A System Level Framework for Streaming 3-D Meshes over Packet Networks

Ghassan Al-Regib and Yucel Altunbasak

School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0250, USA
{gregib,yucel}@ece.gatech.edu

Abstract. In this paper, a system-level framework is proposed for 3-D graphics streaming. The proposed architecture is scalable with respect to the variations in both bandwidth and channel error characteristics. We consider the end-to-end system, and jointly optimize pre-processing and post-processing solutions. In particular, forward error correction (FEC) and multiple description (MD) codes are applied to the base-layer mesh; FEC codes are applied to enhancement layer connectivity data; and unequal error protection (UEP) and error-concealment methods are applied to the geometry data. Re-transmission is utilized wherever it is applicable as determined by the delay requirements. Furthermore, the performance analysis of the system is also provided.

1 Introduction

The internet evolves from being a place for users to communicate through media objects into a place where users meet and interact with each other within a virtual world. 3-D graphics applications are the driving force for this "virtual reality over IP" concept. These applications utilize highly detailed 3-D models, giving rise to huge amount of data to be stored, transmitted, and rendered. To alleviate these limitations, several single-layer compression techniques have been developed to minimize the storage space for a given model [1,2]. However, the resulting compressed mesh still requires significant time to be fully downloaded before being displayed on the client's screen. To reduce this time, progressive compression techniques have been designed so that a coarse representation of the model is downloaded first and then the enhancement layers are streamed out to the client [3,4,5].

In a typical network, there are several factors that considerably affect the quality of the received 3-D mesh. These factors are:

1. The bandwidth between the server and the client
2. The clients' graphics card capabilities
3. The channel/link packet bit-error characteristics
4. The channel/link packet loss characteristics

P. Lorenz (Ed.): ICN 2001, LNCS 2094, pp. 745–753, 2001.

The existing 3-D graphics compression techniques are designed to reduce the bandwidth and storage requirements, and do not deal with transmission aspects, hence they only provide a solution for the first and the second constraint. In this paper, we propose a system-level framework that provides a solution considering all these aspects.

The research efforts in 3-D graphics are mainly limited to compression and simplification. Not many research efforts have been undertaken to stream out 3-D graphics except MPEG-4 related research. However, even MPEG-4 3-D mesh research efforts concentrate on compression and integration with video, but they do not provide error handling methods specific to 3-D graphics streaming and/or transmission. In this paper, we provide a robust 3-D graphics streaming framework that borrows several ideas from existing robust video streaming research, but adapts them for our particular purpose. In particular, we consider server-side pre-processing, network processing and client-side post-processing at the same time, and jointly optimize them.

2 Progressive Compression

Progressive compression schemes can be categorized into two main classes: Wavelet-based [5] and vertex-split-based [3,4] schemes. We adopted the latter class due to its popularity, simplicity and its good performance for a variety of 3-D models. A progressive compression scheme has been proposed by Hoppe [3] for the first time. In his work, Hoppe simplifies a mesh by applying a sequence of edge-collapse operations to arrive at the coarsest mesh M^0. In practice, this base mesh, M^0, is first transmitted to the client, and then a sequence of vertex split operations are streamed out until the fully refined mesh, M^n, is constructed at the client. These two operations are illustrated in Figure 1. Each vertex split command specifies: i) the new vertex, ii) two new faces, and iii) the attributes of the new vertices and the faces.

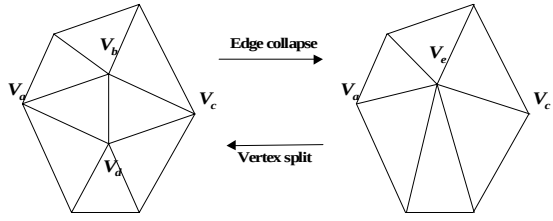

Fig. 1. Edge-collapse and vertex-split commands.

