Compression of Time Varying Isosurfaces

Ilya Eckstein
University of Southern California
ilyaeck@usc.edu

Mathieu Desbrun
Caltech
mathieu@caltech.edu

C.-C. Jay Kuo
University of Southern California
cckuo@sipi.usc.edu

ABSTRACT

Compressing sequences of complex time-varying surfaces as generated by medical instrumentations or complex physical simulations can be extremely challenging: repeated topology changes during the surface evolution render most of the previous techniques for compression of time-varying surfaces inefficient or impractical. In order to provide a viable solution, we propose a new approach based upon an existing isosurface compression technique designed for static surfaces. We exploit temporal coherence of the data by adopting the paradigm of *block-based motion prediction* developed in video coding and extending it using local surface registration. The resulting prediction errors across frames are treated as a static isosurface and encoded progressively using an adaptive octree-based scheme. We also exploit local spatiotemporal patterns through context-based arithmetic coding. Fine-grain geometric residuals are encoded separately with user-specified precision. The other design choices made to handle large datasets are detailed.

Keywords: Surface Compression, Isosurface Compression, Dynamic Surface Compression

1 INTRODUCTION

The realm of dynamic surface compression, *i.e.*, encoding and decoding of surface motion in space and time, has been receiving much less attention than its static counterpart. In contrast, the field of signal processing has witnessed a natural evolution from digital images to digital video over the last decade, and as a consequence, the digital encoding of video sequences is now common practice. Now that dynamic 3D contents are generated at a fast growing rate in various applications ranging from characters in movies and electronic games to scientific simulation, the need for dynamic surface compression is becoming greater.

Not all dynamic 3D data need specific compression techniques. Computer-generated movies, while extensively using 3D animated models and gaining an unprecedented popularity, are treated as video once produced: the specific camera position and lighting conditions of each scene are parts of the artistic value of the movie, so this content is rarely encoded in 3D. Another source of 3D data is distributed electronic games, where characters are typically animated using motion parameters (either precomputed or generated at runtime) applied to a reference object: here again, there is no specific need for dynamic surface compression.

There are, however, a number of other sources for which such compression is vital. Scientific simulation data typically requires significant computational resources to be generated (for fluid mechanics simulation for instance), and a vast amount of space to store the results. Similarly, acquired dynamic data such as 3D medical images and sequences of range scans demand a great deal of storage space as well. Compressing this type of animated 3D content is, in these cases, key to their proper handling.

Although some work has been done on compression of fixed-topology mesh animation and time-varying volume data, we will focus on a specific type of dynamic 3D data, namely *time-varying isosurfaces*. Isosurfaces are a useful representation for visualizing volumetric content, be it in scientific simulation or medical applications; this is particularly true for time-varying volumetric data, where one is often interested in observing the evolution of one or more specific iso-contours (corresponding to particular values of intensity, density, etc.), rather than the volume in general. Unfortunately, the size of the data often becomes the bottleneck, even once the volume dataset has been converted to a polygonal mesh. In particular, for applications involving remote data monitoring and analysis, high-resolution isosurfaces become impractical without *efficient compression*. The challenge in compressing isosurfaces is to cope with the level of geometric and topological complexity they often have, as deformation may be arbitrary and topology can change over time. Consequently, *none* of the existing dynamic mesh coding methods are applicable to this case as we explain next.

1.1 Background

Existing methods for compression of dynamic polygonal meshes [1, 5, 9, 12, 14, 17] rely on known inter-frame vertex correspondences—typically, the mesh connectivity is fixed in time. This specificity renders those methods unsuitable in our context, where not only the surface sampling but also the surface topology may vary considerably between frames. Anuar and Guskov [2] proposed to solve the correspondence problem by converting the input meshes to adaptive signed distance volumes, followed by motion estimation using optical flow. Consequently, motion is estimated as a dense vector field, impractical for compression purposes.

Other related work has been done in volumetric data compression. Mascarenhas *et al.* [19] suggest a streaming scheme for volumetric grids, where voxels are ordered to facilitate extraction of isosurfaces for subsequent isovalues, a very valuable feature for data analysis. However, their approach does not support encoding of specific (subset of) isosurface(s) and their motion in time, rendering it inefficient for our purpose. Guthe and Straßer [10], and Sohn *et al.* [23] compress time-varying *volumetric* data using wavelet transforms and motion estimation, but do not report compression rates of single isosurfaces. Another recent work by Gregorsky *et al.* [8] is concerned with the extraction of time-varying isosurfaces from compressed volume data. There again, compression is applied to the whole raw data, rather than just a single isosurface. Finally, Ibarria *et al.* [11] suggest a method for out-of-core compression of *n*-dimensional scalar fields, mostly applied to 3D and 4D datasets. Their main contribution is the so-called Lorenzo prediction, which is a generalization of the parallelogram predictor to an arbitrary dimension. While this predictor could in theory be used to predict corresponding inside/outside grid values for two subsequent frames, they provide no evaluation for values with a binary range.

