Progressive compression of arbitrary textured meshes

F. Caillaud, V. Vidal, F. Dupont and G. Lavoué

Université de Lyon, CNRS, INSA-Lyon, LIRIS UMR 5205, Université Lyon 1

Abstract

In this paper, we present a progressive compression algorithm for textured surface meshes, which is able to handle polygonal non-manifold meshes as well as discontinuities in the texture mapping. Our method applies iterative batched simplifications, which create high quality levels of detail by preserving both the geometry and the texture mapping. The main features of our algorithm are (1) generic edge collapse and vertex split operators suited for polygonal non-manifold meshes with arbitrary texture seam configurations, and (2) novel geometry-driven prediction schemes and entropy reduction techniques for efficient encoding of connectivity and texture mapping. To our knowledge, our method is the first progressive algorithm to handle polygonal non-manifold models. For geometry and connectivity encoding of triangular manifolds and non-manifolds, our method is competitive with state-of-the-art and even better at low/medium bitrates. Moreover, our method allows progressive encoding of texture coordinates with texture seams; it outperforms state-of-the-art approaches for texture coordinate encoding. We also present a bit-allocation framework which multiplexes mesh and texture refinement data using a perceptually-based image metric, in order to optimize the quality of levels of detail.

Categories and Subject Descriptors (according to ACM CCS):
- Mathematics of computing ~ Coding theory
- Computing methodologies ~ Image compression
- Computing methodologies ~ Texturing

1. Introduction

The development of computer graphics applications leads to a global increase in 3D model quality. This increase in quality is generally obtained by an augmentation of the geometric information describing these models (mostly represented by surface meshes), as well as the addition of appearance attributes for improving realism, such as texture maps. Utilization of these high quality meshes over an increasing diversity of devices and applications is quickly constrained by bandwidth, memory and/or processing speed. For instance, a web 3D application requires low-latency visualization and thus is strongly limited by the size of the data to be transmitted over the network. This kind of application requires a considerable decrease in mesh information size. Besides this need for efficient compression, levels of detail (i.e. a multiresolution representation)
are also necessary to adapt the data to the visualization device. These functionalities are offered by progressive compression techniques, which allow a high compression ratio to be attained and different levels of detail (LoD) to be produced. With these techniques, users instantly get a coarse version of the mesh which is then progressively refined as more data are decompressed until the initial model has been restored.

Many progressive compression methods for surface meshes already exist [MLDH15]. However, most of these approaches only deal with triangular manifold meshes and few can compress either polygonal manifold or triangular non-manifold meshes. To our knowledge, none of them provide an implementation of progressive compression of any surface meshes, whatever their connectivity (i.e. including polygonal non-manifold configurations). As for texture coordinates, most previous works consider them as simple per-vertex information. However, this assumption does not hold for most cases due to the presence of discontinuities in the texture mapping (i.e. texture seams).

In this context, we present a lossless progressive compression algorithm suited for arbitrary textured meshes. Our main contributions are:

- Generic edge collapse and vertex split operators, suited for polygonal non-manifold meshes with arbitrary texture seam configurations.
- Efficient geometry-driven prediction schemes and entropy reduction techniques for progressive encoding of connectivity and texture mapping.

We also introduce several minor contributions such as a fast metric for edge collapse selection which preserves both geometry and texture mapping quality, as well as efficient heuristics for adapting the number of simplifications per iteration, which are called batches. To our knowledge, our method is the first progressive algorithm to handle polygonal non-manifold models. For geometry and connectivity encoding of triangular manifold and non-manifold meshes, our method is competitive with state-of-the-art and better at low/medium bitrates. Our algorithm also allows progressive encoding of texture coordinates with texture seams and outperforms existing progressive and single-rate methods for this task. Since the texture map may be also encoded progressively (e.g. using progressive and single-rate methods for this task. Hoppe [Hop96] introduced progressive compression of 3D models. He proposes the edge collapse operator to simplify the mesh while different unitary configurations around the collapsed edge encode connectivity. He also records the position of edge vertices. These two pieces of information are given to the vertex split operator in order to refine the mesh during decompression. The main drawback of this method is its fine granularity (difference between two consecutive LoDs). This allows strong control of distortion but penalizes the compression rate. Moreover, as in most further progressive compression methods (e.g. [TGLH08, PR00, AD01]), it only deals with triangular manifold meshes.

Only a few methods are able to compress polygonal manifold meshes. Maglo et al. [MCAH12] present a generalization of the valence-based algorithm by Alliez and Desbrun [AD01] for this task. This algorithm can be described in two parts. First, a deterministic mesh conquest constructs a set of independent 1-rings where the center vertex is removed. Second, the valence of the removed vertices is encoded. These two parts form a decimation batch which is repeated until a base mesh is reached.

