Eurographics/SIGGRAPH Symposium on Computer Animation (2003)
D. Breen, M. Lin (Editors)

Geometry Videos: A New Representation for 3D Animations

Hector M. Briceño[1], Pedro V. Sander[2], Leonard McMillan[3], Steven Gortler[2], and Hugues Hoppe[4]

[1]MIT, Cambridge MA
[2]Harvard, Cambridge MA
[3]University of North Carolina at Chapel Hill, NC
[4]Microsoft Research, Seattle, WA

Abstract

We present the "Geometry Video," a new data structure to encode animated meshes. Being able to encode animated meshes in a generic source-independent format allows people to share experiences. Changing the viewpoint allows more interaction than the fixed view supported by 2D video. Geometry videos are based on the "Geometry Image" mesh representation introduced by Gu et al. [4]. Our novel data structure provides a way to treat an animated mesh as a video sequence (i.e., 3D image) and is well suited for network streaming. This representation also offers the possibility of applying and adapting existing mature video processing and compression techniques (such as MPEG encoding) to animated meshes. This paper describes an algorithm to generate geometry videos from animated meshes.

The main insight of this paper, is that Geometry Videos re-sample and re-organize the geometry information, in such a way, that it becomes very compressible. They provide a unified and intuitive method for level-of-detail control, both in terms of mesh resolution (by scaling the two spatial dimensions) and of frame rate (by scaling the temporal dimension). Geometry Videos have a very uniform and regular structure. Their resource and computational requirements can be calculated exactly, hence making them also suitable for applications requiring level of service guarantees.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism, Animation

1. Introduction

The problem of representing animated models is of significant importance in the field of computer graphics. There are several applications for animated geometry, such as computer-generated movies and computer games. Due to the unceasing increase in computational power and memory, it is becoming increasingly easier to acquire such animated models. Some systems, such as Matusik et al. [11], are already able to acquire, construct, and render animated geometry in real-time.

We present a new data structure for storing animated meshes called the *geometry video*. Our approach is based on the *geometry image* representation by Gu et al. [2002]. The geometry image represents a static mesh as a 2D square image, which is a completely regular remesh of the original geometry.

Geometry videos are sequences of geometry images, each geometry image being one frame in the video. Thus, a geometry video is a 3D image with two spatial dimensions and one temporal dimension. This paper presents a method to create geometry videos, that is, to parametrize the meshes in the animation sequence and sample them onto a 3D image.

Our new animated geometry representation has several significant advantages over previous ones:

- It inherits the features of geometry images, thus providing a mesh representation for each frame that has completely regular connectivity.
- It allows for a unified way to address level-of-detail, both in terms of mesh resolution and frame rate. That is achieved by scaling the two spatial dimensions or the one temporal dimension of the geometry video.
- It is amenable to hardware parallelization and optimization.

- It allows for the application of numerous available processing and compression methods targeted at videos (such as MPEG encoding) to animated meshes.
- It provides a global *implicit parametrization* for all frames, thus allowing the application of a single high resolution texture-map to be shared by all frames in the animation sequence
- It provides a representation whose resource and computational requirements for rendering can be known exactly, thus providing level of service guarantees.

In Section 2, we place our novel representation in the context of previous work. In Section 3 we describe the Geometry Image approach. In Section 4, we describe our method for creating geometry videos. In Section 5, we present the results of our implementation. Finally, in Section 6, we summarize and propose directions for future work.

2. Related Work

Static Mesh Compression. Taubin and Rossignac [18] introduced a method to encode the connectivity of a static mesh. It uses predictors to compactly encode the vertex positions. The prediction is based on estimating the angles between adjacent faces to predict the location of future vertices. Although this method does very well for encoding connectivity information, geometry information is not as highly compressed.

Karni at Gotsman looked at an alternative method for encoding vertex positions [6]. Instead of looking at building predictors based on vertex locations, they approached the encoding from another perspective. They applied classical Fourier analysis to mesh data. Fourier analysis is already used in JPEG for image compression, so it is well understood and achieves good compression. Here they compute the eigenvectors of the Laplacian Matrix to obtain a basis. The vertex positions are encoded as a weighted sum of these basis.

Remeshing. Higher compression rates can be achieved if the original connectivity is discarded, and instead, a regular or semi-regular representation is used. Several approaches for constructing and encoding meshes with semi-regular connectivity have been proposed [2, 9, 8, 5]. Khodakovsky et al. [7] present an algorithm to encode a mesh with semi-regular connectivity using wavelet techniques, thus achieving very high compression rates. Gu et al. [4] propose the extreme approach of representing the input mesh with fully regular connectivity. We build upon this last work to encode a sequence of time-varying geometries.

Time-varying Geometries. There has also been directly related work in the compression of time-varying geometries. Lengyel [10] proposes an encoding based on predictors. He partitions the mesh into sets of vertices. These sets of vertices (from a reference frame) will be transformed independently and linearly combined to form a predictor. Each frame in the animation is represented as a predictor plus a residual. The residual and the transformation coefficients are quantized for compression. Our approach is very similar to his. We find that our approach is less sensitive to the quantization of transformation coefficients, it rearranges the data (residual) making it suitable for compression, and can make use of video encoding techniques.

Alexa and Müller [1] propose to represent the motion of vertices through principal component analysis. First, all the frames are approximated by a reference frame using affine transformations. The vertex locations at each frame (columns) are then decomposed into basis and weights. Compression is achieved by discarding less important basis. This method is computationally expensive, especially for large meshes. It also suffers from scaling artifacts if important basis are discarded.

Finally Shamir and Pascucci [17, 16] propose a very flexible approach for encoding dynamic meshes. It supports level of detail both in time and space. It works by building a progressive representation of each frame, while tracking reconstruction error information. Their encoding is not very compact, it is actually larger than the original mesh size. Their work is complementary to Lengyel's [10], so it is possible to combine their works to produce a more compact encoding.