1.2 Contributions

In this paper we extend the concept of *block-based motion estimation*, traditionally used in video coding, for compression of dynamic isosurfaces extracted from a series of uniformly sampled volumetric grids. Leveraging the progressive encoding method of Lee *et al.*[16] for static isosurface compression, we provide a codec allowing rapid, progressive display of the sequence during decoding. The unstructured nature of time-varying isosurfaces allows our algorithm to deal naturally with surfaces of high geometric complexity and dynamic topology, unlike any of the existing codecs.

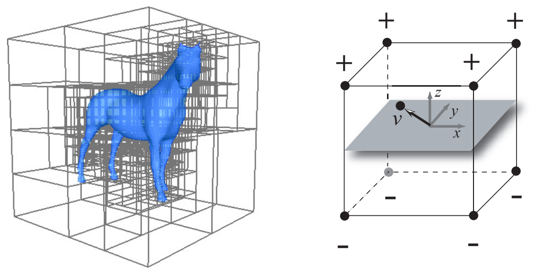

Figure 1: *Static Isosurface Compression: Each leaf of the inside/outside octree (left) intersects the isosurface, and the position of the vertex (obtained through Dual Contouring) is encoded using a sign-based local frame as detailed in [16].*

2 The Static Isosurface Compression Method

Before delving into the details of the dynamic case, we first review the static isosurface compression scheme that we will build upon, and establish terminology. A more detailed coverage can be found in the original paper [16].

Input We are given a 3D grid of uniformly sampled volume data of size $(2^n+1)^3$, along with an isovalue which determines the desired isosurface to encode.

Octree Generation and Surface Extraction Based on the input grid values, a *sign bitmap* of the given volume (i.e., a binary grid of inside/outside values) is generated. An octree representation of this sign bitmap is then constructed bottom-up: whenever all eight child cells are *homogeneous* (i.e., contains no sign change), they are deleted. This results in a tree has two kinds of leaf nodes: (a) coarse level leaves in the homogeneous regions; and (b) heterogeneous leaves, corresponding to the original grid cells intersected by the isosurface (the so-called *narrow band*). For surface extraction, the Dual Contouring method [13] is applied, assigning a vertex to each leaf cell, contrasting with the Marching Cubes algorithm [18] which positions vertices at edge intersections instead. Dual Contouring produces watertight meshes for any sparse or full octree description of the original bitmap and can even handle sharp features if necessary. In the remainder of this paper, we will call *connectivity* the adjacency structure of the octree, and *geometry* the positions of the vertices stored at the heterogeneous leaves of this octree as they roughly correspond to the connectivity and geometry of a usual polygonal mesh (see Figure 1).

Connectivity Encoding At this stage, the octree representation of the sign bitmap is encoded in a breadth-first fashion. Eight children of each cell correspond to 27 grid corners, each of them encoded using a context based arithmetic coder. For each homogeneous child, a *leaf bit* is encoded in a separate stream indicating whether the corresponding cell is indeed a leaf node—to let the decoder know that no subsequent data is needed for that particular subtree of the sign bitmap.

Geometry Encoding The sign bitmap alone is often sufficient to approximate the original surface, by simply applying Dual Contouring after assigning +/-1 values to outside/inside grid corners. A useful feature of this approximation is that the *distortion* (i.e., error introduced by the encoding process) is bounded by the grid cell size. For some applications, this level of precision may prove sufficient (surface smoothing techniques are often applied to improve the visual quality and the quantization error in many cases). Otherwise, *local geometric residuals* are encoded for each leaf cell to provide the final vertex positions. Each residual vector is encoded in a local coordinate frame, as the normal component typically bears more information than the tangential components, and therefore needs a better quantization accuracy. At each leaf cell, the local frame is given by the binary sign configuration of that cell (see Figure 1, right), leading to 256 possible cases, precomputed offline.

Discussion This algorithm provides, to the authors' knowledge, the most efficient compression of static isosurfaces. As such, it can be used on time-varying sequences on each successive isosurface. Obviously, this straightforward approach is rather limited as no temporal coherence in the sequence can be either detected or taken advantage of to reduce the bitrate. A modification of this approach to render the compression method more efficient is presented next.

3 Video vs. Dynamic 3D Surface Coding

Recall that the input to our algorithm is a series of 3D volumetric frames $\{F_i\}$, all sampled uniformly at the same resolution 2^n+1, paired with isovalues $\{\alpha_i\}$ determining the desired isosurfaces. The naïve choice of frame-by-frame compression mentioned above does not take advantage of the temporal coherence. We thus propose to rely on the paradigm of *block-based motion prediction* used in video coding to exploit *both* geometrical and temporal coherence.

3.1 Video Coding

In video coding (e.g., in the MPEG format), a common approach to encode a single frame at time *t* in a given video sequence is as follows (Figure 2):

- predict frame *t* based on frame *t*-1 through motion prediction of small pixel clusters called *macroblocks*;
- encode the estimated motion of each block;
- encode the prediction error image (i.e., the error between the prediction and the actual frame).

