Tetrahedral Mesh Compression with the Cut-Border Machine

Stefan Gumhold[*], Stefan Guthe, Wolfgang Straßer

WSI/GRIS University of Tübingen

Abstract

In recent years, substantial progress has been achieved in the area of volume visualization on irregular grids, which is mainly based on tetrahedral meshes. Even moderately fine tetrahedral meshes consume several mega-bytes of storage. For archivation and transmission compression algorithms are essential. In scientific applications lossless compression schemes are of primary interest. This paper introduces a new lossless compression scheme for the connectivity of tetrahedral meshes. Our technique can handle all tetrahedral meshes in three dimensional euclidean space even with non manifold border. We present compression and decompression algorithms which consume for reasonable meshes linear time in the number of tetrahedra. The connectivity is compressed to less than 2.4 bits per tetrahedron for all measured meshes. Thus a tetrahedral mesh can almost be reduced to the vertex coordinates, which consume in a common representation about one quarter of the total storage space.

We complete our work with solutions for the compression of vertex coordinates and additional attributes, which might be attached to the mesh.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—object representations I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types E.4 [E.4 Coding and Information Theory]: Data compaction and compression

Keywords: compression algorithms, solid modeling, scientific visualization, volume rendering

1 Introduction

Tetrahedral meshes have been around in finite element simulations on volumetric domains for a long time. With the growing need of visualizing simulation data, tetrahedral meshes established themselves also in volume visualization. There are several beautiful properties of tetrahedral meshes which make them the natural choice for volume data representation. The flexibility of a tetrahedral mesh is ideally suited for irregular samplings and multiresolution analysis. The convex nature of a single tetrahedron allows for a simple visibility sorting algorithm[12], which is essential in volume visualization.

In most application areas of tetrahedral meshes some data is attached to the mesh elements. The data can be attached to the vertices, edges, the faces, the border faces or the tetrahedra. A density might be attached to the vertices, the intensity of a flow to the edges or material identifiers to the tetrahedra. The tetrahedral mesh serves several different purposes. It can be used to store nearest neighbors, to subdivide a volume into convex primitives or to sample and, by using the barycentric coordinates, to parameterize the domain of a function. The function can be scalar, a vector field or even a tensor field as for example the stress tensor of an inhomogeneous material. Our compression algorithm can be extended in a natural way to support compression of all three types of data functions defined on all different types of mesh elements.

[*]Email: gumhold@uni-tuebingen.de

1.1 Basic Definitions and Notations

We deal with tetrahedral meshes in the three dimensional Euclidean space given by a set of tetrahedra such that any two tetrahedra either are disjunctive or share a face, an edge or a vertex. We denote the number of vertices, edges, faces, border faces and tetrahedra with v, e, f, b and t respectively.

We will denote the total amount of bits consumed by a tetrahedral mesh with $\mathcal{S}$, where we use a right subscript to express a special representation type. $\mathcal{S}_{\text{std}}$ denotes for example the standard representation with a list of vertex coordinate triples, a list of vertex index quadruples representing the tetrahedra and additional lists for the attached data. We split the storage space $\mathcal{S}$ into the bits $\mathcal{L}$ consumed by the locations of the vertices, $\mathcal{C}$ consumed by the connectivity and $\mathcal{D}$ consumed by the data attached to the mesh elements. If no data is present only the geometry consisting of connectivity and vertex locations has to be encoded in $\mathcal{G}$ bits. For reasonable representations we get:

$$\mathcal{S} \leq \mathcal{G} + \mathcal{D} \qquad \mathcal{G} \leq \mathcal{C} + \mathcal{L}$$

The combined representation of two and more components of the tetrahedral mesh can be more efficient since better predictions might improve delta coding or just because the coding mechanism can combine some fractional bits.

1.2 Basic Equations and Approximations

The basic equation for a tetrahedral mesh as defined in the previous section is the Euler equation:

$$v - e + f - t = \chi, \tag{1}$$

where χ is the Euler characteristic of the mesh and in most cases negligibly small. If we count the tetrahedron-face instances once for each tetrahedron and once for each face we get a second equation including the number of border faces b:

$$f = 2t + \frac{b}{2}. \tag{2}$$

In the case of triangle meshes the corresponding equations are sufficient to determine the average face-order of a vertex and the number of triangles per vertex in a mesh with small Euler characteristic and few boundary edges, but not in the tetrahedral case as Figure 1 illustrates. The tetrahedron-order of a vertex might vary between one as in Figure 1 a) and v for the mesh in b)[1]. Thus for the number of tetrahedra in an arbitrary tetrahedral mesh we only know

$$\frac{v}{4} \leq t \in \Omega(v^2). \tag{3}$$

[1]The mesh is one of the delaunay tetrahedrizations of the shown set of points.

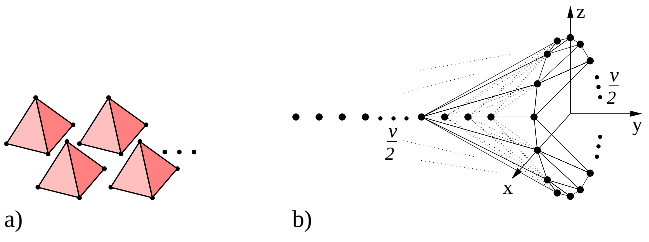


Figure 1: tetrahedral meshes with a) minimum and b) maximum vertex tetrahedron-order $\frac{4t}{v}$.

None of the tetrahedral meshes of Figure 1 are used to sample volumetric functions for volume visualization or finite element analysis. The tetrahedral meshes of interest normally have a limited edge-order of the vertices, a small border portion and low Euler characteristics of the mesh and of the border mesh, respectively. Therefore, we express the fraction between t and v in terms of the average number of edges around a vertex $\bar{o}_{v\to e} = \frac{2e}{v}$, the number of border vertices v_b, χ and χ_b the Euler characteristic of the border. As the border is closed we get $3b = 2e_b$, where e_b is the number of border edges. Using the Euler equation for triangular meshes $v_b + b = e_b + \chi_b$, we get $v_b = \chi_b - \frac{b}{2}$. This equation together with equations 1 and 2 yields

$$\frac{t}{v} = \frac{\bar{o}_{v\to e}}{2} - 1 - \frac{v_b}{v} + \frac{\chi + \chi_b}{v}. \qquad (4)$$