3. Background: Review of Geometry Images

Before we describe our algorithm to create geometry videos, we will briefly review the geometry image construction algorithm of Gu et al. [4]. To construct a geometry image, the original mesh is cut in order to form a mesh that is topologically equivalent to a disk. The cut mesh is then parametrized onto a square domain. Then, the original mesh is sampled onto a square image using the computed parametrization. The normalized XYZ coordinates are encoded as the RGB values of the image. Finally, the resulting image is compressed using image wavelet coding techniques. Sideband information is used to ensure that the boundaries (of the cut) match after lossy compression. We now describe each of these steps. Refer to Gu et al. [4] for further details.

Cutting. In order to parametrize the mesh onto a square, the mesh must be topologically equivalent to a disk. The goal is to find a cut such that the resulting mapping from the cut mesh to the parameter domain can be adequately sampled. The algorithm first finds a topologically sufficient cut to map the initial mesh onto a disk and then improves upon this cut.

For a genus zero surface, this initial cut can be as simple as a cut traversing two edges. For higher genus, a different algorithm is used (see [4] for more details); for example for a genus one object like a toroid, we would need two cuts: a first cut, to unwrap the toroid forming a cylinder; and a second cut, to unfold the cylinder onto a sheet, which is topologically equivalent to a disk.

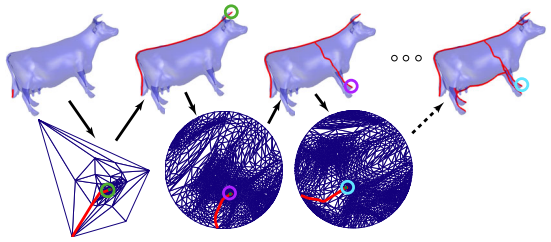


Figure 1: *Iterative Cutting Algorithm. We start a short cut. The resulting parametrization has high distortion at the center. The worst undersampled area is near the horns. We then cut, and reparametrize. We stop when adding a new cut increases the average distortion or undersampling.*

This cut is then improved iteratively. We greedily improve the cut and parametrize the mesh until the quality of the resulting parametrization no longer improves the average distortion or quality. The quality is measured using the geometric-stretch metric of Sander et al. [15], which measures both the change in area and shape of triangles or faces from the parameter space to the object space. During each iteration, in order to improve the cut, an angle-preserving parametrization for the current cut mesh is computed using the method of Floater [3]. The vertex whose adjacent faces have the highest undersampling in the domain is identified. The parametrization is computed using a circular boundary. Then, the cut is extended by the shortest path between the current cut boundary and the identified vertex. Figure 1 shows iterations of the cutting algorithm on a sample mesh.

Boundary parametrization. Once the cut is defined, the mesh is parametrized onto the 2D domain. In order to prevent geometric discontinuities across the cut, the samples on opposite sides of the cut must have the same geometric value. This is achieved by parametrizing the boundary of the mesh onto a square, and assigning corresponding parametric lengths to edges on opposite sides of the cut. Vertices that lie at the junction of multiple cuts are called *cut-nodes*. These will appear more than twice on the boundary of the square. In order to guarantee that they will be sampled exactly, we place these samples on integral grid positions on the boundary.

If the Geometry Image is compressed using a lossy encoding, additional sideband information is recorded. The lossy encoding will blur the boundaries and hence the *cut-nodes* and boundary will not match on the reconstructed mesh, causing a seam to be visible at the cut. The sideband information keeps track of the correspondence of the *cut-nodes*, so that after the Geometry Image is decoded, their location can be aligned to match (in order to prevent seam artifacts).

Geometry images form a geometric sampling of the original surface. We can implement lower resolution versions of the original surface by simply subsampling it – by subsampling the geometry image. This is commonly referred as "level-of-detail" support. When we subsample (decimate) the geometry image to obtain a lower resolution version, it is possible that we might miss *cut-nodes* on the boundary, and therefore cracks may become visible. To avoid this, we place *cut-nodes* on boundary locations that would be sampled, even if the geometry image is decimated.

Interior parametrization. After defining the parametrization of the boundary vertices, the parametrization of the interior is computed so as to minimize the geometric-stretch metric of Sander et al. [15]. This metric penalizes undersampling in the parametric domain and yields a parametrization with samples uniformly distributed over the mesh surface. The algorithm uses the hierarchical parametrization framework of Sander et al. [14] to improve quality and convergence speed.

Compression. Once the parametrization is computed, the mesh is sampled onto a 2D image. In order to sample the image, a bounding box around the mesh is computed, the mesh is scaled to fit within the RGB unit cube, and the samples are quantized to a desired number of bits. The geometry image is then compressed using off-the-shelf image coding techniques (e.g., 2D wavelets). When lossy compression is applied, the corresponding boundary samples will not necessarily match after decoding. This is addressed by storing additional sideband information about the topology of the boundary.

Limitations. There are many advantages of being able to parametrize the surface onto one chart. It arranges the data in a compressible manner. It can be easily manipulated and parallelized. Unfortunately, many topologically and geometrically complex surfaces cannot be parametrized onto a square with low distortion.

Moreover, we are mapping the surface to a square; this in it by itself introduces distortion at the corners. Being of a fixed aspect ratio also imposes a limitation on the unwrapping of the surface. Figure 2 shows the parametrization of a snake. Due to the "aspect-ratio" of the snake body, there is high distortion at the boundaries. This becomes a problem for textures, as these areas might not be rendered well. It is possible to alleviate the problem by smoothing the points away from the boundary. This is sensible in a static context. In a dynamic context the answer is not so clear, as we would need to calculate the effect of this smoothing on all frames.

There are many approaches to this problem, among them chartification and allowing the boundaries to be free. We maintain this parametrization due to its advantages in compressibility and ease of processing. For now, we acknowledge that Geometry Videos may not be for all objects. With this in mind, let us delve into the encoding of animated meshes.