The prediction error image is encoded using JPEG (i.e., block-based DCT) or JPEG2000 (i.e., wavelets), two of the most efficient algorithms in image compression. Note that if two successive frames are too different as it happens in movies when a scene ends and another one starts, the motion prediction and correction can cost more than a direct encoding of the new frame; to deal with this case, each frame is labeled as an *I-frame* or a *P-frame*: I-frames are ***I***ndividually encoded, while P-frames are ***P***redicted from previous frames.

3.2 From Video to Dynamic 3D Isosurfaces

We now wish to mimic the approach used in video compression by first splitting the surface in small block-like parts, estimating local motion of the isosurface parts from the previous isosurface, then encoding the difference between the resulting prediction and the exact shape in a compact bitstream. The most delicate step in this encoding process is the motion estimation between two consecutive frames. Indeed, compression rates will be directly affected the quality of the estimation. Additionally we have to deal with 3D shape matching instead of pixel comparisons, rendering the issue significantly more complex. One simple strategy is to restrict

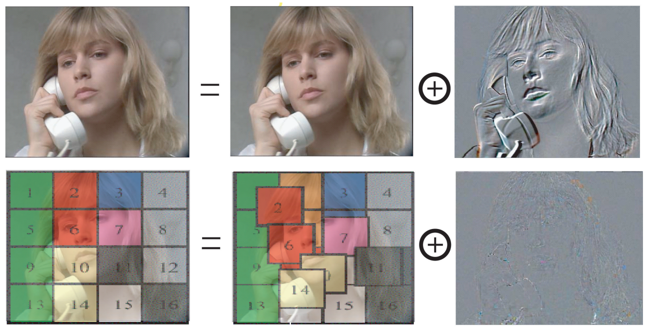

Figure 2: *Illustrating differential prediction vs. block-based prediction in MPEG: instead of recovering a video frame (top left) from the previous one (top middle) with a difference image (top right), a more efficient approach is to partition the new frame in blocks (bottom left), and track these blocks onto the previous frame to their best matching position (exaggerated in that figure for illustrative purposes): the encoding of the few block motion vectors and the resulting difference image (with much less range) requires less bits due to the strong temporal coherence present in usual video sequences.*

our attention to changes in the sign bitmap over time, thus reducing the problem to motion search in three-dimensional binary images. Then, for each macroblock in the predicted frame, we can search for the best matching block in the previous frame, within a limited search range. Matching error can be measured as the sum of absolute differences (SAD) over all the values in a macroblock, just as it is done in video compression. Alas, in general this approach is not very effective. To see why a more complex solution is needed, we have to understand the challenges that are specific to our problem:

- *Lack of Correspondence*: Since each isosurface is extracted independently, there are no correspondences between vertex positions of two consecutive shapes. As a result, even for a simple global translation between two subsequent frames, new vertex positions never reproduce the underlying motion exactly. Moreover, connectivity often changes from frame to frame, which makes tracking vertices impossible in practice.
- *Frame Difference*: In the case of 2D video, the difference between two almost similar frames F_{t-1} and F_t will typically produce an error image with a lower dynamic range compared to the original F_t, that is, an image much cheaper to encode. Unfortunately, this is no longer the case in our setting. Bear in mind that when encoding a isosurface, we are basically working with a binary inside/outside function. Consequently reduction of dynamic range is simply impossible: the difference between two frames $F_t - F_{t-1}$ is equivalent to the exclusive “or” between the two. In fact, the naïve approach of subtracting two binary images and encoding the resulting contour typically *increases* the amount of information to be encoded, as shown in Figure 3 (left).
- *Temporal Resolution*: Unlike most video codecs which assume a high frame rate (typical standards are 24-30 frames/second), here we have to deal with arbitrary time step, thus arbitrary deformation.
- *Improving the geometric rate/distortion curve*: Although the sign bitmap data provides a bounded-error approximation of the shape, overall compression rates are dominated by the geometric residuals (about 90%). Therefore, to achieve a significant compression gain, we need to focus on the latter component.

3.3 General Approach

With the above considerations in mind, we propose the following principles for compression of time-varying isosurfaces:

- Motion estimation needs to be as accurate as possible, and not limited by the grid resolution. We therefore cast this task as a local surface registration problem. As we are only interested in high quality prediction, low-confidence estimates will be discarded, and static encoding will be used in those regions instead. Details are discussed in section 4.2.
- Instead of attempting to track the isosurface directly, we adopt an indirect approach in which we track the surface distance field. Then for each voxel in the narrow band, we use the estimated distance values on its corners to predict a vertex position. See section 4.3 for details.
- As motion prediction errors tend to follow a different distribution from the static residuals, different categories of errors are quantized and encoded separately, as shown in section 4.4.

4 Compression thru Block-based Prediction

We begin describing our contribution by providing a simplified overview of the encoding/decoding scheme:

```
ALGORITHM:
Given two consecutive isosurfaces $S_i$ and $S_{i-1}$
    Estimate motion vectors $M$ from $S_{i-1}$ to $S_i$ (encoder only)
    Encode/decode $M$
    Predict $S_i$ based on $S_{i-1}$ and $M$
    Encode/decode prediction residuals.
```

We now describe the various steps of our encoding/decoding algorithm, pointing out the choices we made to render the technique as scalable as possible.

