A Measure for Mesh Compression of Time-Variant Geometry

Prasun Mathur[†], Chhavi Upadhyay[†], Parag Chaudhuri[‡] and Prem Kalra[‡]
[†]Department of Mathematics, IIT Delhi.
[‡]Department of Computer Science and Engineering, IIT Delhi.
Email: {prasun, chhavi, parag, pkalra}@cse.iitd.ernet.in

We present a novel measure for compression of time-variant geometry. Compression of time-variant geometry has become increasingly relevant as transmission of high quality geometry streams is severely limited by network bandwidth. Some work has been done on such compression schemes, but none of them give a measure for prioritizing the loss of information from the geometry stream while doing a lossy compression. In this paper we introduce a cost function which assigns a cost to the removal of particular geometric primitives during compression, based upon their importance in preserving the complete animation. We demonstrate that the use of this measure visibly enhances the performance of existing compression schemes.

Keywords: Time-Dependent Geometry Compression, Computer Animation

Introduction

Complex static meshes impose large demands on any visualization system in terms of rendering time and storage space. Compression of static geometry has been widely studied as means of overcoming these restrictions. However, complex time-dependent meshes are becoming more frequent in animation sequences. Animation modeling programs, computer vision and scanning techniques, finite element simulations are all rich sources of moving geometry. Real time visualization of such geometry demands the compression of these time-dependent geometry streams. Some compression schemes do address the case of time-dependent geometry. All these schemes try to exploit the temporal coherence of the geometry stream to achieve compression.

The compression of geometry data can be achieved by quantizing the information contained in the mesh. This can further be combined with prediction techniques and entropy encoding of the prediction errors. Another way is to actually remove or delete information contained in the mesh. This is usually a decimation step whereby some mesh primitive i.e. a vertex, an edge or a face is deleted. The selection of which information (or primitive) in the geometry is to be quantized or removed is usually done by taking into consideration its effect on the static mesh

shape only. Applying such static compression techniques to the meshes at every time step may cause simplification of areas or regions of the mesh which are essential to preserving the animation sequence.

We propose a novel cost function which assigns a cost to the removal/quantization of every primitive in the geometry depending on its importance in preserving the animation sequence. This allows a compression algorithm to decide which primitive it can safely discard or approximate coarsely without destroying the animation. We demonstrate that the use of this measure enhances the performance of existing compression schemes and actually helps preserve the characteristics of an animation during compression.

Background

It is beyond the scope of this paper to describe the many approaches that have been developed for compression of static geometric data. We describe in brief some of the approaches.

Earliest attempts at geometry compression are made by Deering [1]. Hoppe [2] defines progressive meshes. A progressive vertex insertion strategy permits the user to transfer a 3D mesh progressively, starting from a coarse mesh and then inserting vertices one by one. The topological surgery method developed by Taubin and Rossignac [3] encode both a vertex spanning tree and its triangle spanning tree. Together, these two provide enough information to recover connectivity of the original mesh. In another approach, Bajaj et. al. [4] propose an algorithm based on vertex-layering and triangle-layering schemes. They break down the mesh into simple geometric primitives which results in approximately concentric layers of vertices and triangles. Data is compressed locally within each layer. Lee et. al. [5] provide a way of adaptively parameterizing irregular connectivity triangulations of arbitrary genus 2-manifolds using a technique called Multi-resolution Adaptive Parameterization of Surfaces (MAPS). MAPS constructs a multi-resolution representation of the given mesh using geometry simplification techniques.

The first work to consider geometry as a first-class media stream is given by Lengyel [6]. It offers a technique for compression of time-dependent geometry. The approach used was to encode gross movement of the mesh with a small set of controls, while the residual differences are quantized and coded at a low bit rate. Shamir et. al. [7] present an adaptive multi-resolution representation for dynamic meshes with arbitrary deformations including attribute, position, connectivity and topology changes called the T-DAG. They also provide an online algorithm for constructing the T-DAG, enabling the traversal and use of the multi-resolution model for partial playback while still constructing it. Ibarria and Rossignac [8] propose the Dynapack algorithm which exploits space-time coherence to compress consecutive frames of the 3D animation of triangle meshes of constant connectivity. Instead of compressing each frame independently or compressing the trajectory of each vertex independently, Dynapack predicts the position of each vertex v of frame f from 3 of its neighbors in frame f and the positions of v and of these neighbors in the previous frame. Briceño et. al. [9] propose *Geometry Videos*, a new data structure to encode animated meshes. The main idea behind their work is that Geometry Videos re-sample and re-organize the geometry information in such a way, that it becomes very compressible. Another approach to animation compression compression is given by Alexa and Müller [10]. They determine the principle animation components based on a PCA analysis of the space of the animation frames. The principal component analysis brings out the principal components of the deformation. Compression is achieved by quantization of the PCA components.