Only a few algorithms are available in the literature for the progressive compression of non-manifold meshes. Popović and Hoppe [PH07] adapted [Hop96] for arbitrary simplicial complexes. The progressive algorithm by Bajaj et al. [BPZ99] is also able to handle non-manifold triangulations; however, the compression efficiency of these last two methods is limited. We particularly set apart tree-based algorithms by Gandoin and Devillers [GD02] and Peng and Kuo [PK05]. They encode mesh geometry over, respectively, a kd-tree and an octree by enumerating the vertices in each cell at the current level. Connectivity is encoded using the simplification and refinement operators of [PH97]. While these approaches [GD02, PK05] handle non-manifold triangular meshes and are very efficient at lossless rates, they generate undesired quantization effects at low and medium bitrates. Moreover, they do not allow any local control of the decimation and, thus, of the distortion. Tian et al. [TJL12] bring elements to limit these drawbacks. Using a bottom-up clustering strategy, Peng et al. [PHK10] propose another progressive method which provides nice low bitrate results. However, it does not guarantee retrieval of the lossless version of the mesh. Furthermore, all these techniques [PH97, BPZ99, AD01, GD02, PK05, PHK10] are limited to triangular meshes.

In comparison, our method (1) handles both polygonal and non-manifold meshes, (2) affords a lossless retrieval of the mesh and (3) allows accurate and customizable local control of decimation, leading to improved rate-distortion performances at low and medium bitrates.

2.2. Textured mesh compression

Several approaches have been proposed for textured mesh compression. Isenburg and Snoeyink [IS03] and, more recently, Váša and
Brunnett [VB14] propose single rate compression methods specifically dedicated to texture coordinates. In [VB14] these coordinates are encoded thanks to a parallelogram prediction described in [VB13], improved by using the mesh geometry information.

In the field of progressive compression, Yang et al. [YLK04] propose an algorithm for multiplexing mesh LoDs obtained using Hoppe’s algorithm [Hop96] and texture LoDs obtained through JPEG2000 compression. Multiplexing is defined in order to obtain the best visual quality (as predicted using an image quality metric applied on rendered views) at any moment during decompression. Tian and Al-Regib [TA08] keep this idea but guide multiplexing using a combination of geometry and texture RMS errors. These works focus on multiplexing the LoDs but they discuss neither the encoding of texture coordinates nor the handling of texture seams. Popović and Hoppe [Hop96, PH97] handle discontinuities in attributes (like for texture seams) in their progressive mesh algorithms. However, as stated above, these algorithms do not provide efficient compression. Lavoué et al. [LCCD16] recently adapted [AD01] to deal with texture seams, however they have to modify the connectivity to create a per-vertex UV representation. Tarini [Tar16] recently proposed a new volume based representation of UV maps. This representation allows to preserve the texture parametrization for different mesh LoDs and could lead to nice compression results. Our goal is, however, to compress the standard texture mapping representation.

In comparison to the algorithms presented above, our scheme handles texture seams and efficiently encodes texture coordinates in a progressive way, using a novel geometry-based prediction involving barycentric coordinates. We propose, moreover, a multiplexing of mesh and texture LoDs based on a perceptually-validated image metric.

3. Overview

The overall procedural flow of our progressive compression method is described below.

- The first step is the uniform quantization of XYZ and UV coordinates, respectively to $Q_x$ and $Q_t$ bits.
- Then, the mesh is iteratively simplified, in batches (see Section 4.3), using our generic simplification operator (see Section 4.1). Decimation is driven by a fast metric (see Section 4.2). This simplification is repeated until each connected component has been reduced to one single vertex.
- At each simplification step, the corresponding batch of vertex-split operations (which will allow to refine the mesh at decompression) is encoded efficiently using dedicated schemes for geometry (see Section 5.2), connectivity (see Section 5.3) and texture coordinates (see Section 5.4).

After these steps, the whole mesh is encoded in the form of chunks of binary compressed data. The first chunk contains only few vertices encoded in a simple binary form; then each chunk contains the encoded refinement information (geometry+connectivity+UV coordinates) that allows the next level of detail to be built. As an application for optimized progressive transmission, we propose a bit allocation framework which multiplexes the encoded refinement information of mesh and texture (obtained using progressive JPEG), to optimize the visual quality of the levels of detail at decompression (see Section 6).

