Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2004)
R. Boulic, D. K. Pai (Editors)

Wavelet Compression of Parametrically Coherent Mesh Sequences

Igor Guskov
Univ. of Michigan

Andrei Khodakovsky
NVIDIA

Abstract

We introduce an efficient compression method for animated sequences of irregular meshes of the same connectivity. Our approach is to transform the original input meshes with an anisotropic wavelet transform running on top of a progressive mesh hierarchy, and progressively encode the resulting wavelet details. For temporally coherent mesh sequences we get additional improvement by encoding the differences of the wavelet coefficients. The resulting compression scheme is scalable, efficient, and significantly improves upon the current state of the art for the animated mesh compression.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Curve, surface, solid, and object representations

1. Introduction

Animated sequences of meshes are commonplace in character animation, computer games, and physical simulation applications. The compact representation of this kind of data is therefore very important for efficient storage and transmission. Most often, these mesh sequences are deformations of a single irregular mesh, that is, they share the same connectivity and differ in the positioning of the vertices from frame to frame. This separation of connectivity and geometry have recently been recognized by a number of animated mesh compression approaches [Len99] [AM00] [KG] [IR03][BSM*03]. In this paper, we propose a novel solution to this problem that explicitly exploits the *parametric* (or sampling) coherence that is often present in the animated sequences of irregular meshes.

Our approach is very simple: we apply a wavelet transform running on top of a progressive mesh hierarchy, and progressively encode the resulting wavelet detail coefficients stored in a local frame. When temporal coherence is present in the input data sequence, we also employ the interframe differencing of wavelet details. The advantage of using a multiresolution representation is evident when the input sequence is a rigid-body motion; in this case the wavelet coefficients do not change from frame to frame and the positions of the coarsest level vertices are sufficient for exact reconstruction of the moving shape. The interesting animated mesh sequences are not typically rigid, and good compression performance is then dependent on the stability of the wavelet transform and its ability to decorrelate geometric information.

Our wavelet transform is a modified version of the multiresolution transform for irregular meshes introduced in [GSS99]. It belongs to the class of second-generation wavelet transforms described by Daubechies et al. [DGSS99]. The important feature of such a transform is that its wavelet filter coefficients are computed based on the geometry (or more precisely, the sampling pattern/local parameterization) of a *parametric* mesh. In the context of animated mesh processing, we can use the first frame of the sequence as such a parametric mesh, and all the other frames are transformed with wavelet filters computed from this parametric frame. The parametric frame is encoded separately with a static mesh compression technique.

The basic idea behind the wavelet transform operating on irregular meshes is to compute a wavelet detail every time a vertex is removed within a progressive mesh hierarchy; the wavelet detail is defined as the difference between the actual position of the vertex being removed and its prediction from the coarser level. The properties of such a wavelet transform are then fully determined by the predictor's ability to predict. One of the contributions of this paper is to introduce an anisotropic modification of the predictor of [GSS99] that can recover shapes with sharp creases, which can be important

for some animated mesh sequences, especially in character animation.

The idea of splitting the mesh information into connectivity, geometry, *and* parameterization has been used in static mesh compression to justify the resampling of the original data to get rid of the parametric information [KSS00][GGH02]. The parameterization information for irregular meshes represents how the mesh vertices sample the underlying shape. Looking at animated mesh sequences, we observe that they typically share not only the same connectivity but also similar local parameterization. Thus, it makes sense to transmit the connectivity and the parameterization component once for all the meshes, and then use it while reconstructing the geometry of all the frames. Our wavelet transform is dependent on the parametric information. Hence it is able to exploit the common parameterization for decorrelating the geometry of each frame, which is very important for further encoding.

Related work Recently, there has been a number of approaches to animated mesh compression. Lengyel [Len99] decomposes the mesh into a number of parts that move affinely and encodes the residual. This approach works well for certain class of animated character models controlled by a skeleton-like structure, but is less applicable to generic surface models coming from simulation.

Alexa and Müller [AM00] represent animation by expressing the difference from the mean shape within the basis of several significant PCA basis vectors. This approach should work extremely well for meshes generated by simulation of elastic materials, however its performance may deteriorate for the skeletal animations without obvious modes. One small disadvantage of PCA-based methods is the necessity of performing the PCA transformation during encoding stage. The very recent work of Karni and Gotsman [KG] improves the compression performance of the PCA based method using linear prediction coding for exploiting temporal coherence. Their method performs very well for long sequences for relatively coarse meshes, as reported in [KG]. For finer meshes, the size of the "payload" archive of eigenmodes becomes a problem if the number of frames in the sequence is small.

The Geometric Video approach of Briceno et al. [BSM*03] resamples the original mesh sequences into a geometric image [GGH02] for every frame of the animation. Remeshing plays an important role in the compression of finely sampled meshes as shown in [KSS00] and [GGH02]. However, for coarser meshes with crease features, special care should be taken to avoid reconstruction artifacts. Our approach operates directly on the original data thus avoiding the costly remeshing step and corresponding resampling artifacts.

Ibarria and Rossignac have recently introduced a Dynapack method [IR03] for compression of animated mesh sequences that exploits the interframe coherence of both temporal and parametric nature. In particular, it uses the parallelogram-like prediction to produce details during the traversal of the mesh, and then performs temporal prediction in the detail space. This is similar to our approach except that the mesh traversal within each frame is determined by a static mesh compression procedure, while our method introduces details in a multiresolution order determined by a progressive mesh. An algorithm similar to [IR03] was also introduced in [YKL02].

Our contribution Our main contribution is to exploit the parametric coherence present in animated mesh sequences for the purpose of compression. We also introduce a novel anisotropic wavelet transform that can handle geometric data with sharp creases. Our compression algorithm is fully automatic, fast, and easy to implement. It gives good compression results for synthetically animated skinned meshes, meshes produced by physically based simulations, and animated mesh sequences coming from the surface motion capture applications. We evaluate our approach on the meshes used in previous publications [BSM*03][KG][IR03], and report significant improvements in the compression performance.