To find a basic approximation for the relation between t and v in a typical tetrahedral mesh with small border portion and low Euler characteristics we are left with the estimation of $\bar{o}_{v\to e}$ for a regular tetrahedral mesh. Unfortunately, the Euclidean space can not be tetrahedralized with equilateral tetrahedra. But the fraction of 4π over the steradian occupied by an equilateral tetrahedron yields[2] with 11.64 a good approximation of the average vertex edge-order. The tetrahedralization of a cubic grid yields $\bar{o}_{v\to e} \overset{v\to\infty}{\longrightarrow} 12$ for an $1:5$ zoning[3] and $\bar{o}_{v\to e} \overset{v\to\infty}{\longrightarrow} 14$ for an $1:6$ zoning. Considering this and the measured average edge-orders in Table 1, we assume in the following an average vertex order of thirteen. For tetrahedral meshes with small Euler characteristic and border portion we get in agreement with Table 1

$$v : e : f : t \approx 1 : 6.5 : 11 : 5.5. \qquad (5)$$

Let us use this approximation to estimate the storage consumption of a tetrahedral mesh in the standard representation, where each vertex is given by three 32bit floating point coordinates and each tetrahedron by four vertex indices:

$$\mathcal{L}_{\text{std}} = 96v, \qquad \mathcal{C}_{\text{std}} = 4t \cdot \lceil \log_2 v \rceil \overset{v\approx 10^5}{=} 374v. \qquad (6)$$

For a typically sized tetrahedral mesh with a hundred thousand vertices the connectivity consumes about four times more storage space than the vertex coordinates. Using our algorithm we can reduce the connectivity to about eleven bits per vertex (see Table 3). This reduces the storage requirements to a quarter without loosing information.

1.3 Entropy and Arithmetic Coding

Let $\mathcal{A} = \{a_1, \ldots, a_m\}$ be an alphabet with m different symbols and $\mathcal{S} = s_1 s_2 \ldots s_n$ a sequence of n symbols $s_i \in \mathcal{A}$. Then we define the relative frequency ν_{a_i} of the symbol a_i by

$$\nu_{a_i}(\mathcal{S}) \overset{\text{def}}{=} \frac{|\{j | s_j = a_i\}|}{n}. \qquad (7)$$

If besides the relative frequencies no further information about the sequence $\mathcal{S}$ is known, it can be shown that at least

$$\mathcal{E}(\mathcal{S}) \overset{\text{def}}{=} \mathcal{E}(n, \nu_{a_1}, \ldots, \nu_{a_m}) \overset{\text{def}}{=} -n \cdot \sum_{a\in\mathcal{A}} \nu_a(\mathcal{S}) \log_2 \nu_a(\mathcal{S}) \qquad (8)$$

bits are needed to encode the sequence S. The quantity $\mathcal{E}$ is called binary entropy. Arithmetic coding (see [13] for an introduction) allows to encode a sequence with only slightly more bits than the binary entropy.

2 Related Work

As shown in the previous section tetrahedral meshes consume disproportional storage space in comparison to the data functions they sample. There are two approaches to reduce the size of tetrahedral meshes. The first one is mesh simplification as described for tetrahedral meshes in [5, 1, 14, 11, 7]. Most methods are based on the edge collapse operation introduced by Hoppe [4] which is easily generalized from the triangular case. All these methods are lossy compression schemes.

In the area of lossless compression we only know of the Grow&Fold method proposed by Szymczak [8]. The compressed representation consists of a tetrahedra spanning tree and a folding string. The spanning tree is rooted at an arbitrary tetrahedron and grown by attaching all other tetrahedra to external triangles of the current spanning tree. For each added tetrahedron three bits encode whether further tetrahedra are attached to the three external triangles of this tetrahedron or not. The folding string contains for each external triangle of the spanning tree a 2-bit code defining one of the three edge or no edge. If an edge is specified, this external triangle is folded together with the triangle adjacent through the specified edge. As the spanning tree contains t tetrahedral and $2t + 1$ external faces the storage requirements so far are $7t + O(1)$bits. External triangles of the spanning tree without folding edge are either border triangles of the tetrahedral mesh or must be glued to a triangle which is not edge-adjacent. The glue operations consume over two bits but appear in reasonable meshes seldom enough such that the total storage space increases only slightly seven bits per tetrahedron. In section 6 we compare the storage space consumed by our compressed representation to the lower bound of seven bits per tetrahedron for the Grow&Fold representation:

$$\mathcal{C}_{\text{G\&F}} \overset{\text{def}}{=} 7t. \qquad (9)$$

The limitation of the Grow&Fold approach is that it cannot handle non manifold borders.

Grow&Fold combines ideas from the triangle mesh compression techniques “geometry compression through topological surgery” introduced by Taubin [9] and “edgebreaker” proposed by Rossignac [6]. Our compression scheme generalizes the cut-border machine proposed by Gumhold in [3] which is similar to the edgebreaker approach. But the cut-border machine is much easier to generalize to the tetrahedral case as the edgebreaker. The ideas of the triangular cut-border machine are briefly repeated in section 3.1. Basically, the cut-border machine traverses the mesh in a region filling way, which is determined by the connectivity of the mesh, and encodes for each newly added triangle one operation, which describes how the triangle is formed upon the current border edge of the growing region.

[2]We applied the Euler equation for spherical triangle meshes.

[3]Each cube is split into five tetrahedra.

Touma's [10] triangle mesh connectivity compression scheme allows the encoding of regular triangle meshes with better space efficiency. This method is in a way similar to the cut-border machine. By encoding the vertex triangle-orders with a run-length encoding scheme half of the operations encoded by the cut-border machine can be saved. For regular triangle meshes the vertex triangle-orders can be encoded very space efficiently such that the compressed representation may only consume half the storage space of the cut-border representation. Touma also proposes a simple prediction for the vertex coordinates. The coordinates are estimated as the fourth vertex of a parallelogram which is formed from the triangle inside the growing region and adjacent to the current border edge of the growing region. The crease angle at this edge is estimated from the other crease angles at the interior triangle. Encoded is only the difference between the estimation and the actual location of the vertex.

Section 4 on coordinate compression is mainly inspired by the work of Deering [2] on delta encoding and the work of Touma [10] on coordinate compression.

3 Connectivity Compression

The connectivity compression is based on the generalization of the cut-border machine described in [3]. Section 3.1 gives a short overview of the method. After that we generalize the ideas to the tetrahedral case in section 3.2 and describe the different cut-border operations (section 3.3) and the compressed representation (section 3.4). The best traversal strategy we found is proposed in section 3.5. In section 3.6 we introduce an improvement for the mesh border encoding which is also helpful for triangle mesh compression with the cut-border machine. In the triangular case the cut-border machine is very simple to implement and also extremely fast. The generalization to the tetrahedral case requires a more sophisticated data structure for non manifold triangle meshes as described in section 3.7.

3.1 Triangle Connectivity Compression with the Cut-Border Machine

The cut-border machine compresses triangle meshes which consist of a list of vertices and a list of triangles, each triangle containing the three vertex indices and the indices of the three edge-adjacent triangles. If the latter adjacency information is not known it can be easily computed through hashing.

The cut-border machine is based on a region growing traversal of the triangle mesh starting with an arbitrary triangle. The border of the growing region is called the *cut-border*. It divides the mesh into the *inner* and the *outer part*, which contain the already compressed and the remaining triangles respectively. Triangles are added to the inner part at a distinguished current cut-border edge which will be called the *gate* as proposed in [6]. After each addition of a triangle the gate moves on to another cut-border edge, until all triangles of an edge-connected component of the triangle mesh have been compressed. This is done for each edge-connected component.