4. Textured mesh simplification and refinement

4.1. Generic edge collapse and vertex split

In order to simplify the mesh, at compression, and to refine it, at decompression, we rely on edge collapse and vertex split operators. These operators offer a finer local control of distortion than other ones (e.g. vertex removal). The manifold edge collapse and vertex split operators introduced by Hoppe [Hop96] were extended by Popović and Hoppe [PH97] to handle non-manifold topology. They can thus handle dangling edges (edges without incident faces), complex edges (edges with more than two incident faces), complex vertices (vertices linking two connected components) and holes (missing faces).

However, the latter operators only apply to triangular non-manifold meshes and are not suitable for polygonal non-manifold meshes. Therefore, we introduce generic edge collapse and vertex split operators, able to handle polygonal faces (faces of degree ≥ 3).

The edge collapse consists of several steps, illustrated in Figure 2. First, incident triangular faces of $v_1v_2$ are suppressed (Figures 2a and 2b). Then, the edge $v_1v_2$ is suppressed (Figures 2b and 2c). Vertices $v_1$ and $v_2$ are merged into $v$. Former incident non-triangular faces of $v_1v_2$ lose one degree. Redundant edges and faces are suppressed so that the connectivity around $v$ continues to be valid (Figure 2c). Finally, $v$ is moved to the middle of the former edge $v_1v_2$ (Figure 2d). This position for $v$ is not optimal regarding geometric error but can be encoded very efficiently. The vertex split reconstructs first the connectivity, in the reverse order to the collapse, using the necessary information. Then, the vertices $v_1$ and $v_2$ are moved to their initial position.

Figure 2: Different steps of our generic edge collapse. (a) Original configuration around the red edge $v_1v_2$ to be collapsed, (b) incident triangular faces of $v_1v_2$ are removed, (c) $v_1$ and $v_2$ are merged into $v$ and non-triangular faces lose one degree, (d) $v$ is moved to the middle of the former edge $v_1v_2$.

For textured meshes, the collapse/split operators have to deal with texture seams. We recall here that UV texture coordinates cannot be considered as simple per-vertex information due to possible discontinuities in the texture mapping (referred to as texture seams) which occur when two adjacent mesh faces are associated with non-adjacent texture faces. A texture map is divided in regions by these seams. Hence UV coordinates are associated with corners which are (vertex,face) tuples. Corners are affected by collapse operations as follows: corners of incident triangular faces of $v_1v_2$ are suppressed with their face. For incident non-triangular faces of $v_1v_2$,
the corner attached to \( v_2 \) is suppressed. To update the remaining corners around \( v_1 \) and \( v_2 \), we first detect all the different regions inside the patch (i.e. the local neighborhood) of \( v_1 \) and \( v_2 \). Note that if two faces from the same region are associated with non-adjacent texture faces (e.g. imagine a texture for a cube unfolded in the typical cross-shaped 2D configuration), we tag one of these faces with an additional region index (the encoding cost is negligible) in order to correctly take into account this seam in the rest of the algorithm. We then define the concepts of matchable and unmatchable corners, illustrated in Figure 3. A corner \( c \) of \( v_1 \) (resp. \( v_2 \)) associated with the region index \( r \) is called a matchable corner if there exists another corner \( c_\text{m} \) around \( v_2 \) (resp. \( v_1 \)) associated with the same index (e.g. \( c_1 \) and \( c_2 \) in Figure 3), while \( c \) and \( c_\text{m} \) constitute a pair of matchable corners. Otherwise, the corner is called a unmatchable corner (e.g. \( c_3 \) in Figure 3). Detection of matchable and unmatchable corners takes place before any corner suppression. We use the pairs of matchable corners to interpolate the UV coordinates of the corresponding corners around \( v \) (e.g. \( c_1' \) in Figure 3), after the edge collapse. Meanwhile, without any possible interpolation, unmatchable corners remain unchanged (e.g. \( c_3' \) in Figure 3). This way, our method can handle any configuration of texture seams (examples are illustrated in Figure 3, on the right).

4.2. Edge selection

Our method simplifies the mesh by successive batches of edge collapses. For each decimation batch, \( k \) edges are collapsed. The edges to be collapsed are selected using a priority queue. Before any simplification, each edge \( e \) is associated with a weight \( w(e) \) which provides its rank in the priority queue. In our algorithm, these weights are computed in order to (1) prioritize the collapse of edges producing the lowest geometric error, (2) minimize the creation of dangling edges as this causes area loss, (3) minimize the collapse of dangling edges as this may lead to a drastic reduction of the bounding box and (4) for textured meshes, minimize moves of the texture seams as their displacement greatly affects the visual quality of the levels of detail.