Pajarola and Rossignac [4] improved Hoppe's work by grouping vertex split operations into batches, where each batch splits half of the previously decoded vertices using edge collapse operations. The encoder works as follows:

1. The base mesh, M^0, is compressed using a single-level 3-D mesh compression scheme, such as [2].
2. At each level i, the vertex spanning tree is traversed. If the vertex is split, a bit "1" is inserted; otherwise a bit "0" is inserted into the bit-stream.
3. For the vertex v to be split, the edges are stored clockwise starting from the edge connecting v to its parent in the spanning tree. The pair of indices corresponding to the two edges affected by the vertex-split operation is encoded using $\lceil \log_2(\frac{d}{2}) \rceil$ bits, where d is the degree of vertex v.
4. A prediction algorithm is applied to predict the position of the vertices from the already encoded vertices. The prediction error is then quantized and entropy-encoded.

Accordingly, the decoder works as follows:

1. The base mesh is received and decoded.
2. Then, the stream containing the batches (*e.g.,* enhancement layers) is received. The bit-stream is parsed one bit at a time. If the current bit is "0", then the next bit is read. However, if it is "1", then the vertex is split, and the next $\lceil \log_2(\frac{d}{2}) \rceil$ bits are decoded to identify which pair of edges are affected by the split. Then, the prediction error is decoded completing the vertex split operation for this particular vertex.
3. Go to step 2 until the bit rate is completely utilized or the whole bit stream is parsed.

The time to display the mesh is the time needed to download the base mesh. As it will be shown in the next section, this accounts for less than 10% of the total download time. However, although progressive compression methods produce a bandwidth scalable bit-stream, they are not robust against packet bit-errors and/or packet losses. The next section discusses the proposed error-handling mechanism for streaming 3-D meshes over packet networks.

3 Error-Handling Framework

3.1 The Bit-Stream Content

The bit-stream is composed of three main parts as follows:

1. *The base mesh data:* Typically, simplifying mesh M^{i+1} to mesh M^i reduces the number of vertices by 30% [4]. Hence, applying n-level batches produces a base mesh, M^0, with $(\left(\frac{2}{3}\right)^n \times V)$ vertices, where V is the number of vertices in the original 3-D mesh. We used the Touma-Gosta (TG) single-level compression algorithm due to its superior performance upon other algorithms. On the average, this method achieves a compression ratio of 10.4 bits per vertex for both the mesh connectivity and the geometry [2]. As a result, the output stream for the base mesh contains $(10.4 \times \left(\frac{2}{3}\right)^n \times V)$ bits.
2. *Enhancement layer connectivity data:* The batch b_i that produces M^{i+1} from M^i contains $(\frac{2}{3} \times V)$ bits that specify either a vertex is being split or not.

In addition, this batch contains $(\frac{2}{3} \times V \times \lceil \log_2(\frac{d_j}{2}) \rceil)$ bits for each vertex split since the two collapsed edges for a vertex can be encoded by $\lceil \log_2(\frac{d_j}{2}) \rceil$ bits, where d_j is the degree of vertex j. Hence, this batch contains $(\frac{1}{2} \times \frac{2}{3} \times V \times \log_2(\frac{d_j}{2}))$ bits, where the $\frac{1}{2}$ factor accounts for the fact that half of the vertices in M^i are split to produce M^{i+1}. Thus, the total number of bits to represent the mesh connectivity in all batches is

$$\sum_{i=1}^{n} \left(\frac{2}{3}\right)^{n-(i-1)} \times V + \sum_{i=1}^{n} \frac{1}{2} \times \left(\frac{2}{3}\right)^{n-(i-1)} \times \lceil \log_2(\frac{d}{2}) \rceil \qquad (1)$$

In practice, 5 bits are adequate to specify the two edges being collapsed per vertex split. In other words, $\lceil \log_2(\frac{d_j}{2}) \rceil$ in the above equation can be replaced by (5) [4].

3. *Enhancement layer geometry data:* Each batch contains bits for the prediction error. The size of this segment is variable due to entropy encoding. Assuming each batch, b_i, contains E_i bits describing the geometry information, then the total number of bits in the enhancement bit stream describing geometry information is $\sum_{i=1}^{n} E_i$ bits.

Table 1 lists these statistics for different 3-D mesh models. The data in the table will be used in performance analysis of the proposed system in the following subsections.

