Symposium of Computer Animation 03 - corrected March 1, 2007

Dynapack: Space-Time compression of the 3D animations of triangle meshes with fixed connectivity

Lawrence Ibarria and Jarek Rossignac

GVU Center, College of Computing, Georgia Institute of Technology
Atlanta, GA, USA

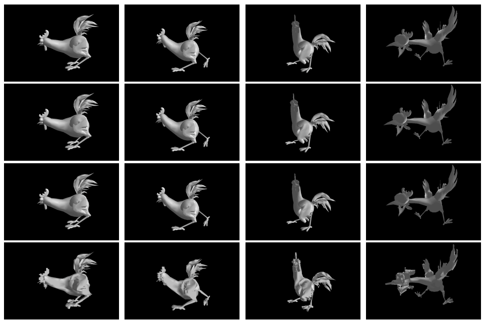

Figure 1: *The top 3D frames were taken from the Chicken Crossing animation (produced at Microsoft, courtesy of Jed Lengyel) contains* 400 *frames of the same connectivity, each having* 41 *components with a total of* 5664 *triangles and* 3030 *vertices. Dynapack quantizes the floating point coordinates of the vertices to* 13 *(respectively* 11*, and* 7*) bits, shown in rows* 2 *(respectively* 3*, and* 5*). It compresses them down to* 2.91 *(respectively* 2.35*, and* 1.37*) bits, resulting in a worst-case geometric error of* 0.0061 *(respectively* 0.024*, and* 0.3*) percent of the size of the minimum axis-aligned bounding box of the animation sequence. Note that the result of the 13-bit quantization is undistinguishable from the original and yields an 11-to-1 compression ratio over the floating-point representation with a* 42.1 *dB signal-to-noise ratio.*

1. Abstract

Dynapack exploits space-time coherence to compress the consecutive frames of the 3D animations of triangle meshes of constant connectivity. Instead of compressing each frame independently (space-only compression) or compressing the trajectory of each vertex independently (time-only compression), we predict the position of each vertex **v** of frame f from three of its neighbors in frame f and from the positions of **v** and of these neighbors in the previous frame (space-time compression). We introduce here two extrapolating space-time predictors: the ELP extension of the Lorenzo predictor, developed originally for compressing regularly sampled 4D data sets, and the Replica predictor. ELP may be computed using only additions and subtractions of points and is a perfect predictor for portions of the animation undergoing pure translations. The Replica predictor is slightly more expensive to compute, but is a perfect predictor for arbitrary combinations of translations, rotations, and uniform scaling. For the typical 3D animations that we have compressed, the corrections between the actual and predicted value of the vertex coordinates may be compressed using entropy coding down to an average ranging between 1.37 and 2.91 bits, when the quantization used ranges between 7 and 13 bits. In comparison, space-only compression yields a range of 1.90 to 7.19 bits per coordinate and time-only compressions yields a range of 1.77 to 6.91 bits per coordinate. The implementation of the Dynapack compression and decompression is trivial and extremely fast. It perform a sweep through the animation, only accessing two consecutive frames at a time. Therefore, it is particularly well suited for realtime and out-of-core compression, and for streaming decompression.

2. Introduction

Although animated 3D models may be produced and represented in a variety of ways [7], they are often stored and transmitted as series of consecutive frames, each represented by a triangle mesh which is defined by the location of the vertices and by a triangle/vertex incidence graph, sometimes referred to as the connectivity or the topology of the mesh. In general, the connectivity of the triangle mesh and even the topology (i.e. number of holes, handles, and connected components) of the surface it represents may evolve with time. Nevertheless, in this paper, similarly to several recent pioneering efforts [10] in animation compression, we restrict our attention to a reasonably large class of animations in which the connectivity is identical in all frames. This class comprises many physic-based animations of cloth deformations [21] and animations produced by warping space [18 4 11]. Through the rest of the paper, we assume that the mesh contains T triangles and V vertices and that the animation has F frames.

The connectivity of a connected, manifold triangle mesh with no handles or holes may always be encoded with less than 4V bits [13]. Small overheads must be added for each handle or hole [12]. More aggressive compressions may be obtained by using entropy or arithmetic codes combined with any one of a variety of recently developed 3D compression techniques for static triangle meshes [15 20 6 22 3 9]. Thus, the number of bits needed for storing or transmitting the connectivity of the mesh in negligible, when compared to the storage needed to encode the vertex motions.

Consequently, in the remainder of the paper, we focus on the compression of the geometry, which basically amounts to the compression of the three coordinates of each vertex, $\mathbf{v}$, in the successive frames of the animation.

Most 3D geometry compression schemes use a predictor, P($\mathbf{v}$), which, based on previously encoded/decoded information, predicts the location of the next vertex, $\mathbf{v}$. Because both the compression and the decompression algorithm perform the same predictor using corrected locations of previous vertices, only the three coordinates of the residue, $\mathbf{v}$-P($\mathbf{v}$), need to be transmitted. If the predictor is good, the residue coordinates are close to zero. Hence, the statistical distribution of the coordinates of the residues of a good predictor is biased towards zero and may be compactly encoded using entropy or arithmetic codes [17]. We distinguish three families of predictors: extrapolating, interpolating, and fitting.

The **extrapolating** predictor considers previously recovered vertex locations and extrapolates the position of each new vertex. The trajectory of a simple vertex may be extrapolated using a linear or a higher order predictor. Note that one may chose to use an extrapolation in time, in space, or both, as discussed below.

The **interpolating** predictor changes the order of transmission and may for instance start by sending the first and the last position of a vertex. Then a linear interpolation may be used to predict an intermediate vertex location. As more vertex locations are received, they refine this interpolation, which may be constructed as a piecewise linear, or higher order, interpolating curve. Thus, the quality of the predictor in general increases with the number of samples that define the interpolating curve. A typical example of this approach is the temporal sub-sampling combined with an interpolation of the decoded key-frames [2].

