Progressive Forest Split Compression

Gabriel Taubin [1] André Guéziec [1] William Horn [1] Francis Lazarus [2]

IBM T. J. Watson Research Center

ABSTRACT

In this paper we introduce the Progressive Forest Split (PFS) representation, a new adaptive refinement scheme for storing and transmitting manifold triangular meshes in progressive and highly compressed form. As in the Progressive Mesh (PM) method of Hoppe, a triangular mesh is represented as a low resolution polygonal model followed by a sequence of refinement operations, each one specifying how to add triangles and vertices to the previous level of detail to obtain a new level. The PFS format shares with PM and other refinement schemes the ability to smoothly interpolate between consecutive levels of detail. However, it achieves much higher compression ratios than PM by using a more complex refinement operation which can, at the expense of reduced granularity, be encoded more efficiently. A *forest split* operation doubling the number n of triangles of a mesh requires a maximum of approximately $3.5n$ bits to represent the connectivity changes, as opposed to approximately $(5 + \log_2(n))\,n$ bits in PM.

We describe algorithms to efficiently encode and decode the PFS format. We also show how any surface simplification algorithm based on edge collapses can be modified to convert single resolution triangular meshes to the PFS format. The modifications are simple and only require two additional topological tests on each candidate edge collapse. We show results obtained by applying these modifications to the Variable Tolerance method of Guéziec.

CR Categories and Subject Descriptors:
I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling - surface, solid, and object representations.

General Terms: Geometric Compression, Algorithms, Graphics.

1 INTRODUCTION

Although modeling systems in Mechanical Computer Aided Design and in animation are expanding their geometric domain to free form surfaces, polygonal models remain the primary 3D representation used in the manufacturing, architectural, and entertainment industries. Polygonal models are particularly effective for hardware assisted rendering, which is important for video-games, virtual reality, fly-through, and digital prototyping.

A polygonal model is defined by the position of its vertices (geometry); by the association between each face and its sustaining vertices (connectivity); and optional colors, normals and texture coordinates (properties). In this paper we concentrate on manifold polygonal models described by *triangular meshes* without attached properties. However, we address issues of non-triangular polygons, properties, and non-manifolds in Section 7. A method to triangulate arbitrary polygonal faces is described by Ronfard and Rossignac [16]. A method to convert non-manifold polygonal models to manifold polygonal models is described by Guéziec *et al.* [7].

[1] IBM T.J.Watson Research Center, P.O.Box 704, Yorktown Heights, NY 10598, {taubin,gueziec,hornwp}@watson.ibm.com

[2] IRCOM-SIC (UMR CNRS 6615), SP2MI, Bvd. 3, Teleport 2, B.P. 179, 86960 Futuroscope Cedex, France, lazarus@sic.univ-poitiers.fr

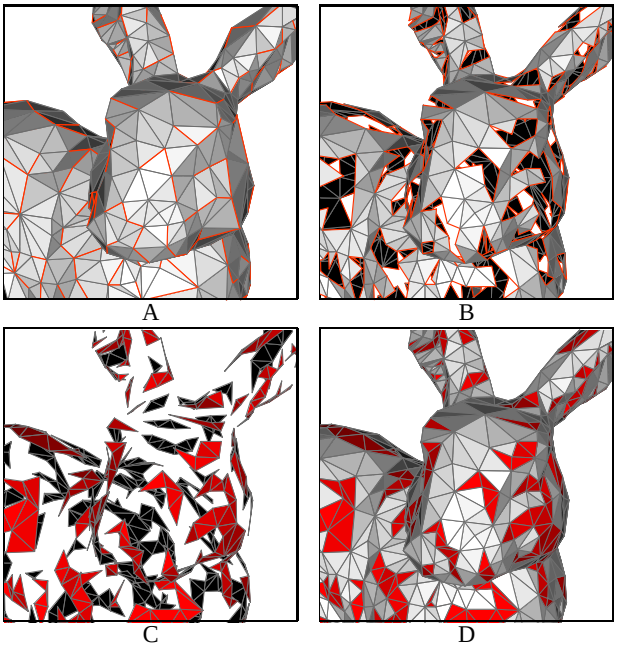

Figure 1: The forest split operation. A: A triangular mesh with a forest of edges marked in red. B: Resulting mesh after cutting through the forest edges and splitting vertices in the resulting tree boundary loops. C: Simple polygons to be stitched to the boundary loops. The correspondence between polygon boundaries and tree boundary loops is implicit. D: The refined mesh. Normally, to produce a smooth transition, the vertices are displaced only after the boundary loops are triangulated. In the figure they have been displaced immediately after the cutting to illustrate the connectivity refinement process.

Polygonal models are typically stored in file servers and exchanged over computer networks. It is frequently desirable to compress models to reduce storage and transmission time requirements. Effective single-resolution-compression schemes have been recently introduced by Deering [3] and Taubin and Rossignac [20].

While single resolution schemes can be used to reduce transmission bandwidth, it is frequently desirable to send the model in a progressive fashion. For example a progressive scheme may start by sending a compressed version of the lowest resolution level of a level-of-detail (LOD) hierarchy. After the lowest level has been sent, a sequence of additional refinement operations may be sent in parallel to the rendering operation. In this manner, successively finer levels of detail may be displayed while even more detailed levels are still arriving.

To prevent visual artifacts, sometimes referred to as “popping”, it is also desirable to be able to transition smoothly, or *geomorph*, from one level of the LOD hierarchy to the next by interpolating the positions of corresponding vertices in consecutive levels of detail as a function of time.

The Progressive Forest Split (PFS) scheme is introduced in this

paper and features a new adaptive refinement scheme for storing and transmitting triangle meshes in progressive and highly compressed form. In this scheme a manifold triangular mesh is represented as a low resolution polygonal model followed by a sequence of refinement operations. The scheme permits the smooth transition between successive levels of refinement. High compression ratios are achieved by using a new refinement operation which can produce more changes per bit than existing schemes. The scheme requires only $O(n)$ bits to double the size of a mesh with n vertices.

The *forest split* operation, the refinement operation of the PFS scheme, is illustrated in Figure 1. It is specified by a forest in the graph of vertices and edges of the mesh, a sequence of simple polygons (triangulated with no internal vertices), and a sequence of vertex displacements. The mesh is refined by cutting the mesh through the forest, splitting the resulting boundaries apart, filling each of the resulting *tree boundary loops* with one of the simple polygons, and finally displacing the new vertices.

In Section 3 we describe the algorithms for efficiently encoding and decoding PFS meshes. We show how any surface simplification algorithm based on edge collapses can be modified to convert single resolution triangular meshes to PFS format. The modifications require performing two simple additional topological tests on each candidate edge collapse.

In Section 6 we show results obtained by applying these modifications to the Variable Tolerance surface simplification method of Guéziec [6]. Using this method we guarantee simplification error bounds on all levels of details as a measure of the approximation quality. We finish the paper with a short discussion on extensions and future work.

2 PREVIOUS WORK

Single-resolution mesh compression schemes Deering's method [3] is designed to compress the data for transmission from the CPU to the graphics adapter. The method uses a stack-buffer to store 16 of the previously used vertices instead of having random access to all the vertices of the model. The triangles of the mesh are partitioned into *generalized triangle meshes*, and the connectivity of the triangular mesh is lost. The vertex positions are quantized and the differences between consecutive values are entropy encoded.

The Topological Surgery (TS) method of Taubin and Rossignac [20] was designed for fast network transmission and compact storage. In this method the connectivity of a manifold triangular mesh is encoded without loss of information, with storage rates approaching one bit per triangle for large models. In this scheme the vertices are organized as a spanning tree, and the triangles as a simple polygon. The vertex positions and properties are quantized, predicted as a linear combination of ancestors along the vertex tree, and the corrections are entropy encoded. A more detailed description is presented in Section 3.1. The method has been extended to handle all the polygonal models which can be represented in the Virtual Reality Modeling Language (VRML) [19], including all properties and property bindings permitted by the language. Compression ratios of up to 50:1 or more can be achieved for large VRML models.