By using more coefficients, a progressively refined animation can be obtained. In another work Kwatra and Rossignac [11] give a scheme for compression of 2D cel animations. They treat the stack of frames as a 3D volume, and then simplify the bounding surface of the volume and then encode it using the Edgebreaker compression scheme.

None of the above techniques suggest a way of assigning a priority based on a criteria or measure, which is then applied for the purpose of either decimating the mesh or quantizing the vertex information. We propose a novel cost function which assigns a cost to the removal/quantization of every primitive in the geometry depending on its importance in preserving the animation sequence.

Defining a Measure

Definitions

Let us begin by fixing a terminology and some definitions which we use while developing our measure.

We represent a triangular mesh as a pair (P,K), where P is a set of N point positions $p_i = (x_i, y_i, z_i) \in \mathbf{R}^N$ with $1 \leq i \leq N$, and K is an *abstract simplicial complex* which contains all the topological, i.e., adjacency information. The complex K is a set of subsets of $\{1,\ldots,N\}$. These subsets are called simplices and are of three types: vertices $v = \{i\} \in K$, edges $e = \{i,j\} \in K$, and faces $f = \{i,j,k\} \in K$.

For a vertex $p_i \in P$, we consider its one ring neighborhood and denote its area as a_i and its curvature as κ_i. We define curvature in the following manner.

Let there be n faces in one ring neighbourhood of the vertex p_i. We define κ_i as

$$\kappa_i = \frac{\sum_{l=1}^{n} a_{f_l} \cdot \kappa_{f_l}}{\sum_{l=1}^{n} a_{f_l}} \tag{1}$$

where a_{f_l} is the area of the l^{th} face f_l and $\kappa_{f_l} = \frac{1-(n_{p_i} \cdot n_{f_l})}{2}$, with n_{p_i} and n_{f_l} being the unit normals at the vertex p_i and face f_l respectively. An alternate definition of curvature is given by [12]. We have experimented with both the definitions and found that they give equivalent results. We have used the definition of curvature given in Equation 1 to generate our results.

All the above definitions are for a static mesh. We consider an animation as a time sequence of meshes $M_{t_0}, M_{t_1}, \ldots, M_{t_m}$ where $t_o < t_1 < \ldots < t_m$. So all the mesh attributes become a function of time i.e. $M_{t_j} = (P_{t_j}, K_{t_j})$. The actual time step values are irrelevant, so we normalize the time-steps to unitary intervals $\{t_j\} \leftarrow j$. Since there are m frames in the animation we have $0 \leq j \leq m$. Let a_{i_j} be the area associated with the vertex $p_i \in P_j$ and similarly, let κ_{i_j} be the curvature associated with it.

The cost function

Now we will actually define the measure. The measure is best defined as a cost function:

$$c(f_1, f_2, \ldots, f_r) = w_1 \cdot f_1 + w_2 \cdot f_2 + \cdots + w_n \cdot f_r \tag{2}$$

where the f_i are various criteria describing the animation sequence. The value of this function for a particular mesh primitive is its importance value i.e. the value of the function is a measure of the importance of the information carried by that geometry primitive to the animation sequence. The weights w_i denote the relative importance of the various criteria used to compute the cost function to the animation. The primitives for which the function returns a high value are more likely to be preserved than primitives for which the functions returns a low value during a geometry compression step.

We develop two such criteria - a change in curvature criterion, and a dynamic joint criterion.

