Implant Sprays: Compression of Progressive Tetrahedral Mesh Connectivity

Renato Pajarola, Jarek Rossignac and Andrzej Szymczak

Implant Sprays: Compression of Progressive Tetrahedral Mesh Connectivity

Renato Pajarola, Jarek Rossignac, Andrzej Szymczak
Graphics, Visualization & Usability Center
Georgia Institute of Technology

8 May 1999 10:30 PM

Abstract

Irregular tetrahedral meshes, which are popular in many engineering and scientific applications, often contain a large number of vertices. A mesh of V vertices and T tetrahedra requires 48·V bits or less to store the vertex coordinates, $4 \cdot T \cdot \log_2(V)$ bits to store the tetrahedra-vertex incidence relations, also called connectivity information, and k·V bits to store the k-bit value samples associated with the vertices. Given that T is 5 to 7 times larger than V and that V often exceeds 32^3, the storage space required for the connectivity is larger than 300·V bits and thus dominates the overall storage cost. Our "implants spray" compression approach introduced in this paper reduces this cost to about 30·V bits or less – a 10:1 compression ratio. Furthermore, implant spray supports the progressive refinement of a crude model through a series of vertex-splits operations.

1. Introduction

A large portion of finite element meshes, and scientific and engineering analysis results are expressed in terms of sample points distributed through 3D space, associated scalar values, and connectivity information which defines how the sampled scalar values are to be interpolated. The simplest and most commonly used interpolation is based on a decomposition of the 3D space into a tetrahedral mesh, which may be defined by a tetrahedra-vertices incidence table that takes $4T \cdot \lceil \log_2(V) \rceil$ bits for a mesh with V vertices and T tetrahedra. In this paper we propose a compression technique which reduces this storage cost to about 5·T bits.

Our compressed format, called *implant sprays*, describes a coarse mesh and a series of implants, refinement operations. Each implant inserts a new vertex and a series of incident tetrahedra. The *implants* are simple extensions of the vertex split operation introduced in [HRD+93] that are applied to tetrahedral meshes [SG98]. The implant is defined by the selection of an existing vertex *v*, called the split-vertex, a displacement vector defining the location of a new vertex, and a cycle of triangle-faces incident upon the open vertex *v*, called the *skirt*. We describe here an encoding of the selection of split-vertices that requires less than 3 bits per tetrahedron. Furthermore, we propose an encoding of the skirt that requires less than 4 bits per tetrahedron. Combined, these two techniques reduce the connectivity costs per tetrahedron from the initial $4 \cdot \lceil \log_2(V) \rceil$ bits to less than six bits. Even for modest size meshes, we obtain a compression ratio better than 10:1. These savings are important, because otherwise the connectivity data dominates the storage costs. Indeed, there are roughly 6 times as many tetrahedra as vertices – this number varies with the structure of the mesh. Even for a small mesh of 32^3 vertices the uncompressed connectivity cost is 360·*V* bits (i.e. 4·6·15·*V*). In comparison, vertex location may be represented with only 48·*V* bits using 16-bit integer coordinates in an optimally chosen coordinate system [Dee95].

The rest of the paper is organized as follows: Section 2 provides an overview of related work on triangular and tetrahedral meshes, in Section 3 the basic simplification and refinement operations are described, Section 4 presents detailed explanation of the implant sprays encoding and Section 5 pro-

vides an implementation framework, experiments are reported in Section 6, and Section 7 concludes the paper and gives an outlook over future work.

2. Related work

Recently, a lot of work has been done in short encodings of the connectivity of triangular meshes. With few exceptions, i.e. *Progressive Forest Split Compression* (PFS) [TGHL98] and *Compressed Progressive Meshes* (CPM) [PR99a], most methods only work for single-resolution meshes. Very successful approaches for triangular mesh encoding are the *Topological Surgery* method [TR98], *Edgebreaker* [Ros98] and the triangle mesh compression presented in [TG98]. A comprehensive overview can also be found in [Ros98] or [TR98b]. These single-resolution mesh compression algorithms are able to encode the connectivity of a triangular mesh with less than 2 bits per triangle. The multiresolution mesh compression methods PFS and CPM achieve a connectivity encoding of less than 5 (PFS) or less than 4 bits (CPM) per triangle while providing a progressive triangulation with several levels of detail of increasing approximation accuracy.