During compression, a **fitting** predictor performs an a priori analysis of the data, selects from a set of supported transformations the one that best approximates the global behavior of the vertices, and transmits a description of that transformation first [10]. The model may for example be a linear transformation characterized by 12 coefficients or a space warp characterized by 2 points and a radius of influence. Then, the position $\mathbf{v}(f)$ of a vertex $\mathbf{v}$ in a frame f is predicted by applying the appropriate fraction of the transformation to the initial location $\mathbf{v}(0)$ of $\mathbf{v}$.

Although interpolating predictors support progressive transmission and in the final stage produce smaller residues than extrapolating predictors of the same degree, they are, according to our experience, less effective for compression, because the residues for the early samples are typically large and thus do not compress well. Furthermore, the interpolating predictors are less suited for animation streaming than extrapolating ones.

Fitting predictors may be extremely powerful, but rely on an expensive optimization and usually require splitting the data into chunks that are each fit by a different predictor. The overhead of describing the chunks and the different predictors decreases the compression ratios of fitting predictors, especially when high accuracy is desired.

Thus, to explore an alternative to interpolating and to fitting predictor, we focus here on extrapolating predictors.

When compressing the animation of a triangle mesh, we distinguish three types of extrapolating predictors: space-only, time-only, and space-time.

A **space-only** predictor compresses each animation frame independently of the others. It predicts a vertex location from the location of its previously transmitted neighbors in the same frame. Thus, it does not exploit the time coherence resent in most animations.

A **time-only** predictor considers the motion of each vertex independently of the motions of the other vertices. It extrapolates the position of a vertex in frame f from its position in frame f-1 and possibly other antecedent frames. However, time-only predictors ignore the spatial coherence between the motion of one vertex and the motion of its neighbors, and hence must encode the similar motion changes independently.

A **space-time** predictor exploits both the space and time coherence. We advocate the used of space-time predictors and demonstrate on a few examples that they are about 2.5 times more effective than space-only predictors and 2.4 times more effective than time-only predictors. Although clearly these rations may vary considerably with the nature of the animation and with the sampling density in time and space, they are indicative of the benefits of exploiting space and time coherence together.

Note that, we cannot use a space-time extrapolating predictor, for the first frame, nor for the first few vertices of each new frame, because we do not have all of the neighbors needed by the predictor. Thus, the initial frame is compressed using a space-only predictor and a few initial vertices in each component of each frame are compressed using a time-only predictor.

We propose two new space-time extrapolating predictors.

The first extrapolating predictor introduced here is an extension of the Lorenzo predictor originally developed for compressing regularly spaced higher-dimensional scalar fields [8]. We will call it the Extended Lorenzo Predictor, abbreviated **ELP**. It is a **perfect predictor**[†] for translations and provides excellent results for more general deformations. The trivial formulation of ELP makes it particularly attractive for real-time decompression of streamed animations and possibly for hardware assist.

The second extrapolating predictor introduced here will be referred to as the **Replica** predictor, because it is capable of perfectly replicating the local geometry at various positions, scales and orientations. Although slightly more complex than ELP, it is a perfect predictor for any combination of translation, rotation, and uniform scaling.

These predictors require that for each new vertex, except the first 3, a particular configuration of neighboring vertices be available in the current and previous frame. The availability of this configuration requires re-ordering the vertices of the mesh. We describe here a trivial algorithm, called Dynapack, which performs this reordering dynamically, during a topological traversal of each connected component of the mesh. It visits the triangles and vertices in the same order as several connectivity compression schemes [19] [6] [22].

For each new vertex, Dynapack issues a call to a predictor. Thus, it makes it particularly easy to compare various predictors. We have used this facility to compare four predictors: time-only, space-only, ELP, and Replica.

Note that this paper is focused on the presentation and evaluation of two new extrapolating predictors. Hence, to keep things separate, it does not discuss or evaluate their possible combinations with other, possibly lossy, compression schemes, such as an adaptive time sub-sampling or mesh simplification, which have been discussed elsewhere. To facilitate comparison with other competing or complementary techniques, we have used a popular model of Microsoft's Chicken Crossing 3D animation, presented at SIGGRAPH 1996, to report our compression ratios.

Most 3D geometry compression techniques are dependent upon a vertex quantization step, which truncates vertex coordinates to the nearest integer in some unit. The amount of quantization controls the error and influences compression ratio. Therefore, we have reported the compression ratios for several quantization levels and have produced a video showing their effect side-by-side with the original. A selected set of frames from the video is shown in Fig. 2.

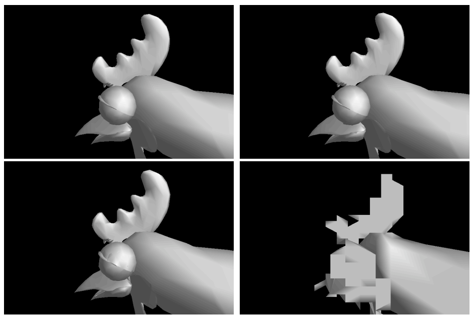
Figure 2: *This shows the head of the chicken at full precision (up left), 13-bit quantization (up-right), 11 bit quantization (down-left) and 7-bit quantization (down-right).*

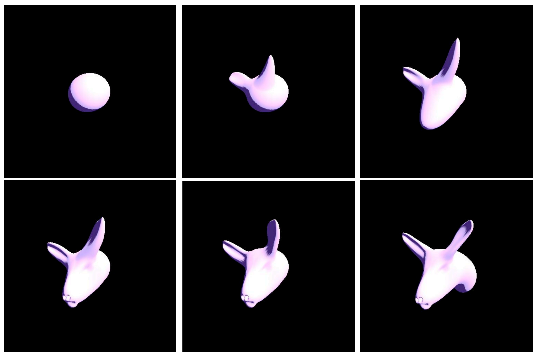
Figure 3: *This shows a model build with Twister, courtesy of Ignacio Llamas. The figure represents the animation that deforms a sphere into a kangaroo head.*

Furthermore, because the use of shape simplification and temporal down-sampling may impact the spatial and temporal coherence of the animation, and thus the compression ratios, we report compression results for various combinations of spatial and temporal sub-sampling of the animated deformation of a sphere into a Kangaroo's head, produced using the Twister system [11], see Fig. 3.