Change in curvature criterion - CURV

Every vertex has associated with it a curvature as defined by Equation 1. The change in curvature criterion is meant to capture the deformable regions of the time-dependent dynamic mesh. It is based on an assumption that groups of vertices in the mesh whose curvature undergoes substantial change during the animation are more important to preserving the animation than vertices whose curvature does not change much. The basis of this assumption is that animation is primarily composition of various movements, and preserving vertices where curvature changes helps preserve the geometry essential to describing the movement. We define this criteria in the following manner. We define the average curvature of each vertex p_i, with m and κ_{i_j} as defined earlier, as $\bar{\kappa_i} = \frac{\sum_{j=0}^{m} \kappa_{i_j}}{m}$. If the deviation in curvature at any frame j for a vertex p_i be $\delta\kappa_{i_j} = abs(\bar{\kappa_i} - \kappa_{i_j})$ then the average deviation in curvature is defined as $\bar{\delta\kappa_i} = \frac{\sum_{j=0}^{m} \delta\kappa_{i_j}}{m}$. Now, we define the change in curvature criterion function as

$$f_{curv}(p_i, \alpha) = \alpha \cdot \bar{\delta\kappa_i} + (1 - \alpha) \cdot \bar{\kappa_i} \tag{3}$$

This criterion function would give higher values for those vertices who's curvature changes more during an animation sequence. This function also captures *rate of change of curvature* at a vertex. For example, a curvature change spread over a 100 frame animation will be given less importance than the the same curvature change spread over 25 frames, repeated 4 times. Note that only including $\bar{\delta\kappa_i}$ in the criteria is not sufficient in itself, because we want to preserve all vertices which have a high curvature even if it does not change during the animation. This is based on the assumption that high curvature areas define important shape features of the mesh and hence must be preserved. This is achieved by including the $\bar{\kappa_i}$ term in the criteria.

Dynamic joint criterion - SKEL

Animations that are generated by embedding an articulated skeleton in a mesh are better described by this criterion. In the animation some joints of the embedded articulated structure can be static while others may be dynamic. The dynamic joints move and so the mesh vertices influenced by the joint move with it. This criterion tends to give more importance to those vertices which are associated with a dynamic joint than a static joint. The basis for this argument also lies in preserving the moving elements in an animation as was the case in defining the CURV criteria.

We first figure out which joints of the skeleton are dynamic joints, i.e., which joints move the most over time. For every joint, we find out the absolute sum of the change in angle at the joint. The angle change is computed for all the segments of the skeleton incident on the joint. The joints for which this sum is more than a specified threshold are labeled as dynamic.

In order to find out the region of influence of a dynamic joint, we construct a *kd-tree* [13] for the mesh and select the vertices of the mesh in the nearest neighborhood of the dynamic joint. Now we define the dynamic joint criteria in the following manner.

Let

$$dyn(jt,\alpha) = \begin{cases} \alpha & \text{if } jt \text{ is a dynamic joint} \\ (1-\alpha) & \text{if } jt \text{ is not a dynamic joint} \end{cases} \tag{4}$$

and

$$\begin{aligned} joint(p_i) \quad = \quad & \{jt|\ jt \text{ is the joint in whose region of} \\ & \text{influence the vertex } p_i \text{ lies}\} \end{aligned} \tag{5}$$

Then we define the dynamic joint criteria function as

$$f_{skel}(p_i,\alpha) = dyn(joint(p_i),\alpha) \tag{6}$$

The complete measure : CURV+SKEL

The complete measure is defined as a linear combination of the criterion functions defined previously and is given by $c = w_1 \cdot f_{curv} + w_2 \cdot f_{skel}$.

Here w_1 and w_2 are the relative weights given to the two above defined criterion. It can be seen here that the cost function is very flexible. More criteria can be very easily added to it to better describe the animation sequence, which would help preserve to a larger extent the primitives of interest during simplification. For example, a criteria can be defined based on the distance of the animated mesh from the rendering viewpoint i.e. what level of detail is sufficient to describe the object at the distance in question. We can also define a criteria based on the velocity of movement of the dynamic mesh in the animation.