Much less work has been performed on compressing the connectivity of tetrahedral meshes, even though in the tetrahedral case the incidence information dominates the geometry data, the 3D coordinates of vertices. Only recently the *Grow & Fold* method was presented in [SR99] that encodes the connectivity of tetrahedral meshes with roughly 7 bits per tetrahedron. However, as most of the triangle mesh compression methods, this approach encodes one single-resolution tetrahedralization. Thus it provides one level of detail only. Progressive multiresolution tetrahedralizations [SG98, THJW98] have only recently been presented. However, no concise encoding of the refinement operations has been provided. Each refinement operation of the progressive tetrahedralization in [SG98] needs to specify one vertex that will be split and 5 to 6 incident triangular faces that will be cut. The split-vertex can be identified using $\lceil \log_2(V) \rceil$ bits in a tetrahedral mesh with V vertices. The cut-faces can be encoded locally with respect to the split-vertex. Because a vertex in a tetrahedral mesh has about 36 incident faces, the cut-faces can be encoded with roughly $6 \cdot \lceil \log_2(36) \rceil$ bits. A similar coding scheme can also be applied to the multiresolution tetrahedralization of [THJW98].

3. Progressive meshes

In [HRD$^+$93] the *edge collapse* operation, and its inverse the *vertex split*, was introduced for triangular mesh simplification, see also Figure 1 for an example. *Progressive Meshes* [Hop96] apply a sequence of edge collapse operations $\mathcal{M}_i \rightarrow \mathcal{M}_{i-1}$ to a given triangular, high resolution input mesh $\mathcal{M}_{l\max}$ to create a series of simplified meshes $\mathcal{M}_{l\max}, \mathcal{M}_{l\max-1}, \ldots, \mathcal{M}_i, \mathcal{M}_{i-1}, \ldots, \mathcal{M}_1, \mathcal{M}_0$ with decreasing approximation accuracy. The meshes $\mathcal{M}_i$ ($0 < i \leq l_{\max}$) can be reconstructed by performing the inverse sequence of vertex splits $\mathcal{M}_{i-1} \rightarrow \mathcal{M}_i$ starting with a crude base mesh $\mathcal{M}_0$.
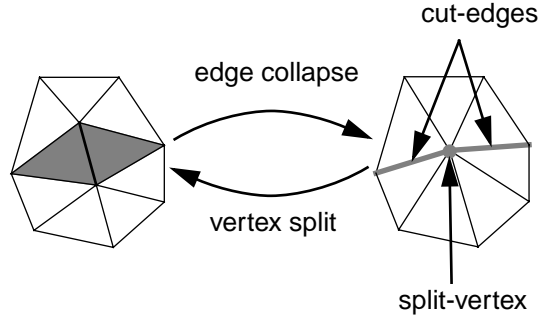
FIGURE 1. Edge collapse and vertex split for triangle mesh simplification and reconstruction.

The same basic principles of edge collapses and vertex splits can be extended and applied to more complex meshes such as *simplicial complexes* [PH97] and *tetrahedral meshes* [SG98]. In tetrahedral meshes, a collapse of an edge eliminates all tetrahedra incident to that edge and reduces the number of vertices in the mesh by one, see also Figure 2 for a graphical example. Again, a sequence of edge collapses and its inverse, the vertex splits, define a *progressive tetrahedralization*, as proposed in [Hop96, SG98], as a series of tetrahedral meshes $\mathcal{T}'_0$, …, $\mathcal{T}'_{l\max}$ of increasing precision. In the remainder of the paper $\mathcal{T}'$ will refer to a tetrahedral mesh if not specified otherwise.
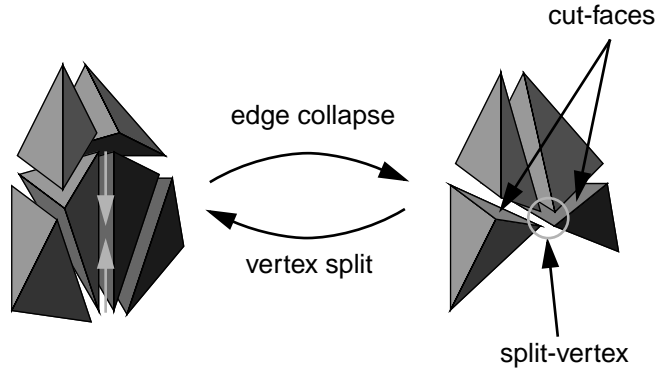

FIGURE 2. Edge collapse and vertex split for tetrahedral mesh simplification and reconstruction.