The remainder of the paper contains a review of prior art, a description of the overall Dynapack algorithm, a discussion of the space-only predictor, of the time-only predictor, and of the ELP and Replica space-time predictors, and a small comparative study of their compression power.

3. Prior Art

Several 3D compression techniques for static models exploit the fact that the decoder has reconstructed a sufficient portion of the connectivity and geometry to know several neigh-

[†] Consider that the motion of a group of vertices, for some time span, can be perfectly modeled by a transformation M(t,**v**), which returns the position of vertex **v** at time t. We say that predictor P is a perfect predictor for M if P(**v**)=M(t,**v**) for all frame-times t and for all predicted vertices **v**, in the neighborhood

bors of the next vertex **v** to be decoded. Then, the decoder uses a predictor P to compute an estimate P(**v**) of the location of **v**. It receives and decodes a corrective vector, **c**, from which it recreates **v** as P(**v**)+**c**. Several space-only extrapolating predictors have been proposed. Deering [5] uses as P(**v**) one of the previously decoded neighbors of **v**. Taubin and Rossignac [19] use a weighted combination of ancestors of **v** in vertex spanning tree, which also defines the order in which the vertices are transmitted. Touma and Gottsman have popularized the parallelogram predictor [22], which has been widely adopted [13] [3]. Given triangle (**a**,**b**,**c**), whose vertices have already been decoded, the third vertex **v** in an adjacent triangle (**c**,**b**,**v**) may is predicted to be the forth corner of a parallelogram, as P(**v**)=**b**+**c**-**a**. We compare the two space-time predictors proposed here to this space-only, popular parallelogram predictor.

The coherence between neighboring vertices in meshes of finely tiled smooth surfaces reduces the average magnitude of the residues (i.e. of the coordinates of **c**). Still, some of the residues may be large. Thus, good prediction, by itself may not lead to compression. However, the distribution of the residues is usually biased towards zero, which makes them suitable for statistical compression [17]. Entropy or arithmetic compression is particularly effective if the coordinates or the residues are quantized to a small number of bits, typically ranging between 8 and 12. Such a quantization truncates the vertex coordinates to a desired accuracy and maps them into integers that can be represented with a limited number of bits. To do this, we first compute a tight (minmax), axis-aligned bounding box around the space swept by the model during animation. The minima and maxima of the x, y, and z coordinates, which define the box, will be encoded and transmitted with the compressed representation of he animation of each object. Then, given a desired accuracy, e, we transform each x coordinate into an integer $i = INT(\frac{x-xmin}{e(xmax-xmin)})$, which ranges between 0 and 2^B, where $B = \log_2(\frac{xmax-xmin}{e})$ is the maximum number of bits needed to represent the quantized coordinate i. The y and z coordinates are quantized similarly. In practice, for static meshes, the combination of the quantization, prediction, and statistical coding reduce the storage of vertex location data to between 3 and 9 bits per coordinate, depending on the quantization and the sampling rate of the surface relative to the size of its features. Consequently, if we were to encode the geometry of each frame of the animation independently, the total cost of geometry would range between $19VF$ bits and $57VF$ bits. Deering [5] has demonstrated that streaming animations as sequences of independently-compressed 3D models is a viable approach for rendering compressed animations on a graphics board that supports real-time decompression. We believe that the proposed Dynapack scheme will help further reduce the storage and transmission costs of 3D animations and, due to its simplicity, could be considered as a possible extension of the graphics hardware capabilities for the real-time decompression of streamed animations.

In order to ensure an apparent continuity in the behavior of the animated shape, most animations are finely sampled in time, and hence exhibit a significant amount of temporal coherence, which is untapped if the frames are compressed independently of each other. Inspired by this observation, one may consider encoding the trajectory of each vertex independently. Because we need not only to encode the path followed by each vertex, but its position as a function of time, we can cast this problem as the compression of a curve in the four-dimensional space-time domain or as the problem of computing a concise representation of a parametric curve **v**(t). A variety of curve fitting approaches, reviewed in [16], could be considered here. We have decided not to pursue these trajectory compression approaches because they do not exploit the spatial coherence present in most animations. For comparison however, we report results on simple experiments, where the vertex trajectories are compressed using the simplest time-only extrapolating predictor, which encodes the displacement between the position of a vertex in frame f and its position in frame $f-1$, or a linear or quadratic predictor, which take into account the positions of the vertex in frames $f-2$ and $f-3$, assuming constant velocity or constant acceleration.

Several relatively recent efforts have pioneered the space-time compression of 3D animations of freely deforming surfaces, as opposite to rigid body motions or to physically plausible simulations of articulated bodies.

Lengyel [10] proposed several fitting predictors to compress the motion of the vertices of animated triangle meshes of a constant connectivity. His method divides the vertices of the mesh into groups and computes a transformation that best matches the average evolution of the vertices in each group. The types of transformations it can fit include Affine Transformations, Free-Form Deformations, Key-Shapes, Weighted Trajectories and Skinning. All other deformations are approximated by one of those. Then it encodes the differences (residues) between the real position of each vertex in each frame and the position predicted by applying the corresponding transformation. When large portions of the model are subject to perfect instances of these transformations, the approach is extremely effective. But the optimal partitioning of the mesh and the fitting of good transformations remains a delicate and computing intensive task. Instead of attempting to generate optimal partitions and optimal transformations, Lengyel proposes a simpler, suboptimal approach. It selects a subset of the triangles and, for each one for these triangles, computes the transformation that interpolate the evolution of their geometry through the desired frames. Triangles undergoing similar transformations may be merged. Then, other vertices are associated with the triangle whose motion best matches theirs.

