Compression of Isosurfaces for Structured Volumes with Context Modelling

Dietmar Saupe, Jens-Peer Kuska
University of Leipzig, Computer Science Institute
Augustusplatz 10/11, 04109 Leipzig, Germany
Email: saupe,kuska@informatik.uni-leipzig.de

Abstract

In many applications surfaces with a large number of primitives occur. Geometry compression reduces storage space and transmission time for such models. A special case is given by polygonal isosurfaces generated from gridded volume data. However, most current state-of-the-art geometry compression systems do not capitalize on the structure that is characteristic of such isosurfaces, namely that the surfaces are defined by a set of vertices on edges of the grid. In a previous paper we had proposed a compression method for isosurfaces that is designed to exploit this feature. In this paper we use the same coding approach, however, including context models for the encoding of the symbol streams. We report improved compression ratios by about 20% for complex isosurfaces from a CT scan of a human head. For this data set our new coder outperformed state-of-the-art general purpose geometry compression methods by a factor of 2.6 to 3.4 in terms of compression ratio. We also report results obtained by two predictive coding methods based on least squares function fitting and a surface relaxation algorithm.

1. Introduction and previous work

This paper extends the methods and results of our recent work [11]. Using context modelling we improve the performance of our geometry compression of isosurfaces from structured grids. The first two sections provide an introduction and review the method as presented earlier [11]. Section 3 describes the enhancements contributed in this work: context coding of parts of the code symbol stream and predictive coding. The last sections contain empirical results demonstrating the improvements and point to further work.

The number of installed 3D image capturing systems such as magnetic resonance imaging, computed tomography, and confocal laser range scanners is growing. Moreover, as technology advances the image resolutions increase. Similarly, due to the increased availability of high performance computing 3D simulations (e.g., fluid flow and meteorology) become more common. Overall, there is a rapidly growing amount of 3D data awaiting processing, evaluation, and storage for archival. Isosurfaces provide a natural approach to view scalar volume data which is the most common type of volume data. In order to generate an isosurface at a client workstation it generally is not economical to transmit the full volume data from the corresponding server. In such cases the transmission of the surface data may be much faster. However, common isosurfaces still contain many polygons, and corresponding VRML encodings produce very large files, posing a major obstacle for rapid transmission over networks. In other applications special isosurfaces may be selected for archival for which compression methods can reduce large file sizes. This paper advances compression techniques tailored for isosurfaces derived from structured volumes as a tool to support their efficient transmission and storage.

The standard format for storage and transmission of 3D polygon models is the Virtual Reality Modelling Language (VRML) [7]. The scene files contain the coordinates of the points as an indexed point set and the topological connection of the points to build polygons. The ASCII scene description for 3D models often is compressed with the gzip program [3].

Several methods for *geometry compression,* i.e., for compact encoding of 3D meshes have been proposed. Typically, the topology is encoded independently of the vertex coordinates. For a given vertex the quantized coordinates are predicted based on the topology and geometry that has been encoded before. Then the prediction residuals in each coordinate are entropy coded.

Certain methods for the compression of general polygonal models and surfaces were proposed to the VRML consortium [13, 14, 15]. A coder based on these works is in the public domain. Another approach for triangle mesh compression and general polygonal surface compression was proposed in [16] and [8]. The compression software available from Virtue Ltd. (www.virtue3d.com) uses some of these algorithms. In [4] Devillers and Gandoin describe a geometry compression approach that is different in spirit from the others. The coding order of the vertices is used to compress their coordinates. The topology of the mesh is reconstructed from the vertices. Another different coding principle using the spectral representation of the adjacency matrix of the mesh is described in [1]. It allows graceful degradation for lossy surface compression. The usage of a multiresolution representation for the compression of a triangular mesh is discussed in [2, 6]

Most of these methods are designed to handle an individual connected component of a polygonal surface. Models with more than one component need preprocessing by additional algorithms to identify and group the connected parts of the surface. Unfortunately isosurfaces from complex volume data tend to produce a large number of connected components requiring extensive preprocessing.

All of the above methods may be applied to compress isosurfaces. However, these techniques are general purpose methods, designed for a large class of 3D meshes. They do not consider isosurfaces derived from volume data as a special case. Thus, improved performance may be expected using methods specially designed for this case.

The basis of our method [11] lies in the fact that for the encoding of an isosurface from gridded volume data it suffices to encode the cells of the volume data that intersect the surface along with the values of the volume data on the vertices of these cells. There is no need to encode the topology of the polygon mesh since after decoding the cells and their voxel values an isosurface extraction method automatically regenerates the topology from the decoded data. Furthermore, as in Engel et al [5] we observed that in place of the voxel values at the vertices of the cells it suffices to encode the intersections of the isosurface with the edges of the cells [11]. More precisely, the following information needs to be encoded for each cell that intersects the isosurface: one pointer to the cell location, an index describing the marching cubes (MC) configuration of the cell [9], and for each cell edge that intersects the isosurface an interpolation value, that prescribes the (quantized) intersection point on the edge.

The emphasis in [5] was on methods for efficient distributed isosurface reconstruction. For the purpose of compression we significantly extended the approach of [5] in several ways. The pointers to cell locations were replaced by a more efficient octree structure. In [5] an interpolation weight had to be encoded multiple times, once for each cell that shares the edge on which the intersection point lies. We encoded each point only once. The MC configuration index can be reconstructed by the decoder and does not need to be encoded. Finally, we applied entropy coding to the code symbol stream. In [11] we obtained for a complex isosurface from a CT head scan compression ratios that are more than two times as large as those yielded by the general state-of-the-art geometry compression methods (binary VRML, [13]).

In this paper we further improve the compression performance by using a context model for the interpolation values. The contexts of an intersecting edge are given by the up to 2^8 possible configurations of adjacent orthogonal intersecting grid edges that are connected to the end points. Also contexts for the bits that represent the octree for the cell locations are defined.