3.2 Error-Handling and Packet Format

According to the bit-stream content-analysis in section 3.1, the base mesh constitute approximately 1-3% of the total mesh data (See Table 1). This portion of the bit stream is of utmost importance, hence, it should be strongly protected against channel degradations and delivered error-free to the client within a certain period of time as specified by the synchronization protocol. We protect the base mesh bit-stream by two mechanisms: i) Reed-Solomon (RS) code against the bit-errors, and ii) multiple description codes (MDC) against the packet losses [6]. MDC is only utilized in cases where packet re-transmission is prohibited due to the stringent time-delay contraints.

The enhancement layer mesh connectivity information also needs to be strongly protected against channel errors. Any bit-error and/or packet-loss (in this part of the bit-stream) at a certain level causes catastrophic errors in all subsequent levels. For example, if a single bit describing a vertex split command is received erroneously, then it will cause a considerable distortion in the mesh in all subsequent levels. Therefore, we apply strong channel encoding to the enhancement layer data by utilizing the RS code. However, we do not employ MDC in this case, since the connectivity data constitute a substantial percentage (30%) of the total rate. The packet-errors are handled through re-transmission at the expense of increased delay. However, since the base-layer mesh is already

Table 1. The bit stream content for different models. The numbers in the parenthesis are percentage of the bits in that category to the total number of bits in the bit stream.

Model (♯ Vertices)	♯ Levels (n)	M^0 (bits)	Batch-0 (bits)	Batch-n (bits)	Total Batches (bits)	Geometry (bits)
horse	8	4048	1362	24970	70810	153190
(10811)		(1.7)	(0.6)	(11.0)	(31.2)	(67.4)
skull	8	4101	1380	25300	71730	163940
(10952)		(1.7)	(0.58)	(10.6)	(29.9)	(68.4)
fohe	7	2272	764	9250	25730	80713
(4005)		(2.2)	(0.75)	(9.1)	(25.4)	(79.4)
fandisk	7	3674	1236	14960	41590	99411
(6475)		(2.5)	(0.85)	(10.3)	(28.7)	(68.6)
triceratops	7	1607	540	6540	18190	32718
(2832)		(3.1)	(1.03)	(25.5)	(34.6)	(62.1)
shape	7	1454	489	5920	16460	35236
(2562)		(2.7)	(0.92)	(11.1)	(30.9)	(66.2)
bunny	9	8609	2897	53110	231040	570570
(34834)		(1.1)	(0.36)	(6.6)	(28.5)	(70.4)
phone	10	13545	4558	83560	55360	1144900
(83044)		(0.8)	(0.27)	(4.9)	(33.2)	(66.8)
happy	9	12306	4141	75920	330270	875350
(49794)		(1.0)	(0.34)	(6.2)	(27.1)	(71.8)

available at the client, the delay induced by the re-transmission of the enhancement layer connectivity data packets is more tolerable to the end-user in most applications.

The remaining portion of the bit stream is the geometry information in the batches. Such information constitutes more than 60% of the total bit stream. Here, we propose to utilize unequal error protection (UEP). We label each vertex either as "critical vertex" or "non-critical-vertex" depending on whether its position can be predicted from already decoded vertex locations within a predetermined threshold. Since the non-critical vertex locations can be interpolated, no error protection is applied. On the other hand, "critical vertices" are protected through RS since there is no way of recovering/concealing these vertices. If the synchronization and delay requirements permits, re-transmission is used in all cases.

The bit stream is organized as shown in Figure 2. In this figure, RSR, C_{b_i} and G_{b_i} stand for Reed-Solomon Redundancy, enhancement layer mesh connectivity and geometry information in the batch b_i, respectively. To prevent error propagation, re-synchronizing codewords are inserted after every batch as well as the base mesh.

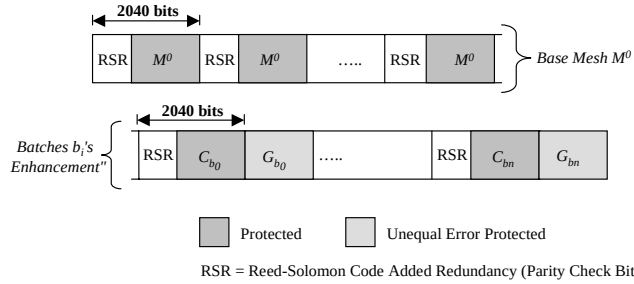

Fig. 2. Bit stream format using $RS(255, 223)$ code.