4.1 Isosurface Macroblocks

We chose to treat each frame as an array of *macroblocks* of user-defined size (e.g., $8 \times 8 \times 8$ voxels in most of our tests), and encode one set of motion parameters per macroblock. Note that while using simple blocks for motion prediction may be very suboptimal in some 3D animation scenarios, it becomes a practical choice when dealing with time-varying topology.

4.2 Motion Estimation

While video coding algorithms typically use pixel-based translational motion of blocks (i.e., displacements with granularity defined by a fixed fraction of the pixel size), this fixed granularity is too limiting in our context as our type of data carries also sub-voxel information. To achieve better motion estimation through accurate geometric fitting, we change the approach as follows.

Splitting the Grid in Blocks Given an isosurface S_i to be predicted at time step i, let B_i be the set of heterogeneous macroblocks intersected by S_i. We first partition the surface S_i into block-sized regions P_j such as $\bigcup_j P_j = S_i$ and $P_j \cap P_k = \emptyset$ for any $j \neq k$. That is, for each macroblock $b_j \in B_i$, we define P_j the subset of vertices of S_i that are in b_j.

Block Backtracking The goal of the next step is to find a region of the previous surface S_{i-1} (that both the coder *and* the decoder know at this point) that best approximate the shape present in a given block. As the deformation field between two successive frames can be arbitrary, we cannot expect to find a “perfect fit” between the shape of the surface within a block and a part of the previous surface. We will therefore only look for a good-enough fit; in other words, we wish to find a translation T_j for each block b_j that minimizes the squared distance metric between T_jP_j and a region of S_{i-1} (see Figure 3). Note that we restrict our search to *translational* transformation for the same reasons that video compression techniques use translation only: this simple temporal domain prediction technique (known as motion predictive coding in the context of video) is adopted because of its computational simplicity as well as its coding gain (very few parameters need to be sent). Since prediction is applied to local blocks in temporally adjacent frames, the union of translations provides a reasonable approximation.

Block-based Shape Matching To find the best local fit for each P_j, we employ a local registration approach where each macroblock patch is registered against the previous isosurface through the geometric optimization procedure proposed by Mitra *et al.* [21]. This method is a generalization of the Iterative Closest Point algorithm [7], offering improved stability and convergence rate. Although the original technique is designed for registration of point clouds, we adapted the algorithm to handle our isosurface shape matching as follows:

1. Given the previous isosurface S_{i-1} (as known to the decoder), construct an approximation $\widetilde{d^2}$ of the square of the distance to S_{i-1}. To achieve a high-quality approximation, we sample the squared distance to S_{i-1} at twice the original grid resolution.

2. Given the set of vertices P_j in the macroblock b_j, find a translation T that minimizes $\widetilde{d^2}(T(P_j), S_{i-1})$. This is done by a Gauss-Newton optimization procedure, employing a gradient descent step at each iteration [21]. Note that $\widetilde{d^2}$ gives us a closed form quadratic error potential for every point in space: as a result, a gradient descent direction can be easily found for every point $q \in P_j$, and a general minimization step can be computed by solving a least-squares linear system (see Appendix A for more details).

Algorithmic Details To compute and store the $\widetilde{d^2}$ field, we use the *d2Tree* octree-based structure [21], where each cell stores the polynomial coefficients of the locally fitted quadratic approximant. The fitting process involves solving a local linear least-squares system for each relevant cell, making the evaluation of $\widetilde{d^2}$ the most computationally demanding step in the encoding process. However, we render the process efficient by restraining the computations to within a fixed distance from the block: a perfect match with the previous surface is most unlikely to be at a long range, but rather nearby the block's spatial position. Moreover, we implemented a lazy evaluation of $\widetilde{d^2}$, i.e., approximants are evaluated on demand only, thus avoiding unnecessary computations for cells that are not used during the registration process. Finally, the least-squares matrix is the same for all non-boundary cells (one other advantage of only using translations!), and therefore we precompute its inverse once and for all. These optimizations save a significant amount of computation and memory, making the approach practical yet still robust.

Checking Quality of Match The best fit is only valid if it is a good-enough estimate; otherwise, sending the motion estimate followed by a large error correction may be require more bits than sending the right shape in the first place. Consequently, in cases when the final registration error of a block exceeds an acceptable threshold, we discard the prediction and issue a "no-motion" flag instead. The corresponding macroblock is then treated by the decoder as an I-block, i.e., encoded by the static method.

Handling Fast Motion So far we have assumed only moderate inter-frame deformations, i.e., low-speed motion. Naturally, the above algorithm can handle faster motion by simply increasing the approximation coverage around the surface. In practice, however, this change turns out to be too computationally extensive. An alternative approach is to find a coarse motion vector by a simple voxel-based search (MPEG-style) over the distance field which needs to be evaluated anyway, then refine the motion estimate step using the registration-based local search. Our fast voxel-based motion search extends the method of Chalidabhongse and Kuo [6] to the 3D case.