Another isosurface compression method has recently been proposed by Zhang et al in [19] and works by encoding two 3D bitmaps and a list of voxel values. The first bitmap distinguishes those grid points that are end points of cell edges that intersect the isosurface. For each of these grid points the corresponding voxel value is entered in a list, which is encoded using second order differences. The second bitmap identifies the intersecting cells. The encoding proceeds slice by slice. Arithmetic coding is applied to all components.

2. Isosurface compression method

We assume that the volume data for isosurface extraction is given on a regular grid, consisting of grid points denoted by $x_{i,j,k} \in \mathbb{R}^3$ with $i = 0, \dots, i_{\max}-1$, $j = 0, \dots, j_{\max}-1$, and $k = 0, \dots, k_{\max}-1$. For example, on an integer cubic lattice we may have $x_{i,j,k} = (i, j, k)$. Scalar values $v_{i,j,k} \in \mathbb{R}$ are attached to grid points, defining the volume data as a collection of voxels $(x_{i,j,k}, v_{i,j,k})_{i,j,k=0,1,\dots}$ We call the scalars $v_{i,j,k}$ *voxel values*. A *cell* in the 3D grid has eight voxels at its corner grid points. For a cell $C_{i,j,k}$ with grid point $x_{i,j,k}$ at its lower, left, front corner (see Figure 1) the corresponding eight voxel values are collected in a set $V_{i,j,k}$

Let $\phi : \mathbb{R}^3 \to \mathbb{R}$ be a continuous interpolation function of these volume data, i.e., $\phi(x_{i,j,k}) = v_{i,j,k}$ for all grid points $x_{i,j,k}$. This interpolation is required to have the property that the values of ϕ in a cell are restricted to the interval spanned by the corresponding voxel values. From the continuity of ϕ it follows that the range of ϕ in a cell $C_{i,j,k}$ is

$$\phi(C_{i,j,k}) := \{\phi(x) \mid x \in C_{i,j,k}\} = [c_{\min}, c_{\max}],$$

where we have set

$$c_{\min} = \min V_{i,j,k}, \quad c_{\max} = \max V_{i,j,k}.$$

Now let an isovalue $c \in \mathbb{R}$ be given. The isosurface for the isovalue c is given by the solution set of the equation

$$\phi(x) - c = 0.$$

Strictly speaking it is not clear that the solution set really is a surface, but we do not discuss this issue here. Then the

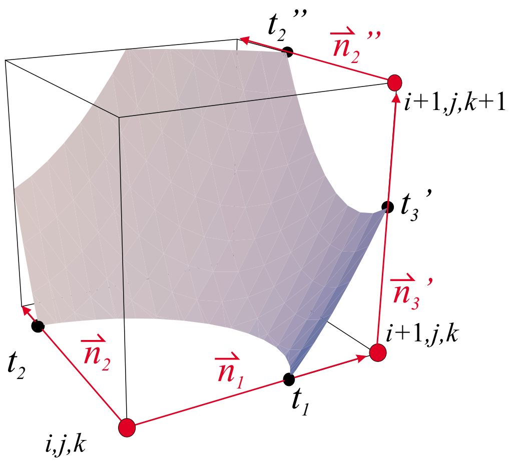

Figure 1. Notation for intersection points in cell $C_{i,j,k}$.

first step of the isosurface computation is the cell extraction, i.e., reporting all cells with c contained in the corresponding interval $[c_{\min}, c_{\max}]$. Actually, it is sufficient to report cells with c being in the half open interval, $c \in [c_{\min}, c_{\max})$. We call these cells *intersecting*. All other (non-intersecting) cells can be regarded as "empty" containing no part of the isosurface. In the second step an isosurface generator processes the intersecting cells $C_{i,j,k}$ one by one, computing and rendering

$$\phi^{-1}(c) \cap C_{i,j,k}.$$

Commonly, the isosurface within each cell is modelled by one or more polygons whose vertices are on straight lines connecting grid points of the cell. Thus, the isosurface renderer computes solutions to equations

$$\phi(x_{i,j,k} + t\,\vec{n}_\alpha) - c = 0, \quad t \in [0,1] \tag{1}$$

where $\vec{n}_\alpha$ is a 3D vector pointing from $x_{i,j,k}$ to another grid point of the cell. For a solution $t \in [0,1]$ of (1) we have that the point

$$x = x_{i,j,k} + t\,\vec{n}_\alpha$$

is an intersection point of the isosurface and the straight line connecting the grid point $x_{i,j,k}$ with another cell grid point in the direction $\vec{n}_\alpha$, i.e., $x_{i,j,k} + \vec{n}_\alpha$.

The popular "marching cubes" algorithm [9] uses the three direction vectors $\vec{n}_\alpha \in \mathcal{N} := \{\vec{e}_1, \vec{e}_2, \vec{e}_3\}$, where $\vec{e}_1, \vec{e}_2, \vec{e}_3$, are the unit vectors in the three coordinate directions. The isosurface renderer is able to produce all necessary polygons based upon these intersection calculations. Thus, if a surface decoder can produce the complete list of these intersection points, an isosurface renderer may proceed to compute and render the surface.

For the computation of the intersection points of the isosurface with the grid edges we may assume that the interpolation function is linear on the edges of intersecting cells. Therefore, equation (1) is solved for t by

$$t_\alpha = \frac{c - \phi(x_{i,j,k})}{\phi(x_{i,j,k} + \vec{n}_\alpha) - \phi(x_{i,j,k})} \tag{2}$$

where $t_\alpha \in [0,1]$ is guaranteed. For an encoding of the isosurface it suffices to encode a set of 5-tupels of the form

$$(i, j, k, \vec{n}_\alpha, t_\alpha) \tag{3}$$

in which the components are from the proper ranges as explained above, i.e.,

$$\begin{gathered} i \in \{0, \ldots, i_{\max} - 1\}, \\ j \in \{0, \ldots, j_{\max} - 1\}, \\ k \in \{0, \ldots, k_{\max} - 1\}, \\ \vec{n}_\alpha \in \mathcal{N}, \\ t_\alpha \in [0,1]. \end{gathered}$$

The decoder is able to regenerate all intersection points $x_{i,j,k} + t_\alpha \vec{n}_\alpha$ from which the marching cubes algorithm may produce polygons. Let us call this set of 5-tupels the *code list*.