Recursive subdivision and refinement Recursive subdivision schemes [2, 4, 13] provide a truly progressive representation for a limited family of meshes. Most have the ability to transition smoothly between consecutive levels of detail. In a recursive subdivision scheme a polygonal mesh is defined as a low resolution base mesh followed by a sequence of subdivision steps. Each subdivision step can be further decomposed into a connectivity refinement step, and a smoothing or geometry update step. In the connectivity refinement step more vertices and faces are added to the mesh. These additions usually do not change the topological

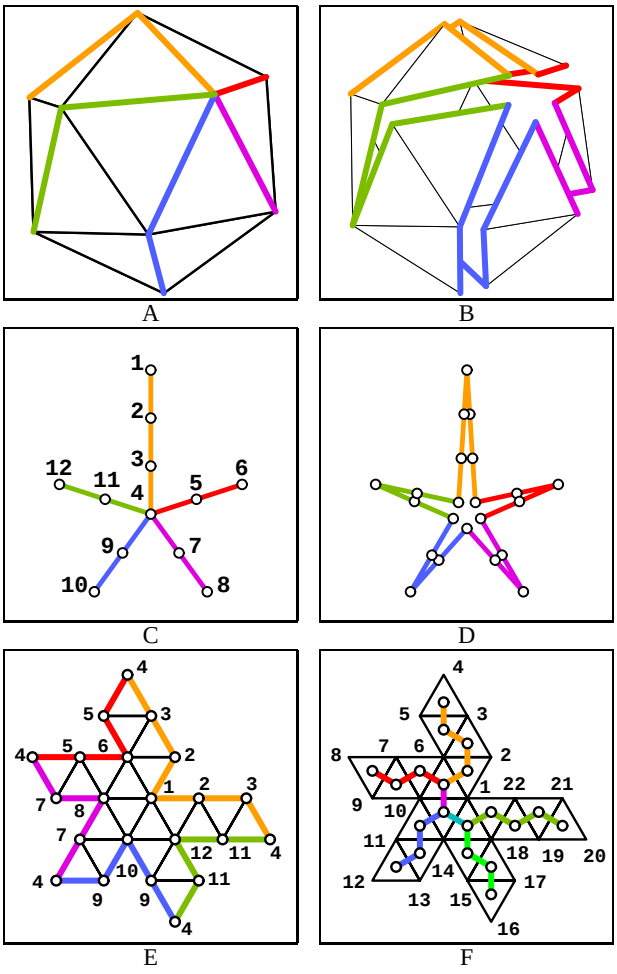

Figure 2: The topological surgery method of Taubin and Rossignac [20]. A: The vertex tree on the surface. Each run is painted with a different color. B: The result of cutting through the edges of the vertex tree (the vertex positions have been modified here to artificially enlarge the gap created by the topological cut) is the simple polygon. C: The structure of the vertex tree with the vertex tree traversal indices. D: The vertex tree becomes the boundary loop after cutting through its edges. E: The simple polygon (artificially flattened) has no internal vertices. Simple polygon vertices are labeled here with their corresponding vertex tree traversal indices. F: The dual graph of the simple polygon is the triangle tree. One marching bit per regular node is also required to describe the triangulation of its runs. The order for the vertex indices is derived from the vertex tree traversal order. Simple polygon vertices are labeled here with their boundary loop indices. The correspondence between boundary loop indices and vertex tree indices is stored in a lookup table constructed by traversing the vertex tree.

type and new vertices are positioned such that the overall geometry does not change. In the smoothing step, depending on whether the method is approximating or interpolating, some or all of the vertices of the mesh with refined connectivity are displaced. Usually the goal is to show that after an infinite number of subdivision steps, the sequence of polygonal meshes converges to a smooth continuous surface, but in practice only a few subdivision steps are applied.

Uniform subdivision schemes can be regarded as the optimal compression schemes. Both the connectivity refinement and the smoothing steps are defined globally by a few parameters per subdivision step [18] and the schemes are optimal in the sense that the number of parameters is independent of the number of vertices and faces created at each subdivision step. However, these high compression ratios are obtained only for meshes with recur-

sive subdivision connectivity. Eck *et al.* [5] describe a method to approximate a triangular mesh by a new mesh with recursive subdivision connectivity and approximately the same geometry, but very often the option of changing the connectivity of the mesh in this way is not possible.

Adaptive subdivision schemes [9, 21] must specify how and where the mesh connectivity is to be refined. This may require as little as only one bit per vertex, edge, or face to specify each connectivity refinement step. These schemes may also require vertex displacements for the newly created vertices, instead of, or in addition to, the global parameters defining the smoothing step.

As noted above, the main problem with existing uniform and adaptive subdivision schemes is that a general triangular mesh usually does not satisfy subdivision requirements and cannot be compressed with these methods. The Progressive Mesh (PM) scheme introduced by Hoppe [11] solves this problem. The scheme is not a subdivision scheme but an *adaptive refinement* scheme where new faces are not created by subdividing existing faces, but by inserting them in between existing faces. Every triangular mesh can be represented as a base mesh followed by a sequence of refinements referred to as *vertex splits*. Each vertex split is specified for the current level of detail by identifying two edges and a shared vertex. The mesh is refined by cutting it through the pair of edges, splitting the common vertex into two vertices and creating a quadrilateral hole, which is filled with two triangles sharing the edge connecting the two new vertices. The PM scheme is not an efficient compression scheme. Since the refinement operations perform very small and localized changes the scheme requires $O(n \log_2(n))$ bits to double the size of a mesh with n vertices.

The refinement operation of the PFS scheme introduced in this paper, the forest split, can be seen as a grouping of several consecutive edge split operations into a set, instead of a sequence. In the PFS scheme there is a tradeoff between compression ratios and granularity. The highest compression ratios are achieved by minimizing the number of levels of detail. Most often the high number of levels of detail produced by the PM scheme are not required. Hoppe typically defines the levels of the LOD hierarchy of his PM representation using exponential steps.

3 THE PFS REPRESENTATION

A multi-resolution mesh represented in the PFS format is composed of an initial low resolution level of detail followed by a sequence of forest split operations. Although any method could be used to represent the lowest resolution level of detail, we use the TS method because the PFS representation is a natural extension of the representation used in this scheme. For both the lowest resolution base mesh and the forest split operations we make a distinction between the representation and the encoding of the representation.

3.1 Topological Surgery

In this section we give a brief description of the TS representation for a *simple mesh*, that is, a triangle mesh with sphere topology. With a few minor additions, manifolds of arbitrary genus, with or without boundaries, and orientable or non-orientable can also be represented. Since these additional concepts are not needed to describe the PFS format, we refer the interested reader to the original reference for the details.

Representation Figure 2 illustrates the main concepts of the TS representation. In this method the vertices of a triangular mesh are organized as a rooted spanning tree in the graph of the mesh, called the *vertex tree* (Figure 2-A). As shown in Figure 2-B,E, when a simple mesh is cut through the vertex tree edges, the connectivity of the resulting mesh is a simple polygon. The edges of the simple polygon form a *boundary loop*.

The order of traversal of the vertex tree (Figure 2-C) defines a one-to-two correspondence between the edges of the vertex tree and the edges of the *boundary loop* (Figure 2-D). This correspondence defines which pairs of boundary loop edges should be stitched together to reconstruct the connectivity of the original mesh.

Encoding The encoding of this representation in the compressed data stream is composed of, in order: the encoding of the vertex tree, the compressed coordinate information, and the encoding of the simple polygon.

The vertex tree is run-length encoded. The tree is decomposed into *runs* (shown in different colors in Figure 2-A,C). A run connects a leaf or branching node to another leaf or branching node through a path of zero or more regular nodes. The order of traversal of the tree defines an order of traversals of the runs, and a first and last node for each run. Each run is encoded as a record composed of three fields *(is-last-run,length-of-run,ends-in-leaf)*. The *is-last-run* field is a bit that determines if the runs shares the first node with the next run or not. It determines the pushing of branching node indices onto a *traversal stack*. The *length-of-run* field is a variable length integer (same number of bits for all the runs in the tree) with a value equal to the number of edges in the run. The *ends-in-leaf* field is a bit which determines if the run ends in a leaf or branching node, and the popping of branching node indices from the traversal stack.

