POMAR: Compression of Progressive Oriented Meshes Accessible Randomly

Adrien Maglo[a], Ian Grimstead[b], Céline Hudelot[a]

[a] *MAS laboratory, Ecole Centrale Paris, France*
[b] *Cardiff School of Computer Science & Informatics, Cardiff University, United Kingdom*

Abstract

This paper presents a new random accessible and progressive lossless manifold triangle mesh compression algorithm named POMAR. It allows to extract different parts of the input mesh at different levels of detail during the decompression. A smooth transition without artefacts is generated between adjacent regions decompressed at different levels of detail. Our approach allows selective decompression with low delay and compares favourably to previous random accessible and progressive schemes in terms of compression rates.

1. Introduction

With the ever-increasing precision of 3D meshes in application domains such as computer-aided design, simulation, medical imaging, digital heritage and entertainment, solutions must be found for their efficient storage, transmission and visualization. Lossless mesh compression allows the size of the input data to be reduced without losing any information, except a tolerated geometry quantization. Therefore it lowers mesh storage and transmission costs. Static mesh compression codecs can be divided into four types:

Single-rate algorithms build a compact representation of a mesh. The input mesh is fully restored after the decompression.

Progressive algorithms allow successive levels of detail to be rebuilt as more data is decompressed.

Single-rate and progressive algorithms need to decompress the full model to access a specific region.

Random accessible algorithms allow to decompress only parts of the mesh that interest the user. They allow an efficient access to the requested regions of the mesh, but the user does not have any overview of the other regions.

Progressive and random accessible algorithms combine the features of the two previous types of codecs. They provide an efficient access to the requested regions and an overview of the other regions. The user can select the decompression level of detail for each part of the mesh. Few approaches have been proposed in the literature.

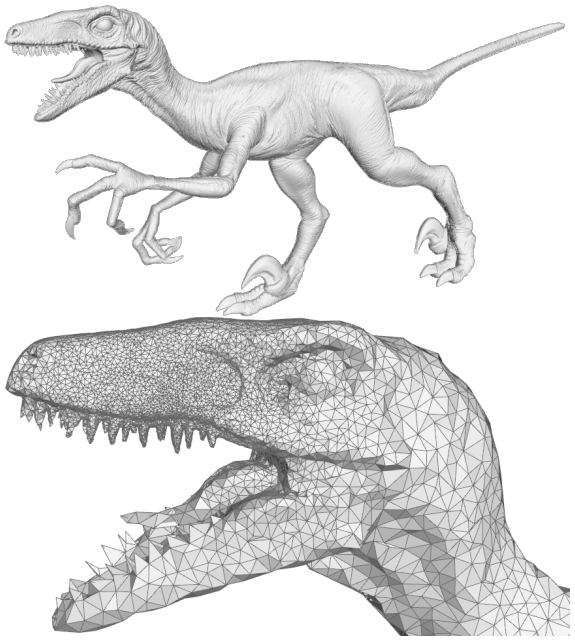

Figure 1: Selective decompression (bottom) of the original raptor model (top) compressed at 15.0 bits per vertex with a 12 bit quantization.

In this paper, we describe a new algorithm, called POMAR, that belongs to this last category: it enables the progressive and random accessible decompression of manifold oriented triangle meshes. During the decompression, each region of the mesh can be decompressed at the level of detail requested by the user. The restriction is that two adjacent regions must have at most one level of detail of difference.

POMAR has been designed for efficient mesh storage and transmission but it also offers interesting features for the interactive visualization. It generates decompressed models with visually good triangles and no artefacts. Unlike some previous approaches, no post-processing operations have to be performed on the boundary between the regions decompressed at different levels of detail. The decompression algorithm requires a low delay and produces smooth transitions between the fine and coarse regions of the mesh.

This paper begins with a study of the related work focused on progressive and random accessible mesh compression. After a general presentation of the proposed scheme, the decimation algorithm, the compression and decompression are described. The influence of the number of clusters for the random-accessibility is then studied. Finally, the article ends with some experimental results, a discussion and a conclusion.

2. Related work

Since the end of the 1990's, mesh compression has been an active research topic. A good review can be found in [1]. Single-rate schemes mainly target the reduction of the mesh storage size. This is often achieved at the cost of inserting dependencies between the mesh simplices. Consequently, if the user wants to access a specific part of a mesh, he must wait for the decompression of the whole mesh. When the mesh is big, the decompression can be time and memory consuming. Some out-of-core approaches such as streaming mesh compression [2, 3] have a limited resource usage, but the data dependency is still persistent.

2.1. Random accessible mesh compression

The aim of random accessible mesh compression is to partially remove the dependencies between the mesh elements. Before the decompression, the user can select which regions of the mesh he is interested in and the algorithm will decompress just these. Two paradigms were proposed in the literature: the cluster-based and the hierarchical representations.

The algorithm of Choe et al. [4] first segments the input mesh and compresses the obtained clusters using a single rate codec [5]. In order to not duplicate the geometry information of the border vertices, their positions are compressed independently. The connectivity between the clusters is encoded in the form of a polygonal mesh. Experimental results show that this method has a small overhead that increases with the number of clusters. Yoon and Lindstrom also built a cluster-based random accessible compression scheme [6]. Their method allows to keep the mesh in a streaming format [2] as it is based on streaming mesh compression [3]. Nevertheless, the compression rates are adversely impacted compared to the approach of Choe et al. [4].

Courbet and Hudelot proposed in [7] a hierarchical mesh representation. The input mesh is recursively split into two partitions until each partition contains only one polygon. Each of the generated sub-meshes can be decompressed independently. The random accessibility granularity is high and polygon meshes can be compressed. However, the compression efficiency for triangle meshes is inferior to cluster-based approaches.

2.2. Progressive mesh compression

Progressive mesh compression techniques allow the extraction of levels of detail during the decompression. The user can see the mesh refining while data is received and decompressed. Levels of detail are also interesting for interactive visualization. Depending on the viewpoint or the visualization device capabilities, the most appropriate level of detail can be displayed. The main focus of progressive mesh compression algorithm is to achieve the best rate-distortion performance. The generated levels of detail must be as close as possible to the initial mesh.

Hoppe first introduced the concept of progressive meshes [8]. During the compression, the input mesh is simplified using the edge collapse operator until the base mesh, which is a coarse version of the original, is obtained. The decompression starts from the base mesh and the vertex split operations, the reverse of the edge collapse operations performed during the compression, are progressively decoded and performed until the initial mesh is restored. Hoppe's method has the advantage of having a high progressiveness granularity but later methods obtained better compression rates.