2. Overview of our approach

Our compression algorithm separates parametric and geometric information for parametrically coherent mesh sequences. We first encode a specific mesh that we call the *parametric mesh (frame)* separately from the rest of the mesh sequence, and then use the information in this parametric mesh to compress the remaining frames. For time-dependent mesh sequences we typically take the mesh from the first frame as the source of the parametric mesh (see Figure 1). Our encoder and decoder thus have the *initialization stage* that processes the parametric frame (prepares a mesh hierarchy and computes wavelet filter coefficients), and the *online stage* that processes the geometric information of the mesh sequence frame by frame. The computational costs of these two stages differ, so that for a typical mesh considered in this paper (around 10K vertices), the initialization stage takes on the order of a second, while the online stage is much faster and can run in real-time at thirty frames per second.

Formally, given a sequence of meshes with the same connectivity $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_T$, we first pass the first frame $\mathcal{X}_1$ through a lossy static compression routine [KADS02]. It generates the parametric mesh archive a_{par}. Upon reconstruction it becomes an approximate version $\mathcal{M}_{par}$ of the first frame. This mesh $\mathcal{M}_{par}$ serves as the parametric region required for the operation of the wavelet transform (Section 3). Additionally, a progressive mesh hierarchy $\mathcal{L}$ is also required for running the wavelet transform, and it is built based on the parametric mesh $\mathcal{M}_{par}$. This concludes the initialization stage of the encoding algorithm, and from this

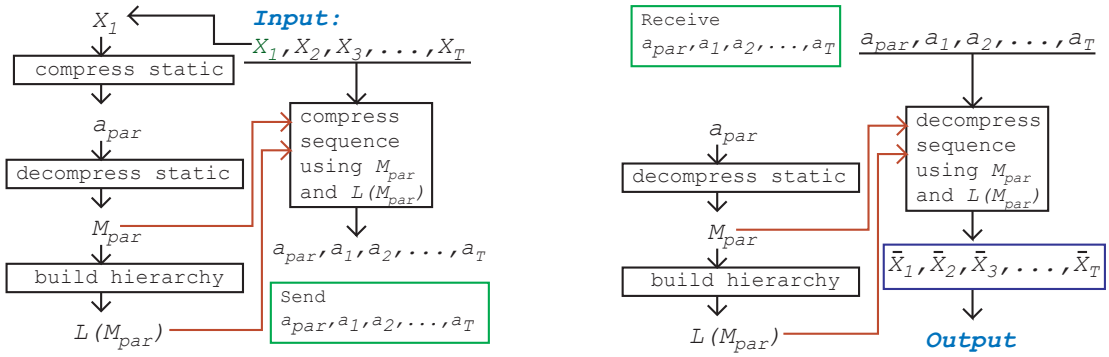


Figure 1: Overview of the encoding/decoding algorithm.

point the geometry of the animated mesh can be compressed frame by frame to obtain the individual frame archives a_t. The specifics of this online stage are described in Section 4. The resulting compressed sequence consists of the parametric archive a_{par} and the frame archives $a_1, a_2, \ldots, a_T$.

The decoder algorithm needs to receive at least the parametric archive a_{par} in order to initialize itself. The initialization replicates the last two steps of the encoder initialization (decompression of a_{par} to create the parametric mesh $\mathcal{M}_{par}$ and its simplification to create the multiresolution hierarchy $\mathcal{L}(\mathcal{M}_{par})$. After this, the decompression can be run on geometric frames. The details of the encoding of geometric frames are described in Section 4.

3. Wavelet transform

The goal of applying a wavelet transform is to decorrelate geometric data and to produce a sequence of geometric details whose distribution is amenable to further compression. We employ a modified version of the multiresolution mesh representation introduced in [GSS99]. In particular, we use a simple wavelet detail computation that takes the difference of the actual vertex position and its predicted position from a decimated mesh; thus, a single wavelet detail is stored for every vertex removal, in contrast to [GSS99] which had an average of seven details per vertex. The following section describes the anisotropic predictor used in our transform, and the detailed description of the wavelet transform itself follows in Section 3.2.

3.1. Anisotropic predictor

We shall use the notation of [GSS99] to describe our multiresolution transform. Given a non-boundary mesh edge $e = (j,k)$, we consider the four vertices $j,k,l_1,l_2 \in \mathcal{V}$ of the two triangles adjacent to e (here $\mathcal{V}$ is the vertex set of the mesh). The associated second difference operator represents the difference between the gradient values on the corresponding faces. Thus, for a function $g : \mathcal{V} \to \mathbf{R}$ we define

$$D_e^{[2]} g := \sum_{i \in \omega(e)} c_{e,i} g(i),$$

where $\omega(e) = \{j,k,l_1,l_2\}$, and coefficients $c_{e,i}$ are computed as in [GSS99]. These coefficients depend on the geometry of a particular *parametric* mesh $\mathcal{M}$ used for the parameterization. In [GSS99] the original mesh serves as the source of the parameterization; in the case of a mesh sequence, we will choose a particular single mesh as the source of parameterization. The isotropic prediction (relaxation) operator is defined via the minimization of the following discrete fairing functional

$$E(g) = \sum_e \left(D_e^{[2]} g \right)^2.$$

We modify the discrete fairing functional and make it anisotropic in order to improve its relaxation and prediction properties near sharp surface creases. Our approach is similar to recent work on anisotropic filtering for meshes[FDCO03][JDD03]. The basic idea is to weigh the contribution of each second difference based on the dihedral angle of the corresponding edge in the parametric mesh, so that the weight of edges with high dihedral angles is smaller. Thus, there will be less smoothing across creases, and more smoothing along creases. Specifically, the following anisotropic functional is used in the derivation of our prediction operator:

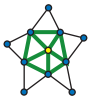
$$E_{aniso}(g) = \sum_e w_e \left(D_e^{[2]} g \right)^2,$$

where for an edge e adjacent to two faces t_1 and t_2, the corresponding weight is computed as $w_e := \varepsilon_w + e^{-(\mathbf{n}_{t_1} \cdot \mathbf{n}_{t_2})/2\sigma_w^2}$. Here the normals $\mathbf{n}_t$ are computed in the parametric mesh. We use $\varepsilon_w = 0.1, \sigma_w = 0.3$.