The coordinate information is placed in the compressed stream in the order of traversal of the vertex tree. The coordinate data is compressed by storing errors instead of absolute coordinates. The errors are calculated with respect to a predictor. The predictor is computed as a linear combination of several ancestors in the tree and is quantized to a certain number of bits per coordinate with respect to a bounding box. The errors are then Huffman-encoded. Once the coordinate information is received, the geometry of the mesh can be reconstructed as an array of vertex coordinates.

The dual graph of the simple polygon is also a tree (Figure 2-F). The structure of this *triangle tree* is run-length encoded in the same way as the vertex tree, except that the *is-last-run* field is not necessary, because the triangle tree is a binary tree. The structure of the triangle tree does not completely describe the triangulation of the polygon. To complete the description, an extra bit per triangle associated with each regular node of the triangle tree must be included. These *marching bits* determine how to triangulate the runs of the tree by advancing either on the left or on the right on the boundary of the polygon. This encoding scheme produces very good results for polygons with very few and long runs. Another encoding scheme for simple polygons is described in the next section.

3.2 The forest split operation

Representation A forest split operation, illustrated in Figure 1, is represented by: a forest in the graph of vertices and edges of a mesh; a sequence of simple polygons; and a sequence of vertex displacements. The mesh is refined by cutting the mesh through the forest, splitting the resulting boundaries apart, filling each of the resulting tree boundary loops with one of the simple polygons, and finally, displacing the new vertices.

Applying a forest split operation involves: 1) cutting the mesh through the forest edges; 2) triangulating each tree loop according to the corresponding simple polygon; and 3) displacing the new vertices to their new positions. As will be explained in the next section, some of the information required to perform these steps, such as the correspondence between trees of the forest and simple polygons, and between tree boundary loop edges and polygon boundary loop edges of each corresponding tree-polygon pair, is not given

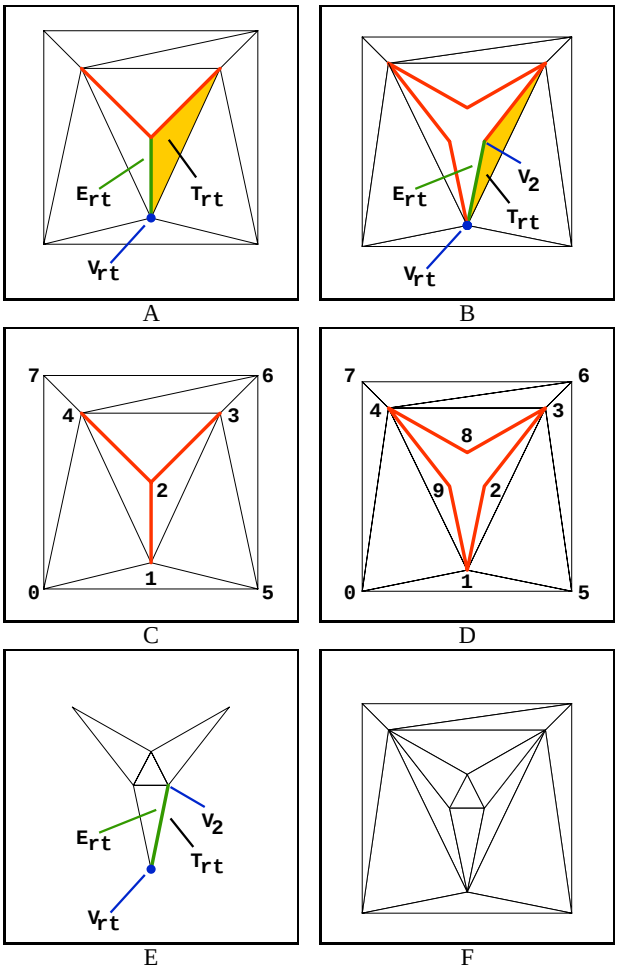

Figure 3: When a mesh is cut through a tree of edges (red and green edges in A), a tree boundary loop (red and green edges in B) is created with each edge of the tree corresponding to two edges of the boundary loop. Some vertex indices are assigned before cutting (C) to new tree boundary loop vertices, others are assigned subsequent indices (D). The hole created by the cutting operation is filled by triangulating the boundary loop using a simple polygon (E) resulting in a refined mesh (F) with the same topological type as the initial mesh.

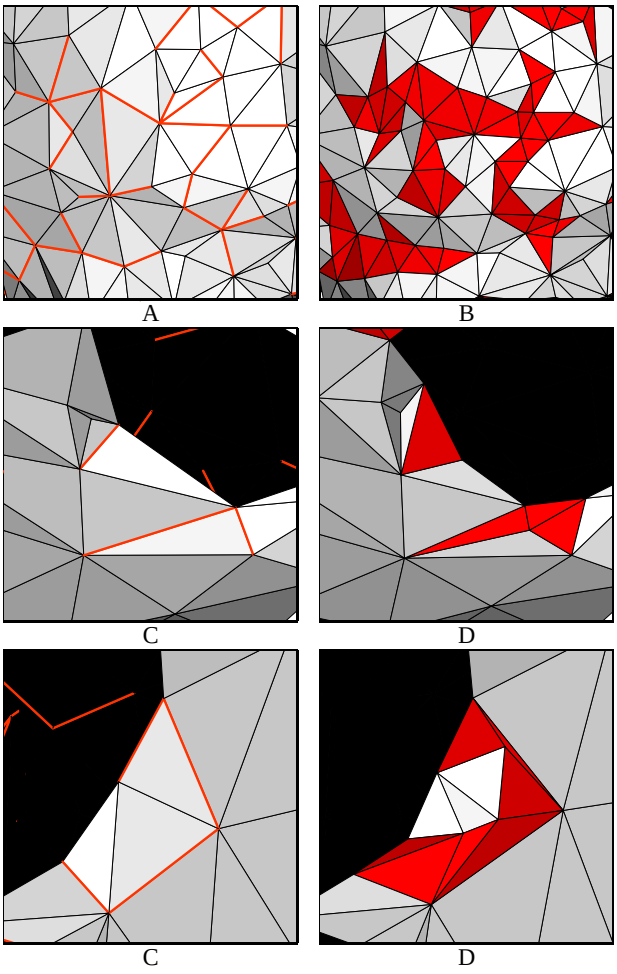

Figure 4: Construction and triangulation of tree boundary loops. A,B: No tree vertices in the mesh boundary. C,D: A tree vertex isolated in the mesh boundary requires an extra tree loop edge. E,F: A tree edge on the mesh boundary edges does not require an extra tree loop edge, but some of the new vertex indices may only be used by new triangles. Note that the tree may have several contacts with the mesh boundary.

explicitly, but is based on an implicit convention for enumerating mesh elements.

Enumeration of mesh elements Given a triangular mesh with V vertices and T triangles, we assume that the vertices have consecutive *vertex indices* in the range $0, \ldots, V-1$, and the triangles have consecutive *triangle indices* in the range $0, \ldots, T-1$. The edges of the mesh, which are represented by pairs of vertex indices (i, j) with $i < j$, are ordered lexicographically and assigned consecutive *edge indices* in the range $0, \ldots, E-1$. The trees in the forest are ordered according to the minimum vertex index of each tree. The *root vertex* v_{rt} of each tree in the forest is the leaf of the tree with the minimum index. Starting at the root, the boundary loop created by cutting along the tree can be traversed in cyclic fashion in one of the two directions. The *root edge* e_{rt} of the tree is the only edge of the tree which has the root vertex as an endpoint. Of the two triangles incident to the root edge of the tree, the *root triangle* t_{rt} of the tree is the one with the minimum triangle index. The root triangle of the tree determines the direction of traversal of the tree boundary loop. Of the two edges of the tree boundary loop corresponding to the root edge of the tree, the *root edge* e_{rt} of the tree boundary loop is the one incident to the root triangle. Figures 3-A,B illustrate these concepts.