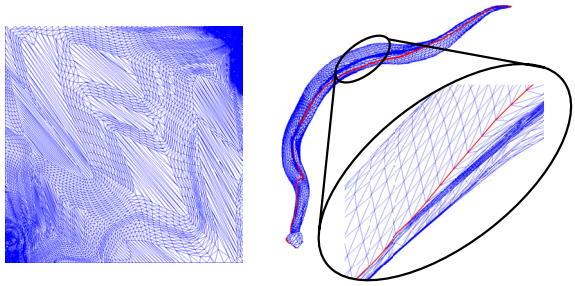


Figure 2: *Limitations: one of the drawbacks of using a single parametrization is that regions of high-distortion might be unavoidable.*

4. Our Approach

In this section, we describe how we extend each of the three steps of the geometry image construction (cut, parametrization, and compression) in order to generate geometry videos from animated input meshes.

The straightforward way to implement 3D animation encoding with geometry images, is to apply the algorithm statically to each frame. The disadvantages of this approach are many. Notice that the geometry image parametrization forms an implicit texture map. Hence if we encode each frame with a separate parametrization, we would need to update the texture map for each frame. The same applies for any other attribute information that is fixed through time. Furthermore, since we are resampling the input mesh, it is unlikely that the two meshes will be sampled in the same way, therefore, on low-motion scenes, a jittering effect will be seen. Lastly, using different parametrizations makes it difficult to exploit the temporal coherence that exists between frames. Different parametrization yields complex correspondence relationship between samples in different geometry images.

We seek to find a common parametrization for all the frames in the animation sequence. Using a single parametrization yields better results because the compression algorithm can exploit the frame-to-frame coherence that is present with a single parametrization; the loss due to using a "non-optimal" parametrization in general is very small. By using a fixed parametrization for all frames one can then store fixed per-vertex attributes (e.g., texture, material id, etc) in a single separate image. Notice that by fixing the parametrization, we are also fixing the cut, since the cut defines how the boundary will be mapped, and changing the boundary mapping would change the parametrization.

Picking a single cut and parametrization for all frames in the sequence does not work well if the features of the model change drastically from frame-to-frame (i.e., a human morphing into a two-legged animal).

Currently our parametrization is fixed and computed from a single frame. Our algorithm assumes all meshes in the animation have the same connectivity. This presents a problem if the input mesh changes connectivity or topology. The first challenge is not an inherent problem of geometry images. We could apply a re-meshing algorithm (e.g. [13]) to all the frames to generate frames of equal connectivity as a preprocess. The second challenge is more difficult. Changes in topology force changes in the cuts which make re-using or adapting another parametrization next to impossible. This is a current limitation of the system. Our current solution would involve approaching this problem as a scene change, where a new parametrization would be used. We would need ensure a smooth transition between frames with different parametrizations. 3D morphing techniques could be used for this purpose.

4.1. Cutting

We now describe the cut algorithm that converts the mesh into one that is topologically equivalent to a disk. As described above, we seek to construct a single parametrization for the entire animation sequence, the cut in all of the meshes must be equivalent (connectivity-wise).

Initially, our approach used the cut found in a single frame and applied it to all frames. We found that frames that showed all the important features of the object (i.e., a human in anatomical position, or a standing cow) worked best, but only slightly better than the worse chosen frame. That is because cuts reach out to regions of high distortion releasing the "tension" that exists in that area. All these areas were usually reached when the cutting algorithm was applied to any frame.

The drawback of this approach, besides being heuristic, is that it will miss areas of high-distortion if these occur at different times. Figure 11, shows an example where this can happen, where the eyes and the tail are regions of high distortion, but they occur exclusively in different frames. This approach also biases regions that have an average of high distortion, such as joints.

In order to "catch" these potential areas, we seek a global cutting algorithm. This cutting algorithm works by applying the single-frame cutting algorithm simultaneously to all frames. Recall that the single-frame cutting algorithm works iteratively by cutting, computing the Floater parametrization on the resulting mesh, finding the vertex next to the face of highest distortion and then making the shortest cut from that vertex to the current boundary. The algorithm stops when additional cuts do not improve the average distortion. The problem then becomes how to find the face of highest distortion and how to compute the "shortest-path" to the boundary when considering all the frames.

In the single frame case, the vertex of highest distortion is located by computing the geometric-stretch for each face. The vertex next to the face with the highest stretch in then

selected. The geometric-stretch measures how circles in the parameter space are mapped into the object space. The distortion is measured by the magnitude of the major-axis of the resulting ellipsis.

For the global case, we have to compare or average the maximum stretch on many frames. We also do not want a single frame with very distorted faces to dominate or drive the algorithm astray. Therefore, for each frame, we normalize the stretch by the maximum stretch on that frame. Then, the final stretch of each face is the average of the normalized stretch of that face in all frames.

We compute the shortest distance to the boundary by using Dijkstra's algorithm using a proxy mesh. This non-realizable mesh has the same connectivity of our animation, but the length of its edges is the average length of that same edge for all frames of the animation. This method yielded good results.

The single-frame algorithm stops when adding a cut would increase the average distortion. This is the average distortion across of faces of all frames. The average distortion of a face is measured using the geometric-stretch metric. This measurement is normalized by the surface area in object space for each frame. It is comparable between frames, hence our criteria for stopping the algorithm considers the average of the average distortion across all frames without any modifications.

The global cut does not consistently do better than choosing a cut and parametrization from a single frame either by hand or by brute force. The advantage is that is does consistently well; this reduces human-intervention or computation time. Additionally, it can identify and relax regions of high-distortion that do not occur in the same frame.

4.2. Parametrization

Given the global cut, we wish to find a parametrization that minimizes the errors across all frames. We have additional requirements which also constrain the parametrization.