Alexa and Müller [2] propose an interpolation predictor. They start by normalizing the animation. To do so, they translate all frames so that the origin lies at the center of mass of the model. Then they apply an affine transformation to it minimizing the sum of the square of the displacement for each vertex with respect to the initial frame. Put together, those modified frames form a large matrix, having for dimension the number of frames and three times the number of vertices in the mesh. Using an expensive PCA (Principal Component Analysis), they compute the eigen values of the product of that matrix with its transpose. These define eigen vectors and a coordinate system, whose axes are aligned with the principal components of the deformation. Thus the deformation is represented by this change of coordinate systems and by its coefficients in it. By setting to zero most of these coefficient, except the largest ones, they produce an approximating animation, which may be encoded with fewer bits. By sending more coefficients, they progressively refine the animation.

Al-Regib et al. [1] propose a combined approach where a possibly different set of key vertices is selected in each keyframe. Their trajectory is encoded as along as they retain the status of key vertices. The trajectories of other vertices are estimated through the interpolation of these key vertices.

We may think of these various approaches as computing and encoding a predictor for the motion of each vertex. The predictor may be phrased in terms of a global transformation or of the motion of key vertices. Because different portions of the mesh require different predictors, the vertices must be divided into groups.

In contrast, Dynapack uses the same trivial predictor for all of the vertices and for all key frames. Thus, it does not require segmenting the model nor fitting optimal transformations to it. In fact, it automatically performs the normalization and recovers rigid body and uniform scaling transformations. Thus it may be viewed as a simple and viable alternative to the more elaborate approaches listed above, although some of them may, in certain situations, yield better results.

4. Details of the Dynapack Algorithm

To precisely describe the Dynapack algorithm, we first discuss in this section the data structure used to represent the connectivity of the triangle mesh. We then explain the traversal of the mesh and its use to perform calls for the various predictors when encoding the vertex locations.

4.1. Corner table data structure and operators

We use the Corner Table [14] to store the connectivity that is common to all the frames and to traverse their vertices in an order suitable for the various predictors discussed here.

The **geometry** (i.e., vertex coordinates) is stored in the **coordinate table**, **G**, where G[v,f] contains the triplet of the coordinates of the location of vertex number v, in frame f. For conciseness, we denote it by v.g(f).

Triangle-vertex **incidence** defines each triangle by the three integer references to its vertices. These references are stored as **consecutive** integer entries in the **V table**. Note that each one of the $3T$ entries in **V** represents a **corner** (association of a triangle with one of its vertices). Let the integer c define such a corner. (We will abuse the language and speak of corner c, rather than of the corner number c.) Let c.t denote its triangle and c.v its vertex. Remember that c.v and c.t are integers in $[0, V-1]$ and $[0, T-1]$ respectively. Let c.p and c.n refer to the previous and next corner in the cyclic order of vertices around c.t.

Although the **G** and **V** tables suffice to completely specify the triangles and thus the surface they represent, they do not offer direct access to a neighboring triangle or vertex. We chose to use the reference to the **opposite** corner, c.o, which we cache in the **O table** to accelerate mesh traversal from one triangle to its neighbors. For convenience, we also introduce the operators c.l and c.r, which return the **left** and **right neighbors** of c (see Fig. 4).

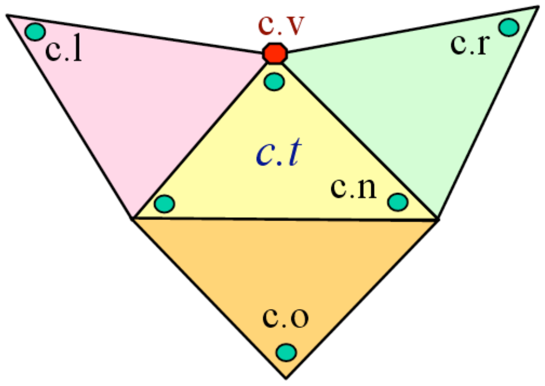

Figure 4: *Corner operators for traversing a corner table representation of a triangle mesh.*

Note that we do not need to cache c.t, c.n, c.p, c.l, or c.r, because they may be quickly evaluated as follows: c.t is the integer division c.t DIV 3; c.n is $c-2$, when c MOD 3 is 2, and $c+1$ otherwise; and c.p is c.n.n; c.l is c.n.o; and c.r is c.p.o. Thus, the storage of the connectivity is reduced to the two arrays, **O** and **V**, of integers.

We assume that all triangles have been consistently **oriented**, so that c.n.v=c.o.p.v for all corners c.

Note that **V** may be trivially extracted from most formats for triangle meshes and that **O** may be efficiently recovered from **V** [14].

To discuss the traversal of the mesh, we use the Boolean c.t.m to indicate that triangle c.t has already been visited. Similarly, the Boolean c.v.m indicates that vertex c.v has been visited. For each frame, except the first one, and for each connected component of these frames, we start by encoding the 3 vertices of a first triangle, c.t, using a time-only predictor. (The integer c may be arbitrarily chosen to be for example 0.) We mark this triangle and its vertices as visited. Then, we issue three calls: dynapack(c.o), dynapack(c.l), dynapack(c.r). These invoke the simple dynapack compression procedure presented below. It visits the other triangles of this component of the frame in a depth-first order of the triangle spanning tree and encodes its vertices. We use the convention which sets c.0 to -1 for corners that do not have an opposite corner, because their opposite edge is a border and has a single incident triangle.

```
dynapack($c$){                               #frame's component compression
  IF$c == -1$THEN RETURN;                    #return if a border is reached
  IF NOT$c.t.m$THEN{                         #if triangle c.t not yet visited
    IF NOT$c.v.m$THEN{                       #if tip vertex not yet visited
      $encode(c.v.g(f) -$ predict(c,f));     #encode residue coordinates
      $c.v.m := TRUE$;}                      #mark the tip vertex as visited
    $c.t.m := TRUE$;                         #mark the triangle as visited
    dynapack(c.r);                           #try to go to the right neighbor
    dynapack(c.l);}}                         #try to go to the left neighbor
```