The progressive forest split representation of Taubin et al. [9] encodes a manifold triangular mesh with a base mesh and a sequence of *forest split* operations. The forest split operation consists of cutting the mesh through several sets of connected edges, filling the generated holes with triangles and relocating the vertices. Pajarola and Rossignac [10] proposed to perform as many as possible edge collapse operations to generate new levels of detail. Vertex splits are then encoded by constructing a vertex spanning tree on the mesh and storing each split vertex. The algorithm of Alliez and Desbrun [11] removes vertices at the centre of patches and retriangulates the remaining holes. The connectivity of the mesh is encoded with the valence of the removed vertices. The geometry is encoded by the vector between

the patch barycentre and the position of the removed vertex.

Gandoin and Devilliers [12] and then Peng and Kuo [13] based their algorithm on a space subdivision approach by using spatial kd-tree and octree data structures respectively. These methods produce high compression ratios. However the rate-distortion performance at low rates suffers of the low geometry quantization of the first levels of detail.

Wavelet decomposition techniques are traditionally reserved for progressive compression of semi-regular meshes. Nevertheless, Valette and Prost proposed the Wavemesh algorithm [14] that contains a wavelet formulation with a lifting scheme for the compression of irregular meshes.

Valette et al. [15] redefined the problem of progressive mesh compression to a mesh generation problem. In their approach, the mesh is first simplified to obtain the base mesh. The compression algorithm progressively adds vertices to build an intermediate mesh that is as close as possible to the initial mesh. The initial connectivity is restored by flipping edges where needed when all the vertices have been added. This technique leads to good rate distortion performance.

More recently, Maglo et al. [16] proposed an algorithm that can compress manifold meshes with arbitrary face degrees. The compression algorithms decimates the input model with a specific local operator which combines vertex removal and polygonal remeshing.

2.3. Progressive and random accessible mesh compression

As explained above, random accessible mesh compression schemes allow the decompression of only the required parts of the mesh, but they do not provide any overview of the other parts. Progressive mesh compression techniques allow levels of detail to be extracted during the decompression. However the full mesh must be decompressed even if only a specific part is required at the finest level of detail. These drawbacks are addressed by the combination of progressive and random accessible techniques, which allow the decompression of different parts of a mesh at different levels of detail.

Kim et al. [17] proposed a multiresolution random accessible mesh compression algorithm based on their previous mesh refinement framework [18]. During the decompression, a vertex can be split even if its neighbours are not identical to the neighbours when the edge was collapsed. This approach leads to a fine-grained multiresolution random access. It breaks the traditional progressive mesh symmetry of operations but the compression performance is limited.

The CHuMI viewer from Jamin et al. [19] partitions the mesh bounding box into a hierarchical structure called the SP-tree. Each SP-cell has a minimum quantization precision. The vertices that belong to several SP-cells are duplicated to allow independent decoding of each cell. SP-cells are encoded with the original Gandoin and Devilliers algorithm [12]. After the decompression, the difference of resolution between adjacent SP-cells is corrected by moving vertices of the cell at the highest resolution.

The approach of Du et al. [20] decomposes the kd-tree of the original Gandoin and Devilliers algorithm [12] into two layers. The top tree and each child sub-trees are compressed independently to enable the random access. The border vertices of each sub-tree cell can be decompressed separately from its internal vertices. The sub-trees must be compressed and decompressed in a canonical order. Thus, to fully decompress a requested sub-tree, the border vertices of the previous sub-trees in the set order must also be fully decompressed. These vertices can be later collapsed if the user does not want them. As this scheme does not duplicate the border vertices, the compression performance should be higher to the one of the CHuMI viewer [19]. Yet, the prefix dependency implies that not requested data must be decompressed.

Maglo et al. [21] extended the cluster-based random-accessible approach from [4] to support the progressive compression of the clusters. They replaced the original single-rate cluster compression algorithm by a progressive one [22], which is based on [11]. This approach, as the original, integrates a time consuming clustering step. Once the clusters are decompressed at the targeted levels of detail, a post-processing step needs to stitch together the parts in order to fill the boundary holes.

2.4. Data structures for view dependent visualization

Progressive and randomly accessible compact mesh data structures were also studied in the context of visualization. Most of these structures are not compressed but they allow the rendering of huge meshes by dynamically coarsening and refining the mesh regions depending on the visualization view point.

Based on progressive mesh [8], the VDPM data structure [23] allows local refinements by encoding the dependency between vertex splits. An extension for huge terrain grid rendering that allow out-of-core processing and geomorph transitions was then proposed in [24]. Pajarola later described the FastMesh data structure [25, 26] that requires less memory than VDPM. However it is restricted to manifold meshes as it is based on an halfedge data structure. The method proposed

by Yoon et al. [27] first clusters the input mesh and then builds a progressive mesh representation for each cluster. Guthe et al. [28] used a geometry octree to partition the mesh. Each node of the octree is simplified with a bottom-up approach. The approach of Shaffer and Garland [29] also decomposes the mesh with an octree. It then assigns to each node a representative vertex computed to minimize the approximation error. Cignoni et al. [30] proposed to decompose the mesh with a tetrahedral structure. Other methods were also proposed to build parallel view dependent progressive meshes [31, 32].

3. Overview

The POMAR compression algorithm performs three main tasks.

Mesh decimation. The input model is first simplified with *halfedge collapses* to generate discrete levels of detail. The decimation stops when a coarse version of the input model, called the *base mesh*, is obtained. All the performed operations are kept in memory to allow the later mesh reconstruction by the encoding steps.

Global level of detail compression. Starting from the base mesh, the successive levels of detail generated by the decimation are reconstructed by performing the reverse operations of the halfedge collapses, the *vertex splits*. The symbols needed to rebuild the levels are encoded for the whole mesh, in the same way as a progressive compression algorithm. We call *cluster* a set of vertices of the input mesh that hierarchically collapsed into a common vertex during the decimation. The global level of detail compression stops when, in the current level of detail, there are as many vertices as desired random accessible clusters. This level is called the *base clustered mesh*.

Clustered level of detail compression. Starting from the base clustered mesh, the levels of detail reconstruction assigns inserted vertices to clusters in function of the vertex splits hierarchy. Each vertex of the base clustered mesh has a corresponding random-accessible cluster. The vertices that collapsed into a common vertex v_c of the base clustered mesh belong to the cluster of v_c. The independent encoding of the vertex splits for each cluster enables the random-accessible decompression.

The mesh decompression happens in the same order as the two compression steps. However, only the desired regions of the mesh are decoded and reconstructed with the constraint that there must be at maximum one level of detail of difference between adjacent clusters.