The minimization of the quadratic functional $E_{aniso}(g)$ with respect to the value of $g(i)$ at a vertex i results in the

prediction operator

$$P[g](i) = \frac{1}{A_i} \sum_{j \in \mathcal{V}_2(i)} a_{i,j} g(j)$$



where the coefficients are given as

$$a_{i,j} = - \sum_{\{e \in \mathcal{E}_2(i) | j \in \omega(e)\}} w_e c_{e,i} c_{e,j},$$

and their sum $A_i = \sum_{j \in \mathcal{V}_2(i)} a_{i,j}$ is used for normalization. Here, $\mathcal{E}_2(i)$ is the set of edges of the triangles adjacent to a vertex (shown in green in the picture on the left), and $\mathcal{V}_2(i)$ is the star-like neighborhood of the vertex i defined as in [GSS99]. In the picture on the left the blue vertices form $\mathcal{V}_2(i)$ for the yellow vertex i. For clarity, we sometimes add the parametric mesh used for computing the predictor coefficients as superscript $P^{\mathcal{M}}[g](i)$.

One can evaluate the qualities of a predictor by using it in the noisy mesh smoothing application. Figure 2 illustrates different smoothing results with predictors based on second difference minimization. One can see the preserved features in the anisotropically smoothed model. Also, using the true original model as the source of parameterization is not possible for practical denoising application but shows the power of the anisotropic predictor when the "true" creases are known. For the comparison of the results of the anisotropic and isotropic predictors within the compression framework, see Figure 3.

3.2. Analysis and synthesis

A progressive mesh [Hop96] is given by a sequence of edge collapses: a particular level $\mathcal{S}_{n-1}$ is obtained from a finer level $\mathcal{S}_n$ by performing a half-edge collapse that removes a vertex v_n from the mesh by collapsing it onto a neighboring vertex. The wavelet transform operates on top of a given progressive mesh sequence, and transforms a given mesh geometry into a sequence of detail coefficients and the coarse base mesh coordinates. The main purpose of the wavelet transform for our compression application is to decorrelate the data and improve their distribution for the purpose of further encoding. Preferably we would like to obtain a lot of coefficients that are close to zero and can be ignored in low bit reconstructions without sacrificing the surface quality.

Analysis The computation of wavelet detail coefficients happens every time an edge is collapsed in the progressive mesh. Consider a single edge collapse that removes a vertex v_n from the current level of progressive mesh hierarchy; the resulting *coarser* level of the transformed mesh is $\mathcal{S}_{n-1}$. The wavelet transform also uses the parametric mesh sequence $\mathcal{M}_n$ for the computation of filter coefficients, and the connectivity of both the parametric mesh $\mathcal{M}_n$ and the currently transformed mesh $\mathcal{S}_n$ is the same on all the levels of the hierarchy. The wavelet coefficient associated with the vertex v_n is computed as the difference between the prediction from the coarser level and the actual value in the mesh $\mathcal{S}_n$ expressed in a local frame $F_{v_n}^{(n-1)}$ that is computed from the coarser level mesh $\mathcal{S}_{n-1}$:

$$d_n = F_{v_n}^{(n-1)}(s_n(v_n) - P^{\mathcal{M}_n}[s_n](v_n)), \qquad (1)$$

here $s_n(v) \in \mathbf{R}^3$ denotes the coordinate vector of the mesh S_n at a vertex v.

Note that this transform does not change any vertices of the coarser mesh and only records a single detail vector per removed vertex. Therefore, our wavelet transform has no oversampling. Given any particular sequence of edge collapses for a parametric mesh $\mathcal{M}$ the described computation of wavelet details can be applied to any mesh $\mathcal{S}$ that has the same connectivity as $\mathcal{M}$; its result is a sequence of wavelet detail coefficients d and some coarse base mesh S_{base}.

Synthesis The synthesis of a mesh from a base mesh S_{base} and a sequence of wavelet details d is straightforward: one only needs to reverse the computation in formula (1) and invoke it on every edge uncollapse. We obtain the following atomic reconstruction step:

$$s_n(v_n) = P^{\mathcal{M}_n}[s_{n-1}](v_n) + \left(F_{v_n}^{(n-1)}\right)^{-1} d_n, \qquad (2)$$

note that with exception of the vertex v_n the position of all the remaining vertices in meshes $\mathcal{S}_n$ and $\mathcal{S}_{n-1}$ are identical. Also, the same parametric mesh $\mathcal{M}_n$ should be used in both reconstruction and analysis algorithm.

We observed that a good quality normal and tangent vector computation is needed for the local frame computation $F_{v^n}^{(n-1)}$ in order to ensure a stable separation of detail information in tangent and normal direction. We used the tangent plane computation as described in [ZS99], it only requires the knowledge of the positions in the one ring of the vertex and does not need the position of the vertex being predicted.

Mathematically speaking, our primal wavelet functions are simply the odd scaling functions. Potentially, a more complicated subdivision step described in [GSS99] results in smoother shapes of the wavelet and scaling functions which can be very important for geometric modeling operations. For compression purposes however, we found the performance of our simple approach to be adequate. Moreover, the simplified algorithm results in a faster reconstruction performance and less memory overhead which can be important for real-time applications.