4. Progressive encoding

As mentioned above, a progressive tetrahedralization is defined by a crude base mesh $\mathcal{T}'_0$ and a sequence of vertex split refinement operations. Given an intermediate mesh $\mathcal{T}'_i$, an individual refinement operation is fully specified by the identification of the split-vertex and the set of incident cut-faces, both given in $\mathcal{T}'_i$. After locating the split-vertex v, identifying the cut-faces is a local process on the neighborhood of v in mesh $\mathcal{T}'_i$. The following Section 4.1 describes our new method of locally encoding the set of cut-faces for an individual vertex split operation. Next, the progressive encoding of the split-vertex locations is described in Section 4.2.

Note that our algorithms presume the existance of a canonical ordering and numbering of the vertices of any tetrahedral mesh $\mathcal{T}'_i$.. Such an ordering can arbitrarily be specified, i.e. sorted by coordinates, or given by a mesh traversal, i.e. depth-first vertex tree traversal.

4.1 Cut-faces

The cut-faces around a split-vertex, also called the *skirt*, define how the incidence relations have to be modified for a vertex split operation. To better understand the coding of the skirt, let us define the *orbital surface* of a split-vertex v as the triangular surface consisting of all faces of tetrahedra incident to v that are not themselves incident faces of v, see also Figure 3. The skirt forms a connected path, a cycle, on the triangulated orbital surface, see also Figure 4. Note that the number of different cycles of length k, also called k-cycles, without repetition of edges or nodes, on a triangular planar graph G with d vertices is much smaller than the number of all subsets of edges of G of size k, also called k-sets, because the k-cycles are a subset of the k-sets. Therefore, the corresponding method to the triangular approach in [PR99a] of encoding the skirt as one particular k-set out of all $\binom{3d}{k}$[1] possible ones, is not optimal in a progressive tetrahedralization.
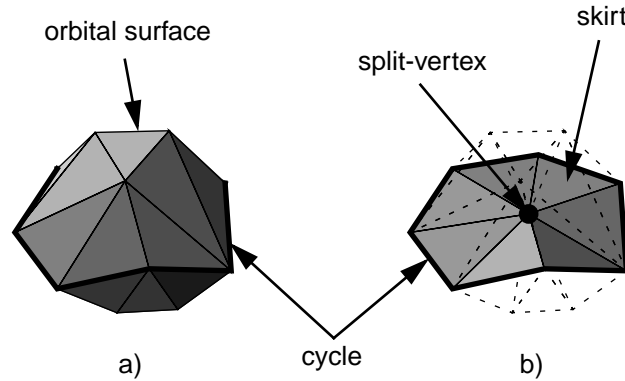

FIGURE 3. a) The *orbital* of a split-vertex v consists of all faces of tetrahedra surrounding v that are not incident to v. b) The cut-faces form a ring of triangles, also called the skirt.

An optimized encoding of the skirt could be achieved by identifying the particular k-cycle of a vertex split operation out of all possible k-cycles on the orbital surface. Given a canonical numbering of the vertices, an enumeration of all k-cycles on the orbital surface, the triangular planar graph G_{orb} (Figure 4), can be achieved by a recursive backtracking algorithm. For every vertex, starting with the youngest one, initiate a depth-first vertex traversal of G_{orb}, again youngest first, for finding paths of length k. Backtracking occurs when a path is longer than k, a k-path is not a cycle, or a node or edge of G_{orb} is used twice in the current path. Moreover, backtracking also occurs when a younger vertex than the start-vertex is found on the currently explored path because k-cycles are only reported for their youngest vertex. Note that otherwise the same k-cycle would be reported several times, for each vertex in the cycle. A correct path is found when it is of length k and the end-vertex is equal to the start-vertex of the initiated traversal. For this method of encoding the skirt, the number k of cut-faces has to be known to the decoder in advance.