Each simple polygon has a boundary edge identified as the *root edge* e_{rt}, with one of the two endpoints labeled as the *root vertex* v_{rt}, and the other endpoint labeled as the *second vertex* v_2. Figure 3-E illustrates these concepts. The cyclical direction of traversal of the polygon boundary loop is determined by visiting the root vertex first, followed by the second vertex. The correspondence between vertices and edges in a tree boundary loop and the polygon boundary loop is defined by their directions of cyclical traversal and by the matching of their root vertices.

Cutting through forest edges Cutting through a forest of edges can be performed sequentially, cutting through one tree at time. Each cut is typically a local operation, affecting only the triangles incident to vertices and edges of the tree. However, as in the TS method, a single cut could involve all the triangles of the mesh. Cutting requires duplicating some tree vertices, assigning additional indices to the new vertices and fixing the specification of the affected triangles.

As illustrated in Figure 4-A,B, if no tree vertex is a bound-

ary vertex of the mesh, then the tree is completely surrounded by triangles. Starting at the root triangle, all the corners of affected triangles can be visited in the order of traversal of the tree boundary loop, by jumping from triangle to neighboring triangle, while always keeping contact with the tree. This process produces a list of triangle corners, called the *corner loop*, whose values need to be updated with the new vertex indices. While traversing this list, we encounter runs of corners corresponding to the same vertex index before the cut. A new vertex index must be assigned to each one of these runs. To prevent gaps in the list of vertex indices we first need to reuse the vertex indices of the tree vertices, which otherwise would not be corner values of any triangles. The first visited run corresponding to one of these vertices is assigned that vertex index. The next visited run corresponding to the same vertex index is assigned the first vertex index not yet assigned above the number of vertices of the mesh before the cut. This procedure performs the topological cut. For example, in Figure 3-C, the vertex index values of the corners in the corner loop list are:

$$[12333332444444421111] .$$

The list can be decomposed into 6 runs $[11111]$, $[2]$, $[33333]$, $[2]$, $[444444]$, and $[2]$. As shown in Figure 3-D, the vertex indices assigned to these runs are $1, 2, 3, 8, 4, 9$.

A tree with m edges containing no mesh boundary vertices creates a tree boundary loop of $2m$ edges. This may not be the case when one or more tree vertices are also part of the mesh boundary. As illustrated in Figures 4-C,D,E,F, several special cases, must be considered. These special cases treat collapsed edges incident to or on the mesh boundary produced by the PFS generation algorithms as described in Section 5.

Triangulating tree boundary loops By replacing each run of corners in the corner loop with the assigned vertex index, we construct a new list representing the tree boundary loop, If the tree boundary loop has m vertices, so does the corresponding polygon boundary loop. Each triangle $t = \{i, j, k\}$ of the simple polygon defines a new triangle of the refined mesh by replacing the polygon boundary loop indices i, j, k with their corresponding tree boundary loop indices. This is done using the list representing the tree boundary loop as a lookup table. The triangles of the simple polygon are visited in the order of a depth first traversal of its dual tree. The traversal starts with the triangle opposite to the root triangle and always traverses the left branch of a branching triangle first.

Displacing vertices To satisfy the smooth transition property, vertex coordinates corresponding to new vertices are first assigned the same coordinates as the corresponding tree vertices before the cut. To prevent the appearance of holes, these vertices are displaced after the boundary loops are triangulated. Optionally, all affected vertices may be repositioned.

4 COMPRESSION AND ENCODING

In this section we describe how a model represented in PFS format is encoded/compressed for efficient transmission and storage. Compression and encoding of the first (lowest) resolution level of detail was discussed in Section 3.1. This block of data is followed by the compressed/encoded forest split operations in the order they are applied. The encoding of each forest split operation is composed of, in order: 1) the encoding of the forest of edges, 2) the encoding of the sequence of simple polygons, and 3) the compression/encoding of the vertex displacements.

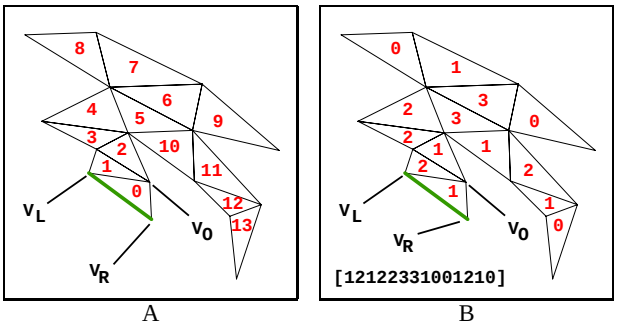

Figure 5: Constant-length encoding of a simple polygon. A: Triangles labels according to their order of traversal. B: Triangles labels according to their two bit code. The encoding of the polygon is the sequence between the brackets.

Encoding the forest A simple encoding of the forest requires one bit per edge, for example a value 1 for the edges which belong to the forest and 0 for the rest. These bits are placed in the compressed data stream in the edge index order defined above. However, since any subset of edges of a forest form a forest, the edges with bit value 1 read up to a certain point may determine that certain edges with higher edge index should have bit value 0, otherwise they would create loops in the forest defined so far by the bits with bit value 1. These edges can be skipped in the compressed data stream. When very few edges belong to the forest, run-length encoding this bit-stream may result in fewer bits. In the experiments that we have performed so far, where the number of triangles increase by 50-70% with each forest split operation, the skipping of predictable bits described above trims the length of the forest bit-stream by only about 1-8% and the simple encoding of the resulting bit-stream is usually shorter that the run-length encoded stream.

Encoding a simple polygon The variable length encoding scheme described in section 3.1 uses one record per triangle tree run and one marching bit per regular node of the triangle tree to encode a simple polygon. This encoding is not very efficient when the simple polygon is composed of a few triangles or short runs. A constant length encoding scheme, requiring exactly 2 bits per triangle, has been proposed by Frank Bossen as an alternative for the MPEG4 standard [1]. This scheme produces better results for polygons with few triangles or short runs.

In the current implementation we compute both encodings for all simple polygons and put the shortest one in the compressed data stream, preceded by one bit to indicate which encoding is used. That is, the simple polygons are either all constant length encoded or all run-length encoded.

The constant-length encoding scheme, illustrated in Figure 5, is performed by traversing the triangle tree. The traversal starts by entering the first triangle through the root edge, with the root vertex assigned to *left vertex* v_L, and the second vertex assigned to *right vertex* v_R. The third vertex of the triangle is assigned to the *opposite vertex* v_O, the edge $e_L = (v_L, v_O)$ is the *left edge*, and the edge $e_R = (v_R, v_O)$ is the *right edge*. One bit is used to indicate whether each edge (left and right) is a boundary edge or an internal edge of the polygon. If only the left edge is internal, we set $v_R = v_O$ and we continue with the other triangle incident to e_L. If only the right edge is internal, we set $v_L = v_O$ and we continue with the other triangle incident to e_R. If both edges are internal, we push v_O and v_R onto a traversal stack, we set $v_R = v_O$ and we continue with the other triangle incident to e_L. If both edges are boundary and the traversal stack is not empty, we pop v_R and v_L from the traversal stack, and we continue with the other triangle incident to the edge (v_L, v_R). If both edges are boundary, and the traversal stack is empty, we

have finished visiting all the triangles of the simple polygon. For example, in Figure 5-A, the triangles are labeled with their order of traversal, and in Figure 5-B, with their corresponding two-bit code, as a number in the range $0, \ldots, 3$. Here, for each digit the first bit equals 0 if the left edge is boundary, 1 if the left edge is interior. The second bit represents the right side and uses the same convention. The encoding of this polygon is $[12122331001210]$. Note that there is a very simple transformation from the constant-length encoding to the variable-length encoding. The 3s and 0s in the constant-length encoded sequence mark the end of the triangle runs. In this example, the triangle-runs are defined by the sub sequences $[121223]$, $[3]$, $[10]$, $[0]$, and $[1210]$, which are in the same order defined by the variable-length scheme. The length-of-run field value is the number of two-bit codes in the corresponding sub sequence (6,1,2,1, and 4 in this case). The ends-in-leaf bit value is determined by the last code in the sub sequence ($3 \rightarrow 0$, $0 \rightarrow 1$), and the marching bits by the other codes in the sequence, skipping the 3s and 0s ($(1 \rightarrow 0, 2 \rightarrow 1)$. In this example the marching pattern is the following sequence of bits $[010110010]$. The transformation from the variable-length encoding to the constant-length encoding is also straightforward.