In order to support level-of-detail, we would like a parametrization that uniformly samples the surface. This allows us to simply decimate in both dimensions each frame of the geometry video, to obtain a low resolution version of the animation. If this feature is not a requirement of the application, then it is possible to tweak the points in the parameter space to align better with features in the object space (like sharp creases). Additionally, having a uniform parametrization works well for frames that may not have been taken into account in building the cut and parametrization. Without a priori information, uniform sampling is likely to sample arbitrary areas better than a specialized sampling based on another frame.

The algorithm already used by Gu et al. [4], minimizes the geometric stretch metric in order to parametrize the mesh. This metric distributes the samples uniformly over the surface and minimizes the reconstruction error assuming a piecewise constant signal. Ideally we would like to modify it to take all frames into account, or determine a reference frame in our animation to be used for the parametrization of the animation.

We have experimented with an approach that parametrizes the cut mesh while considering the geometry of all frames. For an animated mesh sequence, instead of a single 3D geometry channel, we have N 3D geometry channels, where N is the number of frames. This yields a $3N$-dimensional signal. We seek to find a parametrization that minimizes the stretch over this $3N$-dimensional space, thus minimizing reconstruction error over all frames in the sequence. But our results in terms of reconstruction error (after compression) were slightly worse than using the parametrization from a single frame. In terms of worst-case distortion, this algorithm did significantly better than the parametrization using a single frame. We have not done enough experiments to reliably use this global parametrization. This is a direction for future work.

Currently we have found that using an arbitrary frame (i.e., the first frame) of the animation as the reference frame for the parametrization algorithm to yield good results. For medium deformation animations, we have found that the cut has a large effect on the parametrization. It already defines the boundary, and in a sense, it has spread the regions of high distortion around the unit square. So even choosing an arbitrary frame in our animation, yields similar results. Figure 5 shows some data to support our theory.

4.3. Sampling and Rendering

Once we have a cut and parametrization, we can uniformly convert each frame of the animation into a geometry image. The collection of geometry images is called a geometry video. The conversion works by sampling over a grid the original vertex values mapped onto the parametrization. By using the same parametrization, the samples in the same coordinates in different frames will correspond to the same location on the original surface.

The mesh resulting from the geometry image will have regular connectivity. It will look like a grid. We triangulate it uniformly along one chosen diagonal. We do not change the original diagonalization once fixed, because doing so would yield popping or jittering artifacts in the encoded animation when the diagonalization changes.

4.4. Compression

The cut and parametrization allows us to generate a sequence of geometry images. The next logical step is to compress these images taking advantage of the coherence or redundancy that exists within each image and between each pair of frames in the sequence.

This sequence of images is very much like a 2D video. Therefore we can apply standard mature video encoding techniques to the encoding and manipulation of such sequences. There are two important differences that allow us to make better use of these techniques.

First, the "colors" of these geometry images correspond to a 2D manifold surface in 3D space. The neighboring "colors" tend to be similar since we are assuming a single connected surface. This allows us to use wavelet techniques which can represent smooth functions very compactly.

Second, mesh vertices, due to the fact that they are connected, usually do not move in a chaotic manner, but rather tend to locally move in conjunction. We can represent this motion by affine transformations, which can be computed or resolved precisely or at least to a best approximation.

We can use this transformation to represent changes between frames as a first approximation. Then we only need to encode the difference, or delta between this approximation and our target image. This is similar to the Predictive frames in MPEG, with the difference that we don't compute motion vectors, but rather affine transformations. This is possible because we know the exact motion of the object, not a 2D flat vision of the world or motion. This idea of using transformations as a form of predictors is not new. Lengyel [10] mentions it in the direct context of compression of time-varying geometry.

Our approach is simple. Given a sequence of geometry images, we encode each image in one of two ways:

I-Frames The geometry image is encoded independently of any other information. We encode the geometry image using wavelets, since this method proves to compress very well. We have found the geometry images to be sufficiently smooth to use wavelet functions with large support (18/10 taps for synthesis/analysis). This allows the compressor to use more information to predict sample locations.

Furthermore, we achieve good compression because the data is well-ordered. Recall that the parametrization is optimized for uniform sampling. This means that information in the vertical and horizontal directions in the parameter space will be independent. This allows the algorithm to exploit the redundancy that exists locally in *both* directions of the surface. This contrasts with many static mesh encoders which sometimes can only exploit the redundancy of the geometry in only "one" direction.

P-Frames The alternative way to encode a geometry image, is to use a transformation of the previous geometry image as a predictor. Any transformation function and any number of reference frames maybe used. The choice is limited by resources and the complexity needed to choose the function optimally. Our idea, is not to resolve exactly how the data was generated (free form deformations, skinning, inverse-kinematics...), but rather to develop a mechanism that works well across many domains and is simple. We use an affine transformation (4x4 matrix with 12 degrees of freedom) to match, in the least square sense, the previous geometry image to our target geometry image. We find that we can quantize the transformation coefficients to 11 bits with negligible negative effects. Finally, we encode the difference between the current geometry image and the transformed reference geometry image using also wavelets – exactly as if it was an I-Frame. The transformation coefficients are saved separately.

We have also implemented "B-Frames" or Bi-directional frames, where the prediction is based on a weighted average of two transformed frames. We find that it is a black art to choose the right combination and selection of different kinds of frames. Nevertheless, it is another tool that works well for certain applications. For example, B-Frames are useful for encoding key-frame animations, and allowing users to have reverse playback with low overhead. Currently we only use P-Frames.

For small meshes, we find that the geometry images tend to supersample the original mesh. If the lossless version of a geometry image for a given mesh is a good approximation of the surface, then the difference between the encoded geometry image and the original geometry image will form a good estimate for reconstructed surface error. This is because the reconstructed surface error (Hausdorff distance) is usually measured by supersampling a surface, and computing the closest point on the reference surface. Although a "diff" between geometry images only considers one point (the corresponding point), it provides a good upper bound on the quality of the reconstructed mesh assuming that both geometry images were constructed from the same parametrization. This is useful to guide any bit allocation and compression decision.