Multiresolution levels Image and semi-regular mesh hierarchies have a well defined notion of scale; the multiresolution bases built in these settings are typically resampled to be normalized in L^2. In a progressive mesh, there is no such predefined scaling; however, in order to improve the compression performance we find it necessary to rescale the coefficients so that the fine and coarse level details carry

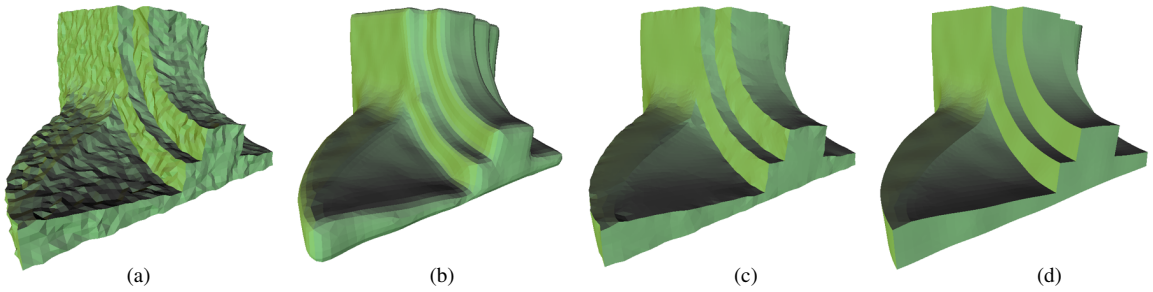


Figure 2: Comparison of different smoothing algorithms for the fandisk model: (a) noisy model; (b) isotropically smoothed; (c) anisotropically smoothed with the noisy parametric mesh; (d) "cheating" – anisotropically smoothed with the original fandisk as the parametric mesh.

the same penalty in terms of some error metric. Our approach to this is as follows: we employ a "batch" simplification method using memoryless quadrics when building the hierarchy [PR00][Hop99], and thus split the wavelet coefficients into levels. In order to make edge size distribution more even, we enforce an edge length constraint on edge removals gradually relaxing it as we proceed to coarser levels. We normalize wavelet details by the square root of the area of the one-ring of triangles adjacent to a vertex in the parametric mesh, which corresponds to approximate L_2 normalization of wavelet basis.

Notation The wavelet transform relies on the knowledge of the parametric mesh $\mathcal{M}_{par}$ and a simplification mesh hierarchy $\mathcal{L}(\mathcal{M}_{par})$ that is built from $\mathcal{M}_{par}$. It will be convenient to denote the application of this wavelet transform to the mesh $\mathcal{X}$ as follows:

$$d = W_{\mathcal{L}(\mathcal{M}_{par})}\mathcal{X},$$

where d is the resulting vector of details coefficients (encoded in local frames) appended with the positions of the base mesh vertices. The corresponding inverse wavelet transform will be written as:

$$\mathcal{X} = W^{-1}_{\mathcal{L}(\mathcal{M}_{par})}d,$$

Encoding of the parametric frame For the compression of the parametric frame, we can use potentially any static mesh compression algorithm. In our experiments we have used the algorithm of [KADS02]. Our experiments showed that the parametric frame should be encoded with at least as much precision as the expected precision of the geometry frames; this makes sense because poor reconstruction of the parametric frame can have direct effect on the reconstruction quality of all the other frames.

4. Compression of wavelet sequences

4.1. Encoding of mesh sequences

The wavelet transform of a single frame from the mesh sequence produces many coefficients that are close to zero. If a given mesh sequence represents a time-dependent evolution of shape, an additional advantage can be gained by exploiting temporal coherence within such a sequence. In order to preserve the possibility of random access to a time-dependent sequence however, one can use an approach analogous to the ones used in video compression [Say00]. We have run simple experiments that would be indicative of the performance to be expected in various temporal encoding scenarios. Specifically, we shall now describe two basic approaches for sequence compression (both rely on the knowledge of the initial parametric mesh): I-frames compress geometry of the mesh independent of other geometry frames, and P-frames encode the differences between wavelet details of adjacent frames. Both methods produce a per-frame detail stream c_t whose encoding is described in the next section. (Note that the detail stream c_t of each frame is quantized and encoded independently of other frames.)

I-frames I-frames encode the geometry of a mesh $\mathcal{X}_t$ independently of other frames, but relying on the parametric mesh hierarchy $\mathcal{L}_{par} = \mathcal{L}(\mathcal{M}_{par})$. Specifically, the detail stream c^t is computed as follows:

$$c^t = W_{\mathcal{L}_{par}}\mathcal{X}_t.$$

I-frames can be useful to allow random access to the frame sequence and also to serve as initial frames of P-frames sequence representing different "clips" of unrelated motion of the same character (e.g. walking, jumping, etcetera).

P-frames P-frames encode the differences between the adjacent frames of the animation. Note that in order to not get the accumulation of error over time it is important to take the difference between the current exact wavelet coefficients and the previous frames' *reconstructed* wavelet coefficients, that is

$$c^t = W_{\mathcal{L}_{par}}\mathcal{X}_t - \tilde{d}_{t-1},$$

where $\tilde{d}_t$ is the reconstructed wavelet details from the previous frame. For a temporally coherent sequence, this typically results in a substantial improvement of the compression

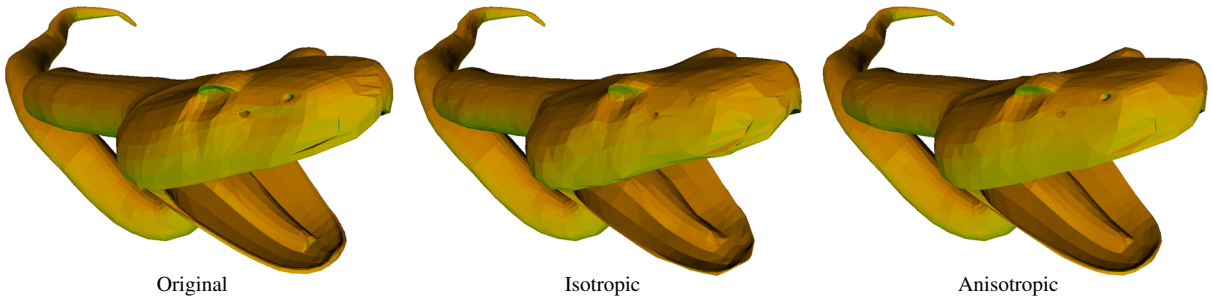


Figure 3: Left to right: original, isotropically compressed, anisotropically compressed meshes of the frame 100 of the snake sequence. Both reconstructed meshes are I-frames compressed at 2 bits per vertex (see Section 4.1). Note that anisotropic version reconstructs both nose holes, and results in a sharper reconstruction of the tongue, as well as the smoother curve of the bottom jaw silhouette.

performance, however, for a sequence consisting of only P-frames one cannot access a given frame randomly as it would require decoding of all the preceding frames. In practice, this can be alleviated by introducing an I-frame at regular intervals within a P-frame sequence. Note that in contrast to [BSM*03] we take simple differences to encode P-frames and no additional search or optimization is needed.