---

1. A planar triangulation has roughly three times as many edges as vertices. Corollary, a vertex in a 3D tetrahedralization has roughly three times as many incident faces as incident edges.
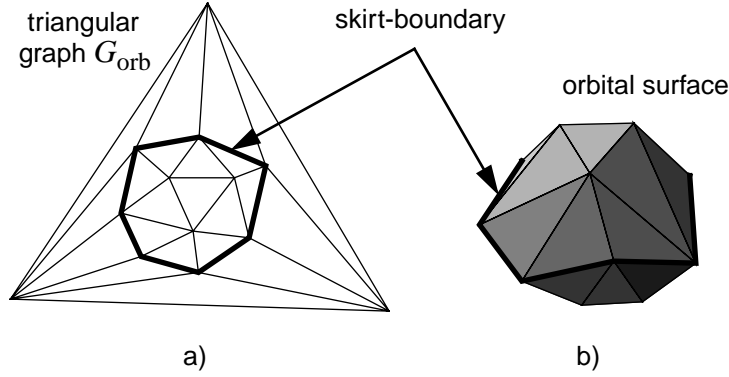
FIGURE 4. a) Shows a planar embedding of the triangular graph G_{orb} of the orbital surface b). The boundary of the cut-faces, forming a cycle in the planar graph a), is highlighted using thick lines.

Although providing an optimized encoding in terms of space cost (bits), the previously described approach of encoding the skirt is inefficient in time cost. Because the enumeration of all possible k-cycles for every split-vertex is quite time consuming we implemented a faster encoding of the skirt. The path on the graph G_{orb} which defines the skirt, see Figure 4 a), can be encoded as a walk along edges of G_{orb}. The start of the path, a vertex of G_{orb}, is encoded using $\lceil \log_2(d) \rceil$ bits for a split-vertex of degree d. Since the average degree of split-vertices in a progressive tetrahedralization is more than 14 – thus also the number of vertices of G_{orb} – the start vertex encoding requires roughly 4 bits on average. The skirt can then be specified by a path of length k, forming the k-cycle, as a set of consecutive edges on G_{orb}. Each edge can be specified using less than $\lceil \log_2(6) \rceil = 3$ bits since the degree of vertices in a planar triangulation is 6 on average. Thus the cut-faces encoded as a walk along edges of G_{orb} can be expected to cost about $\log_2(14) + 6 \cdot \log_2(6) \approx 19$ bits.

Note that in general the orbital surface could be non-manifold due to edge collapse operations. In the following section we describe the constraints that prevent non-manifold orbital surfaces.

4.2 Split-vertex

Instead of refining the current mesh $\mathcal{T}_i'$ by one single vertex split at a time, we perform a series of vertex splits simultaneously, also called *implants sprays*, to achieve the next refined *level of detail* (LOD) $\mathcal{T}_{i+1}'$. Identifying one isolated split-vertex in $\mathcal{T}_i'$ would require $\log_2(V_i)$ bits in a mesh with V_i vertices, which is very costly for large meshes. However, identifying a set of V_i / k independent split-vertices by a flag, marking all vertices in $\mathcal{T}_i'$ with one bit only, amounts to a constant of k bits per split-vertex. The decoding process just needs to visit all vertices in $\mathcal{T}_i'$ in the same order as the encoder, and read the respective marking bits from the data stream to identify the set of vertices that are to be split.

To optimize coding efficiency one would like to maximize the fraction V_i / k of independent vertex splits that form a refinement step $\mathcal{T}_i' \rightarrow \mathcal{T}_{i+1}'$. In a planar triangulation k can be guaranteed to be less than 4 by the vertex coloring theorem, and experiments show an average of about 3 [PR99a] for independent vertex splits. In a tetrahedral mesh k cannot be bounded easily. Vertex coloring of non-planar graphs depends on the maximal degree of incident edges on a vertex, which is not bounded in a tetrahedral mesh. Larger independent sets than induced by the vertex coloring can be constructed, however, the maximization of k is limited by the simplification process. During the construction of the progressive mesh, the choice of independent edge collapses in $\mathcal{T}_{i+1}'$ is restricted. To be able to distinguish the individual vertex splits in $\mathcal{T}_i'$ without ambiguities, the following three requirements for edge collapses in $\mathcal{T}_{i+1}'$ are sufficient:

1. The two sets of tetrahedra in $\mathcal{T}'_{i+1}$ intersecting two edges to be collapsed in one implant sprays simplification batch are disjoint. (Figure 5)
2. For each edge $e = (v_1, v_2)$ that will be collapsed and a vertex w that is incident to both v_1 and v_2, the triple (v_1, v_2, w) must define a valid face of $\mathcal{T}'_{i+1}$. (Figure 6 a)
3. For each edge $e = (v_1, v_2)$ that will be collapsed and two vertices w_1, w_2 such that the triangles (v_1, w_1, w_2) and (v_2, w_1, w_2) are faces of $\mathcal{T}'_{i+1}$, the quadruple (v_1, v_2, w_1, w_2) must define a valid tetrahedron of $\mathcal{T}'_{i+1}$. (Figure 6 b)
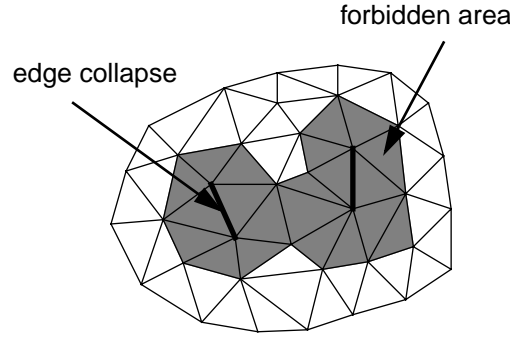

FIGURE 5. Independent edge collapses may not have common incident tetrahedra. The figure shows an analogous case in a 2D triangulation with two selected edge collapses. All faces of the forbidden area must not be incident to another edge collapses.
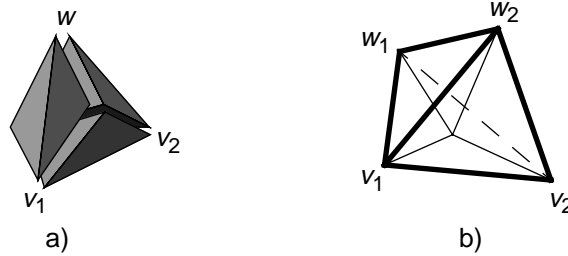

FIGURE 6. Examples of non-valid edge collapses, where a) fails test number 2 and b) fails test number 3 of the constraints mentioned above.

Thus during progressive mesh construction, the simplification process must select sets of edge collapses of maximal size that form a simplification step $\mathcal{T}'_{i+1} \rightarrow \mathcal{T}'_i$ according to the requiremenst mentioned above. In our current implementation independent edge collapses are selected greedily, see also Algorithm 2 in Section 5. Our experiments suggest that in practical situations k is in the range of 12 to 16 for the very restrictive selection of edge collapses mentioned above. A less restrictive selection of edge collapses would allow an even larger fraction V_i / k of split-vertices. The average number of nearly 6 removed tetrahedra per edge collapse results in a split-vertex encoding with less than 3 bits per tetrahedron.

Additional geometric properties and constraints of edge collapses assure that the simplified tetrahedral meshes are good approximations of the initial tetrahedralization. However, these constraints are not the subject of this paper and the interested reader is referred to [SG98] for further details.

5. Implementation

The main problem of an actual implementation is the construction of the sequence of increasingly simplified meshes $\mathcal{T}'_{l\max}, \dots, \mathcal{T}'_0$, or levels of detail (LODs), such that each set of edge collapses C_i that forms one simplification step $\mathcal{T}'_i \rightarrow \mathcal{T}'_{i-1}$ can unambiguously be encoded using the techniques

described in the previous section. The simplification and encoding procedure EncodeOneStep() is iteratively called by the main encoding loop of Algorithm 1 for the current tetrahedral mesh starting with the best LOD $\mathcal{T}'_{l_{max}}$. After arriving at a sufficiently simplified and small mesh $\mathcal{T}'_0$, a simple single-resolution encoding can be used for this base mesh. The encoding of the crude mesh $\mathcal{T}'_0$ followed by popping the codes from the stack obtained by the EncodeOneStep() procedure calls builds the input data stream for the decoding procedure.

```
PROCEDURE Encode (mesh: Tetrahedralization);
VAR code: Stack; data: OutputStream;
BEGIN
   code.initStack();
   WHILE mesh not simplified enough DO
      mesh := EncodeOneStep(mesh, code)   (* encode simplification steps *)
   END;
   data.output(mesh.simpleEncoding());   (* encode base mesh T0 *)
   WHILE code.notEmpty() DO
      data.output(code.pop())
   END
END Encode;
```

ALGORITHM 1. Pseudocode for the main simplification and encoding algorithm.