Decompression follows the same pattern. For each frame, except the first one, and for each connected component of these frames, it starts by decoding the 3 vertices of a first triangle, c.t, using a time-only predictor. It marks this triangle and its vertices as visited. Then, it issues three calls: dynaunpack(c.o), dynaunpack(c.l), dynaunpack(c.r), to the decompression procedure below.

```
dynaunpack($c$){                             #decompress frame's component
  IF$c == -1$THEN RETURN;                    #return if a border is reached
  IF NOT$c.t.m$THEN{                         #if triangle c.t not yet visited
    IF NOT$c.v.m$THEN{                       #if tip vertex not yet visited
      $c.v.g(f) :=$ predict(c,f) $+ decode()$;  #decode residue, add prediction
      $c.v.m := TRUE$;}                      #mark the tip vertex as visited
    $c.t.m := TRUE$;                         #mark the triangle as visited
    dynaunpack(c.r);                         #try to go to the right neighbor
    dynaunpack(c.l);}}                       #try to go to the left neighbor
```

The first frame is compressed and decompressed using a space only predictor. We advocate the use of the inexpensive ELP and of the more general Residue space-time predictors. However, as discussed earlier, for sake of comparison, we have also implemented space-only and time-only predictors, producing the four series of results by simply changing the formula for predict(). These formulae for the four predictors are discussed in details below, using the Corner Table notation.

5. Extrapolating Predictors

We describe here four formulae for computing the extrapolating predictor, predict(c,f), from a selected set of the previously visited immediate neighbors of c.v in frames f and possibly $f-1$.

5.1. Space-only Predictor

The space-only predictor is used for encoding the first frame in Dynapack. For comparison, we also provide the result of using it to encode all the frames independently. It is exactly the parallelogram predictor popularized by Touma and Gotsman [22]. With this approach, predict(c,f) returns $c.n.v.g(f) + c.p.v.g(f) - c.o.v.g(f)$, as shown Fig. 5.

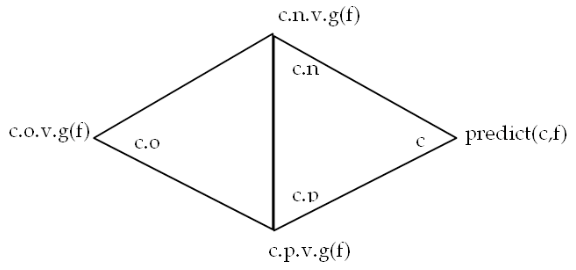

Figure 5: *Space-only predictor: predict(c,f) = c.n.v.g(f)+ c.p.v.g(f)-c.o.v.g(f).*

5.2. Time Predictor

We have also implemented a time-only predictor, which does not exploit any spatial coherence and simply returns c.n.v.g($f-1$), which is the position occupied by the vertex in the previous frame. This time-only predictor has also been used by Lengyel [10] as a "row Predictor" and is a special case of Linear Predictive Coding.

5.3. Space-time Extended Lorenzo Predictor (ELP)

The first space-time predictor, proposed here, is a generalization of the Lorenzo predictor [8] developed for compressing regular samplings of four-dimensional scalar fields. The proposed generalization simply evaluates predict(c,f) as $c.n.v.g(f) + c.p.v.g(f) - c.o.v.g(f) + c.v.g(f-1) - c.n.v.g(f-1) - c.p.v.g(f-1) + c.o.v.g(f-1)$, as illustrated in Fig. 6.

Note that ELP predicts perfectly the locations of the vertices of regions of the mesh that have been transformed by a pure translation from the previous frame. Indeed, if for all corners c, $c.v.g(f) = c.v.g(f-1) + \mathbf{d}$, then $predict(c,f) = c.v.g(f-1) + \mathbf{d}$. Thus, the residues are null.

5.4. Space-time Replica Predictor

To make our predictor capable of perfectly predicting rigid body motions and uniform scaling transformations, we have developed the new Replica predictor. It computes the coefficients a, b, and c, such that the vertex, c.v.g(f-1), can be written as $c.o.v.g(f-1) + aA + bB + cC$, with $A = c.p.v.g(f-1) - c.o.v.g(f-1))$, $B = c.n.v.g(f-1) - c.o.v.g(f-1)$, and $C = \frac{A \times B}{\sqrt{\|A \times B\|}}$.

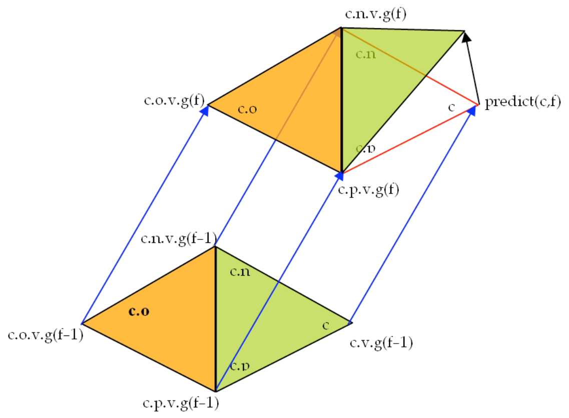

Figure 6: *ELP:* $predict(c,f) = c.n.v.g(f) + c.p.v.g(f) - c.o.v.g(f) + c.v.g(f-1) - c.n.v.g(f-1) - c.p.v.g(f-1) + c.o.v.g(f-1)$.