We have also experimented with using the parameterization of the previous frame to compress the geometry of the next frame; this can accommodate meshes with gradually changing sampling patterns; however, we did not observe significant improvements on our example data sets.

4.2. Quantization and progressive encoding of detail stream

This section describes encoding of the detail stream within a single time frame, all the interframe dependence is described in the previous section.

Some sophisticated hierarchical schemes such as the zerotree coding exist for encoding of wavelet coefficients in regular settings [SP96][KSS00]. There are no such methods for irregular meshes, though. Therefore we use a simpler, non-hierarchical progressive encoding method which consists of the two main stages: first, it initializes the stream by transmitting the positions of the most significant non-zero bits (S-bits) of all the coefficients, and then it proceeds by performing a number of bitplane refinement passes. We shall now go through these stages in more detail.

We start by quantizing the detail coefficients, that is by remapping them into the range of integer numbers:

$$c_i \mapsto |c_i| * INTEGER\_MAX / \max_j |c_j|.$$

We also store the signs of all the coefficients. Next we perform the initialization pass of the encoder: for each coefficient we send the position of its S bit. An S bit is the most

	Init	P1	P2	P3	P4	P5	P6	P7
+ 10111011	ac(8), 1	0	1	1	1	0	1	1
- 00001110	ac(4), 0					1	1	0
+ 00000001	ac(1), 1							
+ 00000000	ac(0)							

Figure 4: Simple illustration of encoding process. Four 8-bit coefficients are transmitted. During initialization pass the positions of significant bits are sent to the arithmetic coder (ac), and signs are encoded. The rest of the bits are sent during refinement passes.

significant non-zero bit of a coefficient. Therefore, these positions are numbers in the range $(0,31)$ and they are sent as symbols for the adaptive arithmetic encoder. For non-zero coefficients we also immediately send their sign bits. In the following passes we transmit the refinement bits. Obviously, we do not need to send any zero bits preceding the significant bit. Note, that only position of S bit is encoded with an entropy encoder. We found that signs and refinement bits are fairly uniformly distributed, and we send them as a raw uncompressed stream.

	Init	P1	P2	P3	Init	P4	P5	Init	P6	P7
+ 10111011	ac(8), 1	0	1	1		1	0		1	1
- 00001110	ac(0)				ac(4), 0		1		1	0
+ 00000001	ac(0)				ac(0)			ac(1), 1		
+ 00000000	ac(0)				ac(0)			ac(0)		

Figure 5: Illustration for the modified encoding process. The same four 8-bit coefficients are transmitted more optimally. Bitplane passes are grouped and significant position is send for each group. For this example we consider 4 bits in the first group and 2 in the rest.

The described scheme is progressive. Right after the initialization pass is received, it is possible to start reconstruction. The later bitplanes reduce the error and improve the reconstruction quality. On the other hand, this approach is not optimal, since we first need to send significant positions for *all* the coefficients, but the significance positions of small

coefficients are not relevant at low bit rates. To address this problem we slightly modify our approach. We organize bitplanes into groups and send significant positions within each group. For example, we first process four most significant bitplanes $(28,31)$ Any coefficient which is not significant in this group is considered to be zero. Therefore, for this group the position symbols are $(0,28-31)$. We then process these symbols as described above. After processing this group the decoder will not have any unnecessary knowledge about next bitplanes leading to more optimal rate-distortion performance. Then we consider the next group (say two bits) and continue the process. The optimal rate-distrotion perfomance is achived when each subsequent group has only one bit, since no extra information is sent. Since a length of per-bitplane symbol sequences for our models are relatevely small for an adaptive arithmetic coder to stabilize we found that sending significance symbols for subsequent bitplanes in pairs produces slightly better total rate.

Note that there are two different notions of *progressivity* and our coder possesses both of them. The first one comes from the usage of the mesh hierarchy which allows to reconstruct coarser approximations of the model if needed. This can directly affect the reconstruction time (running the wavelet transform) which may be important for real-time LOD applications. The second progressivity is the ability of our decoder to stop at an arbitrary point in the bitstream and use all the bits received so far to reconstruct the model. Every extra bit that is received decreases the total error. We can also use this functionality to vary the rate in-between frames.

5. Results

We evaluated the performance of our algorithm on the original (irregular) animated meshes from the Geometric Video project [BSM*03], and on a larger animated sequence extracted from the human skin data of Sand et al. [SMP03]. We have also run comparisons on some of the animated meshes from the work of Ibarria and Rossignac [IR03] and Karni and Gotsman [KG]. For a fair comparison, we distributed the size of the compressed parametric mesh archive uniformly to all the geometric frames, so that the reported rate is equal to the total size of all the archives in bits divided by the number of vertices and the number of frames in the sequence. The table below reports the characteristics of all the meshes used in our experiments. We included a demo version of our decoder and example archives with the supplementary materials.

Name	Num. of verts	Num. of frames	Param. archive size	Adjusted frame size (1/2/4/8bpv) in Bytes
Cow	2,904	204	7,115B	-/-/1417/2869
Dance	7,061	201	13,780B	813/1695/-/-
Snake	9,179	134	19,169B	1003/2150/4444/-
Jump	15,830	222	35,516B	1818/3797/7755/-
Face	539	10,001	2,200B	variable rate used
Kanga	4,002	65	7,212B	variable rate used

Among the three approaches we compared against, the Geometry Videos algorithm of Briceno et al. [BSM*03] was the easiest to compare against, since it also operates in a fixed rate setting, and the animated meshes considered in that paper have large vertex counts (the smallest mesh of about 3,000 vertices), which is the class of meshes we target in our multiresolution approach.