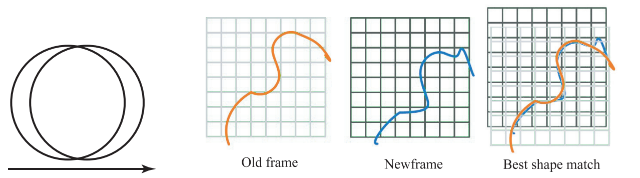

Figure 3: *Left, for a surface slightly shifted (here, a disk moved to the right), the difference contour between two time steps is more verbose than each of the steps. Right, shown is a local (i.e., block-sized) surface motion between two subsequent frames, and the result of a local registration procedure.*

4.3 Motion Compensation

From the previous frame and the motion vectors, the decoder must now start deducing the new frame: this process is typically referred to as *motion compensation* in video coding. We now review the main steps involved in our 3D extension.

Connectivity Encoding/Decoding We first proceed by sending the connectivity (i.e., the inside/outside function of all grid nodes) of the new frame using the static encoding described in [16]. Our tests has shown that there is very little to gain to try to predict connectivity from the motion vector, as the encoding of the full-blown information only requires a few bits (connectivity typically represents 10% of the total bitstream). This design choice also allows rapid progressive display of the decoded isosurface, which is not possible when connectivity and geometry bitstreams are interleaved.

Vertex Position Prediction Given the correct connectivity, the decoder knows that each heterogeneous cell needs to find an inside vertex position. As stated earlier, due to connectivity changes and sampling artifacts we cannot in general trace vertex positions directly. We therefore proceed on a cell-by-cell basis, trying to make a good prediction of each vertex position given the previous frame, the new motion vectors, and the new, correct connectivity. Two cases can happen:

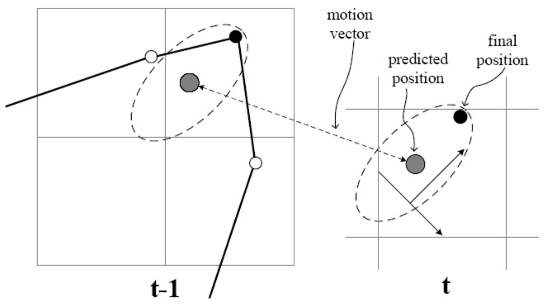

Figure 4: *Predicting vertex position. The grey vertex is extracted from the predicted distance function. The likelihood ellipsoid is aligned with the local frame, describing the expected error distribution. Using the motion vector, the position is mapped back to the previous frame and the prediction is refined using the nearest original vertex within the likelihood ellipsoid.*

- First, a prediction on the vertex position of a heterogeneous cell is used *only* if the predicted distance values in the narrow band agree with the actual connectivity. Otherwise, we know that any temporal prediction of the new vertex position is pointless: too much deformation has happened locally. Note that this case happens quite often for fast motion. For those unpredictable positions, the decoder simply expects the coder to send the correct position using the spatial encoding used in [16], i.e., using a normal and tangential components from a prediction purely deduced from the connectivity.
- Otherwise, the decoder first extracts a vertex position $\mathbf{x}_1$ from the predicted distance function. The final predicted position is obtained as follows. Using the motion vector, we map $\mathbf{x}_1$ back to the previous frame, and find the vertex $\mathbf{x}_2$ closest to it within the *likelihood ellipsoid*. The ellipsoid is defined in the local frame (as defined by the voxel connectivity) with its radii being parameters specified a-priori. These correspond to typical standard deviations of the error in the normal/tangential directions. If there is no such vertex, prediction is discarded. Otherwise, $\mathbf{x}_2$ is the predicted position. This heuristic has been, in all our tests, the most reliable way to filter out predictions that do not help reduce the total entropy. See Figure 4 for an illustration.

In fact, whether or not a prediction is available, our tests has shown that the tangential components of the prediction are mostly noise (therefore, not helpful for prediction): the decoder thus only uses the normal component of the prediction. Finally, we encode, send, and decode the geometric residuals, i.e., geometry prediction error, using the static approach from [16] (as we have already exploited temporal coherence), which completes the process. The next section focuses on arithmetic coding of different data components.

4.4 Context-based Arithmetic Coding

Further reduction in bit rates are obtained using context-based coding as explained next.

Motion Vector Encoding

As motion vectors represent the least dominant component in the overall bit rate, we use a straightforward encoding scheme. For each macroblock, a single bit flag is first generated, indicating if the block has a motion vector. Then for each motion vector, X, Y and Z components are uniformly quantized and encoded separately, using a context-based arithmetic coder. The context for each component is based on the values of the already encoded motion vectors of the immediate neighbors.

Connectivity Encoding

Our connectivity encoding scheme uses exactly the same context as the one for static isosurfaces [16]: based on our tests, this context appears to be superior to others.

Geometry Encoding

Geometric information, i.e., the positions of the vertices inside heterogeneous cells can also benefit from context encoding. The results reported in this paper have been generated by separating the encoding of the residuals for predicted vs. unpredicted vertices (see Section 4.3). Indeed, they show very different distributions, and are therefore better encoded using two streams. The stream for predicted vertices is best encoded using a Lloyd quantization on the residual, as the error distribution shows a significant peak with few outliers. Finally, the context used in both cases is the connectivity-based, 8-bit context used in [16].