The compressed representation contains for each triangle a bit code of an operation identifier which tells how the triangle was formed upon the gate. There are three cases: the gate is an edge of the mesh border, the gate forms a triangle with a vertex on the cut-border or the triangle is formed with a new vertex. The different operations are called *"border"*, *"new vertex"* and *"connect"*. The *"connect"*-operations take one parameter which specifies the offset of the third vertex relative to the gate. The *"connect"*-operations with offset one and minus one are also called *"connect forward"* and *"connect backward"*. All other *"connect"*-operations split the cut-border into two loops. As the triangle meshes describe two dimensional surfaces in three dimensional space, two cut-border loops can grow together again, actually once for each handle and each hole of the triangle mesh. The operation which unifies two loops is called *"union"* and takes two parameters, the index of the second loop and the offset of the third triangle vertex within this loop.

The cut-border data structure basically consists of a stack of doubly linked lists containing the vertices – or their indices –, which are adjacent to triangles of the inner and the outer part at the same time.

Data at the faces, edges or vertices such as their coordinates are included in the compressed representation each time a new mesh-element is added to the cut-border – for example the vertex coordinates of vertex v_i are encoded after the *"new vertex"*-operation which introduces v_i into the cut-border.

The decompression algorithm builds the mesh in the same order as the compression algorithm traverses the original mesh. With the help of the indices attached to the *"connect"*-operations the original connectivity can be reconstructed with permuted vertices and triangles. During decompression the edge-adjacency information can be reconstructed with no additional cost.

The success of the method results from the high frequency of the *"new vertex"*- and the *"connect forward"*-operations. Together they constitute in most meshes over 95% of all operations. The high frequency of the *"connect forward"*-operation and the low frequency of *"connect"*-operations with large offsets depends in a high degree on the traversal order, which is determined by the choice of the gate after each operation.

3.2 From Triangle Connectivity to Tetrahedral Connectivity Compression

As in the triangular case the uncompressed tetrahedral mesh is stored as a list of vertices and a list of tetrahedra, each tetrahedron containing the indices of the incident vertices and the face adjacent tetrahedra.

The inner and the outer part consist of a set of tetrahedra. The cut-border is the triangular surface between the inner and the outer part and the gate is a triangle of the cut-border. For each face-connected component of the mesh the traversal begins with an arbitrary tetrahedron and successively adds face-adjacent tetrahedra at the gate to the inner part. The different cut-border operations are described in the next section. The cut-border may become the surface of an arbitrary face-connected tetrahedral mesh and therefore contain non manifold vertices and edges. In section 3.7 we describe an appropriate data structure. We assume that the tetrahedral mesh is embedded in three dimensional space and that the tetrahedra do not penetrate each other.

As in the triangular case the traversal order highly influences the distribution of the *"connect"*-operations with different offsets. Section 3.5 describes the best heuristic we could find.

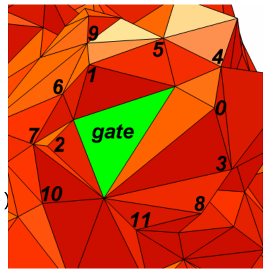
3.3 Cut-Border Operations and Situations

There are three possibilities for the fourth vertex of a newly added tetrahedron at the gate: the gate is a border triangle of the tetrahedral mesh, the gate forms a tetrahedron with a new vertex or the gate is connected through a tetrahedron to another cut-border vertex. The corresponding cut-border operations will again be called *"border"*, *"new vertex"* and *"connect"* and are abbreviated with the symbols Δ, $*$ and ∞_i.

Although only three different types of cut-border operations exist, we distinguish ten different situations which describe the surrounding of the cut-border around the gate for the different cut-border operations. All the situations are illustrated in figures 4 and 5. Figure 4 shows the situations which do not introduce non manifold vertices or edges. For the *"border"*- and the *"new vertex"*-operation only one situation exists which is depicted in figure 4 c)

Algorithm 1 *Vertex Enumeration*

```
fifo.pushback(gate.zeroEdge())
fifo.pushback(gate.oneEdge())
fifo.pushback(gate.twoEdge())
while not fifo.empty do
    edge = fifo.popfront()
    tgl = edge.rightTriangle()
    if not marked(tgl) then
        mark(tgl)
        vtx = tgl.oppositeVtx(edge)
        if not marked(vtx) then
            mark(vtx)
            enumerate(vtx)
        fifo.pushback(tgl.nextEdge(edge))
        fifo.pushback(tgl.prevEdge(edge))
```



and b), respectively. The *"connect"* operation comes along with a whole variety of situations. The most frequent of these is the *"flip"* operation shown in figure 4 a). Here the newly added tetrahedron connects the gate to an adjacent triangle of the cut-border. The common edge of these two cut-border triangles is kind of flipped if the two former cut-border triangles are replaced by the two new cut-border triangles introduced by the new tetrahedron. The *"top"* and the *"close"* operations are very similar to the *"flip"* operation. The only difference is that not only two faces of the newly added tetrahedron are part of the cut-border but three of them in the case of the *"top"* operation or even all in the case of the *"close"* operation. The *"close"*-operation eliminates or closes an edge-connected component of the cut-border triangle mesh. The *"close"*-situation cannot be seen from the outside of the cut-border. Therefore, in figure 4 e) the cut-border is rendered with transparent triangles. In the front long edges of outer triangles are visible.

As mentioned earlier, the cut-border can be a non manifold triangle mesh. Figure 5 portrays all types of situations which introduce a non manifold vertex or edge. In figure 5 a) the non manifold counterpart of the *"flip"* situation is shown. Here the free edge of the *"flip"* situation is touched by the cut-border and therefore already belongs to the cut-border. The touched edge becomes non manifold after application of the *"connect"* operation. The *"join"* situation in figure 5 b) is the non manifold counterpart of the *"new vertex"* operation. The fourth vertex of the newly added tetrahedron is part of a region of the cut-border triangle mesh which is further apart from the gate. This vertex becomes non manifold. Finally, in the *"join"* situations depicted in figures 5 c), d) and e) not only the fourth vertex of the newly added tetrahedron belongs to the cut-border but also one two or all three free edges of the *"join"* situation. Thus one, two or three non manifold edges are introduced.

The situations depicted in figures 4 and 5 constitute all possible situations, which can be easily verified by considering a newly added tetrahedron: the three triangles of the tetrahedron which are unequal to the gate may all be part of the cut-border or not be part. The same holds true for the fourth vertex and the three edges not incident to the gate. All of these seven mesh elements might be present in the cut-border or not. The presence of one of the three triangles implies the presence of the fourth vertex and the two incident edges. If we take such implications into account each possible assignment of presence to the three triangles, three edges and the fourth vertex yields exactly one of the discussed situations. Thus each face-connected component of the tetrahedral mesh can be compressed without any vertex repetitions. Only if two components of the tetrahedral mesh are exclusively connected through edge-adjacency and vertex-adjacency the involved non manifold vertices are repeated. In a simple way the *"border"*-operation allows for the encoding of all possible border surfaces of tetrahedral meshes including non manifold borders.