We generally observed approximately a factor of two improvement on the compression performance for the snake, cow, and dance sequences for both the P- and I-frames cases as compared to the results reported in [BSM*03] as can be seen in Figures 3 and 6. This improved performance can be explained by the good parametric coherence of the input animated mesh sequences that is exploited by our parameterization-conscious wavelet transform. (Note that all the mean-square error numbers on the vertical axis have to be multiplied by a factor 10^{-4} to get the relative error with respect to the bounding box diagonal of the first frame of the original sequence; errors were measured with Metro tool [CRS98]).

We have also compared the performance of our anisotropic (AWC) versus isotropic (IWC) wavelet transform. While the mean-square error performance of the two algorithms are similar (see the top left plot in Figure 6), we see an improved reconstruction of certain surface features and a better handling of the sharp creases as can be seen in Figure 3. We have also compared the mean-square error of the normals and saw about five to ten percent improvement in the anisotropic version.

Figure 7 shows the results of running our compression on a larger animated model of a jumping human. The four shaded meshes show the reconstruction results compared to the original frame; frame 30 used for this illustration corresponds to the time frame in which the error peaks at all three rates. Potentially, an adaptive rate adjustment can help to more uniformly distribute the error over the frames. Even in the fixed rate mode , the algorithm is able to reconstruct a good quality mesh at the rate of four bits per vertex (per frame).

The accompanying material contains the original and reconstructed meshes for the 2bpv fixed-rate archives for the snake, cow, jump, and dance animated models. The compressed archives and rendered movies of the reconstructed models are also included.

The paper of Karni and Gotsman [KG] introduces its own metric for measuring the error of the reconstructed animated mesh with respect to the original animated sequence. The metric is easy to compute and it corresponds to the relative discrete L_2 norm both in time and space. For brevity, we shall call it KG-error metric. We use this metric to evaluate our method on the face sequence considered in [KG]. The face sequence has 10,001 frames and only 539 vertices. The repetitive nature of this data sequence and its coarse mesh

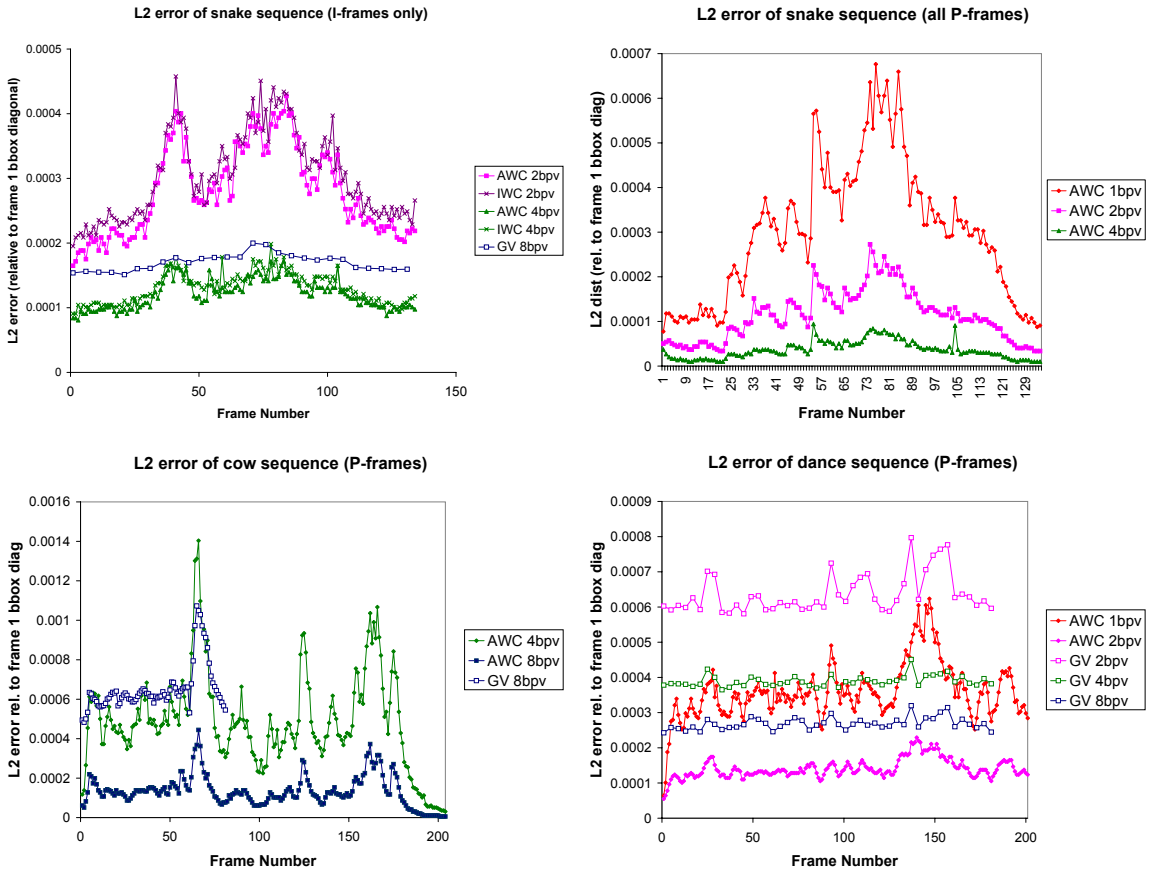


Figure 6: Results of error measurement for three animated mesh sequences encoded with P-frames only, compared with the corresponding data from Geometry Videos paper. For details, see the table in Section 5. Top left: I-frames of snake sequence; top right: P-frames of snake sequence; bottom left: P-frames of cow sequence; bottom right: P-frames of dance sequence. (Note: the error numbers on the vertical axes have to be multiplied by a factor 10^{-4}*).*

size make the application of PCA-based methods very attractive. On the other hand, the multiresolution nature of our method is practically not exploited for such a coarse mesh. The results of the comparison for the three methods are given in Figure 8. We see that the method of Karni and Gotsman is significatly better for this data sequence. However, this method considers the sequence as a whole and needs to run a PCA mode extraction. The Dynapack algorithm [IR03] and our method does not have the advantage of such a global view and have to encode the sequence as they go, from one frame to the next, possibly exploiting local interframe coherence. Their performance in this case is therefore worse than that of the method of Karni and Gotsman.