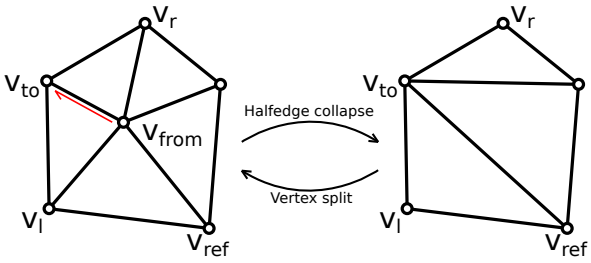

Figure 2: A patch modified by halfedge collapse and vertex split operations. The collapsed halfedge is in red.

4. Decimation

The aim of the decimation step is to generate discrete levels of detail. The decimation operator used is the halfedge collapse. It consists of merging a vertex with one of its neighbours and removing the degenerated faces. A patch is the set of faces modified by an halfedge collapse. We call v_{from} the vertex that is going to merge with the vertex v_{to}, which does not move. After the collapse, v_{ref} is the vertex of the patch that makes with v_{to} the longest edge generated by the collapse. v_l and v_r are the two vertices that are connected to v_{from} and v_{to} before the collapse. v_l is on the left of the collapsed halfedge and v_r on the right. All these vertices are represented on Figure 2.

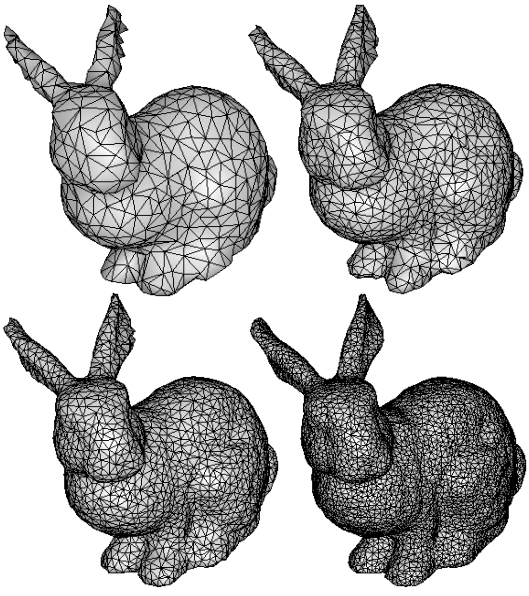

Figure 4: Examples of levels of detail generated by our decimation algorithm.

To generate a new level of detail l, all the halfedges that can be collapsed without violating the manifold

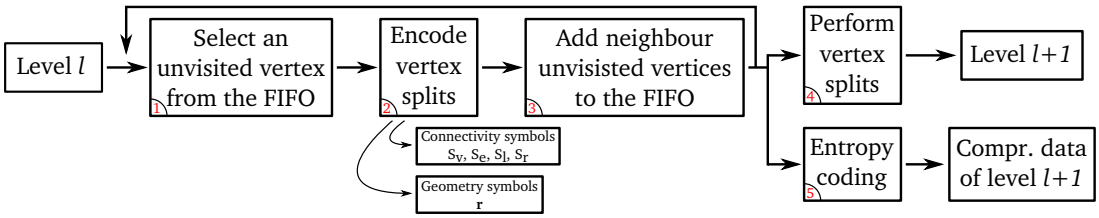

Figure 3: Encoding process of a level of detail.

property of the mesh are ranked according to an edge length metric. The value of this metric corresponds to the length of the edge v_{to} v_{ref}. The halfedges with the smallest metric values are collapsed first. An halfedge cannot be collapsed if its collapse would generate face normal flips. We ensure that the normals are not altered beyond a certain threshold. We verify for each face of the patch that the cosine of the angle between the normal before and after the collapse does not exceed a threshold (set to 0.7 in our experiments). After a collapse, no more halfedges that would modify the patch can be collapsed for generation of the current level of detail. So these halfedges are marked with a flag.

The generation of the new level of detail l is finished when there are no more halfedge candidates to be collapsed. The generation of the level $l-1$ then begins after having reset all the halfedge flags. In this way, successive levels of detail are generated until the targeted number of vertices for the base mesh ($l = 0$) is reached.

Other existing decimation metrics, such as the Hausdorff distance in [10], the volume metric from [11] or the well-known Quadric Error Metric [33], could be used. Nevertheless, our simple edge length metric goes together with the predictions involved in the compression scheme (see Section 5). Therefore, the generated levels of detail can be efficiently compressed. Our decimation scheme produces levels of detail with uniform triangle sizes as illustrated by the Figure 4.

All the performed halfedge collapses are kept in memory because the reverse vertex split operations will have to be performed during the mesh compression. Vertex positions are quantized before the reconstruction and compression steps.

5. Compression and reconstruction

Starting from the base mesh ($l = 0$), the successive levels of detail l of the mesh are reconstructed and encoded. The vertex splits, the reverse operations of the halfedge collapses performed during the decimation, are applied. A vertex must be split if it was the v_{to} vertex of an halfedge collapse operation performed during the decimation of the current level of detail.

The reconstructed levels of detail are split into two layers. The coarse levels are not clustered. During the reconstruction, the mesh is uniformly refined as it would be with a progressive mesh compression algorithm. However, the finest levels of detail are clustered. A cluster can be refined more or less than its neighbours. This enables the random access during the decompression.

5.1. Global levels of detail encoding and reconstruction

The encoding process of a level of detail is illustrated on Figure 3. For each level of detail, vertex splits are first encoded before being performed. Two types of information need to be compressed to perform the same operation during the decompression: the connectivity and the geometry.

The *connectivity* information of a vertex split operation on a triangle mesh can be defined by three vertices: v_{to}, v_l and v_r. We made the choice to encode a vertex split with one additional vertex: v_{ref}. The reason is that v_{ref} can be easily predicted with the length of the edge v_{to} v_{ref}. When v_{ref} is known, v_l and v_r can be efficiently encoded with offsets.

A deterministic traversal of the mesh vertices is performed. It consists of a breadth-first traversal also carried out by the decoder in the same order. In the base mesh, the first halfedge of the first face is pushed into a FIFO queue. This halfedge is selected since it can be retrieved by the decoder with the base mesh data. It will always be the first processed halfedge for all the global levels of detail. To process a new vertex, an halfedge is popped out the FIFO queue (Figure 3, step 1). The data of the vertex it points to is encoded if this vertex has not already been visited (Figure 3, step 2). Then, all the outgoing halfedges of the current vertex are added to the FIFO queue in clockwise order (Figure 3, step 3). The traversal is continued by popping out another halfedge from the FIFO queue.

