MULTI-RESOLUTION STREAMING AND RENDERING OF 3-D DYNAMIC DATA

Ramanathan.S[1], Ashraf Kassim[1], Sumit Gupta[2] and Kuntal Sengupta[1]
[1]Dept of Electrical & Computer Engineering, National Univ. of Singapore, Singapore 119260.
[2]Dept. of Electronics & Computer Engineering, Indian Instt. of Technology, Roorkee 247 667, India.

ABSTRACT

The proposed algorithm attempts to combine the *partitioning based 3D dynamic data compression* and the *multi-level k-way graph partitioning* algorithms to achieve multi-resolution reconstruction of animation sequences. *Partitioning based 3D dynamic data compression* efficiently encodes changes between partitioned meshes while the *multi-level k-way graph partitioning* algorithm provides partitioning information for the input graph (mesh) at various resolutions. Combining the two provides a powerful tool to incorporate both spatial down sampling and temporal compression in a streaming framework where the server can satisfy the needs of a low-end user as well as effectively adapt to dynamically varying bandwidth constraints. At the decoder, the mesh may either be reconstructed at the streamed resolution or at a higher resolution by ingeniously adding vertices so that the viewer does not feel the effects of down sampling, and consequently, the deterioration in quality. Our algorithm deals with meshes whose connectivity does not change with time.

1. INTRODUCTION

Over the years, there has been a substantial increase in the number of 3D graphic models created and accessed through the Internet. Although many representations have been proposed for 3D models, polygon and triangle meshes are the de facto standard for exchanging and viewing 3D models. 3D polygon and triangular meshes consist of two main components, namely *position*, that gives the location of the vertices in the model and the *connectivity* that gives the adjacency relationship between the vertices. Since representation of 3D models requires voluminous data, a number of techniques have been proposed for their efficient compression and transmission. Although the emphasis has been on exploiting spatial coherence for static geometry data [1][2][3], compression techniques for dynamic 3D models whose position and connectivity can change with time have evolved in recent times [4][5][6]. The MPEG4 standard, in its bid to integrate coding and transmission of a wide range of media objects (Audio, Visual and Graphics), also addresses representation and compression of 2D and 3D meshes. [7] and [8] are recent MPEG4 compliant works on 3D model compression. While [7] proposes an error resilient mechanism for streaming of 3D wireframe animations that minimizes perceptual effect of data loss, [8] deals with coding and animation of 3D human body models.

Of late, multi-resolution techniques that address the need to provide the end user with an acceptable image quality within the bandwidth constraints imposed by the network, have gained in importance. Here again, ways have been explored to represent static meshes at various levels of detail as in [10] & [11], while very few research works [6][12] have focused on multi-resolution rendering of dynamic meshes.

The proposed algorithm aims to create a framework for multi-resolution encoding and streaming of 3D dynamic data. It combines the *partition based 3D dynamic compression* [4] and *k-way multi-partitioning algorithm* [9] to dynamically vary the resolution of the encoded frames. This facilitates effective bit rate regulation in accordance with the constantly varying network bandwidth available for the application. At the receiver, data may be reconstructed at the same resolution as that of the server or vertices may be added to the coarser mesh so as to display a model of higher resolution. The scope of the algorithm is restricted to meshes with fixed connectivity and deals only with geometry compression. The paper is organized as follows. Sections 2 & 3 describe the *k-way multi-partitioning* and *partitioning based 3D dynamic data compression* algorithms respectively. Section 4 describes how the above two algorithms could be combined to realize the proposed framework. The results are presented in Section 5 and conclusions in Section 6.