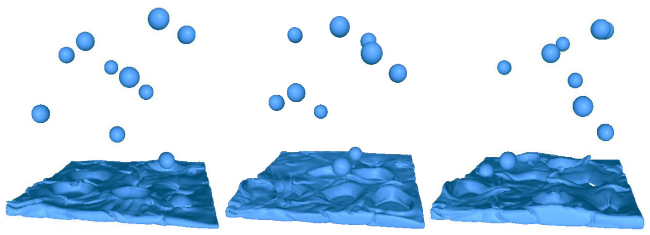
Figure 5: *Fluid simulation with falling droplets generating ripples and splashes.*

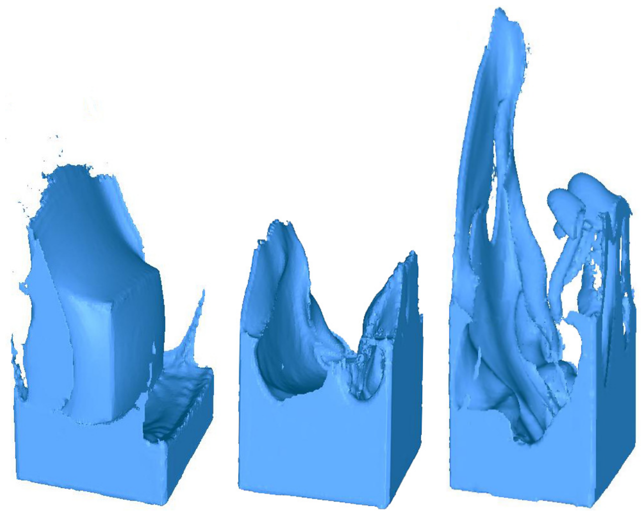
Figure 6: *Another simulation of splashing fluid, with high topological complexity.*

5 Results

We are faced with the same issues as the authors from [16] reported: we are not aware of any existing compression techniques for time-varying isosurfaces, and comparing our compression performance with methods for different data modalities (e.g., volumes) is bound to be biased. This makes the comparisons of our results quite challenging. For a fair assessment of the performance of our motion-prediction technique (MP), we compare it to the frame-by-frame approach (FF) where each frame uses the progressive encoding of [16] with the best published rates for an isosurface encoder. We focused our tests on highly dynamic models (fast motion and/or evolution of the surfaces) as it is the most challenging type of motion compression: obviously, a simple translation motion of an object can be achieved with extremely low rates, but such encoding is not as interesting.

To demonstrate our results, we used three challenging datasets:

- Droplets Dimensions: $257^3 \times 20$. A fluid simulation with topology changes (see Figure 5).
- FluidSplash Dimensions: $257^3 \times 20$. Another fluid simulation, this time of a splash with a high topological complexity (see Figure 6).
- HeadScan Dimensions: $257^3 \times 36$. An MRI head scan, sampled at different isovalues to visualize different tissues (see Figure 7, left).

Figure 8 illustrates the rate/distortion (R-D) performance of both methods. Table 1 provides some concrete rates for the median quantization parameters. The results of our dynamic surface encoder on such fast-pace animation exhibit a typical gain of 12% to 18% over the best frame-by-frame encoding method. It can be seen from the R-D curves that the highest gain is achieved for lower rates, reflect-

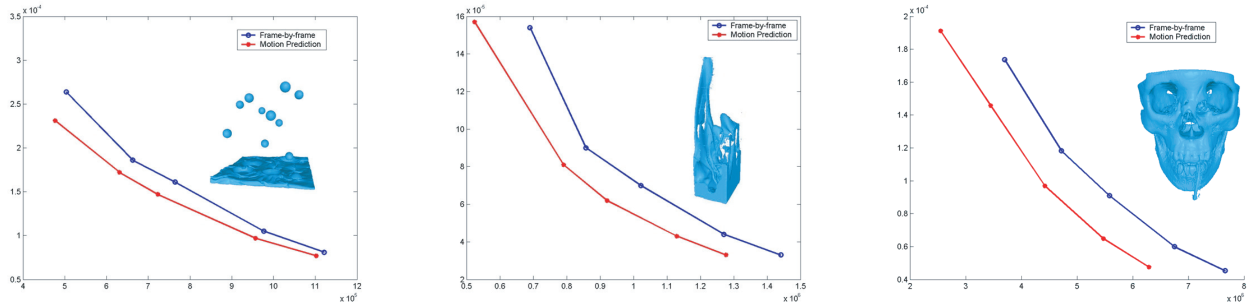

Figure 8: *Rate/distortion curves for the two fluid sequences and the MRI headscan sequence. We compare compression with motion prediction to frame-by-frame encoding for different quantizations. The x-axis represents the number of bytes, while the y-axis represents the $\mathscr{L}^2$ distance between the decompressed version and the original sequence (obtained by summing each $\mathscr{L}^2$ distance for each frame) as measured by MESH [4].*

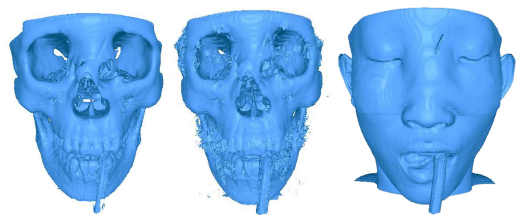

Figure 7: *MRI headscan sequence (the visible human dataset sampled at different iso-values). Note the amount of noise and topological complexity in the middle frame.*

ing the well known fact that fine detail is hard to encode. On the other hand, it is precisely the high frequency data that dominates the bit rate, hence the moderate overall gain.