5. Results

We have implemented an encoder for geometry videos. To compare different cuts, parametrizations, and encoding parameters, we measure the root mean square (RMS) distance between the original and reconstructed mesh for each frame (we take the maximum of both one-sided distances, same as Khodakovsky et al. [7]).

For our results, we use bits per vertex (bpv) as our measure of compression. We always consider the number of vertices of the original mesh when computing the bits per vertex (bpv) of a Geometry Image. Vertices are assumed to be three coordinates of 32-bit floating point numbers; hence they are 96 bpv uncompressed. Therefore, compressing a mesh at 1 bpv would correspond to a 96x compression over the uncompressed format.

We use three animated meshes to test and compare our system: A bouncing cow sequence (deformation) with 2904

vertices, a motion captured dance sequence (skinning) with 7061 vertices, and a curling snake sequence (motion) with 9179 vertices. We have also applied our algorithm to short sequences with meshes with 38K vertices.

5.1. Comparison to Static Approaches

Geometry images are competitive with other static mesh compression approaches. Figure 3 shows rate-distortion curves for a few models. Static compression methods based on quantized deltas usually cannot support very low bit-rates. Also, note that the geometry image algorithm works better with a larger number of vertices. With more vertices, we have a larger bit-budget to encode the object; in both cases we are encoding the same object.

5.2. Cuts/Parametrization

In order to obtain more temporal coherence and have one uniform parametrization for the entire animation, we would like to use only one cut and parametrization. Two experiments show the effect of this choice.

The first experiment shows that using one global cut does not yield much worse reconstruction. Figure 4 compares using an "optimal" cut and parametrization for each frame versus having one fixed cut and parametrization (using the first frame of the sequence). Notice that the fixed global cut does only slightly worse in the Dance sequence and achieves about the same quality in the Snake sequence.

The second experiment shows that once a cut is defined, picking an arbitrary parametrization is not harmful. The reasoning is that the cut already relieves tension in areas of high distortion and fixes the boundary of the parametrization. Hence, picking a different reference frame for the parametrization only changes the internal configuration slightly. Figure 5 shows the effect of fixing the cut and calculating the error for all frames of the sequence using different reference frames for the parametrization. The model on frame 15 is a highly distorted cow, but even using this as the reference frame for the parametrization, does not yield a notably higher errors for the sequence.

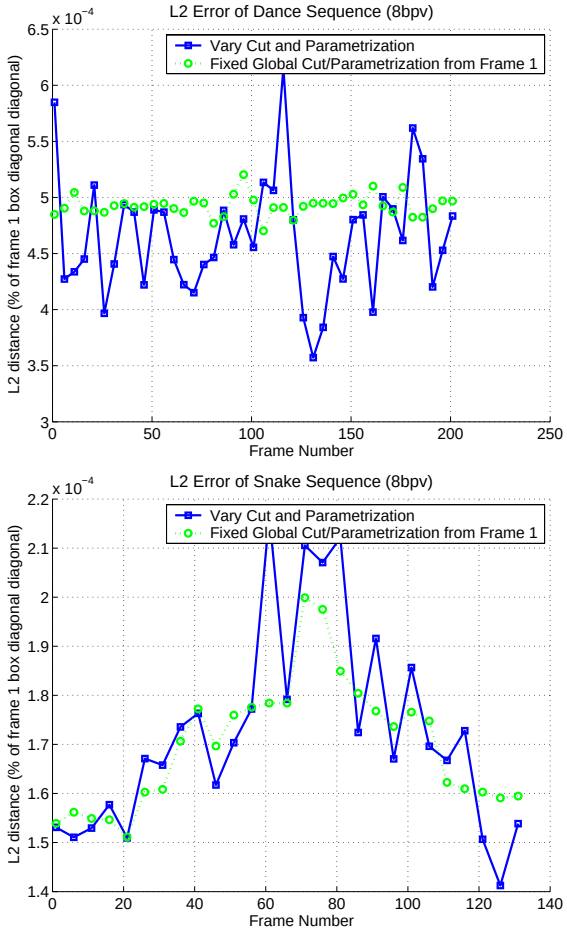
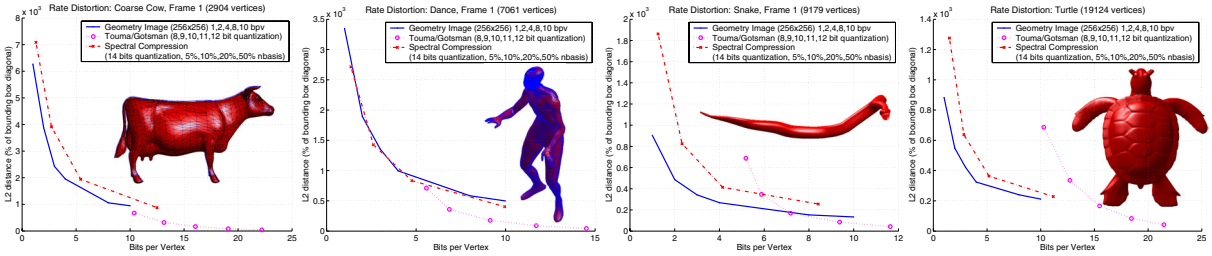


Figure 4: *Dance (7061 vertices) and Snake (9179 vertices) sequences encoded at 8 bits per vertex. We do not lose much by using a fixed cut and parametrization.*

5.3. Level of Detail

Level of detail is naturally supported by geometry images and hence by geometry videos. It is simple to construct a coarse version of the mesh by simply decimating (removing every other pixel in each direction) the geometry image. It is important to have a uniform parametrization such that when decimation occurs, the texture map of the fine geometry image also forms a good mapping on the coarse geometry image. In figure 10 we show the same geometry image at two resolutions; we can combine them to present a multi-resolution version of the mesh.