Hence we have: \( w(e) = w_{\text{gc}}(e) + w_{\text{t}}(e) \), where \( w_{\text{gc}}(e) \) is the weight corresponding to geometry and connectivity errors and \( w_{\text{t}}(e) \) is the weight corresponding to texture error. If \( e \) is a dangling edge (case 3 above) then \( w_{\text{gc}} = \text{diag} + \text{length}(e) \); if the collapse of \( e \) will create a dangling edge (case 2) then \( w_{\text{gc}} = 2 \times \text{diag} + \sqrt{\text{area}(e)} \); otherwise (case 1) then \( w_{\text{gc}} = d_{\text{H}}(e) \). \( \text{diag} \) is the length of the bounding box diagonal, \( \text{area}(e) \) is the absolute value of the area difference caused by an hypothetical collapse of \( e \) and \( d_{\text{H}}(e) \) is the symmetric Hausdorff distance between the mesh before and after the hypothetical collapse. In this way we give priority to case 1, then 2, then 3. For the texture weight, we set \( w_{\text{t}}(e) = d_{\text{F}}(e) \), where \( d_{\text{F}}(e) \) is the maximal displacement of the involved unmatchable corners, in the geometric space.

This metric is used to make the simplification aware of connectivity (by preventing creation and modification of dangling edges), geometry (by minimizing local Hausdorff distance) and texture (by minimizing local texture seam displacement). Moreover, there is no need to introduce complex weighting factors between \( w_{\text{gc}} \) and \( w_{\text{t}} \) since both these measurements are consistent (they represent distances in the 3D Euclidian space). The effect of \( w_{\text{gc}} \), \( w_{\text{t}} \) and their combination on simplification quality is illustrated in Figure 4. This example clearly shows that our metric offers and excellent tradeoff between geometry and texture preservation, while also being much simpler and thus faster than concurrent metrics like that proposed by Popović and Hoppe [PH97]. More quantitative results are presented in the supplementary material.

4.3. Size of batches

The number \( k \) of edges to be collapsed in each batch is critical. Existing progressive compression algorithms based on this principle (e.g. [PR00]) usually maximize the number of edges to be collapsed. While this strategy provides the best compression rate, it may severely impact the quality of the levels of detail. On the contrary, a too small \( k \) will penalize the compression rate. There is actually a tradeoff to be found between the optimized quality of the levels of detail (by collapsing one single edge at each step like [Hop96]) and the optimized compression rate (by collapsing
as many edges as possible like [PR00]). In our algorithm, we adapt this batch size dynamically. For a given batch, we first compute the weights of all edges. We then determine a collapse threshold as the average weight of these edges. Edges with weights higher than this threshold are not collapsed in this batch. This simple yet efficient rule provides an excellent rate-distortion tradeoff. Quantitative comparisons with other strategies are provided in the supplementary material. After each collapse, the weights of all the edges involved in the operations are updated. In order to avoid conflict between two collapses during a batch, edges included in the neighborhood (set of incident faces) of an already selected edge, in the current batch, cannot be selected afterwards. These batched simplification iterations are repeated until each connected component is composed of only one vertex. This vertex is the fixed vertex of the corresponding connected component, as described in Section 5.1.

5. Encoding

We describe below the encoding of connectivity, geometry and texture mapping data. This encoding takes place at the same time as edge collapses. We carefully coordinate the encoding/decoding of these data in order to be able to propose novel and efficient prediction schemes: connectivity information is predicted using decoded geometry, while texture mapping information is predicted using both decoded geometry and connectivity. Within each batch, these data are synchronized using a spanning tree, as described below. Note that all these information are then entropy coded by means of arithmetic coding.

5.1. Spanning tree

After having collapsed all the edges of a batch, we have to record which vertices must be split during decompression. Instead of storing explicitly the index of these vertices (as in [Hop96, PH97]), we follow the strategy of Pajarola and Rossignac [PR00] who consider a spanning tree. This drastically reduces the generated information. In our case, a spanning tree is built over each connected component of the mesh, using a strict natural order over $\mathbb{R}^3$. This operation is renewed for each batch. As the structure is build after the simplification batch, the connectivity of the current LoD remains the same for the decoder. A spanning tree starts on a fixed vertex. These fixed vertices may be arbitrarily chosen on the original mesh. Each fixed vertex remains during all the simplification steps and cannot be removed (its incident edges can be collapsed anyway). During the spanning tree traversal, we specify which vertices will be split by encoding a 1 if the visited vertex results from an edge collapse, otherwise a 0 (Figure 5). As we forbid edge collapse conflict, we do not encode a 0 if, at this state, we already encoded a 1 for an adjacent vertex. This prediction allows us to save about 50% of 0 codes (0.8 bits per vertex or bvp, on average, in the compressed stream). This spanning tree also provides an ordering of the connectivity, geometry and texture mapping refinement data which are described below.