The t-values $t_\alpha \in [0,1]$ are real numbers and must be quantized for encoding. To this end we may assume that the voxel values are given by integers ranging from 0 to some maximal number $2^m - 1$. Typically these voxel values stem from measurements or are rounded results of numerical simulations. A simple error analysis shows that in such a case it suffices to encode unformly quantized t-values using m bits as well [11].

The basic encoding scheme for the code list is as follows. We split the encoding of the code list into three parts, the first for the (i,j,k)'s, the second for the $\vec{n}_\alpha$'s, and the third for the remaining information, the t-values.

The (i,j,k)-triples from the code list correspond to grid points and may conveniently be stored in an octree structure. The octree is defined for a cubic volume with a side length given by a power of 2 greater or equal to $\max(i_{\max}, j_{\max}, k_{\max})$, similar to the branch-on-need octrees in [17]. Every node in the octree corresponds to a cubical array of grid points and is encoded using one bit with the following conventions. For an inner node the symbol 0 indicates that the subtree rooted at the node is empty containing no grid points of the code list. This implies that the entire subtree (below the node) can be pruned. The symbol 1 at an inner node indicates the opposite case, i.e., some leaf node below corresponds to a grid point that is a member of the code list. The symbol 0 at a leaf node indicates that the grid point is not in the code list, while the symbol 1 specifies a grid point in the code list. The resulting pruned octree is encoded by traversing it in depth-first order outputting the binary symbols encountered at each node. As side information the depth of the tree must be transmitted before.

For each of the encoded grid points in the first part above there may be one to three 5-tupels, since there are three possible $\vec{n}_\alpha$-vectors in $\mathcal{N}$. This amounts to seven different cases. Therefore, to encode the $\vec{n}_\alpha$'s, we can equivalently encode a sequence of symbols from the alphabet $\{0, \ldots, 6\}$. Likewise for the third part, the t-values, we encode a sequence of symbols from the alphabet $\{0, \ldots, 2^m - 1\}$.

Finally, the resulting symbol sequences of all three encoding phases are passed through a standard entropy coder which produces a compressed bit stream and writes it to the file system. In our implemention we have employed the zlib functions used by the gzip program [3].

The decoder reads the input stream, carries out the entropy decoding, constructs the octree for the grid points and then interprets the remaining symbols. From the results it computes the corresponding intersection points and stores them in a hash table for further processing by the isosurface renderer.

3. Efficient encoding

Tests using isosurfaces from CT data with 8 data bits per voxel resulted in compressed binary files of which about 62% corresponded to the encoding of the t-values. Thus, we first consider improving the encoding of the t-values. We consider three possible approaches: context modelling and encoding prediction residuals using either prediction by least squares or by surface relaxation.

3.1. Context modelling and coding

The principle of context modelling [10] allows a more efficient coding of the stream of t-values. Intersecting grid edges are classified into a small number of contexts, and corresponding t-values can be encoded using the probability table relating to its context class which on average should exhibit a lower entropy. We propose to use the following simple context model. A grid edge has two grid points at its ends. At each of these end points four other grid edges emanate orthogonally from the given edge (2). We collect these eight emanating orthogonal edges and check whether they also intersect the isosurface. This gives $2^8 = 256$ classes. However, we may exploit the symmetry that is inherent in a regular structured grid. We identify those configurations that be obtained from each other by a rotation around the given grid edge by a multiple of 90 degrees, by a 180 degree rotation around the vertical axis through the center point of the grid edge, by an reflection at the same center point, or a combination of these operations, see Figure 2.

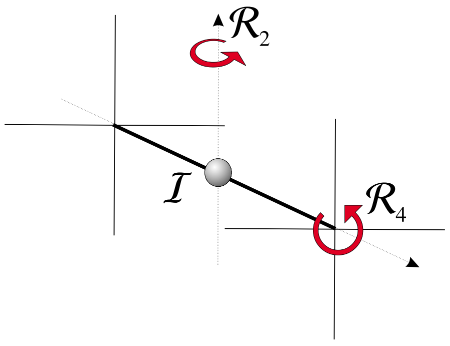

Figure 2. Symmetry of an edge in a cartesian grid: rotation $\mathcal{R}_4$ by multiples of 90 degrees, rotation $\mathcal{R}_2$ by multiples of 180 degrees, and relection at inversion center $\mathcal{I}$.

After these identifications only 34 of the initial 256 classes remain which are shown in Figure 3. When applied to several large volume data sets we found that only 6 of the 34 classes were actually occupied. All others, although theoretically possible, were not observed, probably because they would presume surfaces with very high curvature.

When encoding a t-value for an intersecting edge, one first determines the corresponding context from the 34 possible ones. Then, if the transformation of the configurtation required an odd number of 180 degree rotations (of type $\mathcal{R}_2$) or reflections (of type $\mathcal{I}$) we must use the properly transformed t-value, i.e., $\tilde{t} = 1 - t$. Otherwise, t is used, i.e., $\tilde{t} = t$.

The context classes are designed with the goal to obtain probability distributions with decreased entropy allowing more efficient entropy encoding. On the decoder side the contexts can be reconstructed from the octree and from the list of corresponding $\vec{n}_\alpha$-values, allowing decoding of the entropy coded t-values.

3.2. Prediction residual coding using least squares fitting

In prediction residual coding t-values are predicted (and quantized) using an appropriate estimation algorithm. The residual, i.e., the difference δt between the prediction $\tilde{t}$ and the true t-value is encoded. The goal is to design a prediction method that is capable of producing accurate predictions such that the entropy of the probability distribution of the residuals is less than entropy of the probability distribution of the original set of t-values.