The concept of temporal scalability is the same as in video. B-Frames can be interleaved such that if the user wants to fast forward through an animation, the decoder does not have to decode all the frames. On the downside, inserted B-Frames makes the prediction for P-Frames harder since these are farther apart. Figure 6 shows the reconstruction error between using only I-Frames, using P-Frames, or using a combination of P-Frames and B-Frames. Notice that the use of B-Frames increases the variance of the error, and only increases the average error average slightly. B-Frames are predicted using reference frames that are farther apart, hence are not as good predictors. The variance could be mitigated by adaptive adjusting the frame size. Most video techniques applicable to MPEG are applicable here.



(a) (b) (c) (d)

Figure 3: *Rate Distortion Curves: comparing Geometry Images to Touma/Gostman (TG) encoder, and Spectral Compression*

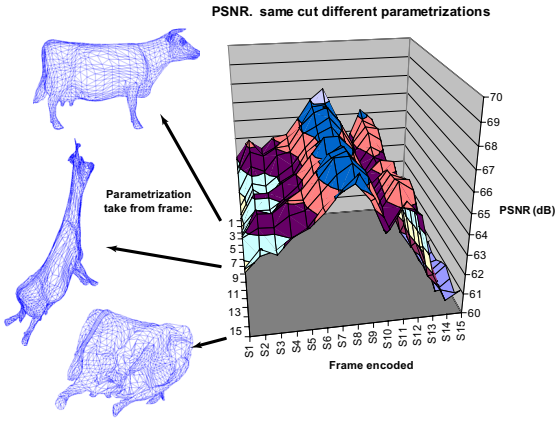


Figure 5: *The cut has a significant effect on the parametrization. This figure shows the Signal to Noise Ratio of the reconstructed key frames of the cow sequence (2904 vertices) encoded at 8bpv. All of them use the same cut. The parametrization is based on different frames. Notice that the error for each frame is about the same for all parametrizations.*

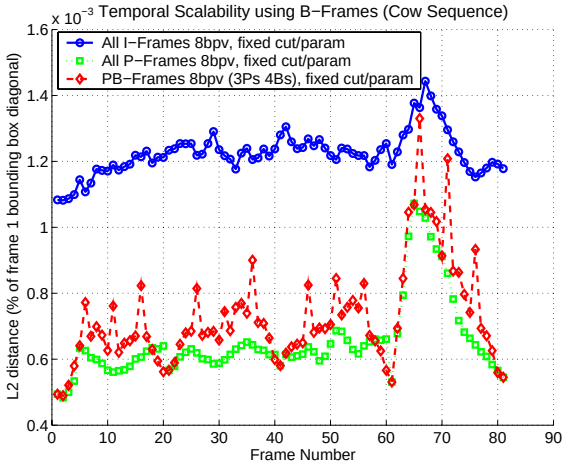


Figure 6: *The use of B-frames allows the decoder to skip them if necessary. One simple way to implement temporal scalability is by introducing B-Frames. The animation can be played at half the frame rate just by dropping the B-Frames.*

5.4. Compression Results

Geometry Videos do a good job in compressing our sample data sets. We compare our results with a straightforward implementation of Alexa and Müller's [1] "Representing Animations by Principal Component Analysis." We find wavelets are very powerful at compressing both the original geometries and the residuals. The intuition is that by re-sampling and re-ordering the vertices, we can extract more spatial coherence. It is possible to calculate this on an irregular mesh, but handling these irregularities can be cumbersome and unpredictable.

Figure 8 shows a comparison between Principal Component Analysis (PCA) and Geometry Videos. Figure 8.b shows that the detail of the single frames with PCA is slightly better (see face), as it does not suffer from some of the smoothing of face that occurs with geometry images. Unfortunately, PCA shows major distortion on some frames of the animation. PCA has worse in reconstruction error, also because without enough basis, it can't capture the exact motion of the object.

Next, we consider how well the predictive encoding works. We find that using P-Frames reduces the reconstruction error by half for the same bit-rate using I-Frames only. P-Frames are encoded at 30% the size of the I-Frames; the overall size is the same. Figure 7 shows the encoding of the Dance sequence. We note that using predictions on a 4 bits per vertex encoding achieves almost the same result as applying a new cut and parametrization to encode frames at 8 bits per vertex. This shows that by fixing the cut and parametrization we can reduce the temporal redundancy between frames.

Finally, we are exploring partitioning the geometry image into regions and applying separate transformations to each

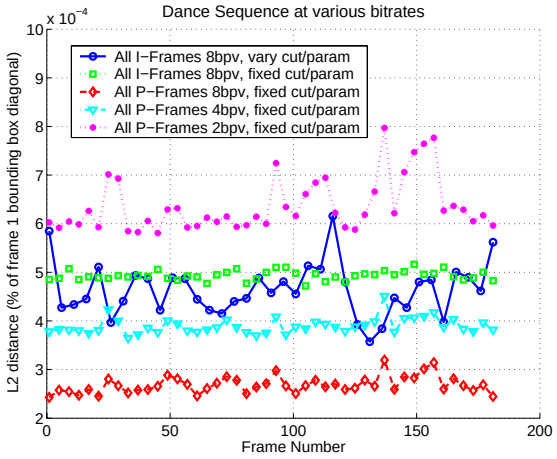


Figure 7: *Dance sequence (7061 vertices) encoded: at 8 bits per vertex (bpv) using a customized cut and parametrization per frame (vary); at 8 bpv using Geometry Images using only I-Frames; and at 8,4 and 2 bpv using P-Frames (0.3 size ratio between P-Frames and I-Frames). Notice that the encoding using P-frames at 4 bpv is better to using only I-frames at 8bpv.*

region. The idea is to capture local changes better. We are inspired by motion compensation in MPEG. Our preliminary results are encouraging. We find that local transformations form a better predictors than using one global transformation. One has to be careful to blend the different transformed regions. Without a smooth transition between transformed regions, this predictor will yield residuals containing high frequencies at the region boundaries which will hurt compression.