Decoding a simple polygon As described in Section 3.2, applying the forest split operation requires the triangles of each simple polygon to be represented by triplets $t = \{i, j, k\}$ of polygon boundary loop indices. These indices are subsequently replaced with the vertex indices assigned to the corresponding tree boundary loop indices.

Since the order in which the polygon vertices are visited during tree traversal is usually not the sequential order of the boundary loop, the following recursive procedure is used to reconstruct the triangles of each simple polygon. As described above the traversal of a simple polygon starts by entering the first triangle crossing the root edge, with the left boundary loop index $i_L = 0$ corresponding to the root vertex, and the right boundary loop index $i_R = 1$ corresponding to the second vertex. In general, when we enter a triangle, we know the values of i_L and i_R, and only the opposite boundary loop index i_O must be determined.

If the two-bit code is 1 (leaf node with next triangle on the left), we set $i_O = i_R + 1$ (addition and subtraction is modulo the length of the polygon boundary loop) and reconstruct the triangle $\{i_L, i_O, i_R\}$, we set $i_R = i_O$, and continue. If the two-bit code is 2 (leaf node with next triangle on the right), we set $i_O = i_L - 1$, reconstruct the triangle $\{i_L, i_O, i_R\}$, we set $i_L = i_O$, and continue. To determine the value of i_O for a branching triangle, if we know the distance d along the boundary loop from the left vertex to the right vertex for the run attached to the left edge, we set $i_O = i_L + d$, reconstruct the triangle $\{i_L, i_O, i_R\}$, push i_R and i_O onto the traversal stack, set $i_R = i_O$, and continue. As explained by Taubin and Rossignac [20], these lengths can be recursively computed for all the runs from the encoding of the polygon based on the formula $d = l - 1 + d_L + d_R$, there d is the distance of one run, l is the length of the run. If the run ends in a branching node, d_L is the distance of the run attached to the left edge of the last triangle of the run, and d_R is the distance of the run attached to the right edge of the last triangle of the run. If the run ends in a leaf node, $d_L = d_R = 1$. If the two-bit code of the triangle has the value 0 (leaf node of the triangle tree), we set $i_O = i_L - 1$ or $i_O = i_R + 1$, and reconstruct the triangle $\{i_L, i_O, i_R\}$. If the stack is empty we have finished reconstructing the polygons. Otherwise we pop i_L and i_R values from the stack, and continue.

Encoding the sequence of simple polygons We encode a sequence of constant-length encoded simple polygons by specifying the total number of triangles in the complete sequence of simple polygons followed by a concatentation of the two-bit encoding sequences. It is not necessary to include special markers in the compressed data stream to indicate the beginning and end of each polygon. The following procedure, which uses a single integer variable called *depth*, determines the limits of the polygons. The depth variable is initialized to 1 before starting traversing a polygon. Each time a two-bit code with the value 3 is found (branching triangle), depth is incremented by one. Each time a two-bit code with the value 0 is found (leaf triangle), depth is decremented by one. The variable depth is always positive while inside the polygon. The end of the polygon is reached after the depth becomes equal to zero. If the polygons are run-length encoded, instead of the total number of runs, the *(length-of-run,ends-in-leaf)* records are put into the data stream in the order of traversal of the trees, preceded by the total number of runs in the sequence, and the number of bits per length-of-run. The same procedure described above (using a depth variable) can be used to determine the limits of the simple polygon.

Encoding the vertex displacements Rather than encoding the new absolute positions of the marked vertices, their positions after and before the forest split operation are first quantized to a certain number of bits per coordinate with respect to a global bounding box enclosing all the levels of detail. The differences between these values are then Huffman encoded. The Huffman encoding table, the specification of the bounding box, and the number of bits per coordinate error, are included at the beginning of the compressed stream. The same bounding box and number of bits per coordinate is used by the TS scheme, described in Section 3.1, to encode the coordinates of the lowest resolution level of detail, but because the errors are computed in a different way, different Huffman tables are used. Also, since the errors keep growing smaller as more forest split operations are applied, we use a different Huffman table for each forest split operation.

Pre and post smoothing The differences between vertex positions before and after each forest split operation can be made smaller by representing these errors as the sum of a global predictor plus a correction. We use the smoothing method of Taubin [18] as a global predictor. The method requires only three global parameters which are included in the compressed data stream. After the connectivity refinement step of a forest split operation is applied, the new vertices are positioned where their corresponding vertices in the previous level of detail were positioned and the mesh has many edges of zero length (all the new triangles have zero surface area). The smoothing method of Taubin, which tends to equalize the size of neighboring triangles, brings the endpoints of most such edges apart, most often reducing the distance to the desired vertex positions. The corrections, the differences between the vertex positions after the split operation and the result of smoothing the positions before the split operation, are then quantized according to the global quantization grid and entropy encoded. To make sure that the resulting vertex positions have values on the quantization grid, the smoothed coordinates must be quantized before computing the corrections. In our experiments, this procedure reduces the total length of the entropy encoded corrections by up to 20-25%.

If the polygonal model approximates a smooth shape, a post-smoothing step can be applied to reduced the visual artifacts produced by the quantization process. In many cases, even fewer bits per coordinate can be used in the quantization process without a significant perceptual difference, also reducing the total length of the encoded vertex displacements. Figure 6 illustrates the effect of post-smoothing.

Compression ratios The simple encoding of the forest with one bit per edge and the constant-length encoding of the simple polygons provide an upper bound to the number of bits required to encode a forest split operation. Since the number of edges in the mesh is independent of the number of triangles added, the minimum

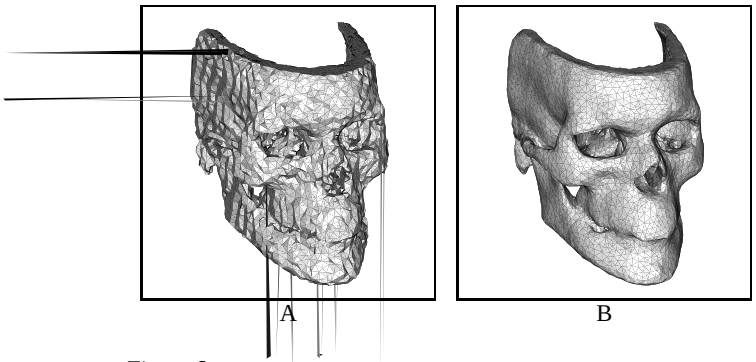

Figure 6: Effect of post-smoothing. A: Coordinates quantized to 6 bits per coordinate. B: Result of applying the smoothing algorithm of Taubin [18] with parameters $n = 16\ \lambda = 0.60\ \mu = -0.64$. Compare with the original in Figure 8-D.

number of bits per triangle are obtained when the most triangles are added with one forest split operation.

The most triangles we can add with one forest split operation is approximately equal to the current number of triangles. As the number of edges in the forest increases, so does the number of triangles added. The forest with the largest number of edges for a triangle mesh with V vertices has $V-1$ edges and corresponds to a forest composed of a single spanning tree. Ignoring the case of meshes with boundaries, this single tree creates a tree boundary loop with $2V-2$ edges, and a simple polygon with that many boundary edges is composed of $2V-4$ triangles. Since typical meshes have about twice as many triangles as vertices, the number of triangles in this simple polygon is approximately equal to T, the number of triangles in the mesh. If we also assume that the mesh has low Euler number ($V-E+T \approx 0$), then $E \approx 1.5T$. If $\Delta T = \alpha T$ is the number of triangles added by the forest split operation ($0 < \alpha < 1$), the total number of bits required to encode the connectivity information of this forest split operation is approximately equal to $(1.5/\alpha + 2)\Delta T$. This is 3.5 bits per triangle for $\alpha = 1$, 4 bits per triangle for $\alpha = 0.75$, and 5 bits per triangle for $\alpha = 0.5$.