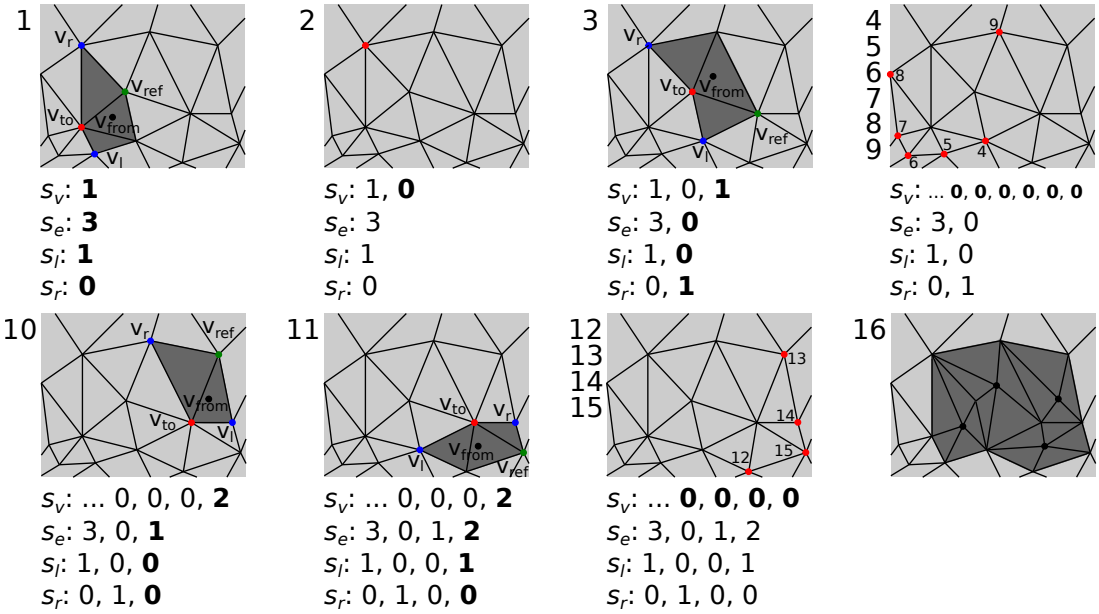

Figure 5: Encoding traversal. The vertices are iteratively processed by a deterministic traversal. The current vertex is in red. 1-15: Vertex splits are encoded with the s_v, s_e, s_l and s_r symbols. The symbols related to the current vertex are in bold. 16: Vertices are finally split once the encoding traversal is finished.

For each vertex, we encode its number of splits, the s_v symbols. Thus, the v_{to} vertices will be known by the decoder. If the current vertex has a number of splits different from zero, then the corresponding split(s) must be encoded. The surrounding edges of v_{to} are ranked from the longest to the shortest. The second vertex of these edges are the possible v_{ref} vertices. This ranking forms a stack, the longest edge being on the top of the stack. The algorithm counts the number of edges it has to unstack before getting the correct v_{ref}. This count is the s_e symbol and thus encodes v_{ref}.

To encode the v_l vertex, the algorithm simply counts the number of vertices belonging to the patch between v_{ref} and v_l and so obtains the s_l symbols. The v_r vertex is encoded in the same way with the s_r symbol.

The *geometry* information of a vertex split is the position of v_{from}. It is encoded with the residual vector $\mathbf{r}$ between the barycentre of the patch b and the position of v_{from}. This residual is projected in a local Frenet frame. This frame is constructed with the direction of the edge v_{to} v_{ref} and the average normal of the two faces adjacent to the same edge as illustrated in Figure 6. To avoid a post-quantization step and slightly reduce the entropy, we use the bijection proposed in [34] to project $\mathbf{r}$ from the global $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ frame to the local Frenet frame $(\mathbf{t_1}, \mathbf{t_2}, \mathbf{n})$.

Once all the split operations of the current level of detail have been encoded, these operations are performed (Figure 3, step 4). So, the level of detail l is obtained. The encoding and then the reconstruction of the next level $l+1$ is performed in the same manner. An encoding traversal is illustrated on Figure 5.

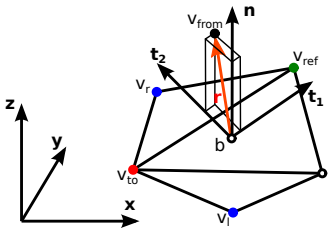

Figure 6: Local Frenet frame $(\mathbf{t_1}, \mathbf{t_2}, \mathbf{n})$ used to encode the geometry residual $\mathbf{r}$.

5.2. *Clustered levels of detail encoding and reconstruction*

The base clustered mesh is obtained once the mesh contains the number of desired clusters for the random accessibility. This corresponds to the level of detail l_c. In the base clustered mesh, each vertex is the parent of one cluster. In the next level of detail, if a vertex v_c of the base clustered mesh is split, then the inserted vertices will belong to the cluster of v_c. In the following levels, these vertices may also be split, thus adding

new vertices to the cluster of v_c. In the end, the cluster corresponding to v_c will be the set of vertices of the input mesh that are the descendant of v_c by vertex splits. The adjacency relations between the base clustered mesh vertices are the same as the adjacency relations between their respective clusters.

To enable the random access during the decompression, the clusters are compressed independently. Separate deterministic traversals that encode the splits are performed for each cluster instead of being performed once for the whole mesh. In this way, only the vertex splits related to the current cluster are encoded in its data block.

For each cluster, the first halfedge to process needs to be selected with a method also available to the decoder. The same breadth-first traversal as during the global levels of detail encoding is performed on the base clustered mesh. When the current halfedge points to a unvisited vertex, it is set as the first halfedge for the encoding of the vertex cluster.

The encoding traversal of each cluster follows the same principle as the encoding traversal of the global levels of detail. The encoded data is also the same. However, if a current halfedge points to a vertex belonging to another cluster, it is simply omitted.

When the encoding traversal has been performed for all the clusters, the vertices are split to reconstruct the new level. The inserted vertices inherit their cluster id from their parents. The encoding of the next clustered level of detail then starts.

To split a vertex v_{to} and obtain v_{from}, the patch must be in the same configuration as during the simplification. The vertices v_l, v_r and v_{ref} among others must be present. v_{to}, v_l, v_r and v_{ref} belong to the level $l-1$, while v_{from} belongs to the level l. If some patch vertices belong to neighbouring clusters, the decoder will simply need to reconstruct these neighbouring clusters at the level of detail $l-1$. The data dependency does not include the same level of detail on the whole mesh as with standard progressive mesh compression algorithms. To decompress a cluster C_k at a set level l, C_k and all its adjacent clusters must be reconstructed at the level $l-1$. The V-partition multiresolution model described in [35] also uses this principle to generate smooth transitions between mesh partitions belonging to several levels of detail.