2. MULTILEVEL K-WAY GRAPH PARTITIONING

Partitions in graphs are analogous to macro-blocks in MPEG video compression. Once the input mesh has been divided into partitions, motion prediction and error coding can be performed for vertices falling in the same partition as for pixels in the same macro-block. The partitioning algorithm divides the input mesh into a specified number of sets on the basis of proximity in the connectivity. For example, assume that the input mesh A has vertices numbered from 1-16. The partitioning algorithm partitions A into Groups A^0 ={13,14,15,16}, A^1= {1,2,3,4} and A^2 ={5,6,7,8,9,10,11,12} as shown in Figure

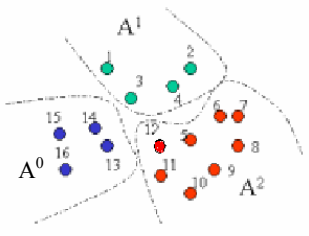

Figure 1: Partitioning algorithm output for input mesh A

Consider a graph G having a total of *n* vertices. Let V represent the positional information of the *n* vertices. The *k*-

way partitioning algorithm divides V into k subsets, where k is user-specified, such that

$$V_i \cap V_j = \varphi \text{ for } i \neq j \quad (1)$$

$$|V_i| = \frac{n}{k} \quad (2)$$

$$\bigcup_{i=1..k} V_i = V \quad (3)$$

Here, $|V_i|$ refers to the cardinality of the i^{th} partition and $\bigcup_{i=1..k} V_i$ denotes the union of the k partitions.

There are 3 stages to the multi-level k-way partitioning algorithm. First, a sequence of smaller graphs is created from the original graph. Secondly, the smallest graph in the sequence is partitioned carefully. It is easier to find a good partition for the coarser graph rather than the original, finer one. Finally, this partitioning is propagated back through the higher resolutions, with occasional local refinements. The entire process is outlined in Figure 2. The coarsening phase involves the construction of a series of coarser graphs $G_i = G(V^i, E^i)$ constructed from the original graph $G_0 = G(V^0, E^0)$ such that $|V_i| < |V_{i-1}|$. The output at every coarsening step, G_i, typically has about half the vertices as its parent G_{i-1}.

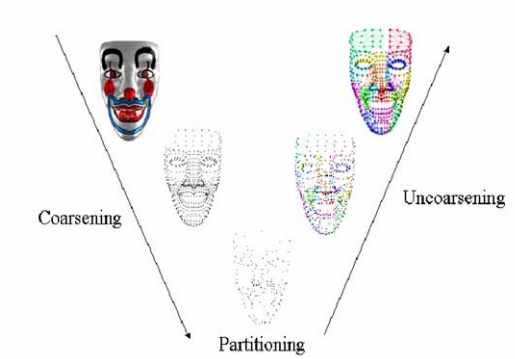

Figure 2: Three stages in graph partitioning algorithm for the *Clown* sequence.

This coarsening procedure preserves graph topology as each vertex in the fine graph is mapped to a unique vertex in the coarse graph. Therefore, any partition of the coarse graph corresponds to a partition of the fine graph. It also executes very quickly. The coarsening phase ends on obtaining the smallest graph G_m that has a very small number of vertices as compared to the original. Figure 3 illustrates the coarsened versions for the *Chicken* sequence. On obtaining the coarsest graph $G_m = G(V^m, E^m)$, it is partitioned into k sets, each containing, $|V^m|/k$ vertices. The algorithm uses a spectral partitioner that computes 1, 2 or 3 eigen vectors of the Laplacian matrix of the graph to partition the graph into 2, 4 or 8 sets respectively. The partitioning is repeated recursively until the desired number of sets is obtained. Now, the partitions obtained for the coarsest graph need to the propagated back to the finer graphs and this is achieved by un-coarsening the coarsened graph in conjunction with an occasional local refinement of the partitions. The final aim is to partition the graph so that the number of *edge cuts* – edges that connect vertices assigned to different sets, is minimized.

(a) (b) (c)

Figure 3: Original and coarsened graphs for *Chicken* sequence. (a) Original graph G_0 - 3030 vertices, 8643 edges (b) G_1-1540 vertices, 4226 edges (c) G_2 - 792 vertices, 1979 edges.