5.5. Timings

In this section we briefly report on the performance of our system. Table 1 reports the performance of different algorithms.

The encoder was developed in Matlab 6.5 for ease of prototyping. It has not yet been ported it to C++. The first five measurements concern the encoder, in particular the parametrization pre-process. Once a cut has been chosen and a parametrization computed, we can encode 3D animations at a rate of few frames per second for an image size of 256×256, and much faster with an smaller image size. The encoding time is dominated by two factors: the wavelet encoding of the geometry images and the cutting and sampling of the frames in the 3D animation. For example, the encoder will encode 100 frames of the Cow Sequence with an image size of 64×64 in 6.24 seconds: 1.69 seconds are spent cutting the meshes of each frame and preparing them to be sampled, 1.90 seconds are spent sampling the input

Model/Sequence # Vertices	Cow Sequence 2904
Find Cut‡	147secs (2m27s)
Find Global Cut (100 frames)‡	12720secs (3h32m)
Parametrize (1 frame)†	125secs
Encode Seq. 256×256 8bpv† (I-Frame only, given cut/param)	3.03 fps
Encode Seq. 64×64 8bpv† (I-Frame only, given cut/param)	16.02 fps
Frame Size	2904 bytes
Decode Stream 256x256 8bpv†	10.57 fps
Decode Stream 128x128 8bpv†	32.99 fps
Decode Stream 64x64 8bpv†	64.03 fps

Table 1: *Geometry Video subsystems timings*

meshes; and finally 1.66 seconds will be spent encoding the resulting Geometry Images for each frame. The wavelet encoder/decoder is a research prototype[12] and its performance can be improved. For example, the wavelet basis is not hardcoded and it does not take advantage of multimedia extensions in the hardware processor. The cutting algorithm is implemented in Matlab; The sampling (rasterization) is done in software without any optimizations and without taking advantage of hardware support.

We can decode Geometry Videos at interactive rates for small image sizes (64). The decoding speed is dominated by the wavelet decoder. The larger the image size, the more computation the wavelet decoder requires. The beauty of Geometry Videos is that the performance for a Geometry Video of size 64×64 and will be the same (assuming same file size) regardless of the actual original mesh it represents.

6. Summary and Future Work

We introduced a new animated mesh representation, called the geometry video. We described an algorithm to create geometry videos from animated meshes. Our algorithm is based on the geometry image construction algorithm from Gu et al. [4]. We extend the algorithm to take advantage of frame-to-frame coherence during compression. We approach this by building a global cut that takes all frames into account. We choose the reference frame for the parametrization arbitrarily, but show, at least for our data sets, that the cut has a significant effect on the parametrization and any choice of frame would be satisfactory.

We also develop intuition for the effectiveness of wavelets in representing the geometry and residuals. The main insight is that by reorganizing and resampling the vertices of a mesh,

† Windows XP, Pentium 4 at 3.06 GHz, 533MHz front side bus
‡ Linux, Pentium 4 at 2.4 GHz, 533MHz front side bus

we can more easily "see" around the vertex to extract the existing spatial redundancy. We can see in two locally independent directions. This allows the wavelets to compress both geometry and residual information well.

The key advantage of this approach is to leverage all of the accumulated knowledge on 2D Video encoding to allow 3D animation to become another natural media. For certain classes of animation this should be possible; for others we still have to address some of the shortcomings of using a single square parametrization. It is also clear that more sophisticated transformations should help in building better predictors. The fact that we also have one continuous surface should make it efficient to select or cluster regions of similar motion together. The possibilities are many, and the advantages unique. We hope to have shown the promise of this approach, and at the very least that it is viable for a subset of animations.

Acknowledgements

We would like to thanks Matthias Müller (ETH Zürich) for providing us with the Snake and Cow animation sequences, and Daniel Vlasic (LCS/MIT) for the Dance sequence.

References

1. Marc Alexa and Wolfgang Müller. Representing animations by principal components. *Computer Graphics Forum*, 19(3):411–418, 2000.
2. Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. *Computer Graphics*, 29(Annual Conference Series):173–182, 1995.
3. Michael S. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14(4):231–250, 1997.
4. Xianfeng Gu, Steven J. Gortler, and Hugues Hoppe. Geometry images. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 355–361. ACM Press, 2002.
5. Igor Guskov, Kiril Vidimce, Wim Sweldens, and Peter Schröder. Normal meshes. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 95–102. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.
6. Zach Karni and Craig Gotsman. Spectral compression of mesh geometry. *Computer Graphics (SIGGRAPH 2000)*, pages 279–286, 2000.
7. Andrei Khodakovsky, Peter Schroder, and Wim Sweldens. Progressive geometry compression. *Computer Graphics (SIGGRAPH 2000)*, 2000.
8. Aaron Lee, Henry Moreton, and Hugues Hoppe. Displaced subdivision surfaces. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 85–94. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.
9. Aaron W. F. Lee, Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. MAPS: Multiresolution adaptive parameterization of surfaces. *Computer Graphics*, 32(Annual Conference Series):95–104, 1998.
10. Jerome Edward Lengyel. Compression of time-dependent geometry. In *Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 89–95. ACM Press, 1999.
11. Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven J. Gortler, and Leonard McMillan. Image-based visual hulls. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 369–374. ACM Press/Addison-Wesley Publishing Co., 2000.
12. Ng Mow-Song. An implementation of EZW. http://pesona.mmu.edu.my/˜msng/EZW.html, September 2002.
13. Emil Praun, Wim Sweldens, and Peter Schröder. Consistent mesh parameterizations. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 179–184. ACM Press, 2001.
14. Pedro V. Sander, Steven J. Gortler, John Snyder, and Hugues Hoppe. Signal-specialized parametrization. *Eurographics Workshop on Rendering*, 2002.
15. Pedro V. Sander, John Snyder, Steven J. Gortler, and Hugues Hoppe. Texture mapping progressive meshes. In Eugene Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 409–416. ACM Press / ACM SIGGRAPH, 2001.
16. Ariel Shamir, Chandrajit Bajaj, and Valerio Pascucci. Multi-resolution dynamic meshes with arbitrary deformations. In *Proceedings of the conference on Visualization 2000*, pages 423–430. IEEE Computer Society Press, 2000.
17. Ariel Shamir and Valerio Pascucci. Temporal and spatial level of details for dynamic meshes. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 77–84. ACM Press, 2001.
18. Gabriel Taubin and Jarek Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17(2):84–115, 4 1998.