In the PM scheme each vertex split operation requires an average of $5 + \log_2(n)$ bits to specify the refinement in the connectivity of the mesh ($\log_2(n)$ bits to specify the index of the common vertex, and an average of 5 bits to specify the two incident edges), to a total storage of about $n(5 + \log_2(n))$ bits to double the number of vertices of the mesh. For example, let us consider a relatively small mesh with 1,000 triangles, and a forest split operation which increases the number of triangles by 75%. The PFS scheme needs about 4 bits per new triangles to represent the connectivity changes while the PM scheme needs about $5 + 10 = 15$ bits per triangle or almost four times the number of bits required by the PFS scheme. For larger meshes the differences become more pronounced.

If no pre or post smoothing is used, the number of bits used to encode each vertex displacement is about the same as in PM. As discussed above, pre and post smoothing can be used to decrease the total size of the encoded vertex displacements by up to 20-25%. Because of the small granularity, it is not practical to apply pre smoothing before each vertex split operation in PM.

Complexity of encoding and decoding Since the two triangles incident to an edge defined by a pair of vertex indices can be accessed in constant time, the forest split operation is linear in the number of triangles added. We represent the edges as quadruplets, with two vertex indices and two face indices, and keep them in a hash table or array of linked lists. For typical surfaces this provides constant or almost constant access time.

5 CONVERSION TO PFS FORMAT

In this section we discuss simplification algorithms to convert a single-resolution triangular mesh to PFS format. We show that most edge-collapse based simplification algorithms can be modified to represent a simplification LOD hierarchy as a sequence of *forest collapse* operations.

Clustered multi-resolution models Several existing methods to generate multi-resolution polygonal models are based on *vertex clustering algorithms* [17]. In the multi-resolution polygonal model produced by these algorithms, the vertices of each level of detail are partitioned into disjoint subsets of vertices called *clusters*. All the vertices in each cluster are collapsed into a single vertex of the next (lower resolution) level of detail. Other examples of clustering algorithms for automatic surface simplification are based on triangle collapsing [10], and edge collapsing [12, 15, 6, 11].

For a clustering algorithm producing an LOD hierarchy with L levels, the vertices of consecutive levels are related by *clustering functions* $c_l : \{1, \ldots, V_l\} \rightarrow \{1, \ldots, V_{l+1}\}$ which map vertex indices of level l onto vertex indices of level l+1. Here, V_l denotes the number of vertices of the l-th. level. Triangles at each level of detail are completely specified by the triangles of the first (highest resolution) level of detail and the clustering functions. If $t_l = \{i, j, k\}$ is a triangle of the l-th. level, then $t_{l+1} = \{c_l(i), c_l(j), c_l(k)\}$ is a triangle of the next level if it is composed of three different indices; otherwise we say that t_l was *collapsed* at the next level.

Clustered multi-resolution polygonal models have the smooth transition property. Vertex i of the l-th. level is linearly interpolated with vertex $c_l(i)$ of the level as a function of time. A closely related, but more compact data structure, optimized for fast switching between levels of detail was recently introduced by Guéziec *et al.* [8] to represent clustered multi-resolution polygonal models.

The forest collapse operation The set of triangles of each level of detail that collapses in the next level can be partitioned into connected components, where two triangles are considered connected if they share an edge. A clustering function defines a forest collapse operation if the following two conditions are satisfied: 1) each connected component is a simple polygon (triangulated with no internal vertices); and 2) no vertex is shared by two or more connected components. If these two conditions are satisfied, the boundary edges of each connected component form a disconnected tree in the next level of detail. If the edges formed a graph with loops instead of a tree then the connected component would not be simply connected. Also, connected components with no common vertices always produce disconnected trees.

To test whether or not a clustering function defines a forest collapse operation, it is sufficient to: 1) check that the Euler numbers of the connected components are all equal to one ($V-E-T$=1), where the edges which separate the triangles of each component from other triangles are consdiered boundary edges; and 2) count the number of times that each vertex of the mesh belongs to the boundary of a connected component and then check that all these numbers are equal to 0 or 1.

Permutations of vertex and triangle indices If all the clustering functions of a clustered multi-resolution model define forest collapse operations, the transition from the lowest to the highest resolution level of detail can be represented as a sequence of forest split operations. The order of vertices and triangles in the highest resolution level of detail and those induced by the clustering functions in the lower resolution levels will, in general, not be the same as those produced by decompressing the low resolution mesh and applying the forest split operations. Special care must be taken to determine the permutations which put the vertices and faces of

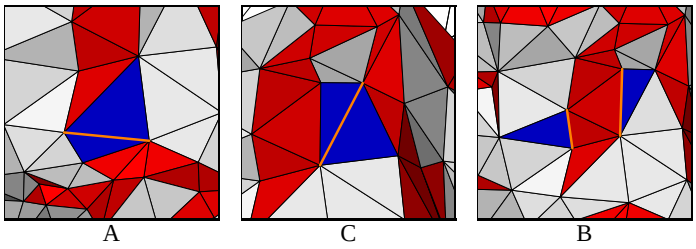

Figure 7: Collapsibility tests. A: Acceptable edge of multiplicity 1. B: Rejected edge of multiplicity 1. C: Rejected edge of multiplicity > 1.

different levels in the order that the decoder expects as described in Section 3.2.

Edge-collapse simplification algorithms In several simplification algorithms based on edge collapsing, the edges of the mesh are ordered in a priority queue according to certain criteria, usually based on the geometry of a neighborhood, and on the changes that the edge would produce if collapsed. The first edge is removed from the queue and certain connectivity and geometry tests are applied to it. If it passes the tests it is classified as *collapsible*, and the edge is collapsed. Otherwise, it is discarded. The collapsing of an edge may require changing the order of some of the neighboring edges in the queue. The process continues removing edges from the queue until the queue is exhausted, or until a termination condition, such as maximum number of collapsed triangles, is reached. To prevent visual artifacts the collapsibility test should take into account the values of properties bound to vertices, triangles, or corners, and the discontinuity curves that such property fields define, as in the PM simplification algorithm [11].

The initial mesh, before the simplification algorithm starts collapsing edges, defines one level of detail, and the resulting mesh, after the collapsing stops, defines the next level of detail. An extended collapsibility test must be performed to ensure that the clustering function defined by a proposed edge collapse also defines a valid forest collapse operation. This extended test requires two additional, simple tests to be performed on each candidate edge collapse. Figure 7 illustrates these tests. Both tests are straightforward.

The first test determines whether or not the connected components of the set of collapsed triangles are simple polygons after the edge is collapsed. The test can be implemented using an auxiliary data structure to maintain a forest in the dual graph of the mesh, with each tree of this forest corresponding to one connected component (simple polygon) constructed so far. When an edge collapse is accepted by the simplification algorithm, it not only identifies the two endpoints of the edge, but also the two remaining edges of each of the two (one for the case involving mesh boundary) collapsed triangles. This defines a *multiplicity* for each edge of the simplified mesh. Initially all edges in the current level of detail are assigned a multiplicity of one. When two edges are identified as the result of an edge collapse, they are removed from the priority queue, and a new edge with the sum of the two multiplicities is reinserted in the queue. Of the four (two for the case involving mesh boundary) boundary edges of the quadrilateral defined by the two (one) collapsed triangles, between 0 and 4 (2) edges are boundary edges of neighboring connected components. The test is passed if all of these neighboring connected components are different, in which case, adding the collapsed edge and the edges shared with the connected components to the dual forest would not destroy the forest structure. Otherwise, one or more loops would be created. Two triangles incident to an edge with multiplicity higher than one correspond to two mesh triangles which each share an edge with a boundary edge of the same connected component but are not neighbors in the original mesh. Either one or two of the edges of each one of these triangles in the original mesh are boundary edges of the neighboring connected component, otherwise the previous test would have been violated. The test is passed if only one edge is shared with the neighboring connected component.