5.2. Geometry encoding

To reconstruct the position of the new edge vertices $v_1$ and $v_2$, resulting from the split of $v$, we just have to encode the displacement vector of $v$ to one of them ($v_1$), since the other one can be deduced easily by taking the opposite vector (the collapse places $v$ in the middle of $v_1v_2$). We represent this displacement vector in a local Frenet frame like in [AD01]. This representation, while not especially optimal for local non-manifold configurations, still remains efficient in these cases.

5.3. Connectivity encoding

To reconstruct, at decompression, the connectivity around the edge $v_1v_2$ resulting from the split of $v$, we use a generalization of the split codes proposed in [PH97]. These codes (described in Figure 6) are generated for each edge (row $(b)$) and each face (row $(c)$) around $v$ and describe the connectivity updates when $v$ is split into $v_1v_2$, for different configurations. For instance, case 1.0 (resp. 1.1) indicates that the edge has to be attached to $v_1$ (resp. $v_2$). Another example is case 2.3, which occurs when a face of degree $n$ becomes a face of degree $n + 1$ during the split. Most configurations are coded in the same way as [PH97] (except for polygonal case 2.3 which is new) using the same topological constraints to reduce redundancy. These constraints usually allow the code associated with a face to be deduced from its edge codes (see Figures 7a and 7b). For instance, if two edges of a face are attached to $v_1$ (they have code 0), then the face code is useless since we can deduce that the face is attached to $v_1$ as well.

On top of these topological constraints, we introduce a novel geometry-based prediction which significantly improves the encoding of cases 1.0 and 1.1 (which are the most frequent). Instead of simply encoding a 0 for case 1.0 and a 1 for case 1.1 we predict the most plausible case using the geometry of vertices $v_1$ and $v_2$. If the vertex to attach to the edge is closer to $v_1$ than to $v_2$, then it more likely shares an edge with $v_1$. The coder thus predicts the closest vertex and generates a 0 if the prediction is verified and a 1 otherwise (see Figure 7c). Since geometry decoding occurs before connectivity decoding, we can use this prediction at decompression. This prediction allows the size of these cases to be reduced, after entropy coding, by about 90% (5.6 bvp, on average, in the compressed stream).

We need to preserve the faces orientations as well at reconstruction. In most cases, the orientation of a removed face can be determined by the decoder from its adjacent faces. Otherwise, we also encode this orientation using one bit.

Figure 5: Spanning tree, in red, built starting from the fixed vertex. Ones are generated for vertices resulting from an edge collapse, zeros otherwise. Red zeros can be predicted and are not generated.
**5.4. Texture mapping encoding**

We also have to encode the information necessary to retrieve, at decompression, the UV coordinates of the corners around $v_1$ and $v_2$. We introduce novel encoding and prediction schemes for this task. As explained in Section 4.1, during simplification, UV coordinates of some corners (unmatchable) remain the same after collapse. So, we only need information to retrieve UV coordinates from matchable corners, as well as the configuration of texture seams.

First, the $n_t$ detected regions around $v_1$ and $v_2$ are ordered by their indices (from 0 to $n_t - 1$). For each face incident to $v_1 v_2$, the index of its region is encoded. If no seams are detected over the mesh, only 0 codes are generated, leading to a null entropy. We actually rarely detect more than one seam over a patch. Practically, in our experiments, we used less than 1 bit per vertex, on average, for compression of these region indices.

The corners around a vertex associated with the same region have the same UV coordinates. Therefore, we only need to encode one displacement vector per region around $v_1$ and $v_2$ to retrieve all the UV pairs. To synchronize the coder and the decoder, we set this vector as the UV displacement vector of the corner belonging to $v_1$. This 2D vector is expensive to encode, even with the quantization of its coordinates. To reduce this encoding cost, we predict this 2D texture vector using the connectivity and geometry around $v_1$ and $v_2$. Since geometry and connectivity decoding occur before texture mapping decoding, we can use this prediction at decompression.

We make a conformality assumption over the texture parametrization and exploit the shape similarity between 3D faces (mesh faces) and 2D faces (texture faces). Note that Váša and Brunnett [VB14] also proposed a geometry-based prediction which exploits conformality for their single rate encoding of texture coordinates. How-

ever, our scheme, as detailed below, is very different.