This leads us to the important question of the most appropriate distortion measure for isosurfaces. Minimizing the $\mathscr{L}^2$ distance to the *originally extracted* isosurface (a usual quality criterion) turns out to be quite costly. The reason is that we are forced to encode the exact vertex coordinates, as chosen by the extraction procedure. The actual ground truth, however, is the original volume itself, and various contouring algorithms exist to extract the actual surface. Therefore, using the volume data for error assessment may be the right choice for many applications. Designing a codec with such error in mind would allow more freedom in vertex placements—thus greatly reducing the bitstream size.

Finally, we note that our compression technique is also progressive "in space": as more bits come in, the decoder can visualize a progressive refinement of the next frame. It was our initial goal to send the animation frame by frame to get a nicely progressive codec; however, we believe that using fully space-time compression, where the user cannot visualize the whole result until the whole sequence is sent but for which more freedom in the way we compress the space-time data is available, could give *dramatically* better compression. Unfortunately, this approach does not lend itself to streaming, while our presented method does.

Dataset	# frames	FF	MP	Gain
Droplets	20	1528932	1343131	12%
FluidSplash	20	2196948	1841577	16%
Headscan	36	7185869	5935626	18%

Table 1: *Comparing compression performance of motion prediction vs. frame-by-frame mode. Rates are shown in bytes.*

6 Discussion and Future Work

We have presented an approach to encoding time varying isosurfaces inspired by the simple framework used in video encoding. Mixed with the progressive encoding method of Lee *et al.*[16] for static isosurface compression, we provided a codec allowing rapid, progressive display of the sequence during decoding. The unstructured nature of time-varying isosurfaces allows our algorithm to deal naturally with surfaces of high geometric complexity and dynamic topology, unlike any of the existing dynamic 3D geometry codecs. Although beyond the scope of this paper, the method may also prove useful to encoding of additional surface properties, such as normals, color, texture, or in the context of fluid simulation, density and velocity fields: the latter is of particular interest here, as it eliminates the need for motion estimation.

Further potential improvements are numerous: nearly all optimization ideas that have been proposed for MPEG and H.264, such as the use of bi-directional prediction (B-frames), variable macroblock size, etc. can all be added to our framework. Our algorithm could also gain from improving the efficiency of the local registration method. Finally, extending our method to other nonconformal representations of time-varying geometry, such as moving point sets and free-viewpoint video could also be of interest.

Acknowledgements

We are grateful to Haeyoung Lee, Sanjit Patel and Yiying Tong for their insights and to Santiago Lombeyda, Andreas Söderström, Adam Bargteil and Richard Keiser for their data sets. The research has been funded in part by the Integrated Media Systems Center (a National Science Foundation Engineering Research Center, Cooperative Agreement No. EEC-9529152), and by DOE (DE-FG02-04ER25657) and NSF (CCR-0133983, DMS-0453145) grants.

References

[1] Marc Alexa and Wolfgang Müller. Representing animations by principal components. *Computer Graphics Forum*, 19(3), 2000.

[2] Nizam Anuar and Igor Guskov. Extracting animated meshes with adaptive motion estimation. In *Vision, Modeling, and Visualization*, pages 63–71, 2004.

[3] Adam W. Bargteil, Tolga G. Goktekin, James F. O'Brien, and John A. Strain. A semi-lagrangian contouring method for fluid simulation. *ACM Transactions on Graphics*, 25(1), 2006.

[4] MESH : Measuring Error between Surfaces using the Hausdorff distance. http://mesh.epfl.ch.

[5] Hector M. Briceo, Pedro V. Sander, Leonard McMillan, Steven Gortler, and Hugues Hoppe. Geometry videos: a new representation for 3d animations. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer animation*, pages 136–146, 2003.

[6] Junavit Chalidabhongse and C.-C. Jay Kuo. Fast motion vector estimation using multiresolution-spatio-temporal correlations. *IEEE*

Trans. on Circuits and Systems for Video Technology, 7(3):477–488, 1997.

[7] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image Vision Comput.*, 10(3):145–155, 1992.

[8] Benjamin Gregorski, Joshua Senecal, Member-Mark A. Duchaineau, and Member-Kenneth I. Joy. Adaptive extraction of time-varying isosurfaces. *IEEE Transactions on Visualization and Computer Graphics*, 10(6):683–694, 2004.

[9] Sumit Gupta, Kuntal Sengupta, and Ashraf A. Kassim. Compression of dynamic 3d geometry data using iterative closest point algorithm. *Comput. Vis. Image Underst.*, 87(1-3):116–130, 2002.