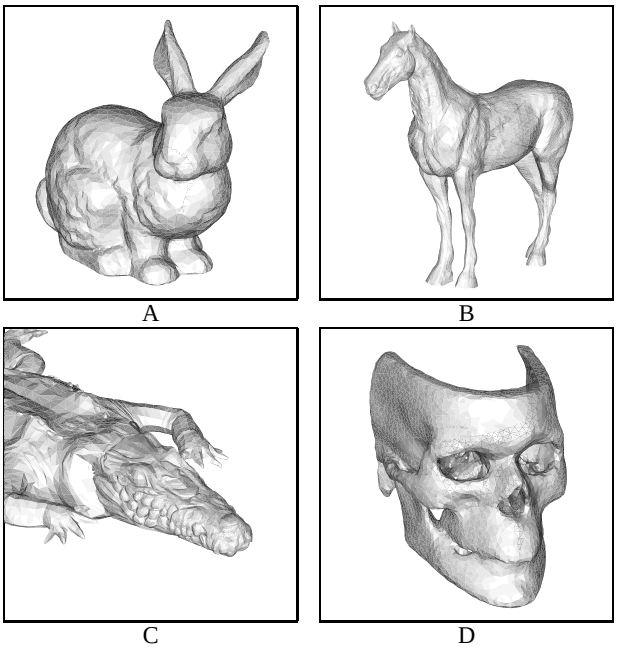

Figure 8: Highest resolution models used to converted to PFS format in Section 6. A: bunny. B: horse. C: crocodile. D: skull. All the models are flat-shaded. Properties are ignored.

The second test is applied only if the edge passes the first test. This test prevents vertices from being shared by two or more connected components and can be implemented using an auxiliary data structure to keep track of the vertices of the mesh which are boundary vertices of simple polygons constructed so far. An array of boolean variables, initialized to false, is sufficient for this purpose. We call a vertex of the quadrilateral defined by the collapse of an edge with multiplicity 1 *isolated*, if neither one of the two boundary edges of the quadrilateral shared by the vertex are shared by neighboring connected components. The same name is given to the two vertices of the two triangles incident to an edge with multiplicity higher than 1, which are not endpoints of the collapsed edge. The second test is passed if neither one of the isolated vertices are boundary vertices of any connected component. If an edge passes both tests then the edge is collapsed, the priority queue and the data structures needed to perform the two new tests are updated, and the process repeats until the queue is exhausted.

As explained in Section 4, the most triangles a forest split operation can add to a mesh with T triangles is approximately T. Therefore, when the simplification algorithm (with the two additional tests) stops because the queue has been exhausted, the resulting mesh cannot have less than half the triangles of the original mesh. In our experiments, a forest split operation adds between 45-90% T to a mesh of T triangles.

6 IMPLEMENTATION AND RESULTS

Our prototype implementation is composed of three programs: 1) a simplification program, 2) an encoding program, and 3) a decoding program. The current implementation handles manifold triangular meshes without properties.

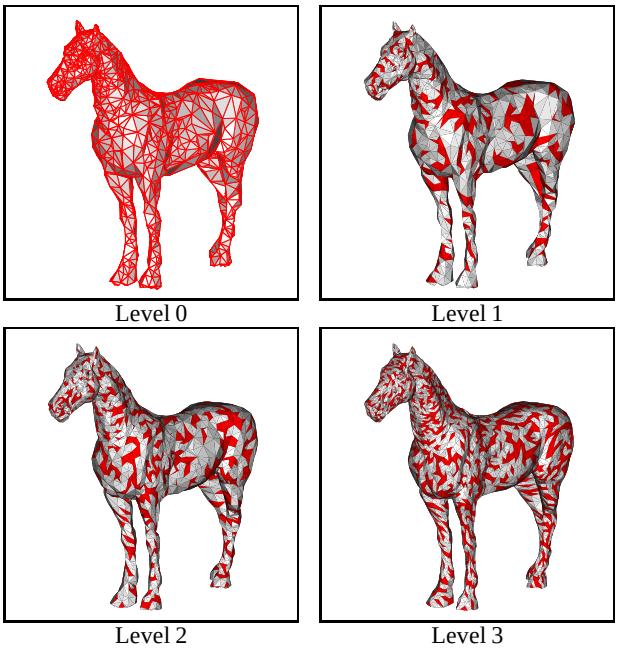

Figure 9: LOD hierarchy generated automatically from the horse of figure 8-B (Level 4). Red triangles collapse in the next level

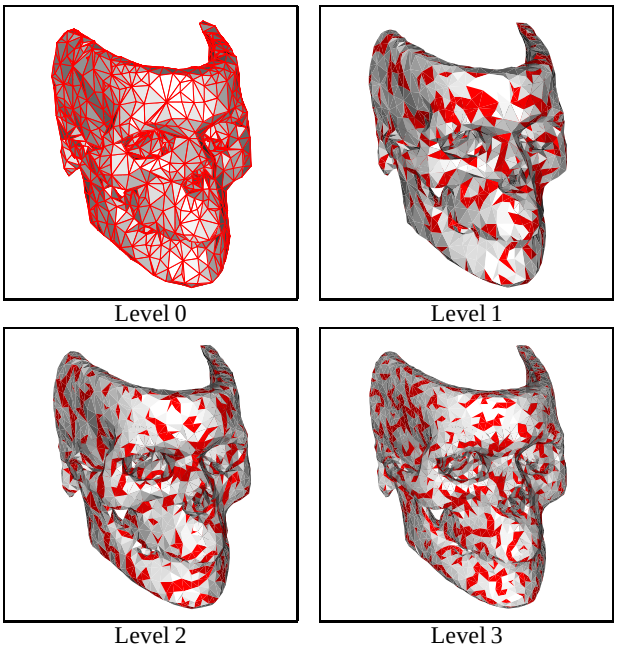

Figure 10: LOD hierarchy generated automatically from the skull of figure 8-D (Level 4). Red triangles collapse in the next level

For the simplification progam we implemented the variable tolerance simplification method of Guéziec [6] with the modifications described in Section 5. The combined procedure takes a single-resolution triangular mesh as input, and produces as output a file in *clustered multi-resolution* format. This is an ASCII file containing the connectivity of the original mesh, the vertices coordinates of all the levels produced, and the clustering functions represented as arrays of vertex indices. The encoder program takes this file format as input, checks whether the data is PFS compressible or not, and if so, produces a compressed data stream as described in Section 4. The decoder program takes this compressed data stream as input and produces a file in clustered multi-resolution format.

We report results for four models, each displayed in full resolution in figures 8-A,B,C,D. The models all have roughly the same number of vertices and triangles (except the bunny) and represent different topological types. The bunny has the topology of a sphere with four holes (in the bottom). The horse is composed of three connected components each with the topology of a sphere without boundary (body and eyes). The crocodile is composed of 65 connected components each with the topology of a sphere (body, mandibles, eyes and teeth) and with large variations in the number of vertices and triangles in each component. The skull is connected, has no boundary and a very high genus.

The simplification program produced four levels of detail for the bunny, and five levels of detail for each one of the other models. Figure 9 shows the LOD hierarchy generated from the horse. Figure 10 shows the LOD hierarchy generated from the skull. The number of vertices and triangles for each level, as well as the number of bytes needed to encode the base meshes and forest split operations are shown in the Figure 11. Overall, the single resolution scheme used to encode and compress the lowest resolution level of detail has better compression rates (bits per triangle) than the forest split operations. However, the forest split operation compression rates approach the single resolution compression rates as the size of the forests grow and the vertex displacements decrease in magnitude.