We propose a prediction of the t-value along an intersecting edge between grid points $x_{i,j,k}$ and $x_{i',j',k'}$ based on a least squares approximation of the interpolation function $\phi(x)$. We search the K intersection points $y_i, i = 1, ..., K$,

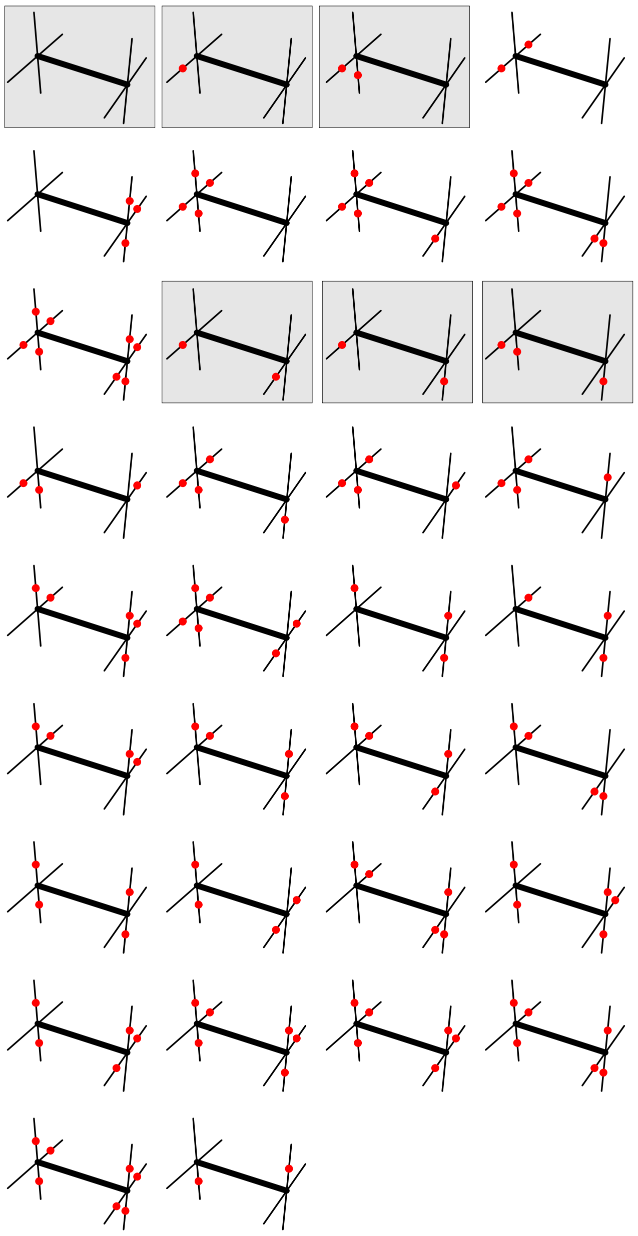

Figure 3. Visualization of the 34 context classes for an intersecting grid edge (thickened line). The dots on the other edges mark those edges that intersect the isosurface. For a collection of sample isosurfaces only six context classes were occupied indicated by grey background.

that are closest to the grid edge center and that have already been encoded. As a suitable function space for the approximation functions $\tilde{\phi}(x)$ we take truncated power series expansions, e.g., all polynomials of degree up to 3. The coefficients are computed as the linear least squares solution of the overdetermined system of K equations $\tilde{\phi}(y_i) = 0, i = 1, ..., K$. In order to avoid the trivial and useless solution $\tilde{\phi}(x) \equiv 0$ one may add a normalizing constraint, e.g., fixing the constant term to be equal to 1.

If $\tilde{\phi}(x_{i,j,k}) \cdot \tilde{\phi}(x_{i',j',k'}) < 0$, then $\tilde{\phi}$ has a zero on the grid edge, i.e.,

$$\tilde{\phi}((1-\tilde{t})x_{i,j,k} + \tilde{t}x_{i',j',k'}) = 0,$$

which can be computed numerically, for example, by using the bisection method. In this case the prediction residual $\delta t = t - \tilde{t}$ is encoded, otherwise the original t-value is used.

3.3. Prediction residual coding using surface fairing

As an alternative to local prediction using a least squares approximation for each intersection point we consider a global procedure by surface fairing. After transmission of the octree the decoder can correctly reconstruct the topology of the 3D mesh using arbitrary chosen t-values, say equal to 0.5 in all instances. Due to the underlying grid structure the resulting surface exhibits aliasing artifacts. For example, the corresponding surface mesh for a sphere as an isosurface is shown on the left in Figure 4. We then apply a surface fairing algorithm with the constraint that vertices are allowed to move only along their corresponding grid edges, see Figure 5. The resulting mesh for the sphere model is shown on the right of Figure 4.

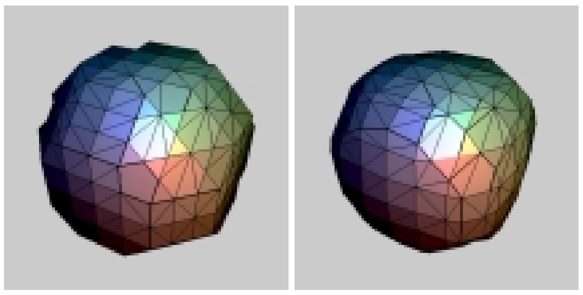

Figure 4. Polygon mesh of a sphere in a grid with low resolution, the initial polygons derived form the intersection parameter $t = 1/2$ (left) and the relaxed polygon mesh.