It is interesting to note that criteria based on velocity and distance from viewpoint can give rise negative weights in the cost function i.e. according to these criteria a primitive may be suitable for deletion while as per the curvature/dynamic joint criteria it may be worth preserving. The cost function is flexible enough to handle such conflicting criteria responses by suitably modulating the weights associated with each criteria. The criteria with higher weight will dominate the outcome of the cost function more than other criteria.

Now we describe how we have incorporated our measure into existing compression schemes.

Our measure and MAPS

We have incorporated our measure in the MAPS [5] algorithm for mesh simplification. First we give a brief description of MAPS.

MAPS : A static geometry simplification algorithm

MAPS first constructs a hierarchy of meshes. The original mesh $(P,K) = (P^L,K^L)$ is successively simplified into a series of homeomorphic meshes (P^l,K^l) with $0 \leq l \leq L$, where (P^0,K^0) is the coarsest or base mesh.

At a level l (of the mesh hierarchy), for a vertex $p_i \in P_l$, we again consider its one ring neighborhood and compute its area $a(i)$ and estimate its curvature $\kappa(i)$. MAPS assigns a priority {i} inversely proportional to a convex combination of relative area and curvature.

$$w(\lambda, i) = \lambda \frac{a(i)}{max_{p_i \in P^l} a(i)} + (1-\lambda) \frac{\kappa(i)}{max_{p_i \in P^l} \kappa(i)} \tag{7}$$

This priority is used to maintain a priority queue of vertices (simplification primitives) which are then subjected to a *vertex remove* step during mesh simplification. The hole that left is then re-triangulated.

The mesh simplification reduces the mesh to a base domain which is the mesh represented by (P^0, K^0). A vertex i in the original mesh is parameterized over the base domain as $\alpha p_i + \beta p_j + \gamma p_k$, where $\{i, j, k\} \in K^0$ is a face of the base domain and α, β and γ are barycentric coordinates. This mapping can be computed concurrently with the hierarchy construction during the simplification process.

Using this parameterization, MAPS does an adaptive, hierarchical remeshing of arbitrary meshes into subdivision connectivity meshes. The remeshed manifold meets conservative approximation bounds.

During the implementation of MAPS we came across a problem in the down-sampling stage of the algorithm whereby intersection of edges can occur during downsampling. We take care of this fact by not deleting a vertex if the deletion causes the new triangles created to intersect with any existing mesh edge.

Incorporating the measure into MAPS

To incorporate our measure into MAPS we replace the MAPS weighing function given in Equation 7 by our cost function as a method of prioritizing the vertices for simplification.

Quantitative error analysis

In order to evaluate the quality of animation obtained, we need some notion of an error associated with each level of mesh simplification. We characterize the error introduced by successive removal of vertices using the approach given in [14]. For every vertex being removed we calculate the error due to addition of new planes. This error gives the sum of distances of the vertex from the planes in its one ring neighborhood. Note that for any vertex which is not removed this error is zero since the vertex passes through all the planes. Since the new planes do not necessarily pass through the vertex, we get a non-zero value of this error. We add the error for all the vertices removed at each compression level. The error metric for deleting a vertex v is given by

$$\triangle(v) = v^T \Big(\sum_{p \in planes(v)} K_p \Big) v \tag{8}$$

where $p = [a\,b\,c\,d]^T$ represents the plane defined by the equation $ax + by + cz + d = 0$, with $a^2 + b^2 + c^2 = 1$ and K_p is the matrix:

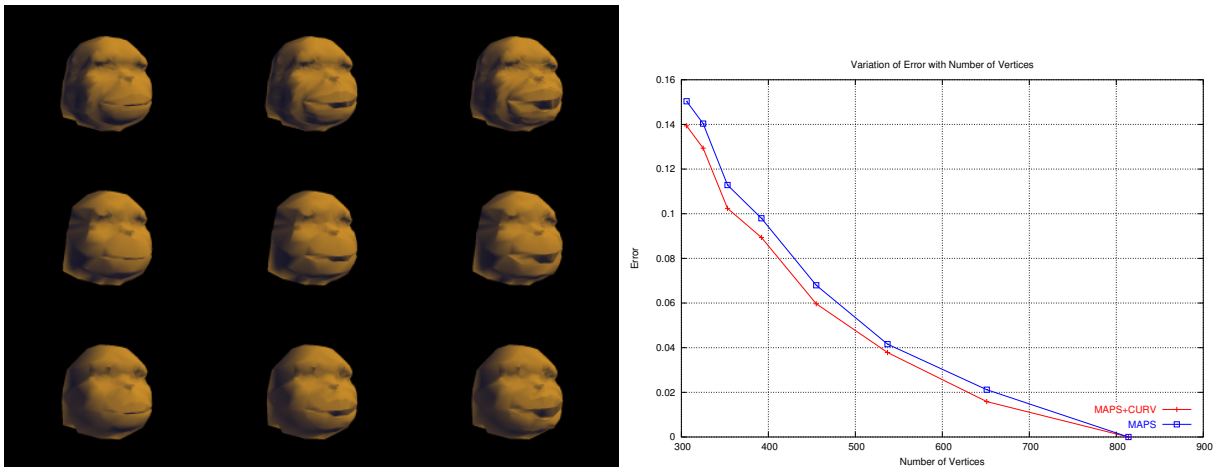
$$K_p = pp^T = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix}$$

The largest dimension of the mesh along the X, Y, or Z axes is scaled to the range 0 to 1, and the other dimensions are scaled proportionately. We do this to be able to compare the errors with respect to a normalized mesh dimension. In the following section we give the results we obtained by incorporating our measure into MAPS.

Results

We have implemented the original MAPS algorithm for mesh simplification. We have then included our measure into MAPS and performed two sets of experiments. We apply MAPS to the static mesh in every keyframe of the animation. We then show that MAPS preserves the animation characteristics of the animation better when it uses our measure to govern its mesh simplification step.

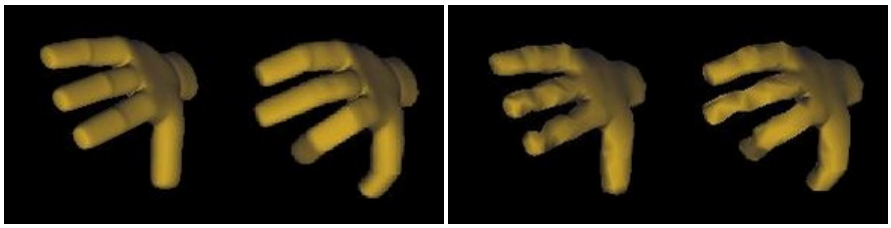
For the first experiment, we use a facial animation of a monkey's face. Here we have used only the CURV criteria to construct our cost function and then included it into MAPS. Top row of Figure 1(a) gives the original uncompressed frames form the facial animation. The middle row of Figure 1(a) gives the frames from the MAPS compressed version while the bottom row gives the frames from the MAPS+CURV compressed version (i.e. when MAPS uses our CURV criteria to govern its simplification step). In both cases, the amount of compression was the same (compression ratio 2.5), but it is clearly observed that applying just MAPS does not preserve the moving lips of the face, while MAPS+CURV does a much better job. Further quantitative validation of our observations can be seen in the quadric error graph given in Figure 1(b) which compares the error when just MAPS is applied with the error when MAPS+CURV is applied. It is clearly seen that use of MAPS+CURV results in a lower error during compression.



(a) (b)

Figure 1: Frames from the Face Animation - uncompressed and compressed, and variation of Quadric Error during compression.

For the second experiment we consider an animation of an articulated hand. The hand has an inbuilt skeletal rig which allows us to detect dynamic joints. We compressed the animation using only MAPS, MAPS+CURV,



(a) Uncompressed Hand Animation (b) Compressed with MAPS

(c) Compressed with MAPS+CURV+SKEL

Figure 2: Various compression schemes applied to the hand animation

MAPS+SKEL and MAPS+CURV+SKEL. We found that MAPS+CURV+SKEL gives the best results in this case. We can clearly observe that use of our complete measure into MAPS (shown in Figure 2(c)) preserves the animation characteristics better than using only MAPS (shown in Figure 2(b) middle row). In all the cases the compression ratio was 6.0.