The *"connect"* operation takes one index as parameter, which specifies the fourth vertex in the cut-border. The fourth vertex is with high probability near to the gate. We can exploit this fact for a more efficient encoding by mapping near fourth vertices to small connect indices. This is achieved by a breadth-first traversal through the triangles of the cut-border starting at the gate as shown in the illustration of algorithm 1. The enumeration is not uniquely defined before one edge of the gate is specified at which the enumeration with the zero connect index will begin. This edge will be called the *zero edge* and is specified by the traversal strategy (see section 3.5). Algorithm 1 gives pseudo code for the vertex enumeration. The algorithm is similar to the cut-border traversal in the case of a triangle mesh. In a *fifo* these edges of the cut-border are stored which are adjacent both to a visited triangle and to a not visited triangle at the same time. The zero edge is firstly placed into the *fifo*. Triangles are visited by extracting the next edge from the *fifo* and addressing the adjacent triangle which has not been visited yet. If the third vertex of the newly visited triangle is reached the first time, the next available connect index is assigned to it. In this way the vertices obtain the indices illustrated in the figure of algorithm 1.

The *"flip"* situation can arise for the operations ∞_0, ∞_1 and ∞_2, the *"top"* situation for ∞_0 and ∞_1 and *"close"* only for ∞_0. The different *"join"* situations correspond to *"connect"* operations with larger index and are less frequent. The traversal strategy described in section 3.5 optimizes the choice of the zero edge in a way that most *"flip"* and *"top"* situations are encoded with ∞_0.

3.4 Compressed Representation

In the triangular case the *"new vertex"*-operation $*$ is performed in about half the cases and is most frequent. In the tetrahedral case the relative frequency of $*$ is only about $\frac{1}{5.5}$, whereas the connect operations with small index are most frequent. For optimal encoding of the operation symbols we use arithmetic coding since the relative frequencies are unequal to 2^{-k} and therefore Huffman-coding is not appropriate.

The connectivity of the tetrahedral mesh is given by the sequence of cut-border operations. As each operation adds one tetrahedron or specifies one border face, $t + b$ operations are encoded. The binary entropy defined in equation 8 gives a good lower bound

$$\mathcal{C}_{\text{CB}} \overset{\text{def}}{=} \mathcal{E}\left(n, \nu_\Delta, \nu_*, \nu_{\infty_0}, \nu_{\infty_1}, \ldots\right) < \mathcal{C}_{\text{CB}}^{\text{adapt}} \qquad (10)$$

for the storage space $\mathcal{C}_{\text{CB}}^{\text{adapt}}$ consumed by our arithmetic coder with adaptive relative frequencies, which are initialized to the average values given in the last row of Table 2. Table 3 shows that our arithmetic coder almost achieves the optimum.

The vertex coordinates and the data at the vertices, edges, faces and tetrahedra are incorporated in the arithmetic coding stream with separate coding models. Each time a cut-border operation produces a new mesh element, the corresponding data is added to the stream. The representation of a 1:6 zoning of a cube with vertex data $v_0, v_1, \ldots, v_7$ and tetrahedral data $t_0, t_1, \ldots, t_5$ might look as follows:

$$\begin{array}{l} t_0 x_0 y_0 z_0 v_0 x_1 y_1 z_1 v_1 x_2 y_2 z_2 v_2 x_3 y_3 z_3 v_3 \Delta\Delta \\ * t_1 x_4 y_4 z_4 v_4 \Delta * t_2 x_5 y_5 z_5 v_5 \Delta \\ * t_3 x_6 y_6 z_6 v_6 \infty_0 t_4 \Delta\Delta * t_5 x_7 y_7 z_7 v_7 \Delta\Delta\Delta\Delta\Delta\Delta. \end{array}$$

3.5 Traversal Order

The traversal strategy chooses after each cut-border operation the next gate and zero edge. The aim is to favorite a small number of different kinds of operations. To avoid most connect operations with large indices it turned out that a good strategy is to stay at one cut-border vertex until all adjacent tetrahedra have been visited. The cut-border vertices are processed in a fifo order. For the choice of the zero edge and the order in which the triangles around a cut-border vertex are added, we tried two heuristics that favorite the ∞_0-operation. The first one cycles around edges and tries to close up with a ∞_0-operation by setting the zero edge of the gate to the edge around which the cut-border machine cycles. The second strategy defines the zero edge of each cut-border triangle at the time when the triangle is created. The zero edge is set to the edge which is shared by the gate and the new triangle. In case of a new vertex operation it is obvious that with this choice the zero edge is the edge with the smallest angle in the outer part. This still holds true to some extent for the other operations. The first heuristic increased the frequency of the ∞_0-operation to 45% and the second heuristic even to 60%. Thus we chose the second strategy, which is documented in Table 2.

3.6 Mesh-Border Encoding

In order to allow for a non manifold mesh border, we explicitly encode the border operations. The border symbol can be avoided when an edge-adjacent triangle of the gate has already been encoded as border triangle. In this case the corresponding connect symbol can be used. This optimization helped to decrease the additional amount of storage for the mesh border to one bit per border triangle as tabulated in Table 3. The same optimization improves the border encoding in the triangular case of the cut-border machine.

3.7 Cut-Border Data Structure

Data Structure 1 *Cut-Border*

```
CutBorder
    CutBorderTriangle          triangles[]
    Fifo<CutBorderVertex>      vertices
    TriangleIndex              gate
CutBorderTriangle
    VertexIndex                vertexIndices[3]
    TriangleIndex              adjacentTriangles[3]
    TetraIndex                 innerTetra
    Boolean                    meshBorder
    Integer                    zeroEdge
CutBorderVertex
    VertexIndex                meshVertexIndex
    Set<TriangleIndex>         adjacentTriangles
```

Data structure 1 shows the cut-border data structure. Three relations between the cut-border vertices and the cut-border triangles are stored: for each triangle the three incident vertices and three edge-adjacent triangles; for each vertex all incident triangles. The latter relation is stored in a set data structure which allows insertion and elimination of elements and the intersection of two sets. This relation allows for the handling of non manifold vertices and edges. For each cut-border triangle the incident tetrahedron of the inner part is stored in order to find the new tetrahedron if the triangle becomes the gate. The `meshBorder`-flag tells us when the cut-border triangle has already been encoded as border triangle of the mesh and therefore does not have to be visited again. With the help of this flag the optimized border encoding is realized. As the traversal order introduced in section 3.5 defines the zero edge for each triangle at creation time, an index between zero and two is stored for each cut-border triangle defining the zero edge. The cut-border vertices are organized in a fifo as demanded by the traversal strategy chosen in section 3.5.

We generate for each vertex of the tetrahedral mesh a field which stores the index of the cut-border vertex and initialize it before compression to minus one. In this way we can not only map a tetrahedral mesh vertex index to a cut-border vertex index but do also know which of the tetrahedral mesh vertices are part of the cut-border.