For the surface fairing we choose a method introduced by Taubin [12]. The surface fairing proceeds in two steps, in a first pass each vertex is attracted by its nearest neighbors

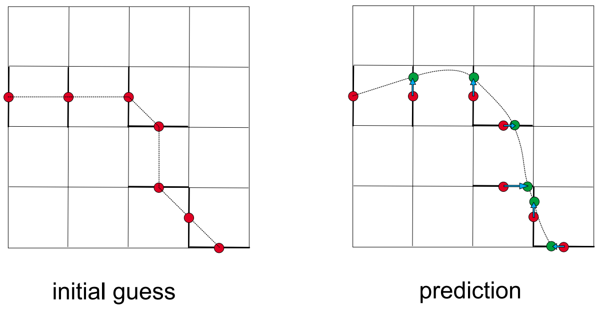

Figure 5. Scetch of the restricted Taubin fairing process on a cross section of the surface.

in the polygon mesh, in a second pass the point is repelled by its neigbor vertices. In each pass the force computed for a vertex is projected on the grid edge before it is applied to the vertex. If due to the movement the vertex passes over one grid point and thus leaves the grid edge we clamp the vertex to the corresponding grid point. This way the restriction of the surface fairing is enforced. Vertices can move only along and within their corresponding grid edges. The original algorithm [12] has no such restriction. This process of attraction and repulsion is repeated L times. The result is a fair surface (see Figure 4, right). For the estimated points the intersection parameter is $\tilde{t}$ is computed, and the difference $\delta t = \tilde{t} - t$ is coded.

The details are as follows. Consider a vertex x on the grid edge from $x_{i,j,k}$ to $x_{i',j',k'}$, the set $\mathcal{M}$ of m indices k such that vertex x_k is a neighboring vertex of x in the mesh for all $k \in \mathcal{M}$. Let $\vec{n} = (x_{i,j,k} - x_{i',j',k'})/||x_{i,j,k} - x_{i',j',k'}||$ be the normalized vector pointing in the direction of the grid edge. For the attraction step for vertex x we compute a displacement $\Delta x = \frac{1}{m}\sum_{k\in\mathcal{M}} x_k - x$. We project the displacement onto the grid edge, i.e., we compute $\Delta x' = \langle \Delta x, \vec{n}\rangle\, \vec{n}$ where $\langle \cdot, \cdot \rangle$ denotes the scalar product of two vectors. Finally, the new position of the vertex x is defined as $x + \lambda \Delta x'$ where $0 < \lambda < 1$ is a parameter of the method. If the updated vertex is outside of the grid edge, then it is clamped to one of the grid points $x_{i,j,k}$ or $x_{i',j',k'}$, whichever is closer.

All vertices of the mesh are processed in this way. The repulsion step is similar with the only difference in the update step in which a vertex x is replaced by $x + \mu\Delta x'$. Here μ is a second (negative) parameter of the method, related to λ by $\frac{1}{\lambda} + \frac{1}{\mu} = \mathrm{const}$ where the constant should be chosen smaller than 1 to ensure the stability if the method [12]. The attraction and repulsion steps are alternatingly iterated a number of times. The procedure differs from that in [12], the formula for Δx is simpler and the projection and clamping is introduced here to keep the vertices on the grid edges.

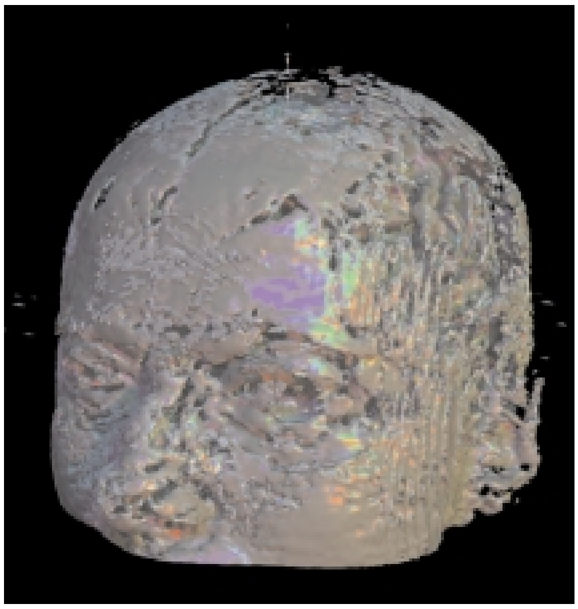

Figure 6. The isosurfaces of the CT data for $c = 70, 90, 110$ **as three transparent surfaces.**

4. Results

We implemented all three encoding methods for the t-values together with the other parts of the encoding. We applied the compression techniques to isosurfaces in a volume data set stemming from a CT scan of a human head. The scan has a resolution of $250 \times 192 \times 168$ voxels with eight bit gray levels. Figure 6 shows several (transparent) isosurfaces together. These surfaces are very complex. For instance, the isosurface for the isovalue $c = 70$ consists of 928,681 polygons that form 16,851 connected components. The t-values are quantized and represented by 8 bits each. The corresponding distribution and its entropy is given in Figure 7.

Due to the discrete nature of the voxel values the distribution is not uniform. For example, the value $t = 0.5$ is occuring about 32,000 times, a count almost 10 times as high as on average for all 256 values of t. This is because voxel values at neighboring grid points equal to $70 - k$ and $70 + k$ for $k = 1, 2, 3, ...$ lead to a t-value of 0.5. For other t-values the corresponding voxel value configurations are fewer and less likely.

We begin by reproducing the results for the three different proposed methods for encoding the set of t-values, using context modelling, prediction by implicit surfaces, and prediction by surface fairing applied to the mesh generated by the octree.