To compute a, b and c, we define $\mathbf{D} = c.v.g(f-1) - c.o.v.g(f-1)$ and write $D = aA + bB + cC$. Given that C is orthogonal to A and B, a dot product of both terms of the equation with vector **A** yields $A \cdot D = aA \cdot A + bA \cdot B$ and a dot product with B yields $B \cdot D = aB \cdot A + bB \cdot B$. Solving this system of linear equations yields:

$$a = \frac{A \cdot D * B \cdot B - B \cdot D * A \cdot B}{A \cdot A * B \cdot B - A \cdot B * A \cdot B} \quad (1)$$

$$b = \frac{A \cdot D * A \cdot B - B \cdot\ D * A \cdot A}{A \cdot B * A \cdot B - B \cdot B * A \cdot A} \quad (2)$$

$$c = D \cdot \frac{A \times B}{\sqrt{\|A \times B\|}} \quad (3)$$

Then, the vertex c.v.g(f) is predicted by $predict(c,f) = c.o.v.g(f) + aA' + bB' + cC'$, where $A' = c.p.v.g(f) - c.o.v.g(f))$, $B' = c.n.v.g(f) - c.o.v.g(f)$, and $C' = \frac{A' \times B'}{\sqrt{\|A' \times B'\|}}$. The situation is illustrated Fig. 7.

Less formally, the Replica predictor looks at the previous frame and expresses vertex c.v.g($f-1$) as in a coordinate system derived from triangle (c.o.v.g($f-1$), c.n.v.g($f-1$), c.p.v.g($f-1$)). More precisely, we compute the projection of c.v.g($f-1$) onto the plane supporting the previous triangle (c.o.v.g($f-1$), c.n.v.g($f-1$), c.p.v.g($f-1$)). Then we compute the distance from c.v.g($f-1$) to that plane and encode a function of its ratio, c, to the area of the adjacent triangle. The function we use guarantees that the replica predictor will not be affected by a change of units. Furthermore, we compute two coefficients, a and b, such that the vector between c.o.v.g($f-1$), and c.v.g($f-1$) may be expressed as $aA + bB$, where A and B are the vectors joining c.o.v.g($f-1$) to the other vertices of the adjacent triangle. Thus, c.o.v.g($f-1$) and c.v.g($f-1$) are the opposite corner of a parallelogram with sides parallel to A and B.

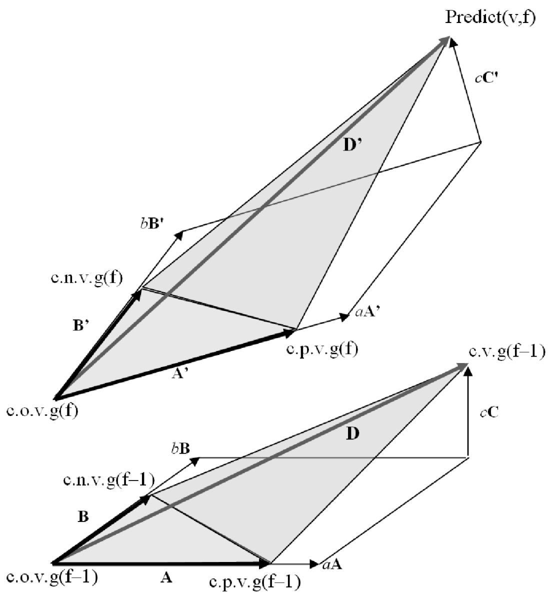

Figure 7: *Replica Predictor.*

Then, we replicate this construction on frame f, using the a, b, and c, and coefficients computed from frame $f-1$, to estimate c.v.g(f). Because this reconstruction only depends on the position of the previously visited triangle in frame f, the Replica is a perfect predictor when both triangles in frame f were obtained by moving the corresponding triangles in frame $f-1$ by the same rigid body motion. Furthermore, as we pointed out earlier, we have chosen the coefficient c to ensure that the predictor is independent of the chosen units and this will also be a perfect predictor if frame f is obtained by a rigid body motion and uniform scaling from frame $f-1$.

Clearly, Replica predicts perfectly the locations of the vertices of regions of the mesh that have been transformed by a rigid body motion, because the construction is relative to the neighboring triangle and thus is not affected by a rigid body transformation.

The normalization of C, dividing $A \times B$ by $\sqrt{\|A \times B\|}$ was introduced to ensure that Replica is also a perfect predictor for uniform scaling.

6. Results

To demonstrate the effectiveness of our two time-space predictors, and to compare them to the time-only and space-only predictors and to results obtained by others, we have tested it on two different animations: "Head-Shaping" and "Chicken Crossing".

For each animation, we report the average number of bits per coordinate when using each one of the four predictors: space-only, time-only, ELP, and Replica.

In order to demonstrate the dependency of these results on the lossy quantization, we provide results for four different quantizations of the vertex coordinates.

We illustrate the errors that result from these quantizations by showing several frames of the Chicken Crossing animation at different degrees of quantization in Fig. 1.

Zoomed images showing the effects of the quantization on a detail are show in in Fig. 2.

Head Shaping	7 Bit	9 Bit	11 Bit	13 Bit
Space-only	3.07	4.94	6.98	9.16
Time-only	0.80	1.13	1.52	2.02
ELP	0.61	0.96	1.42	2.05
Replica	0.60	0.94	1.39	2.02

Table 1: *This shows the compression results in bits per coordinate for the Head Shaping animation. To avoid biasing the results by over-sampling in space or time, we used a sub-sampled version having 64 frames and 250 vertices.*

Chicken Crossing	7 Bit	9 Bit	11 Bit	13 Bit
Space-only	1.90	3.37	5.20	7.19
Time-only	1.78	3.29	5.03	6.91
ELP	1.37	1.79	2.28	3.01
Replica	1.37	1.83	2.35	2.91

Table 2: *This shows the compression results in bits per coordinate of the Chicken Crossing animation having 41 connected components, 3030 vertices, 5664 triangles, 400 frames.*

One may notice that the ELP and Replica predictors yield nearly identical results. Both are consistently better than space-only and time-only predictors.