The procedure EncodeOneStep(), performs one simplification step $\mathcal{T}'_i \rightarrow \mathcal{T}'_{i-1}$ at a time and provides the respective encoding of the vertex splits on a stack. The first foreach-loop in Algorithm 2 over all edges of a mesh $\mathcal{T}'_i$ can be ordered according to an increasing simplification error of edge collapses as used in [SG98]. Validation of edge collapses is performed by the validCollapse() procedure according to the constraints described in the previous section. Next, the method mesh.collapseEdges() performs the actual simplifications, and stores the vertex split information for every edge collapse with its respective vertex. The following foreach-loop traverses the vertices of the simplified mesh $\mathcal{T}'_{i-1}$ in the inverse order of the decoding process, and marks all vertices with one bit. For the marked split-vertices the associated cut-faces code is given as well. All codes are pushed on a code stack such that popping the codes from that stack provides the correct sequence for the decoding process and its traversal of vertices.

```
PROCEDURE EncodeOneStep (mesh: Tetrahedralization; VAR code: Stack)
   : Tetrahedralization;
VAR v: Vertex; e: Edge; ecol: EdgeSet;
BEGIN
   ecol = EmptySet;
   FOREACH e IN mesh.edges()
      IF validCollapse(e, mesh, ecol) THEN
         ecol.insert(e)
      ENDIF
   END;
   mesh.collapseEdges(ecol);
   FOREACH v IN mesh.inversedVertexTraversal()
      IF v.isCollapsedEdge() THEN
         code.pushBit(1);
         code.pushCode(v.cutFacesCode())
      ELSE
         code.pushBit(0)
      ENDIF
   END;
   RETURN mesh
END EncodeOneStep;
```

ALGORITHM 2. Pseudocode for one step of simplification and encoding of vertex splits.

Decompression involves first decoding the crude mesh $\mathcal{T}_0'$ according to the chosen single-resolution encoding method, and then traversing the vertices of the current LOD and simultaneously reading marking bits and vertex split information from the input data stream to create the implants sprays refinement updates. In Algorithm 3 the method mesh.simpleDecoding() performs the task of reading and constructing the base LOD $\mathcal{T}_0'$. The following while- and nested foreach-loops repeatedly traverse the vertices of the current mesh. While traversing the vertices of $\mathcal{T}_i'$, the marking bits indicate the occurrences of vertex split refinement operations.If a marking bit signifies a split-vertex the respective vertex split information, the encoding of the cut-faces, is read from the input data stream by the readCutFacesCode() method in Algorithm 3. The individual vertex splits can emmediately be performed to refine the current mesh. However, note that the so newly created vertices should not affect the current vertex traversal order of $\mathcal{T}_i'$, and that these new vertices should only be included in the vertex traversal of the next LOD $\mathcal{T}_{i+1}'$.

```
PROCEDURE Decode (data: InputStream): Tetrahedralization;
VAR mesh: Tetrahedralization;
BEGIN
  mesh.simpleDecoding(data);              (* decode base mesh T0 *)
  WHILE data.notEndOfStream() DO
    FOREACH v IN mesh.vertexTraversal()(* decode refinement steps *)
      IF data.inputBit() = 1 THEN
        v.readCutFacesCode(data);
        mesh.splitVertex(v);
      ENDIF
    END
  END;
  RETURN mesh
END Decode;
```

ALGORITHM 3. Pseudocode for the main decoding algorithm.

The marking of vertex splits is already shown in above algorithms, the cut-faces encoding depends heavily on the data structure used for maintaining a tetrahedral mesh and is described in Section 4.

6. Results

We have implemented the simplification and encoding algorithms mentioned above in a prototype system. Even though the encoding of enumerated cycles performed well in terms of coding costs, the current implementation was unreasonably slow. Therefore, we have only included results for encoding the cut-faces as walks on a planar triangular mesh. Note that the results presented here do not include an encoding of the base mesh $\mathcal{T}_0'$.