For the prediction method based on least squares function fitting we found that quadratic or cubic function fitting

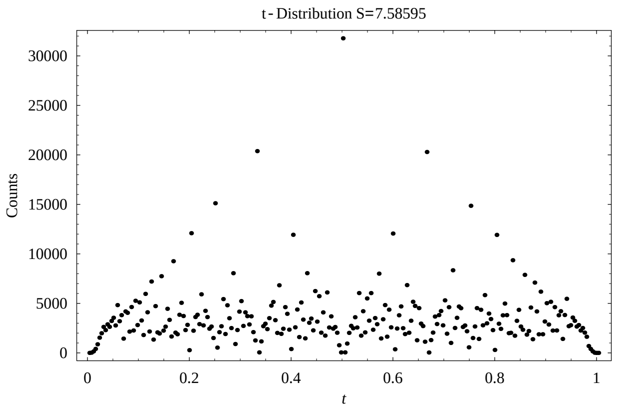

Figure 7. The distribution of the intersection parameters t, quantized to 8 bits. The (first order) entropy is given as S.

did not yield better results than fitting an affine linear function to the K data points. Also choosing $K > 4$ did not yield better results than with $K = 4$. Therefore, we produce the distribution of the perdiction residuals in Figure 8 only for the case of affine linear functions obtained from four nearest neighbor sample points each. The implementation of the fitting algorithm was based on [18]. We found that in 82% of all cases the prediction was located in the correct expected grid edge. Nevertheless, the result was not encouraging, the entropy of the probability distribution of prediction residuals was even larger than that corresponding to the original t-values. Thus, we do not report complete coding results with this approach.

For the method based on prediction residuals obtained by restricted Taubin surface fairing we obtained similar results as by prediction using function fitting. In the fairing algorithm we used the parameters $\lambda = 0.63$, $\mu = -0.67$ and applied $L = 80$ relaxation steps. Figure 9 shows the distribution of the resulting prediction residuals. Again, the entropy of the residuals was greater than that of the original t-values, and therefore we do not proceed to present complete encodings based on this prediction method.

In contrast to prediction coding we found that the method with context modelling did indeed yield an improvement over the isosurface encoding as given in our previous paper [11]. Figure 10 displays the distribution of the t-values in all of the contexts that were found in the data set. In most cases the first order entropies are lower than for the set of all t-values together. Overall, the conditional entropy was 7.54 as opposed to the entropy 7.59 for the full set. This seems to indicate only a small possible savings of 0.05 bits per value. However, since there are 923 thousand t-values to be coded in this data set he savings add up to a significant

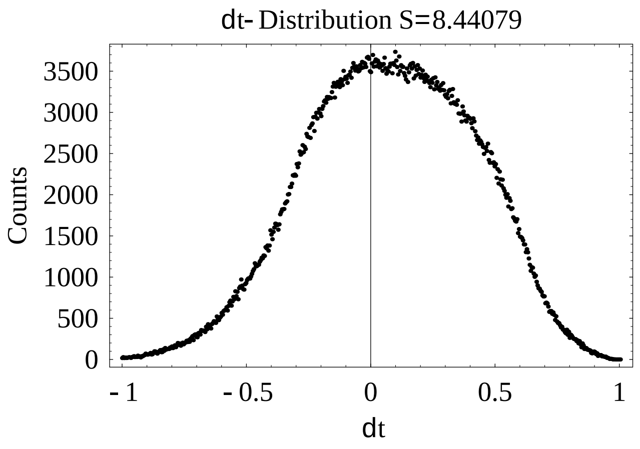

Figure 8. The distribution of the prediction residual δt based on the method of least squares function fitting.

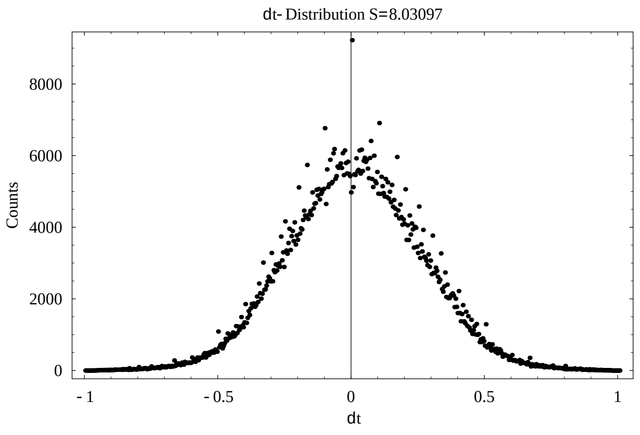

Figure 9. The distribution of the prediction residuals δt obtained from the restricted Taubin fairing.