3. PARTITIONING BASED 3-D DYNAMIC DATA VERTEX COMPRESSION

This algorithm provides a very efficient scheme to code and transmit a 3D mesh model whose vertices change with time, while connectivity remains constant. The mesh at time t (termed frame) is initially partitioned using the topology based partitioning technique discussed above. The initially partitioned frame V has k sets of vertices $V(t) = [V^1(t) V^2(t) \ldots V^k(t)]$ where $V^i(t)$ represents the set of vertices that fall in the i^{th} set. To detect regions of homogeneous motion, the partitioned frames, $V(t)$ and $V(t-1)$, are now fed to the *Iterative Closest Point* (ICP) module [4]. From the ICP algorithm the partitions $V^i(t)$ and their matches at time *(t-1)*, represented by $\tilde{V}^i(t-1)$ are obtained. The best affine match between the partition $V^i(t)$ and $\tilde{V}^i(t-1)$ for $i=1,2,...,k$ is given by the best least square solution.

$$A^i = V^i(t) \times (\tilde{V}^i(t-1))^- \quad (4)$$

where $(\tilde{V}^i(t-1))^-$ represents pseudoinverse of $\tilde{V}^i(t-1)$ .It is to be noted here that the initial partitions were not selected on the basis of motion coherency and were purely based on graph theoretic properties. The initial partitions are re-clustered, split and merged to obtain regions of homogeneous motion for $V(t)$ and $V(t-1)$ and now, the displacement of vertices in these regions can be coded. The estimated value of the vertex j at time t is given by

$$\tilde{v}_j(t) = A^i \times \tilde{v}_j\ (t-1) \quad (5)$$

The error metric $\varepsilon^i = \frac{1}{n}\sum \left| v^i(t) - A^i \times \tilde{v}^i(t-1) \right|$ (6)

is used to evaluate the partitions that have relatively coherent motion where n is the number of vertices in the i^{th} partition. All partitions with an error value of less than τ (typically, 0.25 times the average distance) are deemed as regions with tolerable motion estimation error. The vertices in the frame are grouped into 3 sets based on the error metric:

- First set consisting of vertices whose estimation error is less than threshold $\lambda 1$, and therefore, the associated affine transform provides an accurate match in V(t-1).
- Second set made up of vertices with estimation error between $\lambda 1$ and $\lambda 2$, and require residual errors to be coded in addition to affine transforms.
- Third set of vertices, for which no affine transform is able to give a good match in the previous frame. These are

encoded using Differential Pulse Coded Motion (DPCM) techniques.

The vertex compression algorithm summary is as follows:

*for all (initial) partitions / * Motion Estimation algorithm */*

{

if (error for i^{th} partition V^I from (6) < λ1)

for (all vertices in V^i and neighborhood)

{

Compute estimated vertex position $\tilde{v}_j(t)$ from (5).

if (estimate error < λ1)

associate $\tilde{v}_j(t)$ with affine transform A^i.

else

if (estimate error > λ1 and < λ2)

conditionally associate $\tilde{v}_j(t)$ with A^i.

}

*} / *End of Motion Estimation*/*

Repartition conditionally associated and non-associated vertices using k-means clustering and compute affine transforms between new sets of clusters as in (4).

Repeat Motion Estimation for new clusters.

For vertices with error < λ1, transmit affine transforms.

For conditionally associated vertices, transmit affine transforms with error.

For remaining (non-associated) vertices, encode using DPCM.

I frames may be inserted whenever inter-frame motion is high. For the I frame, the position and the connectivity data are transmitted independent of the data in the preceding frames and only the spatial coherence is exploited for compression. Periodic transmission of I frames helps to minimize the effect of channel losses on animation quality. The algorithm provides high compression ratios with excellent reconstruction quality as against other contemporary techniques.

Compression Technique	Compressed data size (MB)	Compression Ratio
MPEG-4 (static)	2.7	5
Lengyel's [5]	0.5	27
Alexa's [6]	0.3492	39.8
Partition based compression [4]	0.3564	39

Table 1: Compression ratios for various algorithms for *Chicken* data with fixed connectivity (uncompressed data size - 13.9 MB).

Frame No	SNR (db) - I	SNR (db) -II	SNR (db) - III
85	46.56	36.71	32.89
187	63.87	52.96	27.61
255	64.98	52.84	27.9

Table 2: SNR values for the 3 sets of vertices for a few frames of the Chicken sequence.

For SNR calculations, the variance of the frame displacement ($V(t)-V(t-1)$) is considered the signal and the reconstruction error,($V(t)-\tilde{V}(t)$), as noise. Typically, 4 bits are used for encoding errors. As seen from Table 2, the vertices encoded with affine transforms (with and without error) have much higher SNR values compared to the vertices encoded using DPCM techniques. The overall SNR is governed by the SNR for vertices corresponding to sets 2 and 3.