[10] Stefan Guthe and Wolfgang Straßer. Real-time decompression and visualization of animated volume data. In *VIS '01: Proceedings of the conference on Visualization '01*, pages 349–356, Washington, DC, USA, 2001. IEEE Computer Society.

[11] Lawrence Ibarria, Peter Lindstrom, Jarec Rossignac, and Andrzej Szymczak. Out-of-core compression and decompression of large n-dimensional scalar fields, 2003.

[12] Lawrence Ibarria and Jarek Rossignac. Dynapack: space-time compression of the 3d animations of triangle meshes with fixed connectivity. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer animation*, pages 126–135, 2003.

[13] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. *ACM Trans. Graph.*, 21(3):339–346, 2002.

[14] Zachi Karni and Craig Gotsman. Compression of soft body animation sequences. *Computers & Graphics*, 28:25–34, 2004.

[15] Liu Kwan-Ma, Diann Smith, Ming-Yun Shih, and Han-Wei Shen. Efficient encoding and rendering of time-varying volume data. Technical report, 1998.

[16] Haeyoung Lee, Mathieu Desbrun, and Peter Schröder. Progressive encoding of complex isosurfaces. *ACM Trans. Graph.*, 22(3):471–476, 2003.

[17] Jerome Edward Lengyel. Compression of time-dependent geometry. In *SI3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 89–95, 1999.

[18] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Computer Graphics (Proc. of SIGGRAPH)*, volume 21, pages 163–169, 1987.

[19] Ajith Mascarenhas, Martin Isenburg, Valerio Pascucci, and Jack Snoeyink. Encoding volumetric grids for streaming isosurface extraction. In *3DPVT '04: Proceeding 2nd International Symposium on 3D Data Processing, Visualization and Transmission, 2004*, pages 665–672, 2004.

[20] Alexander McKenzie, Santiago Lombeyda, and Mathieu Desbrun. Vector field analysis and visualization through variational clustering. In *Eurographics - IEEE VGTC Symposium on Visualization 2005*, 2005.

[21] Niloy J. Mitra, Natasha Gelfand, Helmut Pottmann, and Leonidas Guibas. Registration of point cloud data from a geometric optimization perspective. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 22–31, New York, NY, USA, 2004. ACM Press.

[22] Ariel Shamir and Valerio Pascucci. Temporal and spatial level of details for dynamic meshes. In *VRST '01: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 77–84, New York, NY, USA, 2001. ACM Press.

[23] Bong-Soo Sohn, Chandrajit Bajaj, and Vinay Siddavanahalli. Feature based volumetric video compression for interactive playback. In *VVS '02: Proceedings of the 2002 IEEE symposium on Volume visualization and graphics*, pages 89–96, Piscataway, NJ, USA, 2002. IEEE Press.

[24] Stephan Würmlin, Edouard Lamboray, and Markus Gross. 3d video fragments: Dynamic point samples for real-time free-viewpoint video. In *VRST '01: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 3–14, Amsterdam, The Netherlands, 2004. Elsevier Ltd.

APPENDIX A

For completeness, we provide a derivation of the geometric optimization procedure used for macroblock registration. Due to limited space, we only explain the process in 2D; details can be found in the original paper [21].

For an isosurface S_i, the local squared distance approximant $\widetilde{d^2}$ at point $p = (x,y)$ is given by

$$\widetilde{d^2}(p) = Ax^2 + Bxy + Cy^2 + Dx + Ey + F,$$

where A,B,C,D,E,F are the approximant coefficients, valid locally around p. Note that for any point q on S_i, $\widetilde{d^2}(q) = 0$ by definition. Now, given a set of points $\{p_j\}$ on isosurface S_{i+1}, we find a translation T that aligns them best with S_i by iteratively minimizing $\sum_j \widetilde{d^2}(p_j + T)$. In each iteration $(x_j, y_j) \mapsto (x_j + t_x, y_j + t_y)$: if for a small translation (t_x, t_y), $\widetilde{d^2}(p_j)$ stays approximately fixed, the residual error for (p_j) is given by

$$\begin{aligned}\varepsilon(t_x,t_y) = \begin{bmatrix} x_j + t_x & y_j + t_y & 1 \end{bmatrix} & W_{p_j} \\ & \begin{bmatrix} x_j + t_x & y_j + t_y & 1 \end{bmatrix}^t,\end{aligned}$$

where W_{p_j} is a matrix representing the approximant $\widetilde{d^2}$ around p_j. Setting $\nabla\varepsilon$ to zero, we obtain the linear system:

$$\begin{aligned}\left[\sum_j \begin{pmatrix} 2A & B \\ B & 2A \end{pmatrix}\right] \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \\ -\left[\sum_j \begin{pmatrix} 2Ax_j + By_j + D \\ Bx_j + 2Cy_j + E \end{pmatrix}\right]\end{aligned}$$

This system is solved at each iteration of the gradient descent process, until a convergence criterion is satisfied or the maximum number of iterations is reached.

The approximant $\widetilde{d^2}(.)$ is fitted locally by solving another least-squares system. The corresponding matrix does not depend on a global coordinate system and therefore can be precomputed for most cases.