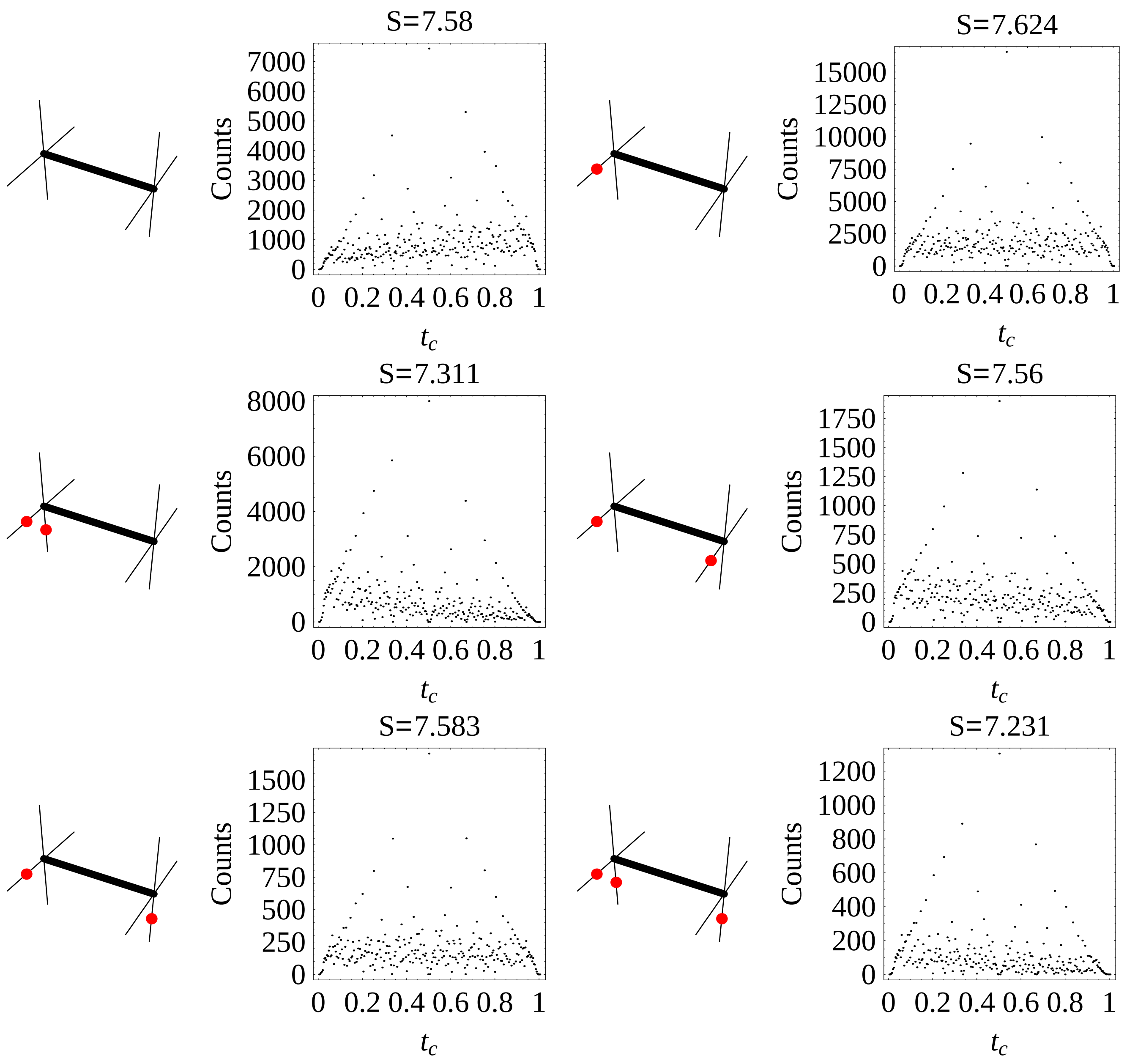

Figure 10. The distributions of the t-values for the CT-head with the corresponding edge configurations.

amount. Moreover, when the t-values are quantized using fewer than 8 bits more significant savings can be had, as we show in the following.

For the implementation of the context coding we applied a simple strategy that avoided programming a context based arithmetic coder. We simply reordered the stream of t-values by sorting it classwise and then applied gzip compression. This was very effective.

In our empirical studies we compare seven encoding methods.

1. **ASCII VRML.** The ASCII VRML format served as a basis for comparison. We ensured that files in this format contained only the topology and the geometry without surface normals or properties.
2. **Gzipped ASCII VRML.** ASCII VRML files can be entropy coded using the gzip program, for example, yielding a significant savings compared to the raw ASCII files.
3. **Method in [5].** The method as described in Section 1.
4. **Virtue3D.** Commercial compression software of Virtue Ltd. (www.virtue3d.com) based on geometry compression algorithms in [16].
5. **VRML compressed binary.** The binary VRML encoder [13, 15] is proposed for a geometry compression standard in MPEG-4.
6. **Our previous method [11].** This is our encoding method as given in Section 2 without context modelling.
7. **Method with context modelling.** This is our encoding method as given in this paper.

The Virtue3D and the binary VRML coders take ASCII VRML files as input, while our coder and that based on [5] work with the original volume data, respectively the set of intersecting cells. The binary VRML coder quantizes vertex coordinates using a representation with 12 bits. To match this choice in the context of our coder (and that based on [5]) we used $m = 4$ bits for encoding t-values, since the voxels were given with 8 bits per coordinate which together with the 4 bits for t-values gave us the same spatial 12-bit resolution for vertex coordinates. With this choice we ensured that the decoded isosurfaces exhibit the same precision for all methods except for the first two VRML-based methods.

The binary VRML compression program has special requirements in cases where the surface consists of several connected components. This surely was the case in our tests. To solve the problem a second program for the optimization of the input files was applied beforehand [13].

isovalue c:	150		70	
cell edges:	408,940		922,947	
polygons:	416,202		928,681	
	kB	CR	kB	CR
VRML	27,059	1.0	62,299	1.0
gzip VRML	6,182	4.4	14,148	4.4
method [5]	2,853	9.5	6,397	9.7
Virtue3D	1,911	14.2		
bin. VRML	1,325	20.4	3,717	16.8
method [11]	598	45.2	1,311	47.5
new method	500	54.0	1,105	56.4

Table 1. Compression results for isosurfaces for different isovalues c of the CT scan in Figure 6. In the top part the number of intersecting cell edges and the number of polygons is given. Below the file size in kB, and the compression ratio (CR) relative to the ASCII VRML format is given for each isosurface and encoding method. Some numbers for the Virtue3D coder are missing because the coder ran out of memory and did not complete the compression.

Unfortunately, it turned out that the surfaces for isovalues $c < 70$ had so many polygons (a million and more), that the preprocessing program ran out of memory on a Windows NT workstation with 512 MByte RAM. The Virtue3D coder had a similar problem, running out of memory already when processing isosurfaces of 855,000 or more polygons.

Table 1 summarizes the results shown for two isosurfaces corresponding to two isovalues c. The complexity of the surfaces ranged from 416,000 to 928,000 polygons resulting in ASCII VRML file sizes from 27 to 62 MBytes. Entropy coding by gzip achieved a compression by a factor of 4.4 in both cases. The Virtue3D coder produced compression ratios of 13 to 14 for the cases where it was able to produce output. Better still was the compressed binary format with compression ratios from 17 to 21. However, our method clearly outperformed all others providing compression ratios from 54 to 56. This amounts to an improvement in terms of compression ratio over the state-of-the-art represented by the binary VRML format by a factor of 2.6 to 3.4 and with respect to our previous approach [11] by about 20%.