Note that we have been using a zero-order time-only predictor because we did not want to have to access more than the current and previous frame. To be fair to time-only extrapolating prediction, we have compared below the zero-order time-only predictor to higher-order ones, computed as follows. Running the time-only predictor a second time on its residues raises it to a first order time-only predictor, which may also be computed as: $2 * c.n.v.g(f-1) - c.n.v.g(f-2)$. Repeating this process a third time produces a second order time-only predictor, which may also be computed directly as $3 * c.n.v.g(f-1) - 3 * c.n.v.g(f-2) + 3 * c.n.v.g(f-3)$. The results are shown in Table 3 for the Chicken Crossing data set. Notice that second-order time-only prediction does not improve upon first order. Nor do subsequent passes. First-order is significantly better than zero-order, but still not competitive with the space-time predictors.

Chicken Crossing	7 Bit	9 Bit	11 Bit	13 Bit
Zero-order Time-only	1.78	3.29	5.03	6.91
First-order	1.62	2.43	3.57	5.01
Second-order	2.19	2.96	3.91	5.07

Table 3: *This shows the compression results in bits per coordinate of the Chicken Crossing animation having 41 connected components, 3030 vertices, 5664 triangles, 400 frames.*

To illustrate the dependency of the compression ratios on the time and space sampling frequencies, we have started with a very high resolution version of the Head Shaping animation with 6482 frames and 16000 vertices and have compared compression results for various combinations of sub-sampling in time and space. We have used the Replica predictor with 13-bit quantization. The results are shown in Table 6. Notice that, as expected, compression results increase with sampling in both time and space. Furthermore, notice that our approach is capable of exploiting coherence in time when the animation is super-sampled in time but not in space; or coherence in space, when the mesh is over-sampled in space but not in time.

Note that, as demonstrated by our experiment, although the benefits of time-coherence diminish with temporal sub-sampling, they are significant (58% savings when 3 frames out of every 4 are dropped, and 33% savings when 15 out of every 16 frames are dropped). Therefore, the proposed extrapolating predictors will be valuable, even if one were to use them to compress a sub-sampled set of key-frames and morph between the transmitted key-frames to restore the missing frames.

Head Shaping	648 Frames	64 Frames	6 Frames
1/4 vertices	0.38	0.92	3.27
1/16 vertices	0.55	1.33	4.83
1/64 vertices	0.78	2.02	7.95

Comparing the compression results achieved with Dynapack to those of other previously published animation compression approaches has proven rather difficult, because the reported results describe lossy compression and because the resulting errors, if at all reported, is measured using a variety of ways, which do not easily map into the quantization errors used by Dynapack, which guarantee a bound on the worst case Hausdorff error between the original models and the compressed ones. In spite of these difficulties, we can safely conclude the guaranteed maximum error of the animations compressed with Dynapack is comparable with the reported error in animations compressed with other approaches that would yield a similar compressed file size, although it is not clear that these reported errors describe the conservative worst case deviation (as we do) or a more forgiving least square statistical measure of it.

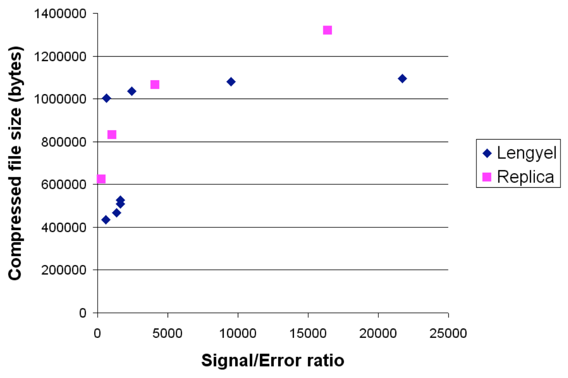

Figure 8: *Comparision results with Lengyel technique.*

Fig. 8 shows a comparison of our method with the results obtained by Lengyel [10], for which the error was expressed in dB using signal/noise ratio. Note that one dB corresponds to $10\log_{10}(Error/range)$. Our interpretation of the reported results is that Lengyel's approach yield 1.0 bit per coordinate when the model is highly quantized. Although Lengyel uses a different quantization from ours, the magnitude of the error he reports, which, to give him the benefits of the doubt, we assume to be the worst case error bound, is similar to the error we obtain when using an 8-bit quantization, for which Dynapack yields 1.5 bits per coordinate with Replica and 1.45 with ELP. Hence, for such over-quantized models Dynapack results in a 45% increase in storage over Lengyel's approach. Note that this penalty may still be acceptable, and that one may chose to trade the better compression results of Lengyel for the simplicity of Dynapack. When using an 11 bit quantization, Dynapack compressed the entire animation to 1.06 Mbytes. Lengyel's approach produces a file of 1.03 Mbytes with a comparable accuracy. Hence, both approaches yield comparable results in this case. Lengyel does not report compression results that would match the accuracy of our 13-bit quantization. Also, Lengyel's result with more accuracy need 6.6 Mbytes, while a comparable compressed mesh of ours would result in 1.35 Mbytes (quantizing to 15 bits).

Alexa and Müller [2] do not provide an explicit error measure and the small size of the illustrations in their paper prevent us from estimating the accuracy of their compression. They report compression results between 3.85 bits per coordinate and 0.8 bits per coordinate. The 0.8 bits per coordinate animation shows errors that are significantly larger that those produced by Dynapack for 7 bit quantization.

We conclude that Dynapack with either the ELP or the Replica predictor yields results that are comparable to, and sometimes better than, results reported in recent animation compression schemes. Its strength stems from the simplicity of its implementation, which does not require preprocessing and works on streaming animations requiring only to buffer the previous frame.

The two animation sequences for which we were able to run the tests reported here do not demonstrate the benefits or Replica over ELP that we were anticipating. Still, we continue to believe that Replica has considerable advantages over ELP for animations in which large portions of the surface undergo major rotations and scaling.