We also compared the performance of our method to the two above methods on data sets with finer mesh sizes and shorter sequences. Figure 8 shows the comparison of Dynapack and our method for the Kangaroo data set (4,002 vertices and 65 frames). There is only a single rate-distortion pair reported in [IR03], and we were not able to obtain other Dynapack data for that dataset. In our algorithm, we employed a variable rate encoding method that uses the progressive nature of our coder and chooses the rate of every frame based on the desired KG-error metric value. This is necessary, since the Kangaroo data is extremely non-uniform in time; the coder of Ibarria and Rossignac operates in the fixed error mode that typically results in a better overall performance for such temporally inhomogeneous data. The reported rate is obtained by dividing the total archive size by the number of frames and the number of vertices in the mesh.

One advantage of using a multiresolution "traversal" is the ability to quantize the detail coefficients rather than the coordinates themselves. We believe that quantizing the details results in less obvious artifacts in the appearance of the shape. The error metrics used in this paper should be augmented to adequately capture the appearance variations across different methods; for instance, Figure 9 shows two reconstructions that have the same L_2 error but differ significantly in appearance quality. This illustrates the staircase

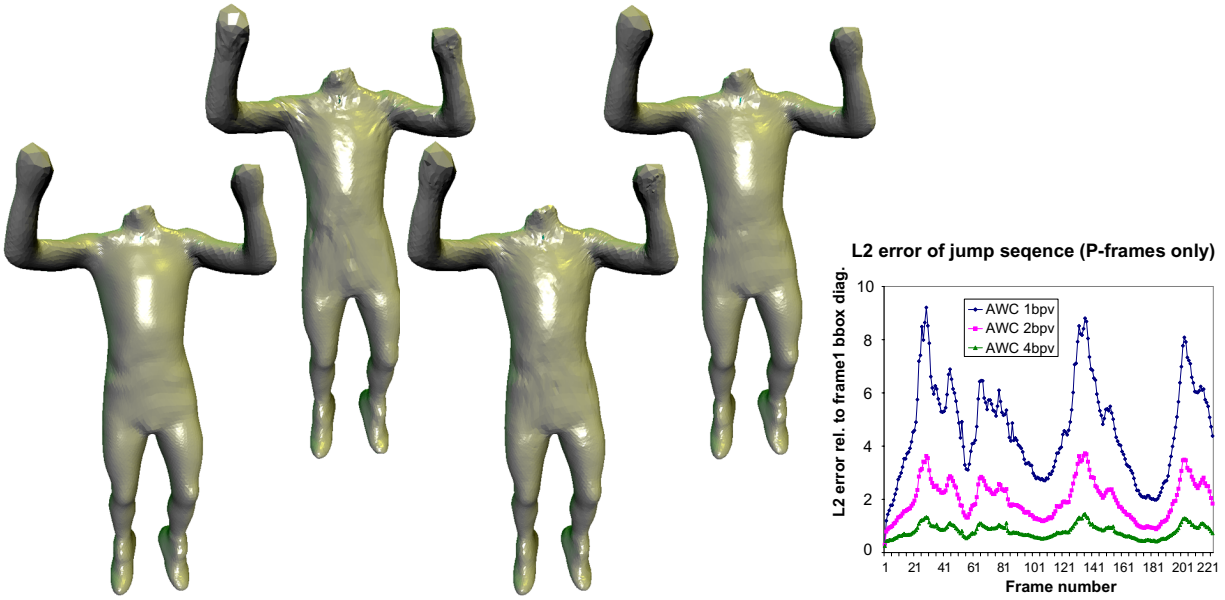


Figure 7: Frame 30 of the jump sequence showing flat-shaded and specularly lit meshes: original, encoded at 1bpv, 2bpv, and 4bpv, and the error plot for the jump sequence encoded with the parameters reported in the table in Section 5. (Note: the error numbers on the vertical axis have to be multiplied by a factor 10^{-4}).

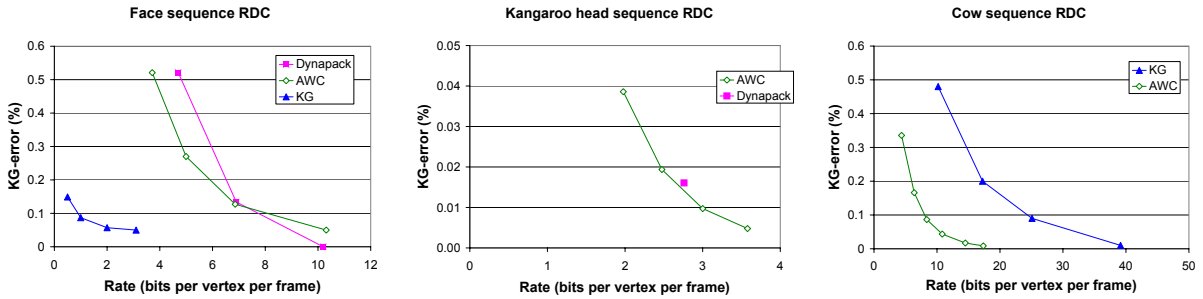


Figure 8: Rate-distortion curves for face, kangaroo, and cow sequences using KG-error.

artifacts typical for the reconstructed shapes that use low bitrate coordinate quantization.