3.3 Performance Analysis

The proposed pre-processing methods, such as FEC and MD codes, provide error-resilience at the expense of increased redundancy. This added redundancy causes an increase in i) the bandwidth requirements, and in ii) the time-delay. Particularly, such delay may considerably affect synchronization among the clients. However, we observe that if the link suffers from packet errors/losses considerably, then the delay induced by the proposed error-resilience methods is much smaller than the delay caused by solely re-transmission based solutions. Furthermore, application of error concealment at the client side to recover the geometry information reduces the re-transmission rate.

Table 2 shows the redundancy introduced by employing $RS(255, 223)$ code for a number of 3-D meshes. The last two columns in this table refer to the amount of time consumed by the redundancy added by the FEC for two different links, $56Kbps$ and $1Mbps$, respectively.

As depicted in Table 2, on a LAN, the redundancy added by the FEC code is in the order of milliseconds while on a $56K$-link, the delay is in the range of $0.06 \sim 1.4$ seconds. The latter delay is significant enough to affect the synchronization in some interactive applications. Fortunately, this simple analysis is not an accurate assessment of the proposed system since the bandwidth scalability prevents streaming the whole bit-stream over such a low bandwidth link, but rather the base mesh with the first few enhancement layers are streamed out to the client. This observation leads to the conclusion that the amount of data being streamed out is determined mainly by two factors: i) the allowable delay from the synchronization algorithm, α (in seconds), and ii) the link bandwidth, β (in bps).

The number of transmitted bits should be less than $(\alpha \times \beta)$ in our method using the $RS(255, 223)$ code. That is,

$$(M^0 + 256 \times \left\lceil \frac{M^0}{1784} \right\rceil + \sum_{i=0}^{n} (b_i + 256 \times \left\lceil \frac{b_i}{1784} \right\rceil)) \leq \alpha \times \beta \qquad (2)$$

The server needs to satisfy the constraint in Equation 2 for each user and the corresponding synchronization and bandwidth requirements. However, the

Table 2. The delay introduced by applying $RS(255, 223)$ on two different links for various 3-D meshes.

Model	Added Redundancy (bits)	Delay on 56Kbps (sec.)	Delay on 1Mbps (sec.)
horse	12032	0.21	0.012
skull	12288	0.21	0.012
fohe	5120	0.09	0.005
fandisk	7680	0.13	0.007
triceratops	3840	0.07	0.004
shape	3584	0.06	0.003
bunny	35328	0.62	0.034
phone	82432	1.44	0.079
happy	50688	0.88	0.048

following inequality has to be satisfied to stream out the base mesh:

$$M^0 + 256 \times \left\lceil \frac{M^0}{1784} \right\rceil \leq \alpha \times \beta \tag{3}$$

Equation 3 describes both the synchronization and the bandwidth scalability requirements.

Note that we do not limit the server to use the $RS(255, 223)$ code. In general, it can choose among several available FEC codes that will satisfy the conditions in Equation 4 and Equation 5:

$$(M^0+(n-k)\times m\times \left\lceil \frac{M^0}{k \times m} \right\rceil + \sum_{i=0}^{n} (b_i + (n-k) \times m \times \left\lceil \frac{b_i}{k \times m} \right\rceil)) \leq \alpha \times \beta, \tag{4}$$

$$M^0 + (n-k) \times m \times \left\lceil \frac{M^0}{k \times m} \right\rceil \leq \alpha \times \beta \tag{5}$$

where the $RS(n, k, m)$ code is used. Such a selection guarantees that the client will at least receive a coarse representation of the mesh in the allowable amount of time.

Equations 4 and 5 can be re-written in terms of $t = \frac{n-k}{2}$ to result in Equations 6 and 7, respectively.

$$(M^0+16\times t\times \left\lceil \frac{M^0}{8 \times (255-2 \times t)} \right\rceil + \sum_{i=0}^{n} (b_i+16 \times t \times \left\lceil \frac{b_i}{8 \times (255-2 \times t)} \right\rceil)) \leq \alpha \times \beta \tag{6}$$

$$M^0 + 16 \times t \times \left\lceil \frac{M^0}{8 \times (255-2 \times t)} \right\rceil \leq \alpha \times \beta \tag{7}$$

Therefore, the server's task is to: i) calculate the value of t that satisfies Equation 7 and maximizes the left hand side of the inequality; and ii) continue streaming out as long as Equation 6 is satisfied.