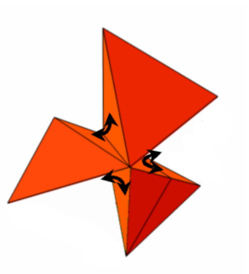


Figure 2: Edge-adjacency of cut-border triangles around non manifold edge.

Let us describe why it is sufficient to keep for each triangle only three edge-adjacent neighbors even at non manifold edges. At any time the cut-border describes the surface of a tetrahedral mesh. Thus the faces around a non manifold edge divide the space into regions alternately belonging to the inner and the outer part. These regions around a non manifold edge are called *inner/outer regions*. The faces bounding the same outer region can be set to be edge-adjacent as illustrated in Figure 2. This definition correctly reflects the proximity needed in enumerating the vertices relative to the gate. Faces of different outer regions can not be connected through a tetrahedron without intersecting an inner region.

Finally, we describe the updates of the cut-border data structure for the different situations depicted in figures 4 and 5. During the *"connect"* operation of a manifold *"flip"* situation (see figure 4 a)) the two present triangles in the cut-border are replaced with two new ones where the common edge is flipped. The vertices and face-adjacent triangles of the two new triangles can be easily determined from the old triangles. For each new triangle the zero edge is set to the edge, which is incident to the gate. The `innerTetra` index of the newly added triangles is set to the newly added tetrahedron, as in all other situations of all operations. Finally, the old triangles are removed from the triangle sets of the vertices and the new triangles are added.

The first step during the update of the *"new vertex"* operation is to create a new cut-border vertex for the fourth vertex of the newly added tetrahedron and store its vertex index of the tetrahedral mesh in the corresponding field. Conversely, the index of the new cut-border vertex is stored within the corresponding field of the tetrahedral mesh vertex. Next the gate triangle is removed and three new triangles are inserted. Again their zero edges are set to the edges incident to the gate. The *"border"* operation just sets the border flag of the gate triangle. For the border optimization the border flags of the three edge-adjacent cut-border triangles are checked and if one of them is set, the operation is encoded with the corresponding *"connect"* operation. The *"top"* situation is similar to the *"flip"* situation except that three triangles are removed and only one is added. As last manifold situation the *"close"* operation eliminates all involved triangles and these vertices for which the set of adjacent triangles becomes empty. If a cut-border vertex is removed the index stored with the corresponding tetrahedral mesh vertex is set to minus one again.

In order to distinguish between manifold and non manifold situations we have to clear up how to decide whether an edge of the newly added tetrahedron belongs to the cut-border or not. The question is trivially answered positively if an incident triangle of the newly added tetrahedron already belongs to the cut-border. Otherwise the answer can be determined by intersecting the set of adjacent triangles of the incident vertices of the edge in question. If the intersection is empty no cut-border triangle contains the edge and therefore the edge cannot belong to the cut-border. The intersection test must be performed for all edges of the non manifold situations in figure 5 which are not incident to a cut-border triangle. In case of the *“nm flip”* situation this is one edge and in case of the four *“join”* situations these are three edges. Only if the non manifold edges are detected, the face-adjacencies can be updated according to figure 2. And this is the only difference in the update process between the *“nm flip”* and *“flip”* situations and between the four different *“join”* situations and the *“new vertex”* operation.

The *“nm flip”* operation is distinguished from the *“flip”* situation by checking if the edge connecting the two newly added triangles belongs to the cut-border or not. This check can be done after the update performed for the *“flip”* situation, such that the face-adjacencies of the two new triangles can be corrected if necessary. This is only possible if we assume that the vertex coordinates are known and given in three dimensional space. For more general tetrahedral meshes the neighbors of the newly added triangles must be explicitly encoded. This can be done with few bits and as the non manifold situations are much less frequent as the manifold situations, the total storage space won’t increase significantly for typical meshes.

The family of *“join”* situations is detected whenever the three triangles of the newly added tetrahedron, which are not equal to the gate, are not part of the cut-border but the fourth vertex is part of the cut-border. The latter condition is checked with the help of the cut-border index field attached to the tetrahedral mesh vertices. The update of the *“join”* situations is the same as in the case of the *“new vertex”* operation accept that the three newly added triangles must also be inserted to the triangle set of the fourth vertex. Finally, the three potential non manifold edges are checked for their presence in the cut-border and the face-adjacencies of the corresponding triangles are corrected if necessary as in the case of the *“nm flip”* situation.

4 Coordinate Compression

In a first step we quantize each vertex coordinate to 16 bits according to the diagonal of the bounding box of all vertices. Thus the compression is lossy and for some applications not appropriate. All the meshes we received came in ASCII format with six to eight valid digits which is equivalent to 19-26 bits. We loose some information in the quantization step and the shape of small tetrahedra changed slightly, but no tetrahedron changed its orientation.

To encode the 16 bit coordinates arithmetically it turned out to be economical to split each coordinate into four packages of four bits. For each package we use a different set of adaptive frequencies for the arithmetic coder. This strategy dramatically reduced the storage space consumed by the arithmetic coder and increased the compression speed.

The next step in coordinate compression is delta coding. We encode the vertex coordinates during the compression of the connectivity. After each new vertex operation the difference vector from the center of the gate triangle to the new vertex is encoded. Thus we use the proximity information given by the tetrahedralization of the vertices. We can estimate the number of bits saved through delta coding with the following simple argument. Suppose the vertices are uniformly distributed. Then there are approximately $\sqrt[3]{v}$ vertices per coordinate axis and it should be possible to save $\log_2 \sqrt[3]{v}$ bits per coordinate. Thus the storage space consumed per vertex can be estimated with $48 - \log_2 v$ bits, which is about three bits above the actually achieved storage space.

A final improvement of two bits less storage space per vertex could be achieved by rotating the coordinate system such that the z-axis is the normal of the gate and the x-axis parallel to the zero edge. Quantization is done after changing to the new coordinate system. To avoid accumulation of rounding errors it is very important that during compression the center of the gate is computed with the same quantized coordinates which are available to the decompression algorithm. The change of the coordinate system saved two bits in the x- and y-axes. The final storage space consumed per vertex by the coordinates is tabulated in Table 3 in the column labeled $\frac{\mathcal{L}_{\text{CB}}^{\text{16bit}}}{v}$.

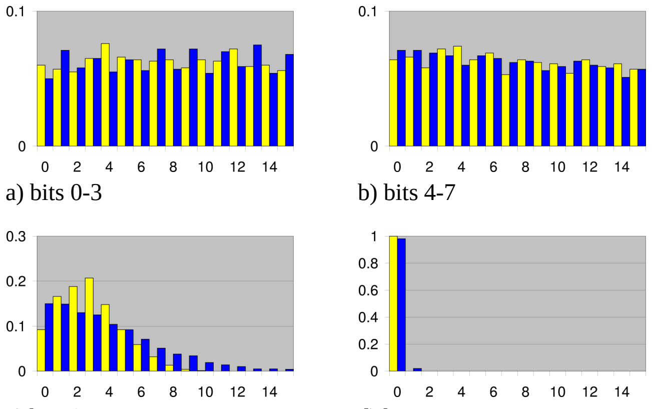


Figure 3: Distribution of coordinates.