Our measure and Dynapack

Dynapack : Space-Time compression of meshes

We have incorporated our measure in Dynapack [8]. The Dynapack algorithm helps reduce storage and transmission costs of 3D animations. Due to the simplicity of this algorithm, it can be considered as a possible extension of the graphics hardware capabilities for real-time decompression of streamed animations. Dynapack uses extrapolating predictors for compression. 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 encoding. Dynapack uses three types of extrapolating predictors, *Space-only predictors*, *Time-only predictors* and *Space-time Extended Lorenzo predictor(ELP)*. In our experiments, we have used the ELP. It is of the form

$$predict(c,f) = c.n.v.g(f) + c.p.v.g(f) - c.o.v.g(f) + c.v.g(f-1) - c.n.v.g(f-1) - c.p.v.g(f-1) + c.o.v.g(f-1) \quad (9)$$

Incorporating the measure into Dynapack

In an animation different vertices can be quantized to different bits depending upon their importance in preserving the shape features. We propose a way to prioritize the vertices according to their importance. Thereby, vertices which have higher priority can be quantized to more number of bits as compared to vertices with lesser priority. This way, we can achieve more compression while still preserving the quality of the animation.

We use our measure as the criteria to select the quantization level of each vertex in the animation. It is logical that vertices which are "important" in an animation (as selected by our measure) need to be quantized with more bits. Vertices which are not marked as "important" (by our measure) can be quantized to fewer number of bits. The algorithm is as given in Algorithm 1.

Algorithm 1 Marking and Quantization Algorithm

1: Run CURV (or SKEL-CURV as may be applicable) on the animation.
2: Obtain an array of flags. A vertex is flagged 1 when the vertex is "important" and flagged 0 otherwise.
3: Quantize the animation using B_1 bits (for vertices with flag 0)
4: Quantize the animation using B_2 bits (for vertices with flag 1) with $B_2 > B_1$

The quantization measure for say, the x coordinate is defined by a given accuracy e, which transforms x into an integer i where, $i = INT((x - x_{min})/(e * (x_{max} - x_{min})))$. This integer lies between 0 and 2^B, where B is the desired quantization level. We choose accuracy e as $max_{i=1}^{3}(X^i_{max} - X^i_{min})/2^B$.

Prediction schemes used in Dynapack require at most 3 neighboring vertices (of current and/or previous frame). Prediction is done as given in Algorithm 2. After Dynapack, we do entropy encoding. These encoded files are passed to the decompressor. The decompressor decodes the files and the vertex information is recovered using Dynaunpack in Algorithm 2.

Algorithm 2 Prediction Algorithm

Require: a vertex v to be predicted while traversing through Dynapack
1: **if** v is flagged 1 **then**
2: Pad neighbouring vertices of v to B_2 bits if any of them was quantized to B_1 bits.
3: **else if** v is flagged 0 **then**
4: Shift out zeroes from the neighbouring vertices of v to make them temporarily to B_1 bits if any of them was quantized to B_2 bits.
5: **end if**

Results

We implemented Dynapack and applied our method for quantization of vertices into two different bits (we will call our method as Hybrid Method). We compared our Hybrid Method with Dynapack and found that visually, the quality of the Hybrid (B_2-B_1, $B_2 > B_1$) Method is better than B_1 bits Dynapack, but not as good as the B_2 bits Dynapack (See Figure 3). Quantitatively, the compressed size of the hybrid lies between the compressed B_1 bit animation and B_2 bit animation (See Table 1).

The Signal to Noise Ratio (SNR) is computed as $SNR = \frac{Error}{Range}$. where $Error$ is the average error per coordinate, and $Range$ is 2^B, B being bits used. The quality of the hybrid animation is contingent on the difference between the bit values B_1 and B_2, and the number of vertices marked as "important" by CURV. The results in Table 1 have been generated for 50 frames, 3030 vertices and 41 connected components.

Quantization Bits	*Filesize (Kb)*	*Bits per coordinate*	*SNR*
13	182.040	3.8758	71.1201
13 - 11	154.750	3.3312	61.8477
11	140.615	3.0309	59.0969
11 - 9	121.142	2.6319	49.788
9	109.316	2.3734	47.0451
9 - 7	91.080	1.9988	37.7463
7	76.699	1.6928	35.0165

Table 1: This shows ELP compression results for the Chicken Run animation. The results have been generated for 50 frames, 3030 vertices and 41 connected components.