Our isosurface coder is fast in comparison to the other coders. On our SGI PC 320 with 512 MB RAM and a 450 MHz Pentium III processor the encoding of the isosurface for isovalue $c = 110$ required 113 seconds plus a few seconds for the context calculations (exact numbers appear in the final version of the paper). On the other hand the encoding times using gzipped VRML, Virtue3D, and binary

VRML (including the preprocessing) were 230, 1320, and 711 seconds respectively. For the decoding and the polygon construction our current not optimized implementation required 174 seconds. We cannot provide decoder timings for the other geometry compression methods, because the decoders are embedded in a browser environment, and it is not possible to isolate the decoding time. Moreover, the Virtue3D decoder is not in the public domain.

It would not be fair to conclude that the current geometry compression methods are inferior in general. Those coders were designed for unconstrained polygonal surfaces, while our coder is restricted to isosurfaces from structured volume data.

5. Discussion and future work

The compression results may be further enhanced by using context models also for the code that is not for the t-values, i.e., for the octree and the $\vec{n}_\alpha$ symbols. The probability tables for the arithmetic coding of the bits corresponding to the octree can be conditioned on the number of already coded cells that bound the block of cells corresponding to the current code bit. If several intersecting cells bordering the boundary of the current block of cells are already known to the coder and decoder, then the probability that the block of cells also intersects the isosurface is very large, allowing to encode the corresponding symbol "1" using much less than one bit. The encoding of the intersecting edges in non-empty cells can be conditioned on the number of known intersecting cells that share the edge. These encoding results will appear in the final version of this paper.

One may introduce multiresolution capabilities to our encoding method by adding information defining surface approximations at all levels of the octree resolution. The required overhead in terms of space and the achievable approximation quality remain to be investigated.

A disadvantage of our coder is that it is designed only for regular structured volume data. It is possible, however, to extend the method to the case of curvi-linear volume data. For this purpose it suffices to separately encode the geometry of the curvi-linear grid, which may or may not require large space depending on the complexity of the grid geometry. A further extension to unstructured volume data is beyond the scope of our method, though.

6. Summary

We have extended our previous method [11] for geometry compression tailored for isosurfaces stemming form regular structured volume data. The method encodes a set of grid points that identifies the non-empty cells in the volume along with the intersection points of the isosurfaces with the cell edges. Our coder is simple and handles all cases of surface topology equally well. We have improved compression performance over our previous method by about 20% by designing an appropriate context model for the intersection values. Compression results for isosurfaces from original CT volume data demonstrated a compression performance that was faster and up to 3.4 times stronger than the best current state-of-the-art 3D geometry coder (binary VRML). We expect further improvements by extending context coding to other portions of the isosurface code.

Acknowledgment. The authors thank Frithjof Kruggel for the CT data.

References

[1] Z. Karni, C. Gotsman. Spectral coding of mesh geometry. *Computer Graphics* 34 (2000) 279–286.

[2] R. Klein, S. Gumhold. Data compression of multiresolution surfaces. In *Visualization in Scientific Computing '98*, pages 13–24. Springer, 1998.

[3] P. Deutsch. Gzip file format specification version 4.3. Technical report, Aladdin Enterprises, 1996.

[4] O. Devillers, P.-M. Gandoin. Geometric compression for interactive transmission. In *IEEE Visualization Proceedings*, 2000.

[5] K. Engel, R. Westermann, T. Ertl. Isosurface extraction techniques for web-based volume visualisation. In *IEEE Visualization Proceedings*, 1999.

[6] S. Gumhold. Compression of discrete multiresolution models. Technical Report WSI-98-1, Wilhelm-Schickard-Institut für Informatik, University of Tübingen, January 1998.

[7] J. Hartman, J. Wernecke. *The VRML 2.0 Handbook*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1996.

[8] B. Kronrod, C. Gotsman. Efficient coding of non-triangular meshes. In *Proceedings of Pacific Graphics*, Hong-Kong, October 2000.

[9] W. Lorensen, H. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics* 21 (1987)163–169.

[10] J. Rissanen, G. G. Langdon, Universal modeling and coding, IEEE Trans. on Information Theory 27 (1981) 12–23.

[11] D. Saupe, J.-P. Kuska, Compression of isosurfaces for structured volumes, in Proc. Vision Modelling and Visualization 2001, Th. Ertl et al (eds.), IOS Press, Amsterdam, 2001.

[12] G. Taubin, A signal processing approach to fair surface design, in R. Cook, editor, SIGGRAPH 95 Conference Proceedings, Annual Conference Series, pages 351–358. ACM SIGGRAPH, Addison Wesley, Aug. 1995.

[13] G. Taubin, B. Horn, F. Lazarus, E. Talnykin, Mitra. Vrml compressed binary format overview. `http://www.research.ibm.com/vrml/binary/index.html`, October 1997.

[14] G. Taubin, W. Horn, F. Lazarus, J. Rossignac. Geometry coding and VRML. In *Proceedings of the IEEE* 86 (1998) 1228–1243.

[15] G. Taubin, J. Rossignac. Gemometric compression through topological surgery. Technical Report RC-20340, IBM Research Division, 1997.

[16] C. Touma, C. Gotsman. Triangle mesh compression. In *Proceedings of Graphics Interface*, Vancouver, June 1998.

[17] J. Wilhelms, A. Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992.

[18] J.H. Wilkinson, C. Reinsch, Linear Algebra, Vol. II of Handbook for Automatic Computation, Springer-Verlag, New York, 1971.

[19] X. Zhang, C. Bajaj, Q. Blanke, D. Fussell. Scalable isosurface visualization of massive datasets on COTS clusters. In *Proceedings IEEE Symp. Parallel and Large-Data Visualization and Graphics.* San Diego, Oct. 2001.