5.3. Entropy coding and compressed file

The connectivity symbols s_v, s_e, s_r and s_l are compressed by a range coder [36] with a quasi-static probability model for each symbol (Figure 3, step 5). The

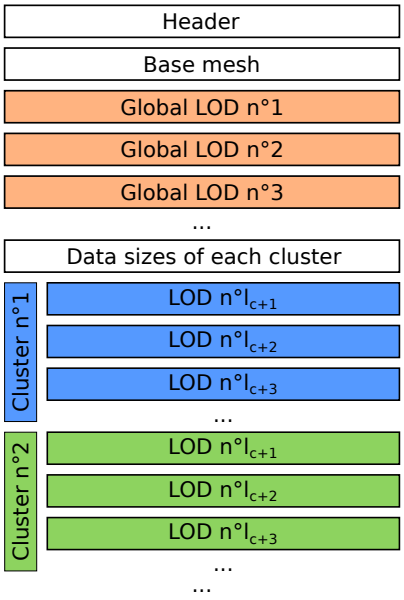

Figure 7: Structure of the compressed file.

tangential components of $\mathbf{r}$ and its normal components are also encoded by a range coder. One model is used for the tangential components and a different one for the normal.

The compressed file starts with header information such as the mesh bounding box and the number of quantization bits. The base mesh is then stored and not compressed. The compressed data of all the global levels of detail is saved just after, as it allows the restoration of the base clustered mesh. To perform random accessible decompression, the decoder needs to know the position in the file of each data block of compressed cluster data. Hence the size of each cluster compressed data has to be stored to generate a location index. The main bulk of the file is stored at end, containing the compressed data of all the clusters. The Figure 7 illustrates the compressed file structure. In a transmission context, each block of the file can be streamed on-demand. Thus, only the data required to decompress the requested parts of the mesh are transmitted to the client.

6. Decompression

The decompression starts by reconstructing the base mesh. The global levels of detail are then decompressed until the base clustered mesh is obtained. A map of the required level of detail M for each cluster C_k must then be generated. The user selects the set R_c of clusters he is interested in by picking their corresponding vertex in

the base clustered mesh. He then chooses at which clustered level of detail l_k he wants to decompress the selected clusters:

$$\forall C_k \in R_c, M(C_k) = l_k.$$

This map must be completed with the following constraint: there must be at maximum one level of detail of difference between a cluster and its neighbours. This completion can be expressed with the formula:

$$M(C_k) = \max(\left\{ \max_{\forall C_l \in R_c} (M(C_l) - d_t(C_k, C_l)), 0 \right\})$$

where $d_t(C_k, C_l)$ is the minimum topological distance in number of edges between the parent vertex of the cluster C_k and the parent vertex of the cluster C_l in the base clustered mesh. In practice, this map is generated with an algorithm that, starting from the parent vertex of each cluster of R_c, traverses all the base clustered mesh vertices and assigns them a required level. Figure 8 shows an example of a required levels of detail map and the obtained decompressed mesh.

Once the map M is complete, the decompression of the clustered levels of detail begins. The random access to each cluster data block in the compressed file is permitted thanks to the location table generated with the stored size of each cluster data block. When the required clustered level of detail of one cluster is null, then its data block is completely omitted.

For each clustered level of detail, only the required levels for all the refined clusters are decompressed. The levels are then reconstructed as during the compression. The difference is that only the decompressed parts of the mesh are refined.

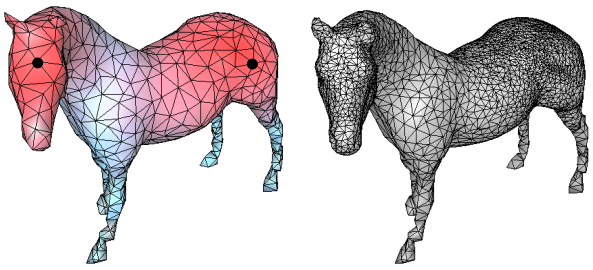

Figure 8: Required levels of detail map M displayed on the base clustered mesh and the obtained decompressed mesh. The two black vertices have been chosen to decompress their clusters at the finest levels of detail. The red areas of the base clustered mesh are decompressed at a high level of detail while the blue areas are decompressed at a low level of detail.

The main point of the POMAR decompression is that it is possible to have up to one level of detail of difference between one cluster and its neighbours. So, the decompression can generate a smooth transition between the clusters that the user wants to see at the highest level of detail and the clusters not of interest. The shape of the triangles in the partially decompressed meshes is good as shown in Figure 9 because they have been generated by the decimation step guided by the metric.

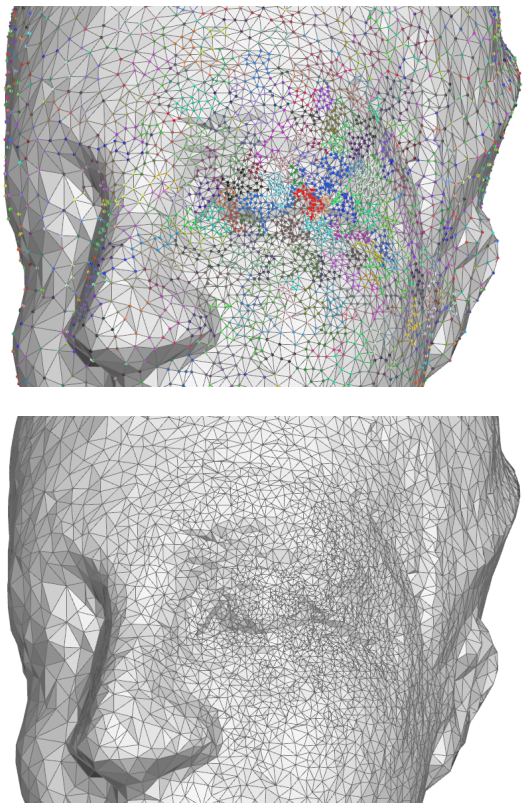

Figure 9: Decompression example of the Ramesses model. On the top image, the vertices belonging to the same cluster have the same colour. The red cluster on the eye was selected to be decompressed at the highest level of detail.

7. Choice of the number of clusters

The choice of the number of clusters is important as it directly influences the random accessibility, the locality of the refinements and the compression performance of the algorithm. The number of clusters is set by the level of detail l_c of the base clustered mesh. The base clustered mesh is the coarsest level of detail the user has access before selecting the clusters to refine more. We will now study the impact of the value of l_c on the random accessibility with two examples. Whatever the value of l_c is, the total number of levels of detail is always constant.

A low value of l_c means a low number of clusters. There will be a high number of clustered levels of detail, but during the decompression there can be only one level of detail of difference between two neighbour clusters. Therefore, if a cluster of the mesh is selected to be decompressed at the finest level of detail, a significant part of the mesh will have to be decompressed to respect this constraint.