Figure 8 shows the comparison of our method with the method of Karni and Gotsman on the cow sequence from the Geometry Videos data sets above [Kar]. We see that for this sequence our method performs better than the one from Karni and Gotsman. This is not surprising since the number of vertices in the mesh is much bigger than the number of frames in the sequence and storing all the required eigenmodes of the mesh takes the bulk of the archive in [KG].

Timings The initialization stage of our decompression algorithm took 2 sec for the largest of our models. Online stage per frame performance ranges between 0.004 sec for the cow model and 0.03 sec for the human jump model, achieved when all the coefficients of our wavelet filters and tangent

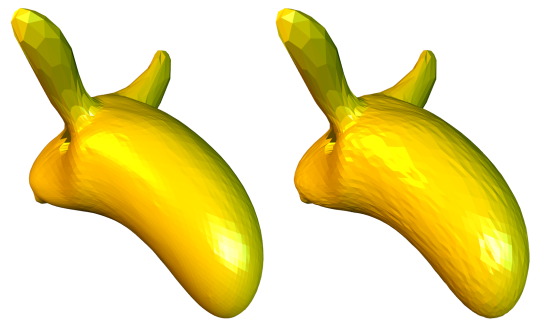


Figure 9: Comparison of two reconstructions of the last frame of the Kangaroo head sequence with the same KG-error of $\approx 0.3\%$. Left: a P-frame of our method; right: quantized at 9 bits per coordinate.

vector masks are precomputed. All timing results were measured on 3Ghz Pentium 4 PC. We do not include time needed for reading and writing the file in the reported timings.

6. Conclusions and future work

We have introduced a simple and efficient animated mesh compression approach that takes advantage of the parametric coherence of mesh sequences, and significantly improves the compression performance. Future work should include adaptive rate coding, and compression of parametrically coherent mesh sequences of changing topology.

Acknowledgements This work was supported in part by NSF (CCR-0133554). The authors would like to thank Peter Schröder and Lee Markosian for useful comments, and Nizam Anuar for technical help and for extracting jump animated mesh. The animated mesh sequences were provided by Hector Briceño, Lawrence Ibarria, and Zachi Karni. The original human jump data was provided by Peter Sand and Jovan Popovic.

References

[AM00] ALEXA M., MÜLLER W.: Representing animations by principal components. *Comput. Graph. Forum 19*, 3 (2000).

[BSM*03] BRICENO H. M., SANDER P. V., MCMILLAN L., GORTLER S., HOPPE H.: Geometry videos: a new representation for 3d animations. In *Proc. of the 2003 ACM SIGGRAPH/EG Symp. on Comp. Animation* (2003), pp. 136–146.

[CRS98] CIGNONI P., ROCCHINI C., SCOPIGNO R.: Metro: Measuring error on simplified surfaces. *Computer Graphics Forum 17*, 2 (1998), 167–174.

[DGSS99] DAUBECHIES I., GUSKOV I., SWELDENS W., SCHRÖDER P.: Wavelets on irregular point sets. *Phil. Trans. R. Soc. Lon. A.* (1999).

[FDCO03] FLEISHMAN S., DRORI I., COHEN-OR D.: Bilateral mesh denoising. *ACM Trans. Graph. 22*, 3 (2003), 950–953.

[GGH02] GU X., GORTLER S. J., HOPPE H.: Geometry images. In *Proceedings of SIGGRAPH 2002* (2002), pp. 355–361.

[GSS99] GUSKOV I., SWELDENS W., SCHRÖDER P.: Multiresolution signal processing for meshes. *Proceedings of SIGGRAPH* (1999), 325–334.

[Hop96] HOPPE H.: Progressive meshes. *Proceedings of SIGGRAPH* (1996), 99–108.

[Hop99] HOPPE H.: New quadric metric for simplifiying meshes with appearance attributes. In *Proceedings of the conference on Visualization '99* (1999), pp. 59–66.

[IR03] IBARRIA L., ROSSIGNAC J.: Dynapack: space-time compression of the 3d animations of triangle meshes with fixed connectivity. In *Proc. of the 2003 ACM SIGGRAPH/EG Symp. on Comp. Animation* (2003), pp. 126–135.

[JDD03] JONES T. R., DURAND F., DESBRUN M.: Non-iterative, feature-preserving mesh smoothing. *ACM Trans. Graph. 22*, 3 (2003), 943–949.

[KADS02] KHODAKOVSKY A., ALLIEZ P., DESBRUN M., SCHRÖDER P.: Near-optimal connectivity encoding of 2-manifold polygon meshes. *Graphical Models 64* (2002), 147–168.

[Kar] KARNI Z.: private communication.

[KG] KARNI Z., GOTSMAN C.: Compression of soft-body animation sequences. To appear in Computers and Graphics, 2003.

[KSS00] KHODAKOVSKY A., SCHRÖDER P., SWELDENS W.: Progressive geometry compression. *Proceedings of SIGGRAPH* (2000), 271–278.

[Len99] LENGYEL J.: Compression of time dependent geometry. In *ACM 1999 Symposium on Interactive 3D Graphics* (1999).

[PR00] PAJAROLA R., ROSSIGNAC J.: Compressed progressive meshes. *IEEE Transactions on Visualization and Computer Graphics 6*, 1 (2000), 79–93.

[Say00] SAYOOD K.: *Introduction to Data Compression*. Academic Press, 2000.

[SMP03] SAND P., MCMILLAN L., POPOVIC J.: Continuous capture of skin deformation. *ACM Transactions on Graphics 22*, 3 (2003), 578–586.

[SP96] SAID A., PEARLMAN W.: A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Trans. on Circuits and Systems for Video Technology 6*, 3 (1996), 243–250.

[YKL02] YANG J.-H., KIM C.-S., LEE S.-U.: Compression fo 3-D triangle meshes sequences based on vertex-wise motion vector prediction. *IEEE Trans. on Circ. and Sys. for Video Tech. 12*, 12 (2002), 1178–1184.

[ZS99] ZORIN D., SCHRÖDER P. (Eds.): *Subdivision for Modeling and Animation*. Course Notes. ACM SIGGRAPH, 1999.