Figure 3 shows the relative frequencies of the 16 different values of each 4 bit package in the case of the randomly generated mesh in section 6. The yellow bars represent the z-coordinate and the blue bars the x- and y-coordinates. The x- and y-coordinates were united as their distributions do not differ at all. The lower eight bits are distributed quite uniformly, whereas the higher four bits are nearly exclusively zero. The bits $8 - 11$ are especially interesting. The x- and y-coordinates frequencies have a Gaussian fall off, whereas the z-coordinate frequencies increase to a maximum at the value 3 and then drop of much faster than the x- and y-frequencies.

This encouraged us to predict the z-coordinate, which is the height of the new tetrahedron in the new vertex operation, from the height of the tetrahedron of the inner part which is adjacent to the gate. But the distribution of the z-coordinate frequencies was even smoothed out and the compression became worse. We also tried to predict the x- and y-coordinates from the interior adjacent tetrahedron but with a similar failure. All these tests were also performed on the more regular meshes of section 6 with no success. The prediction of the tetrahedron height, which is proportional to its volume, should help in meshes where the sampling density changes significantly. But we still have to conclude that tetrahedral meshes are too irregular to predict vertex coordinates much better than with the proximity information of the connectivity alone.

5 Data Compression

The last section showed that tetrahedral meshes are not regular enough for a good prediction of vertex coordinates. Therefore, we propose to encode data given for the mesh elements in a different way. In this section we restrict ourselves to a scalar data function attached to the vertices. This data is transmitted with each new vertex operation after the vertex coordinates. We propose delta encoding for the data function after an appropriate quantization. This time we can additionally use the vertex coordinates to predict the function value at the new vertex.

Let us denote the scalar data function with f and the location of the new vertex with $\vec{v}_n$. A linear approximation f_{lin} of the function f is of the form

$$f_{lin}(\vec{v}) = \vec{f_{lin}}^{T} \cdot \vec{v} + f_{lin}(\vec{0}). \tag{11}$$

Thus the linear approximation must be known at four locations in order to determine the unknowns $\vec{f_{lin}}$ and $f_{lin}(\vec{0})$. In a new vertex operation the new tetrahedron is always adjacent to a tetrahedron of the inner part, where the function f is already known. We can use the corner vertices of this tetrahedron $\vec{v}_0, \vec{v}_1, \vec{v}_2$ and $\vec{v}_3$ and the corresponding values of the data function $f(\vec{v}_i)$ to determine the unknowns of the linear approximation. The linear system of equations is

$$f(\vec{v}_i) = \vec{f_{lin}}^T \cdot \vec{v}_i + f_{lin}(\vec{0}), \quad i \in \{0, 1, 2, 3\}.$$

If this linear system is solved and the values are plugged into equation 11 with $v = v_n$, we get as linear prediction at the location $\vec{v}_n$

$$\begin{aligned} f_{lin}(\vec{v_n}) &= \Delta\vec{f}^T \cdot \Delta V^{-1} \cdot (\vec{v}_n - \vec{v}_0) + f(\vec{v}_0), \text{ with} \\ \Delta\vec{f} &\overset{\text{def}}{=} \begin{pmatrix} f(\vec{v}_1) - f(\vec{v}_0) \\ f(\vec{v}_2) - f(\vec{v}_0) \\ f(\vec{v}_3) - f(\vec{v}_0) \end{pmatrix} \text{ and} \\ \Delta V &\overset{\text{def}}{=} \begin{pmatrix} (\vec{v}_1 - \vec{v}_0) & (\vec{v}_2 - \vec{v}_0) & (\vec{v}_3 - \vec{v}_0) \end{pmatrix}. \end{aligned}$$

The matrix ΔV can be inverted, iff the tetrahedron $(\vec{v}_0, \vec{v}_1, \vec{v}_2, \vec{v}_3)$ is not degenerated. Vector valued data functions can also be compressed with this method – coordinate by coordinate.

6 Measurements & Results

6.1 The Tetrahedral Meshes

Figure 6 shows the six tetrahedral meshes which we used for our measurements. They differ in their sizes and their origin. The "Random" mesh was generated by delaunay tetrahedralization of a cloud of randomly distributed points. In order to show that the interior of this mesh is more complex than the surface, we blended a cut through the mesh with its surface. The "Proto" mesh is a quite regular tetrahedralization of an object with non trivial boundary. The "Bubble" is the output of a simplification algorithm applied to a spherical symmetric scalar function. Again the blending technique shows part of the interior. The "Torso" meshes are regularly tetrahedralized real world meshes and the "Blunt Fin" is a curvy linear grid.

6.2 Measurements

mesh	v	v: e: f: t	$\frac{v_b}{v}$	b	$\bar{o}_{v\to e}$	$\bar{o}_{e\to t}$
Random	2000	1:7.39:12.67:6.29	0.101	400	14.77	5.11
Proto	2896	1:5.94: 9.41:4.47	0.477	2760	11.89	4.51
Bubble	5715	1:6.89:11.63:5.74	0.150	1710	13.78	5.00
Torso I	11140	1:6.55:10.91:5.35	0.197	4380	13.10	4.90
Torso II	15164	1:6.61:11.04:5.43	0.180	5454	13.22	4.93
Blunt Fin	40960	1:5.74: 9.32:4.58	0.165	13516	11.48	4.78
average		1:6.52:10.83:5.31	0.212		13.04	4.87

Table 1: Basic quantities of the measured meshes.

Table 1 shows the basic quantities of the different meshes and average values which confirm equation 5.

In Table 2 the distribution of the cut-border symbols is analyzed. The first column shows for each mesh the total number $t + b$ of encoded operations. In the following columns the relative frequencies of the different cut-border symbols are shown. ∞_0 is with 60% the most frequent operation, followed by $*$, ∞_1 and ∞_2. With the border optimization described in section 3.6 the frequency of the border symbol became negligibly small. The last column shows the fraction of the non-manifold situations in Figure 5 which arose during compression. This number is important for the optimal running time of the compression and decompression algorithms as the non-manifold operations consume more computing power.

Table 3 illustrates different aspects of the consumed storage space and running time for the cut-border machine. The first column shows the storage space consumed by our arithmetic coder for the connectivity. The second and third columns tabulate the binary entropy of the cut-border operations in bits per vertex and bits per tetrahedra. Comparison of the first two columns shows that our arithmetic coder is near the optimum. The cut-border machine consumes on average about two bits per tetrahedron, even for the randomly generated mesh which forces more connect operations with a high index. $\mathcal{C}_{\text{CB},\Delta}$ is the binary entropy of the sequence of cut-border operations which were used to encode the border faces. The fourth column of Table 3 shows that the border could be encoded with about one bit per triangle. As the best triangle mesh compression methods consume also about one bit per triangle, the initializing of the cut-border machine with the border of the tetrahedral mesh would not improve our border encoding described in 3.6. The fifth column of Table 3 documents the compression speed in tetrahedra per second for connectivity alone. The decompression speed is approximately the same. The speed does not depend on the size but more on the frequency of non-manifold operations (compare the last column of Table 2). The last but one column contains the storage space consumed by the vertex coordinates, if compressed with the technique described in section 4. Finally, the last column shows that the vertex compression doesn't decrease the compression speed significantly.