Figure 8 shows the patch around the to-be-collapsed edge $v_1 v_2$. We need to predict the 2D texture vector $\overrightarrow{v_1'}$. For that purpose, we consider an edge of the patch, $v_3 v_4$, chosen in a deterministic way, as reference. We then compute the barycentric coordinates of $v_1$, according to $v_1 v_2 v_3$ (the resulting face after collapse). By applying these barycentric coordinates to the texture face $v_1' v_2' v_3'$, we approximate the texture vertex $v_1'$ by $v_1''$. As the decoder can find the same predicted $v_1''$, we only have to encode the error vector $\overrightarrow{v} = v_1'' - v_1$. To retrieve the correct displacement vector $\overrightarrow{d} = \overrightarrow{v_1'} + \overrightarrow{v}$, in this scheme, the more the corresponding faces have a similar shape, the smaller are the error vectors, and the smaller is the entropy.

For each incident triangular face of $v_1 v_2$, which is suppressed during collapse, the opposite corner of $v_1 v_2$ also needs to be retrieved (e.g. the corner associated with $v_3$ and $v_1' v_2 v_3$). For that purpose, we just copy the UV coordinates of another corner belonging to the same vertex and with the same region index (e.g. the corner associated with $v_4$ and $v_1' v_2 v_4$). There are only very rare configurations where such a corner does not exist. This case can be detected by the decoder. In that case, we use the same prediction scheme as
presented above, to approximate the coordinates and generate an error vector.
During simplification, a triangular incident face of \( \mathbf{v}_1 \mathbf{v}_2 \) can be the last face of a region. In this case, we do not have any other possibility but to send the full UV coordinates of the three corners of the face. For the decoder to understand that this will be a new region, its region index is set to \( n_r \), the number of detected regions in the patch.

6. Multiplexing mesh and texture data
A textured mesh is associated with a texture image. In a real remote visualization scenario, this image has to be compressed as well. It would make no sense to apply a single rate image coding scheme because the texture would have to be fully transmitted before the mesh to obtain meaningful levels of detail, hence removing the benefit of progressive mesh compression in terms of latency. Moreover, the full texture resolution is not necessary for early mesh LoDs.

In this application, we use the progressive JPEG codec which offers good compression performance and provides high quality levels of detail. Of course other multiresolution compression schemes could have been used such as JPEG2000, or some random-accessible texture-specialized methods such as ASTC [NLP*12]. We then propose a multiplexing algorithm between mesh LoDs and texture LoDs. This scheme alternates mesh information and texture information in the compressed stream in order to provide the highest quality levels of detail at any time of the decompression. To determine optimal multiplexing, we start from the coarsest levels of detail for mesh and geometry. At each iteration, we select either to refine the mesh or the texture. This choice is made by optimizing the tradeoff between quality improvement achieved by the refinement and its cost in terms of number of bits. In practice, we choose the refinement which maximizes \( \Delta Q \). Where \( \Delta Q \) is the difference of visual quality before and after the refinement and \( \Delta S \) the data size of the refinement chunk.

Measuring the visual quality \( Q \) of a textured mesh is still an open problem. A recent study [LLV16] suggests that perceptual image metrics (in particular the multiscale SSIM [WSB03]) may be good predictors of the visual quality of 3D models. We thus use this framework for measuring the quality of our textured meshes. As recommended in [LLV16], we use 42 snapshots of the 3D objects to be compared (i.e. the original model and a level of detail). Cameras are placed uniformly around the object using a one-level dyadic split of a regular icosahedron, and an indirect top left lighting is considered. Each corresponding pair of snapshots is then evaluated using the multiscale SSIM (MS-SSIM) metric. Its value ranges from 0 (totally different) to 1 (exactly the same). We keep the mean of the 42 values as the final visual quality score \( Q \) for the level of detail to be evaluated.

7. Results
In this section, we compare our algorithm with state-of-the-art approaches for progressive compression of non-manifold triangular meshes [PK05, PHK*10], and compression of texture coordinates [Hop96, VB14, LCCD16]. We finally provide results of our perceptually-optimized mesh/texture multiplexing.

7.1. Geometry and connectivity compression
We present, in Table 1, lossless bitrates and execution times (on a 3.4GHz Intel Core i5) for different 3D models. Bitrates include both geometry and connectivity. The compression was carried using 12-bit geometry quantization (just like all the following results). Figure 12 illustrates the LoDs for the Hippo model. Our algorithm is able, on average, to decompress between 17K and 44K vertices per second, which is competitive with state-of-the-art progressive algorithms.