If a high value of l_c is chosen, the base clustered mesh will be very refined and there will be a low number of clustered levels of detail. The full decompression of one cluster will only impact a small region of the mesh and there will be a weak gradation between the coarse and refined clusters.

The consequence of this behaviour is that, in order to guarantee a correct random accessibility and good compression rates, we advise to choose l_c so that the number of clusters is between 1% and 5% of the input mesh number of vertices. Figure 10 illustrates the influence of the number of clusters on the random accessibility and the locality of the refinements. We also study the impact of the number of clusters on the compression rates in Section 8.

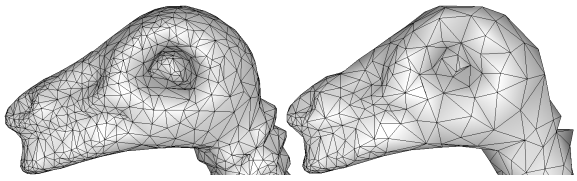

Figure 10: Decompression examples of the dinosaur model with a high (left) and low (right) number of clusters. For both cases, a cluster on the end of the nose was selected to be decompressed at the highest level of detail.

8. Experimental results

We implemented our algorithm using the halfedge data structure from OpenMesh [37] and the range coder from Michael Schindler [36]. We provide in Table 1 experimental results obtained with our first implementation of the POMAR codec, the progressive and random accessible algorithm of the CHuMI viewer [12] and the random accessible hierarchical approach [7]. Experiments ran on a desktop computer with an Intel Core i7 CPU at 2.80Ghz with 8GB of RAM. Comparing the compression performance of the POMAR codec with the performance of the CHuMI viewer is difficult, as both algorithms do not provide the same type of random accessibility. Nevertheless, each cluster generated by our compression algorithm can be decompressed independently. This is also the case of each nSP-cell of the CHuMI viewer. Therefore we tried, for each tested mesh, to have approximatively the same number of clusters as the number of nSP-cells. Some models could not be compressed with the hierarchical approach as it does not handle meshes with a genus superior to zero or the compression took too much time to complete.

We implemented a selective decompression application to demonstrate the features of our scheme. The user selects, by drawing a rectangle, the clusters he wants to decompress at the highest level of detail on the base clustered mesh by their parent vertices. The map M of the required levels of detail for each cluster is computed as described in Section 6. This experiment is captured in the additional video.

We studied for one mesh the influence of the number of clusters on the compression rate. Figure 11 shows the obtained curve. The compression performance decreases when the number of clusters increases. This relation is asymptotically linear with a significant slope. We indeed use a different quasi-static probability model to independently build the probability tables of the entropy coder for each level of detail and each cluster. The more clusters there are, the less the models can well adapt to the data. As a consequence, the choice of the number of clusters must be taken with care as it also influences the random-accessibility (see Section 7).

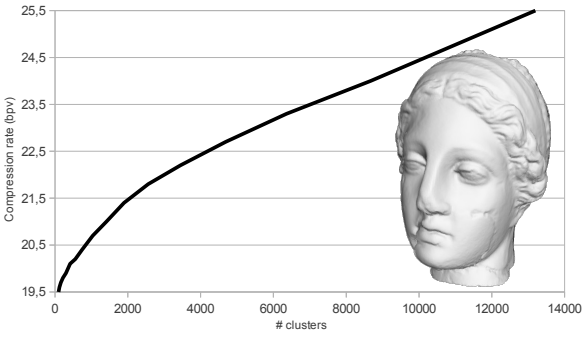

Figure 11: Compression rate in function of the number of clusters for the Igea model (134345 vertices). Geometry in quantized to 12 bits.

The POMAR scheme can also be used in pure progressive compression mode. In this case, only global levels of detail are encoded as described in Section 5.1. We measured the rate-distortion performance of our algorithm and previous approaches in progressive mode. The results are shown on Figure 12.

Model	# vertices	POMAR					CHuMI [19]			Hierarchical [7]	
		# clust.	c.	g.	tot.	time	# cells	tot.	time	tot.	time
Igea	134,345	5192	8.4	15.0	23.4	7s	5249	27.4	3s	27.19	21s
Armadillo	172,974	9268	9.1	15.0	24.1	9s	9217	26.0	3s	24.7	46s
Fertility	241,607	2551	8.1	11.8	19.8	14s	2561	22.7	5s	-	-
Raptor	1,000,080	4869	7.6	7.0	14.6	47s	4801	16.4	10s	-	-
Ramesses	826,266	4480	7.6	8.4	16.0	50s	4481	17.4	11s	22.4	2m 44s
Neptune	2,003,932	10498	7.7	5.7	13.5	2m 7s	10497	14.7	24s	-	-
Dragon	3,609,600	14383	7.4	5.7	13.0	3m 40s	14337	15.0	27s	20.0	53m 13s
Statuette	4,999,996	19094	7.9	5.9	13.8	4m 51s	19073	15.0	35s	-	-

Table 1: Experimental compression results of the POMAR codec, the CHuMI viewer and the hierarchical approach. Geometry is quantized to 12 bits. The compression rates are in bits per vertex. c. stands for connectivity and g. stands for geometry.

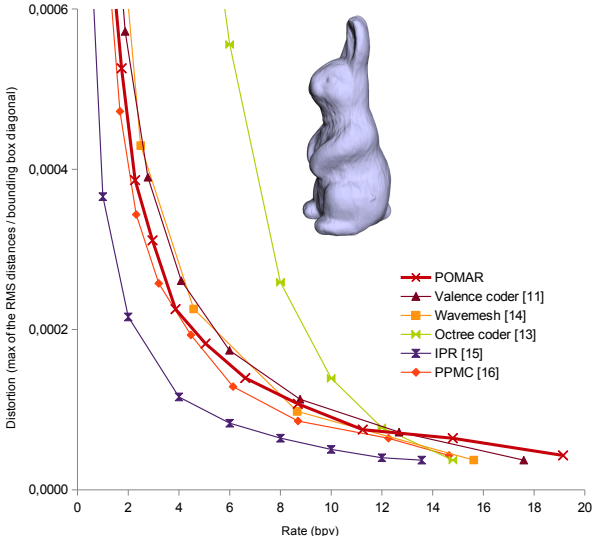

Figure 12: Rate-distortion curves for the progressive compression of the rabbit model (67039 vertices) with a 12 bit quantization.