Table 4 compares the cut-border machine to the standard representation and the Grow&Fold compression of Szymczak [8]. The results of the cut-border machine are convincing and improve the standard representation by a factor of 20 to 50 depending primarily on the size of the tetrahedral mesh but also on the regularity.

7 Conclusion

We presented for the first time a lossless connectivity compression scheme for tetrahedral meshes which can handle non manifold borders. Our implementation of the cut-border machine showed that it achieves very high compression rates and is able to compress tetrahedral connectivity to about two bits per tetrahedron, which is between three and four times better than previously reported results. Lossy compression of vertex coordinates turned out to be not as efficient as in the triangular case but still valuable for most applications. Future work must concentrate on more sophisticated compression techniques for the vertex coordinates and further data attached to the tetrahedral mesh.

8 Acknowledgements

We like to thank Hans-Christian Hege for providing the "Torso"-meshes, Paolo Cignoni for the "Bubble"-mesh and Claudio T. Silva

mesh	$t + b$	ν_Δ	ν_*	ν_{∞_0}	ν_{∞_1}	ν_{∞_2}	$\nu_{\infty_{i>2}}$	ν_{nm}
Random	12971	0.001	0.154	0.519	0.118	0.108	0.101	0.116
Proto	15695	0.001	0.184	0.631	0.073	0.067	0.044	0.046
Bubble	34526	0.001	0.165	0.549	0.106	0.091	0.088	0.109
Torso I	64028	0.002	0.174	0.607	0.080	0.072	0.064	0.069
Torso II	87788	0.001	0.173	0.603	0.083	0.075	0.065	0.069
Blunt Fin	200910	0.000	0.204	0.707	0.045	0.044	0.000	0.000
average		0.001	0.176	0.602	0.084	0.076	0.060	0.068

Table 2: Total number of encoded operations; relative frequencies of cut-border operations; relative frequency of non-manifold situations.

mesh	$\frac{\mathcal{C}_{\text{CB}}^{\text{adapt}}}{v}$	$\frac{\mathcal{C}_{\text{CB}}}{v}$	$\frac{\mathcal{C}_{\text{CB}}}{t}$	$\frac{\mathcal{C}_{\text{CB},\Delta}}{b}$	$\left(\frac{t}{sec}\right)_{\mathcal{C}}$	$\frac{\mathcal{L}_{\text{CB}}^{\text{16bit}}}{v}$	$\left(\frac{t}{sec}\right)_{\mathcal{G}}$
Random	15.12	15.02	2.39	1.37	84831	34.40	73866
Proto	9.55	9.48	2.12	0.90	93603	30.86	74259
Bubble	13.52	13.43	2.34	1.11	85774	30.09	74146
Torso I	11.02	10.99	2.05	1.29	92508	30.41	76749
Torso II	11.15	11.14	2.05	1.20	92574	29.64	76992
Blunt Fin	6.00	5.99	1.31	0.54	98587	26.36	78493
average	11.06	11.01	2.04	1.07	91313	30.29	75751

Table 3: Cut-border machine: consumed storage for connectivity, border and quantized vertex coordinates. Running time for connectivity alone and together with vertex coordinates in tetrahedra per second on a Pentium II 350MHz.

mesh	$\frac{\mathcal{C}_{\text{std}}}{\mathcal{C}_{\text{CB}}^{\text{adapt}}}$	$\frac{\mathcal{C}_{\text{CB}}^{\text{adapt}}}{v}$	$\frac{\mathcal{C}_{\text{G\&F}}}{v}$
Random	18.39	15.12	44.03
Proto	22.61	9.55	31.29
Bubble	22.22	13.52	40.18
Torso I	27.27	11.02	37.45
Torso II	27.29	11.15	38.01
Blunt Fin	48.90	6.00	32.06

Table 4: Comparison of the different approaches.

for the “Proto”-mesh.

This work was supported by the Deutsche Forschungsgemeinschaft (DFG) and is part of the project D1 within the Sonderforschungsbereich 382.

References

[1] Paolo Cignoni, Claudio Montani, Enrico Puppo, and Roberto Scopigno. Multiresolution representation and visualization of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):352–369, October–December 1997. ISSN 1077-2626.

[2] Michael F. Deering. Geometry compression. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 13–20. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[3] Stefan Gumhold and Wolfgang Straßer. Real time compression of triangle mesh connectivity. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 133–140. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8.

[4] Hugues Hoppe. Progressive meshes. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[5] Jovan Popović and Hugues Hoppe. Progressive simplicial complexes. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 217–224. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

[6] J. Rossignac. Edgebreaker: connectivity compression for triangle meshes. Technical Report GIT-GVU-98-35, Georgia Institute of Technology, October 1998.

[7] Oliver G. Staadt and Markus H. Gross. Progressive tetrahedralizations. In *Proceedings of IEEE Visualization ’98*, page 7. ETH Zürich, Institute of Scientific Computing, August 1998.

[8] A. Szymczak and J. Rossignac. Grow&Fold: compression of tetrahedral meshes. Technical Report GIT-GVU-99-02, Georgia Institute of Technology, February 1999.

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

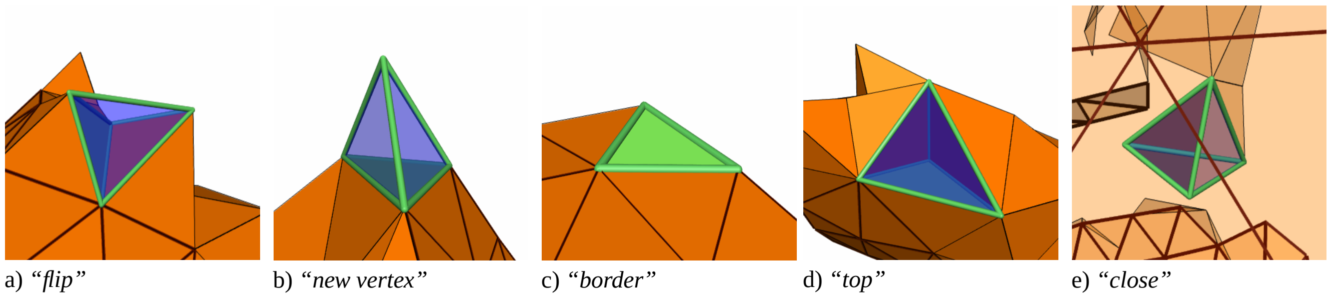
[10] Costa Touma and Craig Gotsman. Triangle mesh compression. In Wayne Davis, Kellogg Booth, and Alain Fourier, editors, *Proceedings of the 24th Conference on Graphics Interface (GI-98)*, pages 26–34, San Francisco, June18–20 1998. Morgan Kaufmann Publishers.