<table>
<thead>
<tr>
<th>Model</th>
<th># Vertices</th>
<th>Type</th>
<th>Bitrate (bpv)</th>
<th>Comp. time</th>
<th>Decomp. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bimba</td>
<td>8,857</td>
<td>Tri. Man.</td>
<td>35.31</td>
<td>0.264</td>
<td>0.056</td>
</tr>
<tr>
<td>Dragon</td>
<td>437,645</td>
<td>Tri. Man.</td>
<td>23.65</td>
<td>4.261</td>
<td>0.022</td>
</tr>
<tr>
<td>DancingArt</td>
<td>1,175,653</td>
<td>Tri. Man.</td>
<td>22.94</td>
<td>6.199</td>
<td>0.026</td>
</tr>
<tr>
<td>Bunny</td>
<td>35,947</td>
<td>Tri. Man.</td>
<td>27.62</td>
<td>0.287</td>
<td>0.057</td>
</tr>
<tr>
<td>Armadillo</td>
<td>125,340</td>
<td>Quad. Man.</td>
<td>25.37</td>
<td>0.438</td>
<td>0.035</td>
</tr>
<tr>
<td>Bimba-q</td>
<td>15,653</td>
<td>Quad. Man.</td>
<td>29.05</td>
<td>0.298</td>
<td>0.061</td>
</tr>
<tr>
<td>Hippo</td>
<td>65,436</td>
<td>Poly. Man.</td>
<td>28.55</td>
<td>0.589</td>
<td>0.048</td>
</tr>
<tr>
<td>Tractor</td>
<td>27,251</td>
<td>Quad. NM.</td>
<td>26.60</td>
<td>0.330</td>
<td>0.055</td>
</tr>
<tr>
<td>HousePlant</td>
<td>35,372</td>
<td>Tri. NM.</td>
<td>31.29</td>
<td>0.339</td>
<td>0.056</td>
</tr>
</tbody>
</table>

Table 1: Lossless compression rate and timings for several 3D models with different connectivities: triangles, quads, higher order polygons, manifold (Man.) and non-manifold (NM.). Timings are in seconds per thousand of vertices.

Algorithm able to handle polygonal non-manifold models. We compared it with state-of-the-art progressive compression methods able to deal with triangular non-manifold models: the octree approach by Peng and Kuo [PK05] and the feature-oriented algorithm by Peng et al. [PHK*10]. Table 2 shows the mean geometric error given by METRO [COS98] for different meshes of various connectivity (both manifold and non-manifold) at different bitrates. Mean errors for [PK05] and [PHK*10] were computed from the LoDs (and corresponding bitrates) provided by the authors. "N/A" values mean early termination. Figure 9 also presents rate-distortion curves obtained for the Horse model (triangular manifold) using Max Root Mean Square error. These results show that our method exhibits an excellent rate-distortion tradeoff at low and medium bitrates thanks to its strong distortion control, better than concurrent methods. Moreover, contrary to [PHK*10], it allows lossless decomposition, even if its lossless rate is more costly than best algorithms (e.g. [PK05]). It is also interesting to notice that compression performance does not drop for non-manifold meshes; for instance, for the Mapple model, our algorithm is better than [PK05] at all bitrates and better than [PHK*10] for the highest ones.

7.2. Texture mapping compression
We first compare the lossless performance of our algorithm with the top performing methods by [VB14]: the weighted parallelogram prediction and the cotan and mean value Laplacian algorithms, which represent state-of-the-art single rate texture coordinate compression. Results are shown in Table 3 for the set of parameterized models used in [VB14] (bitrates are taken from the original paper). For our algorithm, we adjusted the quantization \( Q_t \) in order to obtain the same error as the other methods, at lossless rate. For this...
Table 2: Mean error values (scaled by $10^4$) for our method, [PK05] and [PHK*10], at different bitrates. Objects marked with an * are non-manifold. Best results are highlighted.