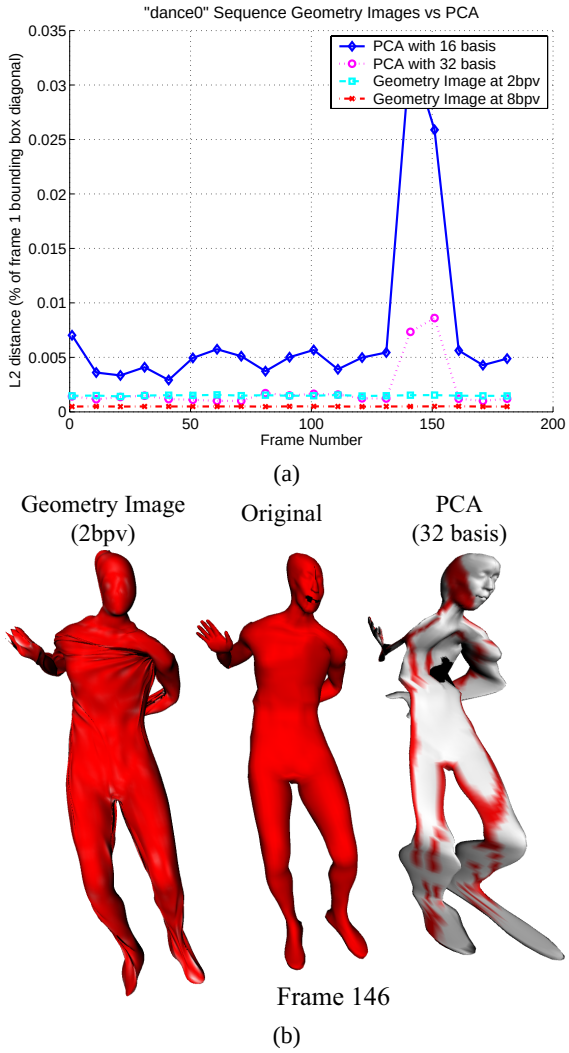


Figure 8: *PCA and Geometry Images Comparison: We compare Alexa and Müller's "Representing Animations by Principal Components" [1] method to Geometry Images. (a) The PCA encoded animation with 16 basis (no compression per basis) on 180 frames corresponds to roughly an encoding at 8bpv, which performs for most frames comparable to Geometry Images at 2bpv. (b) Objects on the bottom show the reconstruction using Geometry Images, the original mesh, and the reconstruction using PCA. The reconstructed meshes are color-coded to show the relative magnitudes of the error from the original mesh. Note the scaling distortion in the PCA reconstruction.*

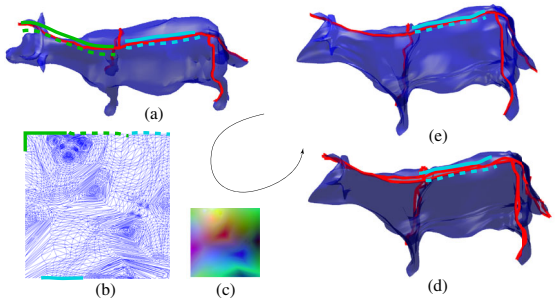


Figure 9: *Pipeline. (a) We start with a cut of the input mesh. (b) the edges of the cut will map to the boundary edges of the parametrization. (c) we raster-scan the parametrization and interpolate the points. (d) We compress, decompress and resample the mesh. The cut or seam will open if the decoding is lossy. (e) We use the boundary information to close seams.*

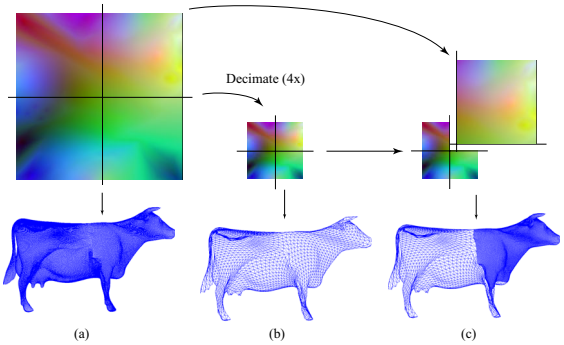


Figure 10: *Level-of-Detail: by decimating the geometry image (a) we obtain a lower resolution version (b); We can combine the two to support a multi-resolution version of the mesh (c); care must be taken at the transition edges, this is a simple operation due to the regularity of the mesh.*

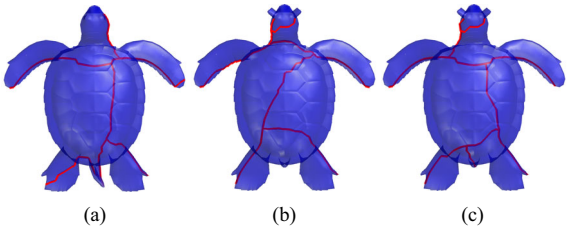


Figure 11: *Global Cut. (a) Shows the cuts of turtle with its tail sticking out. (b) Shows the cuts of turtle now with its eyes sticking out but tail retracted; tail now does not have much distortion. (c) Shows a base turtle with the global cuts built from (a) and (b)*