4. MULTI RESOLUTION STREAMING ALGORITHM

The proposed algorithm aims to take advantage of the fact that the multi-level *k*-way partitioning algorithm can provide partitioning information for a number of resolutions of the input 3D model, starting from the coarsest resolution. Consider a situation where the available bandwidth for the streaming application, is suddenly reduced. If data streaming continues at the same resolution as before, the user would have to wait for a longer period of time to receive the data on account of increased transmission latency. A possible way to overcome the problem is to code and transmit future frames that are spatially down sampled. The process involves down sampling the reference frame and current frame to a lower resolution, coding and transmitting differences corresponding to the current frame data at this new resolution. At the receiver end, the same process is repeated to reconstruct the current frame at a lower resolution. In addition, if the client could improve the resolution by adding data from previous frames obtained at full resolution, the end user may not actually feel the degradation in quality. Alternatively, the same framework could be utilized to stream data at a reduced resolution.

Once the 3D model to be streamed is input to the server, it determines all the resolutions at which the data could be streamed. Typically, the resolution could vary from the finest graph (original) to the coarsest graph obtained from the coarsening algorithm. Let a resolution level be associated to each of these graphs with the finest graph corresponding to level 1 and the coarsest graph corresponding to level *m*. At any given point in time, the server can now invoke the multi-partitioning algorithm to provide the partitions for resolution level *r, r ε [1,m]*. The current frame and the reference frame are coarsened to the r^{th} level, and once the partitions have been obtained, the temporal differences could be coded and streamed. Starting from the coarsest graph partition, the partitions for any of the *m* resolutions can be obtained. This would enable the streaming server to dynamically switch to any of the m resolutions instantaneously.

*/*Algorithm summary - Server module */*

do

Decide on resolution r to be streamed.

Generate reference & current meshes at resolution r.

Invoke partitioning algorithm for partitions at resolution r.

Obtain partitions and transmit vertex displacements between the frames.

until (end of session)

/ Partitioning module - input resolution r & no of sets - s */*

Obtain partitioning information for coarsest graph $G=G_m$.

Obtain partitions for $G=G_{i-1}$ until(($G!=G_r$) and (no of sets != s)

At the client, the mesh corresponding to G_r is reconstructed. In situations where higher resolution display is possible, if the client can improve the resolution from G_r to G_{r-1} instantaneously, the end user does not really perceive the effect of spatial down-sampling. Consider a scenario where the first frame in the sequence (I frame) has been transmitted to the client at full resolution, and the subsequent P frames are transmitted at a reduced resolution. By displacing each of the higher resolution counterparts of a given coarse vertex by the same amount as it's displacement from the corresponding coarse vertex, a speedy algorithm can be developed to provide higher resolution displays. The higher resolution frames are perceptually indistinguishable from those obtained by originally streaming higher resolution data as elaborated in the results section. The proposed algorithm, while being able to transmit data only at a limited number of resolutions, has very less computational complexity The ICP module is most compute intensive and determines the complexity of the overall algorithm. However, when data is coded at reduced resolutions, the algorithmic complexity becomes a lot less dependent on the complexity of ICP. Realization of real time compression and streaming of 3D dynamic data, by transmitting data at reduced resolutions, and reconstructing higher resolution data, could form the basis for future work. The possibility of transmitting data at resolutions other than those obtained from the coarsening algorithm could also be studied. Then the resolution of streamed data would only be limited by the available network bandwidth.

5. RESULTS

For the *Chicken crossing* sequence, reconstructed sequences for various resolutions were generated using partition sizes of about 100 vertices. The coarsening algorithm outputs 5 resolution levels starting from the original graph (3030 vertices) to the coarsest (200 vertices). We find that only resolution levels 1 (3030 vertices), 2 (1540 vertices) and 3 (792 vertices) provide reconstructed sequences of acceptable quality. Also, vertices were added to the mesh reconstructed at level 2 to obtain a level 1 mesh. The results are depicted in Figure 4. Clearly the mesh obtained by adding vertices is almost perceptually indistinguishable from the frame reconstructed at level 1.