<table>
<thead>
<tr>
<th>Mesh</th>
<th># Vertices</th>
<th>Method</th>
<th>1.0</th>
<th>2.0</th>
<th>4.0</th>
<th>8.0</th>
<th>12.0</th>
<th>16.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horse</td>
<td>(19,851)</td>
<td>Our method</td>
<td>20.2</td>
<td>9.3</td>
<td>4.1</td>
<td>1.6</td>
<td>1.0</td>
<td>0.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[PK05]</td>
<td>31.9</td>
<td>19.4</td>
<td>10.2</td>
<td>3.3</td>
<td>0.9</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[PHK*10]</td>
<td>19.0</td>
<td>12.8</td>
<td>5.1</td>
<td>3.3</td>
<td>1.6</td>
<td>1.1</td>
</tr>
<tr>
<td>Rabbit</td>
<td>(67,039)</td>
<td>Our method</td>
<td>5.3</td>
<td>2.4</td>
<td>1.1</td>
<td>0.6</td>
<td>0.3</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[PK05]</td>
<td>18.9</td>
<td>10.2</td>
<td>8.3</td>
<td>2.2</td>
<td>0.6</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[PHK*10]</td>
<td>5.5</td>
<td>4.2</td>
<td>2.1</td>
<td>1.7</td>
<td>1.0</td>
<td>0.5</td>
</tr>
<tr>
<td>Maple*</td>
<td>(45,499)</td>
<td>Our method</td>
<td>28.3</td>
<td>17.7</td>
<td>9.2</td>
<td>4.9</td>
<td>3.2</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[PK05]</td>
<td>29.8</td>
<td>18.6</td>
<td>13.8</td>
<td>11.3</td>
<td>6.1</td>
<td>4.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[PHK*10]</td>
<td>21.5</td>
<td>13.6</td>
<td>8.4</td>
<td>5.4</td>
<td>3.7</td>
<td>2.8</td>
</tr>
<tr>
<td>Tractor*</td>
<td>(27,251)</td>
<td>Our method</td>
<td>31.2</td>
<td>26.3</td>
<td>14.5</td>
<td>6.2</td>
<td>3.8</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Our method</td>
<td>19.0</td>
<td>11.9</td>
<td>10.6</td>
<td>7.3</td>
<td>5.5</td>
<td>3.8</td>
</tr>
</tbody>
</table>

Figure 9: Rate-distortion curves for Horse model. Distortion is measured by the Max Root Mean Square error, using METRO [COS98]. The MRMS values are scaled by 10^4.

comparison the error is computed as the Root Mean Square Error over the texture coordinates. Our algorithm outperforms [VB14] for almost all the models. A visual comparison, at similar bitrates, is illustrated in Figure 10. Most tested models are derived from automatic conformal parametrization and thus are particularly well suited for our prediction. However, even for Victoria and Frog associated with manual parametrization, our bitrates are still good.

We also compare our algorithm with the few existing methods able to handle progressive encoding of textured mesh with texture seams: the progressive mesh from Hoppe [Hop96] and the recent algorithm from Lavoué et al. [LCCD16] which creates invisible triangle strips at texture seams to get back to a simpler per-vertex UV representation, before applying the progressive encoding from Alliez and Desbrun [AD01]. Both algorithms are restricted to manifold meshes. Rate distortion curves for the Tiger Fighter model are illustrated in Figure 11. Bitrates correspond to geometry, connectivity and UV encoding. We used original author’s implementations of [Hop96] and [LCCD16] and adjusted our quantization parameters to stick with theirs (resp. $Q_g = 16$, $Q_r = 16$ and $Q_u = 12$, $Q_t = 10$).

Our method outperforms both others. [Hop96] is very costly (more than 400 bpv at lossless rate), [LCCD16] is quite efficient at lossless rate (26 bpv against 23 bpv for our method) but produces very low quality LoDs for low bitrates (due to artifacts along seams). Other comparisons are illustrated in the supplementary material.
Figure 12: Progressive decompression for different models (Hippo, Creature, Dwarf and Tractor). Textured LoDs are obtained using our mesh-texture multiplexing. We present the total size of the decoded data for the mesh elements (geometry, connectivity, texture coordinates) and for the texture image (in parentheses).
Figure 11: Rate-Quality curves for Tiger Fighter. Visual quality is measured by the mean MS-SSIM values (higher is better). LoDs corresponding to the dots on the curves are illustrated. Models are mapped with the uncompressed texture image.

7.3. Texture multiplexing

Finally, Figures 1 and 12 illustrate some visual results of our mesh/texture multiplexing approach. It is interesting to observe that starting at 150-200KB, the levels of detail become really close to the original models. Quantitative results are given in the supplementary material.

8. Conclusion

In this paper, we have proposed a new progressive compression algorithm suited for arbitrary textured meshes. This algorithm handles polygonal and non-manifold models as well as texture mapping discontinuities. Our approach provides excellent results at low and medium bitrates for geometry and connectivity compression. Moreover, it outperforms the state-of-the-art for texture coordinates compression and allows high quality levels of detail to be obtained, even at very small bitrates.

In a future work we plan to propose fast parallel decoding mechanisms and structure optimizations in order to reach timing performance attained by recent multi-resolution methods (e.g. [PD15]) specifically designed for interactive Web-based rendering, but much less efficient in terms of compression performance. We also envision extending our edge-collapse and vertex-split operators to volume meshes, particularly used in scientific visualization.

Acknowledgements

The Dwarf is courtesy of the Visual Computing Laboratory of ISTI-CNR. We thank Libor Vasa for providing us with his parameterized models.

References