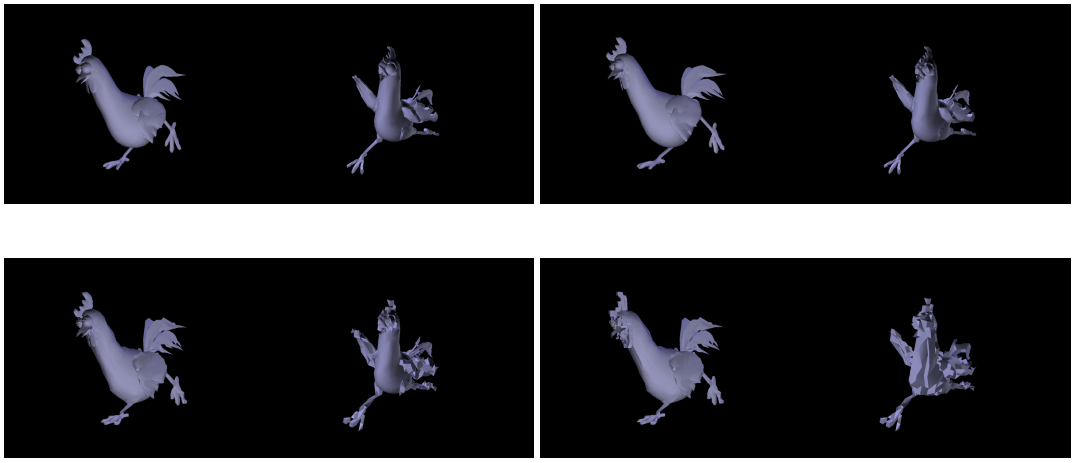


Figure 3: Frames from the Chicken Run animation. We show the frames encoded using 11-9, 9, 9-7, 7 bits respectively in left to right order, from top to bottom.

Conclusions

We propose a new measure for compression of time-variant geometry. The overall cost function accounts for the cost of removing or quantizing vertices (primitives) depending on its importance in preserving the quality of animation. We demonstrate that the use of this measure enhances the performance of existing compression schemes and actually helps preserve the characteristics of an animation during compression.

Acknowledgments

We would like to acknowledge Ms. Vinita Bhagat for her initial implementation of MAPS.

References

[1] Michael F. Deering. Geometry compression. In *Proceedings of SIGGRAPH 95*, Computer Graphics Proceedings, Annual Conference Series, pages 13–20, August 1995.

[2] Hugues Hoppe. Progressive meshes. In *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, pages 99–108, August 1996.

[3] Gabriel Taubin and Jarek Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17(2):84–115, April 1998.

[4] Chandrajit Bajaj, V. Pascucci, and G. Zhuang. Single resolution compression of arbitrary triangular meshes with properties. *Computational Geometry*, 14:167–186, 2001.

[5] Aaron W. F. Lee, Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. Maps: Multiresolution Adaptive Parameterization of Surfaces. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 95–104, 1998.

[6] Jerome Edward Lengyel. Compression of time-dependent geometry. In *1999 ACM Symposium on Interactive 3D Graphics*, pages 89–96, April 1999.

[7] A. Shamir, V. Pascucci, and C. Bajaj. Multi-resolution dynamic meshes with arbitrary deformations. In *IEEE Visualization 2000*, pages 423–430, October 2000.

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

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

[10] Marc Alexa and Wolfgang Müller. Representing animations by principal components. *Computer Graphics Forum*, 19(3):411–418, August 2000.

[11] Vivek Kwatra and Jarek Rossignac. Surface simplification and Edgebreaker compression for 2D cel animations. *International Conference on Shape Modeling and Applications*, May 2002.

[12] Sun-Jeong Kim, Chang-Hun Kim, and David Levin. Surface simplification using a discrete curvature norm. *Computers & Graphics*, 26(5):657–663, October 2002.

[13] László Szécsi. An effective implementation of the k-D tree. *Graphics programming methods*, pages 315–326, 2003.

[14] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, pages 209–216, August 1997.