Table 1 presents results for a real world data set, the turbine, and a Delaunay tetrahedralization [EM94] of a random point set. For the turbine data set an average of $k = 5.77$ tetrahedra were removed with every edge collapse in 49 simplification steps, whereas the random tetrahedralization reported 6.25 tetrahedra per edge collapse in 54 LODs. Overall, the connectivity of the tetrahedral

meshes was encoded with less than 6 bits per tetrahedron which includes a large number of different LODs that are progressively available during decompression time.

data set	LODs	vertices $\mathcal{T}'_{l\max}$	vertices $\mathcal{T}'_0$	tetrahedra $\mathcal{T}'_{l\max}$	tetrahedra $\mathcal{T}'_0$	edge collapses	start vertex	loop	split-vertex	total bits	bits per tetrahedron
turbine	49	106795	24550	576576	101619	82245	4.14	14.5	10.35	2384643	5.020755
random	54	10000	928	66487	9799	9072	4.19	15.69	16.2	327386	5.775230

TABLE 1. Encoding results for the turbine data set and a random tetrahedralization. The start vertices have been encoded with roughly 4 bits, and the cut-faces using about 15 bits for every edge collapse. The marking of all vertices in each LOD resulted in an amortized cost of 10.35 bits per edge collapse for specifying the corresponding split-vertex.

Figure 7 shows the turbine volume data set that was used in our experiments. The volume data consists of solid turbine blades (Figure 7 a) and a large number of sampled data points in between the blades. All vertices of the surface of the turbine blades (Figure 7 b) and the data points are represented in one large tetrahedral mesh (Figure 7 c). For the very common approach of storing the turbine mesh as a set of indexed tetrahedra, the connectivity cost requires 13MB in ASCII and still 4.7MB using a simple binary encoding. The *Grow & Fold* method [SR99] cuts this cost down to 492KB for a single-resolution representation, whereas the method presented in this paper only requires 87KB for representing the base mesh $\mathcal{T}'^1_0$ and 291KB for the 49 refinement steps.

a)   b)   c)
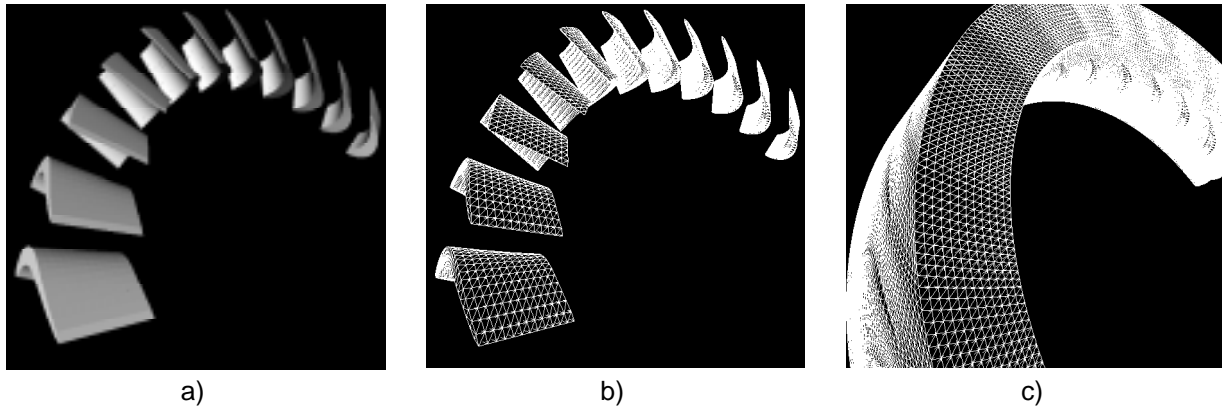
FIGURE 7. Pictures a) and b) only show the solid turbine blades parts of the tetrahedralized volume data set in c) which was used in our experiments. Note that the rim where the blades are mounted on is not shown in images a) and b). (Data set courtesy of AVS Inc.)

7. Conclusion and future work

The presented *implant sprays* method presented in this paper is a simplification, storage and transmission technique for tetrahedral meshes. Implant sprays provides a progressive tetrahedralization, starting from a crude base model, at significantly lower storage or transmission costs than previously known methods. It even improves connectivity encoding on the best known single-resolution tetrahedral mesh compression method [Symzcak]. The low coding cost is achieved by grouping vertex splits into batches, called the implant sprays, and by a concise encoding of the skirt, the cut-faces of a split-vertex.