7. Conclusion

Dynapack is a very simple compression schemes for the 3D animations of triangle meshes of constant connectivity that undergo arbitrary deformations. Because Dynapack requires only accessing the previous frame when compressing or decompressing an animation frame, it is particularly well suited to real-time compression, out-of-core compression, and decompression of streaming animations. Because of its simplicity, it may prove to be a good candidate for a hardware assisted decompression. Dynapack may be implemented using a trivial algorithm that traverses the triangles of the mesh. It supports two space-time predictors. The Extended Lorenzo predictor (ELP) reduces to nearly zero the cost of encoding portions of the animations where a subset of the mesh undergoes a pure translation. Its main advantage lies in the fact that the predictor formula uses only point additions and subtractions. The more elaborate Replica predictor extends the nearly zero-cost prediction capability to combinations of all rigid body motions and uniform scaling transformations. The performance of both decays gradually as the behavior of the mesh departs from these simple transformations and as the space and time sampling density is decreased. Still, even for subsampled meshes, these predictors are superior to space-only and to time-only predictors, and hence will benefit other compression techniques that subsample the data and rely on interpolation for restoring the missing frames.

Acknowledgements

We wish to thank the NSF and DARPA/LLNL for their support of this work. We wish to thank Ignacio Llamas for providing us with the animation model of the Head Shaping and John Snyder for giving us access to the Chicken Run animation data produced by Microsoft. The chicken character was created by Andrew Glassner, Tom McClure, Scott Benza, and Mark Van Langeveld. This short sequence of connectivity and vertex position data is distributed solely for the purpose of comparison of geometry compression techniques.

References

1. AL-REGIB, G., ALTUNBASAK, Y., ROSSIGNAC, J., and MERSEREAU, R., "Encoding of 3d animations for efficient delivery," in *Proceedings of International Conference on Multimedia and Expo (ICME)*, vol. 1, pp. 353–356, 2002.

2. ALEXA, M. and MÜLLER, W., "Representing animations by principal components," in *Proceedings of EUROGRAPHICS 2000*, pp. 411–418, 2000.

3. ALLIEZ, P. and DESBRUN, M., "Progressive compression for lossless transmission of triangles meshes," in *Proceedings of ACM SIGGRAPH* (FUIME, E., ed.), (New York), pp. 198–205, ACM, ACM Press / ACM SIGGRAPH, 2001.

4. BARR, A. H., "Global and local deformations of solid primitives," in *Proceedings of the 11th anual conference on Computer graphics and interactive techniques*, pp. 21–30, 1984.

5. DEERING, M., "Geometry compression," in *Proceedings of ACM SIGGRAPH 95*, pp. 13–20, ACM, 1995.

6. GUMHOLD, S. and STRASSER, W., "Real time compression of triangle mesh connectivity," in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pp. 133–140, 1998.

7. HODGINS, J. and O'BRIEN, J., "Computer animation," *Encyclopedia of Computer Science*, pp. 301–304, 2000.

8. IBARRIA, L., LINDSTROM, P., ROSSIGNAC, J., and SZYMCZAK, A., "Out-of-core compression and decompression of large n-dimensional scalar fields," in *Proceedings of EUROGRAPHICS 2003*, 2003.

9. ISENBURG, M. and SNOEYINK, J., "Spirale reversi: Reverse decoding of the edgebreaker encoding," *Canadian Conference on Computational Geometry 2000*, vol. 20, no. 1, pp. 247–256.

10. LENGYEL, J. E., "Compression of time-dependent geometry," in *Proceedings of the 1999 symposium on Interactive 3D Graphics*, pp. 89–95, ACM, ACM Press, 1999.

11. LLAMAS, I., KIM, B., GARGUS, J., ROSSIGNAC, J., and SHAW, C. D., "Twister: A space-warp operator for the two-handed editing of 3d shapes," in *Proceedings of ACM SIGGRAPH 03*, Computer Graphics Proceedings, Annual Conference Series, 2003.

12. LOPES, H., TAVARES, G., ROSSIGNAC, J., SZYMCZAK, A., and SAFANOVA, A., "Edgebreaker: a simple compression for surfaces with handles," in *Proceedings of the seventh ACM symposium on Solid modeling and applications*, pp. 289–296, ACM Press, 2002.

13. ROSSIGNAC, J., "Edgebreaker," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 1, pp. 47–61, 1999.

14. ROSSIGNAC, J., SAFONOVA, A., and SZYMCZAK, A., "3d compression made simple: Edgebreaker on a corner table," in *Proceedings of Shape Modeling International Conference*, pp. 278–283, 2001.

15. ROSSIGNAC, J. and SZYMCZAK, A., "Wrapzip decompression of the connectivity of triangle meshes compressed with edgebreaker," *Computational Geometry*, vol. 14, no. 1-3, pp. 119–135, 1999.

16. SAFONOVA, A. and ROSSIGNAC, J., "Compressed piecewise circular approximation of 3d curves," *Computer-Aided Design*, vol. 35, no. 6, pp. 533–547, 2003.

17. SALOMON, D., *Data Compression: The Complete Reference*. Springer Verlag Berlin Heidelberg, 2000.

18. SEDERBERG, T. and PARRY, S., "Free-form deformation of solid geometric models," in *Proceedings of ACM SIGGRAPH 86*, pp. 151–160, ACM, 1986.

19. TAUBIN, G., GUEZIEC, A., HORN, W., and LAZARUS, F., "Progressive forest split compression," in *Proceedings of ACM SIGGRAPH 1998*, Computer Graphics Proceedings, Annual Conference Series, pp. 123–132, ACM, ACM Press / ACM SIGGRAPH, 1998.

20. TAUBIN, G. and ROSSIGNAC, J., "Geometric compression through topological surgery," *ACM Transactions on Graphics*, vol. 17, no. 2, pp. 84–115, 1998.

21. TERZOPOLOUS, D. and PLATT, J., "Elastically deformable models," in *Proceedings of ACM SIGGRAPH 87*, Computer Graphics Proceedings, Annual Conference Series, pp. 205–214, 1987.

22. TOUMA, C. and GOTSMAN, C., "Tirangle mesh compression," *Graphics Interface*, pp. 26–34, 1998.