[11] Vivek Verma and Allen VanGelder. DECIMATION OF TETRAHEDRAL GRIDS WITH ERROR CONTROL. Technical Report UCSC-CRL-97-25, University of California, Santa Cruz, Jack Baskin School of Engineering, June 23, 1998.

[12] Peter L. Williams. Visibility ordering meshed polyhedra. *ACM Transactions on Graphics*, 11(2):103–126, April 1992.

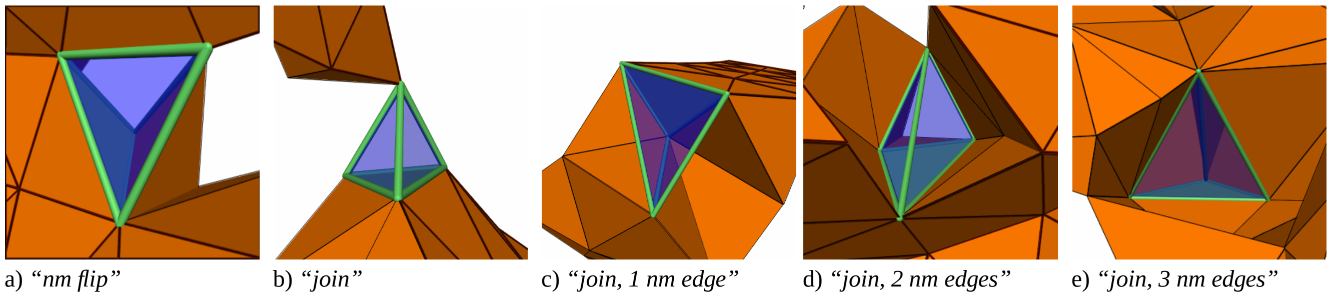
[13] Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, June 1987.

[14] Yong Zhou, Baoquan Chen, and Arie Kaufman. Multiresolution tetrahedral framework for visualizing regular volume data. In Roni Yagel and Hans Hagen, editors, *IEEE Visualization ´97*, pages 135–142. IEEE, November 1997.



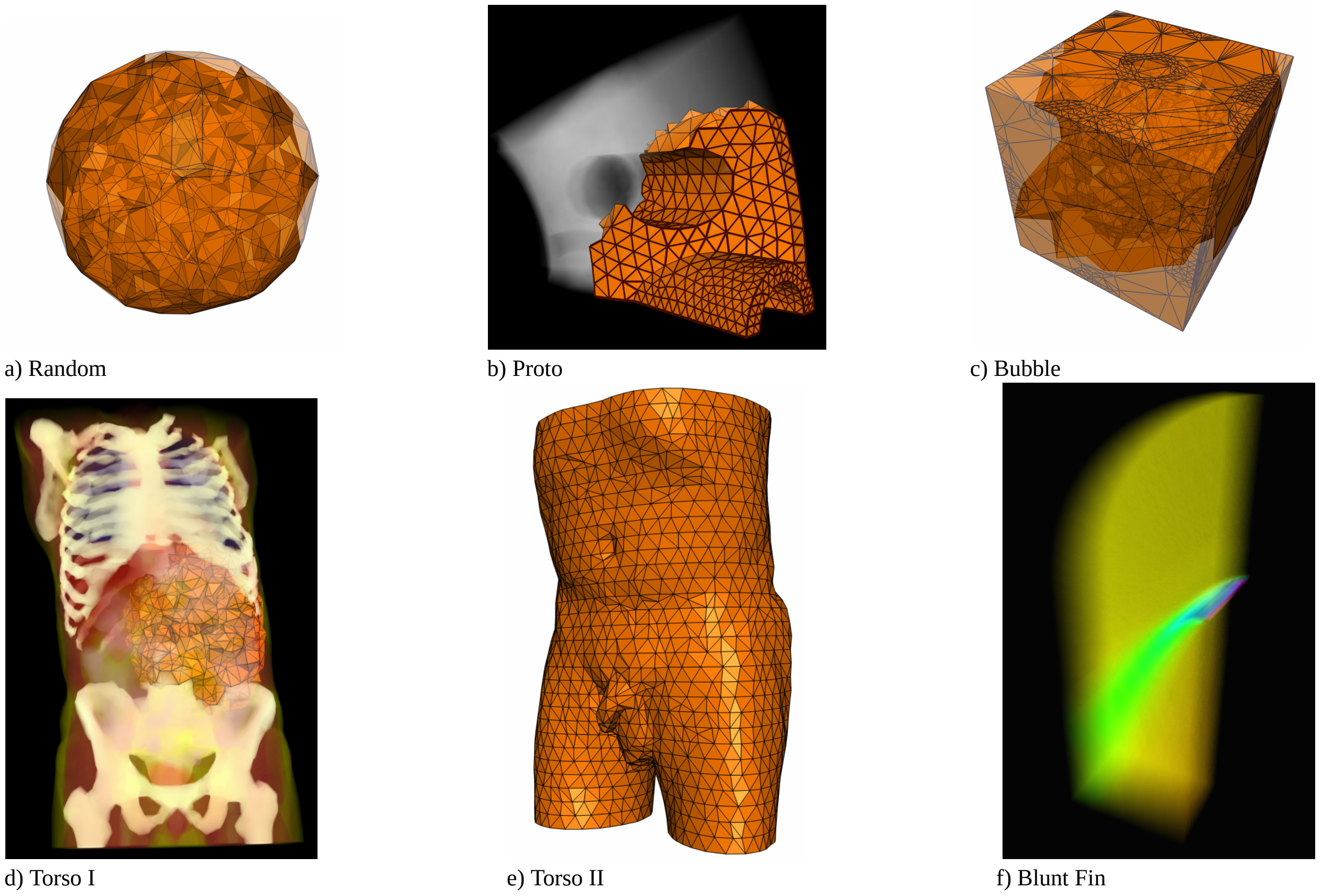
a) *"flip"* b) *"new vertex"* c) *"border"* d) *"top"* e) *"close"*

Figure 4: The different manifold cut-border situations. The gate is shown as green triangle and the newly added tetrahedron with green edges and blue transparent faces.



a) *"nm flip"* b) *"join"* c) *"join, 1 nm edge"* d) *"join, 2 nm edges"* e) *"join, 3 nm edges"*

Figure 5: The different types of non manifold cut-border situations.



a) Random b) Proto c) Bubble

d) Torso I e) Torso II f) Blunt Fin

Figure 6: The measured tetrahedral meshes. The transparent meshes were rendered with projected tetrahedra. To the tetrahedra of the "Torso I"-mesh a material identifier is attached. The "Blunt Fin"-mesh was rendered with false colors.