---

1. i.e. using the grow and fold single-resolution encoding

Combining the implant sprays simplification procedure with an edge collapse error measure for tetrahedral meshes [SG98] provides an efficient mesh encoding of a progressive multiresolution tetrahedralization. Compressed multiresolution tetrahedral meshes can be used for cooperative scientific visualization, fast exploration of volumetric data sets, or provide LODs

Future work includes efforts to reduce the edge collapse selection constraints to increase the vertex split rate, thus lowering the split-vertex coding costs, while still guaranteeing locally unambiguous mesh updates. Furthermore, we will also investigate faster k-cycle enumeration and coding techniques to improve on the cut-faces encoding. However, the optimization in connectivity cost is far less important than inventing an efficient coordinate geometry compression method for tetrahedral meshes.

Because the connectivity cost of a tetrahedral mesh is less than 6 bits per tetrahedron using implant sprays, it is no longer the dominant cost factor. In contrast, uncompressed floating point coordinates now take up most of the storage space with at least 96 bits per vertex, or 18 bits per tetrahedron. However, the progressive encoding of the topological mesh connectivity allows for efficient coordinate compression as it is already exploited in triangular meshes. Based on the local mesh connectivity and geometry around a split-vertex, the displacement vector can be encoded using predictive error compression techniques [Kou95].

Acknowledgments

References

[Dee95] Michael Deering. Geometry compression. In *Proceedings SIGGRAPH 95*, pages 13–20. ACM SIGGRAPH, 1995.

[EM94] H. Edelsbrunner and E. P. Mucke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, 1994.

[GHJ+98] Tran S. Gieng, Bernd Hamann, Kenneth I. Joy, Gregory L. Schussman and Issac J. Trotts. Constructing Hierarchies for Triangle Meshes. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):145–161, April-June 1998.

[HRD+93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *Proceedings SIGGRAPH 93*, pages 19–26. ACM SIGGRAPH, 1993.

[Hop96] Hugues Hoppe. Progressive meshes. In *Proceedings SIGGRAPH 96*, pages 99–108. ACM SIGGRAPH, 1996.

[Kou95] Weidong Kou. *Digital Image Compression: Algorithms and Standards*. Kluwer Academic Publishers, Norwell, Massachusetts, 1995.

[PR99a] Renato Pajarola and Jarek Rossignac. Compressed progressive meshes. Technical Report GIT-GVU-99-05, GVU Center, Georgia Institute of Technology, 1999.

[PH97] Jovan Popovic and Hugues Hoppe. Progressive simplicial complexes. In *Proceedings SIGGRAPH 97*, pages 217–224. ACM SIGGRAPH, 1997.

[Ros98] Jarek Rossignac. Edgebreaker: Compressing the incidence graph of triangle meshes. Technical Report GIT-GVU-98-17, http://www.cc.gatech.edu/gvu/reports/1998, GVU Center, Georgia Institute of Technology, Atlanta, GA, 1998. (to appear in IEEE Transactions on Visualization and Computer Graphics)

[SG98] Oliver G. Staadt and Markus H. Gross. Progressive tetrahedralizations. In *Proceedings Visualization 98*, pages 397–402. IEEE, Computer Society Press, Los Alamitos, California, 1998.

References

[TGHL98] Gabriel Taubin, André Guéziec, William Horn and Francis Lazarus. Progressive forest split compression. In *Proceedings SIGGRAPH 98*, pages 123–132. ACM SIGGRAPH, 1998.

[TR98] Gabriel Taubin and Jarek Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17(2):84–115, 1998.

[TR98b] Gabriel Taubin and Jarek Rossignac. 3D geometric compression. In *Siggraph 98 Course Notes 21*. ACM SIGGRAPH, 1998.

[TG98] Costa Touma and Craig Gotsman. Triangle Mesh Compression. In *Proceedings Graphics Interface 98*, pages 26–34, 1998.

[THJW98] Issac J. Trotts, Bernd Hamann, Kenneth I. Joy and David F. Wiley. Simplification of tetrahedral meshes. In *Proceedings Visualization 98*, pages 287–295. IEEE, Computer Society Press, Los Alamitos, California, 1998.

[SR99] Andrzej Szymczak and Jarek Rossignac. Grow & Fold: Compression of tetrahedral meshes. to appear in Solid Modeling, 1999.