9. Comparisons and discussion

9.1. Progressive and random accessible compression

Compression rates. In the conditions described in Section 8, the results of Table 1 show that, for the tested models, POMAR always provides better compression rates than the CHuMI viewer. The duplication of the vertices belonging to several nSP-cells by the CHuMI encoder may explain why our algorithm provides better compression rates. POMAR does not duplicate any geometry or connectivity information. We expect that the scheme of Du et al. [20] would provide slightly better results than the CHuMI viewer with the same random access granularity. Both algorithms are a random accessible extension of the original Gandoin and Devilliers algorithm [12], but the algorithm of Du et al. does not duplicate border vertices. The compression rates provided by POMAR are about 10 bits per vertex lower than the rates given by the approach of Kim et al [17] and the cluster-based approach of Maglo et al. [21] does not support the high granularity random access of our experiments.

Random-accessibility granularity. With POMAR, to decompress a cluster at a given level of detail l, the global levels of detail must be fully decompressed and the neighbour clusters must be decompressed at the level $l-1$. This constraint spreading on the mesh affects the locality of the refinements and the random-access. To decompress a cell with the approach of Du et al. [20], the top tree and all the border vertices of previous cells in the dependency list must be decoded. Consequently, if the requested cell is at the end of the dependency list, a significant number of border vertices must be decompressed. The algorithm of Kim et al. [17] allows to split a vertex with no neighbourhood restriction. However, the hierarchy allowing selective refinements is divided into data blocks, which are the atomic unit for the random accessibility. Consequently, some vertices must be decompressed even if they are not inserted to the decompressed model. The CHuMI viewer [19] can decompress a nSP-cell independently of its neighbours if its parent cells have been decoded. The charts of cluster-based approach [21] can also be decompressed independently after their border vertices have been decoded.

Mesh quality. POMAR has the advantage of reconstructing without any post-processing one piece decompressed models with good triangle shapes. All the decompressed triangles come directly from the decimation and therefore respect its metric. The CHuMI viewer [19] and the cluster-based approach [21], however, require a post-processing algorithm to stitch adjacent nSP-cells and clusters together. The chart stitching algorithm of the cluster-based approach [21] sometimes

generates artefacts due to normal flips. Regarding the CHuMI viewer, the algorithm that builds the transitions between adjacent nSP-cells with different levels of precision generates anisotropic triangles, as shown on the Figure 5 in [19]. The approach of Du et al. [20] does not need any post-processing step to stitch adjacent cells but anisotropic transition triangles between the border vertices and the inner vertices of the cell are generated by the decompression.

Progressive compression algorithms based on space subdivision are known to generate decompressed models that have a high distortion at low rates because of the low quantization. This fact is illustrated by the rate-distortion curve of the octree coder on Figure 12. To remove this well-known blocky effect, Jamin et al. [19] proposed for the CHuMI viewer to encode some connectivity information before the geometry, thus increasing the complexity of the compression algorithm. Du et al. [20] did not address this issue.

Compression times. The POMAR decimation algorithm is based on a ranking of the halfedge candidates to be collapsed. The mesh simplification needs more time than the CHuMI encoder that uses a spatial kd-tree decomposition to simplify the mesh. As expected, the approach of Du et al. [20] is reported as having similar compression times than the CHuMI encoder.

Decompression times. In our experiments, the full decompression of one cluster and its dependencies of the Ramesses model, as shown on Figure 9, takes with the global levels of detail decompression approximately 280 ms. The selective decompression experiments of the additional video show that the decompression is fast enough to allow interactive decompression like the CHuMI viewer [19] and the cluster-based approach [21]. The schemes of Du et al. [20] and Kim et al. [17], however, did not demonstrate their ability to perform interactive decompression.

9.2. Progressive compression

As shown on Figure 12, compared to previous progressive compression approaches, POMAR achieves competitive rate-distortion performance at low rate. However, the final compression rate, which is illustrated by the right most point of the curve, is worse than the rates obtained by progressive coders. The reason is that our connectivity compression scheme, needed for the progressive random access to clustered levels, is costly. It uses 4 different symbols (s_v, s_e, s_l and s_r) while approaches like the progressive valence encoder [11] use only one symbol. Nevertheless, we would like to point out that another quality of the POMAR encoder is its simplicity of implementation.

9.3. Data structures for interactive visualization

As seen in Section 2.4, progressive and random-accessible mesh data structures have been proposed in the literature for the interactive visualization of large meshes. Most of them were designed to be compact since they must allow the storage of huge meshes either in main memory, in GPU memory or on disk. Their key concept is to enable fast mesh adaptation mechanisms by a quick access to a multiresolution data structure. Consequently, most of these data structures are not compressed. In the recent work of Derzapf and Guthe [32], which integrates a compressed data structure, the connectivity of the mesh is stored at about 4 bytes per vertex, while our scheme requires about 1 byte. Even if the scheme described in [32] stores the normal of the mesh vertices while POMAR does not, it saves a mesh with around 11 to 14 bytes per vertex while POMAR requires around 3 bytes per vertex.

The current implementation of POMAR supports selective refinement of the mesh. However, when the selection slightly changes, the decoder cannot yet perform delta coarsening and refinement as data structure for interactive visualization can. The mesh access is slower than with the previously described GPU data structures. It also allows less flexible mesh adaptation. Yet, we think that the much higher compression performance of our approach with its relative progressive and random access to the data is particularly useful for efficient mesh storage and transmission.

10. Conclusion and future work

We presented in this paper POMAR, our new progressive and random accessible manifold triangle mesh compression algorithm. Unlike previous approaches, it is not based on a space-subdivision data structure or a initial segmentation of the input model. Experimental results show that our algorithm compares favourably with previous progressive and random accessible approaches in terms of compression rates with similar random access. The POMAR decompression algorithm allows the generation of a smooth transition between the fine and coarse decompressed regions of the mesh.

An adaptation of POMAR encoder would enable out-of-core compression without impacting the final compression rate. The mesh decimation would be performed out-of-core such as in [24]. The reconstruction and the compression of global levels of detail would be performed in-core as described in Section 5.1. For the clustered levels of detail, each cluster would be reconstructed and compressed separately. As there must be at

most one level of detail of difference between adjacent clusters, the compression of one cluster would involve the partial reconstruction of its neighbours.

Future work also include the generalisation of the algorithm to non-manifold meshes by replacing the vertex split operator by generalized vertex splits. A decimation algorithm that better preserves the mesh shape and its adapted compression algorithm could also be investigated. Finally, we also wish to work on a mixed CPU/GPU implementation of our framework to enable automatic refinement and coarsening of the model for interactive visualization while maintaining good a compression performance.

Ackowledgements

This work has been founded by French National Research Agency (ANR) through COSINUS program (project COLLAVIZ no. ANR-08-COSI-003) and the LRC with the CEA DAM DIF. The models are courtesy of the AIM@SHAPE Repository and the Stanford Computer Graphics Laboratory.