7 EXTENSIONS

To simplify the description of the representation, algorithms and data structures, and to reflect the limitations of the current implementation, we have limited the scope of this paper to manifold triangular meshes without properties. Some of these extensions are straightforward, others will require further work.

Polygonal faces Just as the TS method was extended to handle the simply connected polygonal faces found in VRML models [19], the PFS method can be extended to handle manifold polygonal models with simply connected polygonal faces. The representation and encoding of the forest of edges do not require any modification. Tree boundary loops would be polygonized instead of triangulated. Tree boundary loops can be polygonized by sending one extra bit per marching edge to indicate which of the internal edges of the triangulated simple polygon are actually face edges.

Properties The current implementation supports topology and vector coordinates. The TS method has been extended to handle properties (colors, normals and texture coordinates) and property bindings for faces and corners [19]. The same techniques can be used to extend the PFS method to handle additional properties and property bindings. Normals can be quantized in a special way, taking into account the unit length nature of normals vectors [19]. When properties are bound to corners, they define *discontinuity curves* [11, 19]. Encoding these discontinuity curves by simply requiring an extra bit per edge may be too expensive. A scheme to progressively update the discontinuity curves may be necessary. We intend to work on this issue in the near future.

Non-manifolds and topology changing refinement The topological type of the lowest resolution level of detail stays constant during the refinement process for bothe the PFS and PM methods. The Progressive Simplicial Complexes (PSC) scheme was introduced by Popovic and Hoppe [14], to allow changes in topological type (genus) to occur during the refinement process.

CROCODILE					(1.89)
Level	0	1	2	3	4
T	3,506	5,128	7,680	12,408	21,628
ΔT	3,506	1,622	2,552	4,728	9,220
ΔT (%)	∞	46	50	62	74
C/ΔT	3.42	5.29	5.04	4.45	4.02
(C+G)/ΔT	13.29	26.71	24.20	20.82	17.27
HORSE					(2.76)
T	2,894	4,306	6,774	11,754	22,258
ΔT	2,894	1,412	2,468	4,980	10,504
ΔT (%)	∞	49	57	74	90
C/ΔT	3.44	5.12	4.64	4.05	3.68
(C+G)/ΔT	13.74	25.04	25.04	19.16	17.91
SKULL					(1.48)
T	4,464	6,612	9,966	14,896	22,104
ΔT	4,464	2,148	3,354	4,930	7,208
ΔT (%)	∞	48	51	50	48
C/ΔT	4.51	5.15	4.97	5.04	5.10
(C+G)/ΔT	14.86	29.85	28.15	26.71	24.58
BUNNY					(1.73)
T	2,008	3,169	5,072	7,698	
ΔT	2,008	1,161	1,903	2,626	
ΔT (%)	∞	58	60	44	
C/ΔT	3.27	4.69	4.56	4.95	
(C+G)/ΔT	14.37	27.97	25.68	26.38	

Figure 11: Numerical results. T: number of triangles. ΔT: increment with respect to previous level. C/ΔT: bits per triangle for connectivity. (C+G)/ΔT: total number of bits per triangle. The number in parentheses is the relative cost of progressive vs. single resolution transmission of connectivity (ratio of connectivity bits per triangle in Progressive Forest Split / Topological Surgery).

The PSC representation retains most of the advantages of the PM representation, including smooth transitions between consecutive levels of detail and progressive transmission. However, the increased generality of the method requires a higher number of bits to specify each refinement operation.

We believe that a recursive subdivision approach related to the one taken in this paper, but with a more complex refinement operation, can solve the problem of dealing with non-manifolds and changes of topological type. We intend to work on this issue in the near future as well.

8 CONCLUSIONS

In this paper we introduced the PFS representation as a new adaptive refinement scheme for storing and transmitting triangular meshes in a progressive and highly compressed form. We started the paper by putting this new method and the PM method of Hoppe in the general context of recursive subdivision/refinement schemes. We showed that PFS allows the user to tradeoff granularity in refinement levels for complexity in the data stream. For a fine granularity (PM) $O(n \log_2(n))$ bits are required to double the connectivity size of a mesh with $O(n)$ levels of detail. For a coarse granularity (PFS) $O(n)$ bits are required to perform the same doubling for $O(1)$ levels of detail.

We have described algorithms for efficiently encoding to and decoding from a data stream containing the PFS format. We also showed how to reduce the number of bits required to encode the geometry the mesh by using pre and post global smoothing steps.

We showed how, with the addition of two simple tests, to modify any simplification algorithm based on edge collapses to convert single resolution triangular meshes to PFS format. We presented results obtained by applying these modifications to the Variable Tolerance method of Guéziec. Finally, we discussed how to extend the scheme to handle polygonal faces and various property bindings.

Acknowledgments

Our thanks to Rhythm & Hues Studios, Inc. and Greg Turk for the Horse model. Thanks to Marc Levoy for the Bunny model.

REFERENCES

[1] MPEG4/SNHC verification model 5.0, July 1997. ISO/IEC JTC1/-SC29/WG11 Document N1820, Caspar Horne (ed.).

[2] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10:350–355, 1978.

[3] M. Deering. Geometric compression. In *Siggraph'95 Conference Proceedings*, pages 13–20, August 1995.

[4] D. Doo and M. Sabin. Behaviour of recursive division surfaces near extraordinary points. *Computer Aided Design*, 10:356–360, 1978.

[5] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *Siggraph'95 Conference Proceedings*, pages 173–182, August 1995.

[6] A. Guéziec. Surface simplification with variable tolerance. In *Second Annual International Symposium on Medical Robotics and Computer Assisted Surgery*, pages 132–139, Baltimore, MD, November 1995.

[7] A. Guéziec, G. Taubin, F. Lazarus, and W. Horn. Cutting and Stitching: Efficient Conversion of a Non-Manifold Polygonal Surface to a Manifold. Technical Report RC-20935, IBM Research, July 1997.

[8] A. Guéziec, G. Taubin, F. Lazarus, and W. Horn. Simplicial maps for progressive transmission of polygonal surfaces. In *VRML 98*. ACM, February 1998.

[9] M. Hall and J. Warren. Adaptive polygonalization of implicitly defined surfaces. *IEEE Computer Graphics and Applications*, pages 33–42, November 1990.

[10] B. Hamann. A data reduction scheme for triangulated surfaces. *Computer Aided Geometric Design*, 11(2):197–214, 1994.

[11] H. Hoppe. Progressive meshes. In *Siggraph'96 Conference Proceedings*, pages 99–108, August 1996.

[12] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *Siggraph'93 Conference Proceedings*, pages 19–25, July 1993.

[13] C. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, Dept. of Mathematics, University of Utah, August 1987.

[14] J. Popović and H. Hoppe. Progressive simplicial complexes. In *Siggraph'97 Conference Proceedings*, pages 217–224, August 1997.

[15] R. Ronfard and J. Rossignac. Simplifying a triangular mesh with multiple planar constraints. Technical report, IBM Research, 1994.

[16] R. Ronfard and J. Rossignac. Triangulating multiply-connected polygons: A simple, yet efficient algorithm. *Computer Graphics Forum*, 13(3):C281–C292, 1994. Proc. Eurographics'94, Oslo, Norway.

[17] J. Rossignac and P. Borrel. *Geometric Modeling in Computer Graphics*, chapter Multi-resolution 3D approximations for rendering complex scenes, pages 455–465. Springer Verlag, 1993.

[18] G. Taubin. A signal processing approach to fair surface design. In *Siggraph'95 Conference Proceedings*, pages 351–358, August 1995.

[19] G. Taubin, W.P. Horn, F. Lazarus, and J. Rossignac. Geometric Coding and VRML. *Proceedings of the IEEE*, July 1998. (to appear) Also IBM Research TR RC-20925, July 1997.

[20] G. Taubin and J. Rossignac. Geometry Compression through Topological Surgery. *ACM Transactions on Graphics*, April 1998. (to appear).

[21] D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. In *Siggraph'97 Conference Proceedings*, pages 259–268, August 1997.