

Let $n = m - k$, we obtain the bit-rate upper bound of the integer sequence:

$$\max H = \log_2 \frac{(n+1)^{n+1}}{n^n} \quad \text{(bits/number)}$$

References

[1] http://www.bbc.co.uk/rd/projects/prometheus/

[2] Pierre Alliez, Mathieu Desbrun, "Progressive Compression for Lossless Transmission of Triangle Meshes", In *Proceedings of ACM SIGGRAPH*, pp. 198-205, 2001.

[3] Pierre Alliez, Mathieu Desbrun, "Valence-Driven Connectivity Encoding of 3D Meshes", In *Proceedings of EUROGRAPHICS*, v.20, n.3, pp. 480-489, 2001.

[4] D. Cohen-Or, D. Levin, and O. Remez, "Progressive compression of arbitrary triangular meshes," in *IEEE Visualization*, pp. 67–72, 1999.

[5] P.-M. Gandoin and O. Devillers, "Progressive lossless compression of arbitrary simplicial complexes", *ACM Transactions on Graphics*, v.21 n.3, July 2002, pp. 372-379. (Proceedings of ACM SIGGRAPH 2002).

[6] C. Gotsman, "On the Optimality of Valence-based Connectivity Coding", *Computer Graphics Forum*, v. 22, n.1, pp. 99–102, 2003.

[7] H. Hoppe. "Progressive meshes", In *Proceedings of ACM SIGGRAPH*, pp. 99–108, 1996.

[8] A. Khodakovsky, P. Schroder, and W. Sweldens, "Progressive geometry compression", in *SIGGRAPH Proceedings*, pp. 271–278, 2000.

[9] H. Lee, P. Alliez and M. Desbrun, “Angle-Analyzer: A Triangle-Quad Mesh Codec”, In *Proceedings of EUROGRAPHICS*, v. 21, n. 3, pp. 383-392, 2002.

[10] R. Pajarola and J. Rossignac. “Compressed Progressive Meshes”. *IEEE Transactions on Visualization and Computer Graphics*, v.6, n.1, pp. 79–93, 2000.

[11] M. Price, et al, “Real-time production and delivery of 3D media”, In *Proceedings of International Broadcasting Convention*, Amsterdam, Netherlands, Sept 2002.

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

[13] G. Taubin, A. Gueziec. W. Horn, and F. Lazarus. “Progressive Forest Split Compression”, In *Proceedings of ACM SIGGRAPH*, pp. 123–132, 1998.

[14] C. Touma and C. Gotsman. “Triangle Mesh Compression”, In *Proceedings of Graphics Interface*, pp. 26–34, 1998.