References

[1] P. Alliez, C. Gotsman, Recent advances in compression of 3d meshes, in: Advances in Multiresolution for Geometric Modelling, Mathematics and Visualization, 2005, pp. 3–26.

[2] M. Isenburg, P. Lindstrom, Streaming meshes, in: Proc. of Visualization, 2005, pp. 231–238.

[3] M. Isenburg, P. Lindstrom, J. Snoeyink, Streaming compression of triangle meshes, in: Proc. of the Eurographics symposium on Geometry processing, 2005.

[4] S. Choe, J. Kim, H. Lee, S. Lee, Random accessible mesh compression using mesh chartification, IEEE Transactions on Visualization and Computer Graphics 15 (1) (2009) 160 –173.

[5] H. Lee, P. Alliez, M. Desbrun, Angle-analyzer: A triangle-quad mesh codec, Computer Graphics Forum 21 (3) (2002) 383–392.

[6] S.-e. Yoon, P. Lindstrom, Random-accessible compressed triangle meshes, IEEE Transactions on Visualization and Computer Graphics 13.

[7] C. Courbet, C. Hudelot, Random accessible hierarchical mesh compression for interactive visualization, in: Symposium on Geometry Processing, 2009.

[8] H. Hoppe, Progressive meshes, in: Proc. of SIGGRAPH, 1996, pp. 99–108.

[9] G. Taubin, A. Guéziec, W. Horn, F. Lazarus, Progressive forest split compression, in: Proc. of SIGGRAPH, 1998, pp. 123–132.

[10] R. Pajarola, J. Rossignac, Compressed progressive meshes, IEEE Transactions on Visualization and Computer Graphics 6 (2000) 79–93.

[11] P. Alliez, M. Desbrun, Progressive compression for lossless transmission of triangle meshes, in: Proc. of SIGGRAPH, 2001, pp. 195–202.

[12] P.-M. Gandoin, O. Devillers, Progressive lossless compression of arbitrary simplicial complexes, in: Proc. of SIGGRAPH, 2002, pp. 372–379.

[13] J. Peng, C.-C. J. Kuo, Geometry-guided progressive lossless 3d mesh coding with octree (ot) decomposition, in: Proc. of SIGGRAPH, 2005, pp. 609–616.

[14] S. Valette, R. Prost, Wavelet-based progressive compression scheme for triangle meshes: Wavemesh, IEEE Transactions on Visualization and Computer Graphics 10 (2004) 123–129.

[15] S. Valette, R. Chaine, R. Prost, Progressive lossless mesh compression via incremental parametric refinement, in: Proc. of the Symposium on Geometry Processing, 2009, pp. 1301–1310.

[16] A. Maglo, C. Courbet, P. Alliez, C. Hudelot, Progressive compression of manifold polygon meshes, Computers & Graphics 36 (5) (2012) 349–359.

[17] J. Kim, S. Choe, S. Lee, Multiresolution random accessible mesh compression, Computer Graphics Forum 25 (3) (2006) 323–331.

[18] J. Kim, S. Lee, Truly selective refinement of progressive meshes, in: Proc. of Graphics interface, 2001, pp. 101–110.

[19] C. Jamin, P.-M. Gandoin, S. Akkouche, Chumi viewer: Compressive huge mesh interactive viewer, Computers & Graphics 33 (4) (2009) 542 – 553.

[20] Z. Du, P. Jaromersky, Y.-J. Chiang, N. Memon, Out-of-core progressive lossless compression and selective decompression of large triangle meshes, in: Data Compression Conference, 2009, pp. 420 –429.

[21] A. Maglo, I. Grimstead, C. Hudelot, Cluster-based random accessible and progressive lossless compression of colored triangular meshes for interactive visualization., in: Proc. of Computer Graphics International, 2011.

[22] H. Lee, G. Lavoué, F. Dupont, Rate-distortion optimization for progressive compression of 3d mesh with color attributes, The Visual Computer 28 (2012) 137–153.

[23] H. Hoppe, View-dependent refinement of progressive meshes, in: Proc. of SIGGRAPH, 1997, pp. 189–198.

[24] H. Hoppe, Smooth view-dependent level-of-detail control and its application to terrain rendering, in: Proc. of Visualization, 1998, pp. 35–42.

[25] R. Pajarola, Fastmesh: efficient view-dependent meshing, in: Proc. of Pacific Conference on Computer Graphics and Applications, 2001, pp. 22–30.

[26] R. Pajarola, C. DeCoro, Efficient implementation of real-time view-dependent multiresolution meshing, Visualization and Computer Graphics, IEEE Transactions on 10 (3) (2004) 353–368.

[27] S.-E. Yoon, B. Salomon, R. Gayle, D. Manocha, Quick-vdr: interactive view-dependent rendering of massive models, in: IEEE Visualization, 2004, pp. 131 – 138.

[28] M. Guthe, P. Borodin, R. Klein, Efficient view-dependent out-of-core visualization, in: The 4th International Conference on Virtual Reality and its Application in Industry, 2003, pp. 428–438.

[29] E. Shaffer, M. Garland, A multiresolution representation for massive meshes, IEEE Transactions on Visualization and Computer Graphics 11 (2) (2005) 139–148.

[30] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, R. Scopigno, Adaptive tetrapuzzles: efficient out-of-core construction and visualization of gigantic multiresolution polygonal models, in: Proc. of SIGGRAPH, 2004, pp. 796–803.

[31] L. Hu, P. V. Sander, H. Hoppe, Parallel view-dependent refinement of progressive meshes, in: Proc. of the symposium on Interactive 3D graphics and games, 2009, pp. 169–176.

[32] E. Derzapf, M. Guthe, Dependency-free parallel progressive meshes, Computer Graphics Forum (2012) 2288–2302.

[33] M. Garland, P. S. Heckbert, Surface simplification using quadric error metrics, in: Proc. of SIGGRAPH, 1997, pp. 209–216.

[34] H. Lee, G. Lavoué, F. Dupont, Adaptive coarse-to-fine quanti-

zation for optimizing rate-distortion of progressive mesh compression, in: Proc. of the Vision, Modeling, and Visualization Workshop, 2009.

[35] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, R. Scopigno, Batched multi triangulation, in: Proc. of Visualization, 2005, pp. 207–214.

[36] M. Schindler, A fast renormalisation for arithmetic coding, in: Proc. of the Data Compression Conference, 1998, 1998, p. 572, http://www.compressconsult.com/rangecoder/.

[37] M. Botsch, S. Steinberg, S. Bischoff, L. Kobbelt, Open-mesh a generic and efficient polygon mesh data structure, in: OpenSG Symposium, 2002, http://www.openmesh.org/.