Equation 7 is a constrained optimization problems. The optimum t can be computed for a given mesh, bandwidth, and delay parameter by solving this equation. Figure 3 illustrates t versus α for different bandwidths for the 3-D mesh *fohe*. As shown, for a specific value of α, t is scaled according to the bandwidth so that Equation 7 is satisfied and hence the client receives the base

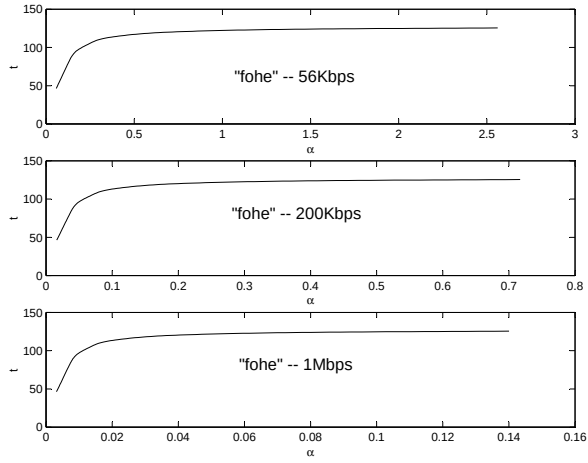

Fig. 3. Three plots of t vs. α for three different links for the mesh *fohe*, which has 4005 vertices.

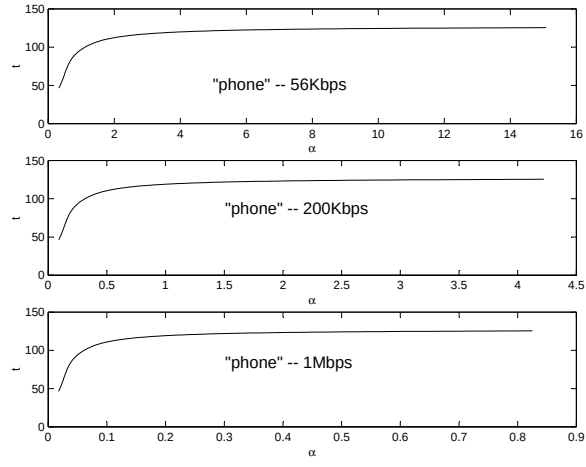

Fig. 4. Three plots of t vs. α for three different links for the mesh *phone*, which has 83044 vertices.

mesh during the allowable window of time. Similarly, Figure 4 plots the same curves for a more complicated model, the *phone* mesh. This increase in the number of vertices has an impact of lowering t for the same delay, α, over the same link, β.

4 Conclusion

In this paper, a system-level framework is proposed for 3-D graphics streaming. The proposed architecture is scalable with respect to variations in both bandwidth and channel error characteristics. We consider end-to-end system, and jointly optimize pre-processing and post-processing solutions. In particular, FEC and MD codes are applied at the client-side to the base-layer mesh; FEC codes are applied to enhancement layer connectivity data; and UEP and error-concealment methods are applied to the geometry data. Re-transmission is utilized wherever it is applicable as determined by the delay requirements.

References

1. G. Taubin and J. Rossignac, "Geometric compression through toplogical surgery," *ACM Transactions on Graphics*, vol. 17, no. 2, pp. 84–115, April 1998.
2. C. Touma and C. Gotsman, "Triangle mesh compression," in *Proceedings of Graphics Interface*, Vancouver, Canada, June 1998.
3. H. Hoppe, "Progressive meshes," in *Proceedings ACM SIGGRAPH'96*, 1996, pp. 99–108.
4. R. Pajarola and J. Rossignac, "Compressed progressive meshes," *IEEE Transactions on Visualization and Computer Graphics*, vol. 6, no. 1, pp. 79–93, January-March 2000.
5. A. Khodakovsky, P. Schroder, and W. Sweldens, "Progressive geometry compression," in *Proceedings of the on computer graphics*, 2000, pp. 271–278.
6. Y. Wang and Q.-F. Zhu, "Error control and concealment for video communication: A review," in *Proceedings of the IEEE*, May 1998, pp. 974–997.