6. CONCLUSION

A multi-resolution streaming framework that can effectively cater to the real time scenario of varying network bandwidth as well as satisfy requirements of lower end clients has been presented. This innovative algorithm is able to reconstruct data at a resolution higher than that of the bit stream data. Only the graphs generated by the coarsening algorithm are considered as the possible resolutions for transmission.

7. REFERENCES

[1] J. Rossignac, Edgebreaker: "Connectivity Compression for Triangle Meshes", *IEEE Trans. on Visualization and Computer Graphics*, Vol. 5, no. 1, pp. 47-61, 1999

[2] S Gumhold, W Strasser., "Real Time Compression of Triangle Mesh Connectivity", *Proc. of SIGGRAPH,* pp. 133-140,1998.

[3] M. Deering, "Geometry Compression", *Proc. of SIGGRAPH,* pp. 13-20, 1995.

[4] S.Gupta, K. Sengupta et al, "Compression of 3D dynamic geometry data using iterative closest point algorithm",*Comput. Vis. Image Understanding*, vol. 87, pp. 116-130, 2002.

[5] J. Lengyel, "Compression of Time Dependent Geometry", *Symposium on Interactive 3D Graphics*, pp. 89-95, 1999.

[6] M. Alexa, W. Muller, "Representing animations by principal components", *EUROGRAPHICS,* 19(3), pp. 411-418, 2000.

[7] Socrates Varakliotis, Stephen Hailes et al, "Optimally smooth error resilient streaming of 3D wireframe animations", *Visual Comm. and Image Processing*, pp. 1009-1022, 2003.

[8] Marius Preda, Titus Zaharia et al, "3D Body Animation and coding within a MPEG4 compliant framework", *Proceedings IWSNHC3DI'99,* pp. 74-78, September 1999.

[9] B.Hendrickson and R. Leland, "A multilevel algorithm for partitioning graphs", *Tech. Rep. SAND93-1301*, 1993.

[10] R. Pajarola , J. Rossignac, "Compressed Progressive meshes", *Technical Report GIT-GVU-99-05*, 1998.

[11] D. Cohen-Or, D. Levin, and O. Remez. "Progressive compression of arbitrary triangular meshes". *Proc. of IEEE Visualization '99*, 1999, 67-72.

[12] S. Capell, S. Green et al, "A Multiresolution framework for dynamic deformations", *Proc. of the ACM SIGRAPH,* 2002, 41-48.

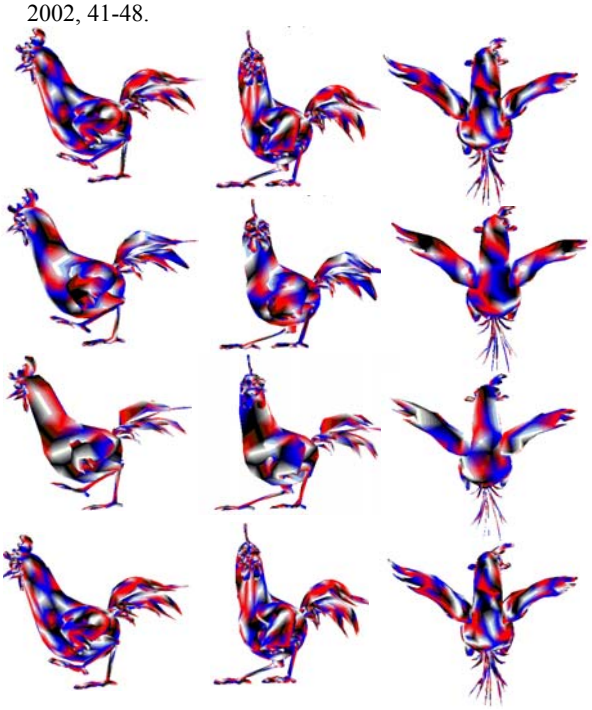

Figure 4: Reconstructed mesh at Level 1 (Row 1), Level 2 (Row 2), Level 3 (Row 3) and Level 1 mesh reconstructed from Level 2 (Row 4). Notice that the 1st and 4th rows are almost perceptually indistinguishable.